Evasion by De-Optimization

Ege BALCI - 2024

~# whoami

- **EGE BALCI** - Security Researcher
- Threat Intelligence Division Manager @ PRODAFT
- Author of multiple A.V. evasion projects


@egeblc


github.com/EgeBalci


infosec.exchange/@ege


linkedin.com/in/egebalci

~# whoami

- **EGE BALCI** - Security Researcher
- Threat Intelligence Division Manager @ PRODAFT
- Author of multiple A.V. evasion projects

@egeblc

github.com/EgeBalci

infosec.exchange/@ege

linkedin.com/in/egebalci

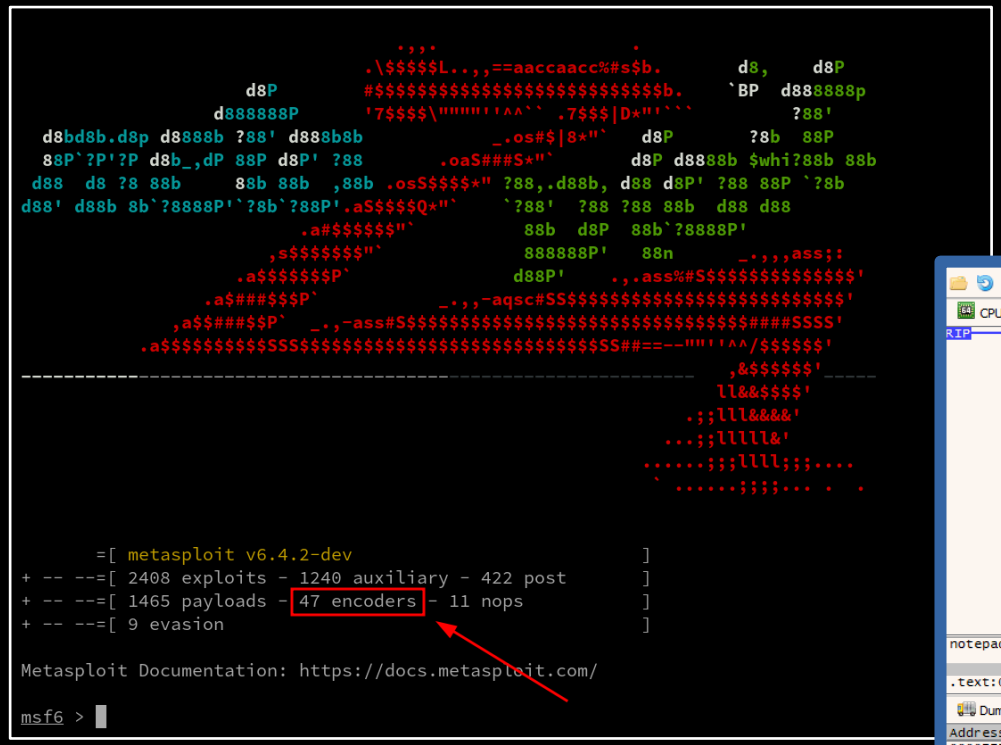
~# What is the challenge?

~# How A.V. Detects Malware?

- Known Hash Values
  - Crypto Hashes (MD5,SHA...)
  - Vhash
  - SSDEEP
  - Imphash
  - Authentihash
- Code/Data Patterns
  - Yara
  - Snort
  - Suricata
  - Zeek
- Heuristic Patterns
  - File format integrity
  - Abnormal entrophy
  - Imported libraries/functions
- Behavioral Patterns
  - File read/write ops.
  - Registery read/write ops.
  - Network traffic
  - Memory contents
  - Function/system calls
- AI Engines
  - ML classifiers
  - Similarity Detection

~# Rule Based Detection

~# Code Encoding

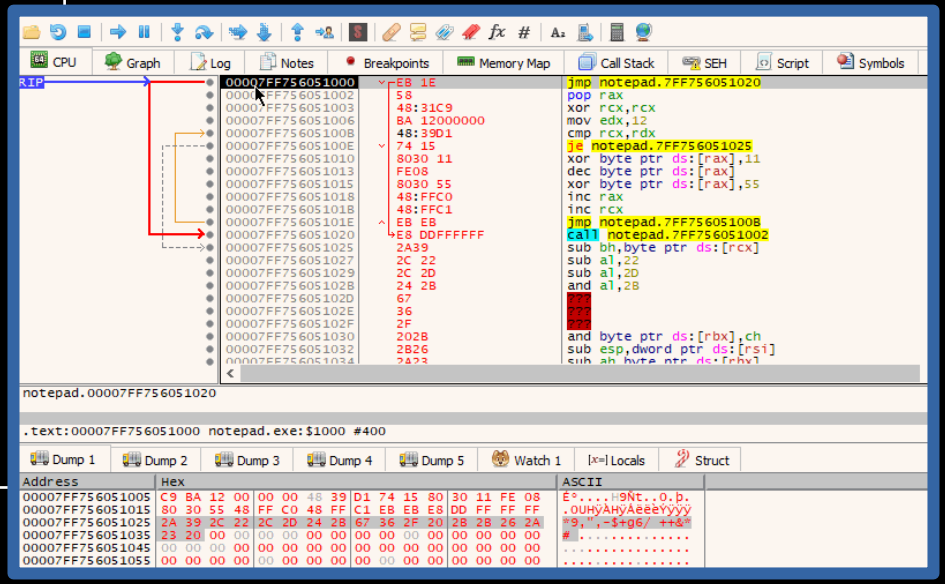


README MIT license

仕方がない

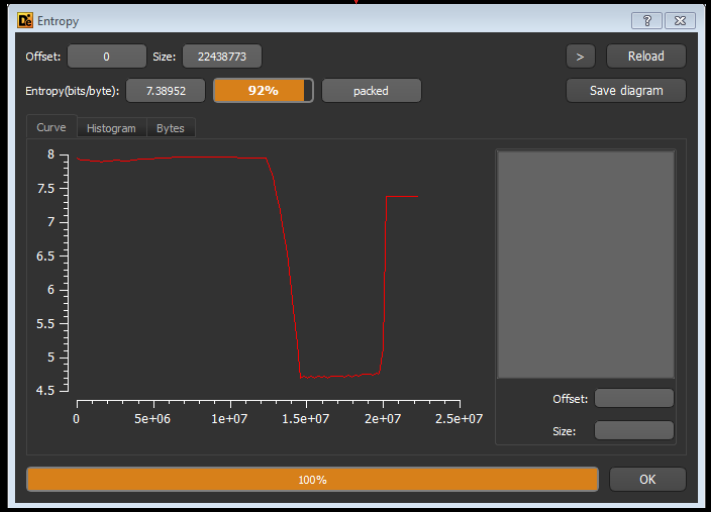
version 2.0.1 | go report A+ | issues 1 open | license MIT | twitter @egebic

SGN is a polymorphic binary encoder for offensive security purposes such as generating statically undetecable binary payloads. It uses a additive feedback loop to encode given binary instructions similar to LSFR. This project is the reimplementation of the original Shikata ga nai in golang with many improvements.



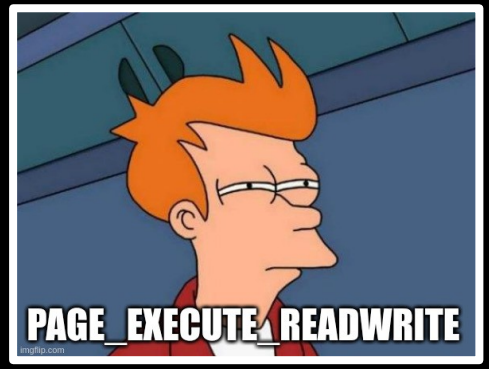
~# Code Entropy

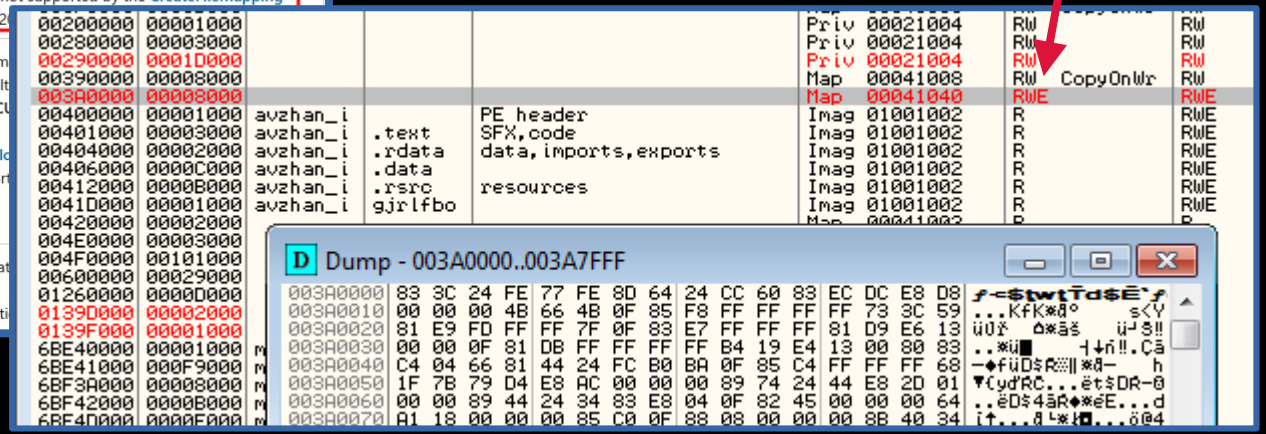
```
.data:004030C3                  cmpsd
.data:004030C4                  jecxz   short near ptr loc_4030CA+1
.data:004030C6                  sub     ebx, [eax]
.data:004030C8                  inc     esi
.data:004030C9                  dec     esi
.data:004030CA
.data:004030CA loc_4030CA:                              ; CODE XREF: .data:004030C4↑j
.data:004030CA                  fimul   word ptr [edi-0Ah]
.data:004030CD                  scasd
.data:004030CE                  and     [eax-56h], dh
.data:004030D1                  daa
.data:004030D2                  mov     cl, 0EAh
.data:004030D4                  and     [edi-265F7DDBh], esi
.data:004030DA                  fdivr   qword ptr [eax]
.data:004030DC                  pop     ebx
.data:004030DD                  imul    ebp, [eax-19h], 0D6438E9Ch
.data:004030E4                  jns     short loc_403109
.data:004030E6                  aas
.data:004030E7                  dec     edx
.data:004030E8                  sub     eax, 0D1ED5790h
.data:004030ED                  push    ss
.data:004030EE                  test    al, 0A3h
.data:004030F0                  mov     esi, 2D57C16Ch
.data:004030F5                  icebp
.data:004030F6                  db      3Eh
.data:004030F6                  popf
.data:004030F8                  add     ecx, 60h ; '`'
.data:004030FB                  cmc
.data:004030FC                  mov     dl, 71h ; 'q'
.data:004030FE                  add     [ecx+7Eh], ah
.data:00403101                  push    ecx
.data:00403102                  cmp     dword ptr [ecx], 0E94AB1D5h
.data:00403108                  cmc
.data:00403109
.data:00403109 loc_403109:                              ; CODE XREF: .data:004030E4↑j
.data:00403109                  and     byte ptr [ebx+52h], 0C6h
.data:0040310D                  mov     ebp, 284A4CB9h
.data:00403112                  aad     0E9h
.data:00403114                  mov     dl, 0E7h
.data:00403116                  dec     eax
.data:00403117                  xchg    eax, ecx
.data:00403118                  xchg    eax, edx
.data:00403119                  mov     eax, 6F8B75D9h
.data:00403119 ; ---------------------------------------------------------------------------
.data:0040311E                  db 0FEh
.data:0040311F                  db  34h ; 4
.data:00403120                  db  5Bh ; [
.data:00403121                  db  1Fh
.data:00403122                  db  92h
```



~# RWE Memory

Constant/value	Description
PAGE_EXECUTE 0x10	Enables execute access to the committed region of pages. An attempt to write to the committed region results in an access violation. This flag is not supported by the CreateFileMapping function.
PAGE_EXECUTE_READ 0x20	Enables execute or read-only access to the committed region of pages. An attempt to write to the committed region results in an access violation. **Windows Server 2003 and Windows XP:** This attribute is not supported by the CreateFileMapping function until Windows XP with SP2 and Windows Server 2003 with SP1.
PAGE_EXECUTE_READWRITE 0x40	Enables execute, read-only, or read/write access to the committed region of pages. **Windows Server 2003 and Windows XP:** This attribute is not supported by the CreateFileMapping function until Windows XP with SP2 and Windows Server 2
PAGE_EXECUTE_WRITECOPY 0x80	Enables execute, read-only, or copy-on-write access to a m attempt to write to a committed copy-on-write page result for the process. The private page is marked as PAGE_EXECU to the new page. This flag is not supported by the VirtualAlloc or VirtualAllo **Server 2003 and Windows XP:** This attribute is not support Windows Vista with SP1 and Windows Server 2008.
PAGE_NOACCESS 0x01	Disables all access to the committed region of pages. An at committed region results in an access violation. This flag is not supported by the CreateFileMapping functi





~# RWE Is Not Ok!

- Indicates dynamic code
- Useless when ACG enabled
- Very uncommon

~# RWE Is Not Ok!

- Indicates dynamic code
- Useless when ACG enabled
- Very uncommon

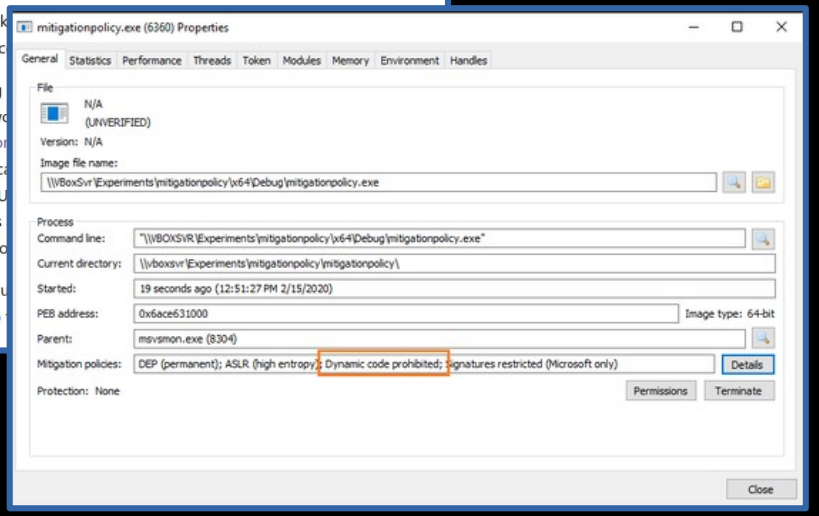
Arbitrary code guard

Description

Arbitrary code guard helps protect against a malicious attack
memory safety vulnerability and being able to execute that c

Arbitrary code guard protects an application from executing
example, from the exe itself or a dll). Arbitrary code guard wo
executable. When an application attempts to allocate memor
with read, write, and/or execute protection flags.) If the alloca
the memory allocation fails and returns an error code (STATU
attempts to change the protection flags of memory that has
flag, then the permission change fails and returns an error co

By preventing the *execute* flag from being set, the data execu
can then protect against the instruction pointer being set to



~# RWE Is Not Ok!

- Indicates dynamic code
- Useless when ACG enabled
- Very uncommon

~# Memory Scanners

README BSD-2-Clause license

PE-SIEVE

build passing | code quality A | commit activity 54/month | last commit february

release v0.3.9 | release date february | downloads 219k | downloads@latest 1.5k

License BSD 2-Clause | Windows | Ask me anything

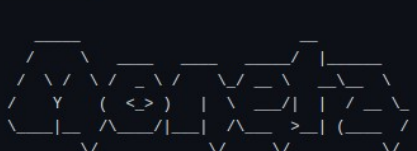
Tweet

Intro

PE-sieve is a tool that helps to detect malware running on the system, as well as to collect the potentially malicious material for further analysis. Recognizes and dumps variety of implants within the scanned process: replaced/injected PEs, shellcodes, hooks, and other in-memory patches.
Detects inline hooks, Process Hollowing, Process Doppelgänging, Reflective DLL Injection, etc.

PE-sieve is meant to be a **light-weight engine** dedicated to scan **a single process** at the time. It can be built as an EXE or as a DLL. The DLL version exposes a simple API and can be easily integrated with other applications.

README GPL-3.0 license



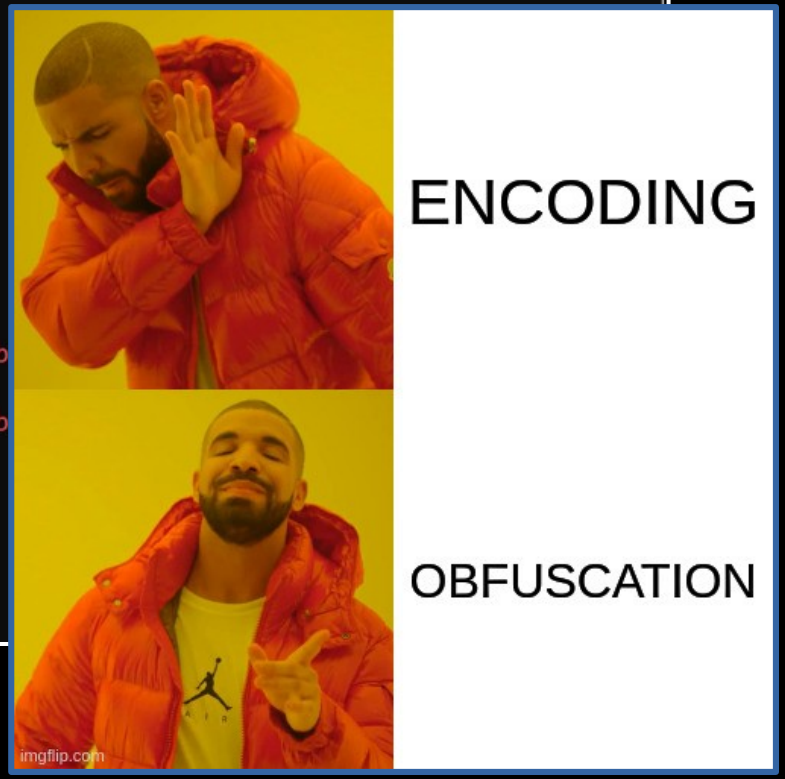
```
Moneta v1.0 | Forrest Orr | 2020

REQUIRED

-m {*|region|referenced|ioc}
-p {*|PID}

OPTIONAL

-v {detail|debug|surface}
-d
--option {from-base|statistics}
--filter {unsigned-module|clr-prvx|clr-heap|metadata-modules}
--address <memory address>
--region-size <memory region size>
```

~# Memory Scanners

```
C:\Users\[redacted]\Desktop>Moneta64.exe -p [redacted] -m ioc
   _____                          __
  /     \   ____   ____   ____   /  |_____
 /  \ /  \ /  _ \ /    \_/ __ \   __\__  \
/    Y    (  <_> )   |  \  ___/|  |  / __ \_
\____|__  /\____/|___|  /\___  >__| (____  /
        \/            \/     \/          \/

Moneta v1.0 | Forrest Orr | 2020


cmd.exe : 12256 : x64 : C:\Windows\System32\cmd.exe
  0x0000000180000000:0x00067000   | Private
    0x0000000180000000:0x00067000 | RWX      | 0x00000000 | Abnormal private executable memory
  0x0000027426000000:0x0004e000   | Private
    0x0000027426000000:0x0004e000 | RWX      | 0x00000000 | Abnormal private executable memory

... scan completed (0.625000 second duration)

C:\Users\[redacted]\Desktop>
```

~# Memory Scanners

```
C:\Users\[redacted]\Desktop>Moneta64.exe -p [redacted] -m ioc
   _____                          __
  /     \   ____   ____   ____   /  |_____
 /  \ /  \ /  _ \ /    \_/ __ \  __\__  \
/    Y    (  <_> )   |  \  ___/|  |  / __ \_
\____|__  /\____/|___|  /\___  >__| (____  /
        \/             \/     \/          \/

Moneta v1.0 | Forrest Orr | 2020


cmd.exe : 12256 : x64 : C:\Windows\System32\cmd.exe
  0x0000000180000000:0x00067000    | Private
    0x0000000180000000:0x00067000 | RWX      | 0x00000000 | Ab
  0x0000027426000000:0x0004e000    | Private
    0x0000027426000000:0x0004e000 | RWX      | 0x00000000 | Ab

... scan completed (0.625000 second duration)

C:\Users\[redacted]\Desktop>
```



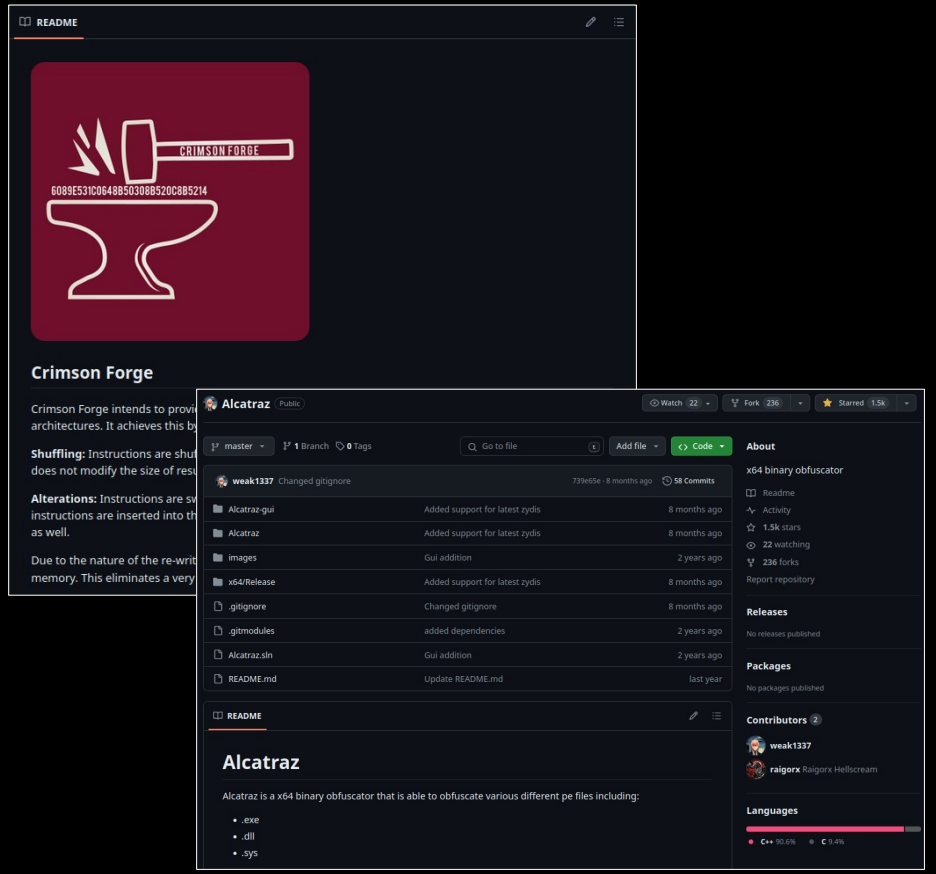
~# The Challenge

We need a way of obfuscating binary code without creating other suspicious indicators.

~# Prior Work

Binary Obfuscators:

- https://github.com/zeroSteiner/crimson-forge
  - Shuffling
  - Alterations
  - Re-ordering
- https://github.com/weak1337/Alcatraz
  - Obfuscation of immediate moves
  - Control flow flattening
  - ADD mutation
  - LEA obfuscation
  - Import obfuscation
  - Anti disassembly



~# Prior Work

Lea obfuscation

The lea obfuscation is quite simple yet effective. We move a different location into the register and decrypt it afterwards. This way, reverse engineers can't cross reference certain data / functions.
Let's say we find the following instruction: `lea rcx, [0xDEAD]`
We will mutate it to:

```
pushf
lea rcx, [1CE54]
sub rcx, EFA7
popf

rcx -> 0xDEAD
```

~# Prior Work

Alcatraz_LEA_Transform.yar

Raw

```
rule Alcatraz_LEA_Transform {
    strings:
        // pushf       > 66 9c
        // lea ?, [?]  > 48 8d ?? ?? ?? ?? ??
        // sub ?, ?    > 48 81 ?? ?? ?? ?? ??
        // popf        > 66 9d
        $lea_transform = { 66 9c 48 8d ?? ?? ?? ?? ?? 48 81 ?? ?? ?? ?? ?? 66 9d }
    condition:
        $lea_transform
}
```

~# Prior Work

Alcatraz_LEA_Transform.yar — Raw

```
rule Alcatraz_LEA_Transform {
    strings:
        // pushf       > 66 9c
        // lea ?, [?]  > 48 8d ?? ?? ?? ?? ??
        // sub ?, ?    > 48 81 ?? ?? ?? ?? ??
        // popf        > 66 9d
        $lea_transform = { 66 9c 48 8d ?? ?? ?? ?? ?? 48 81 ?? ?? ?? ?? ?? 66 9d }
    condition:
        $lea_transform
}
```

You can only search sequences not operands.

~# The Challenge

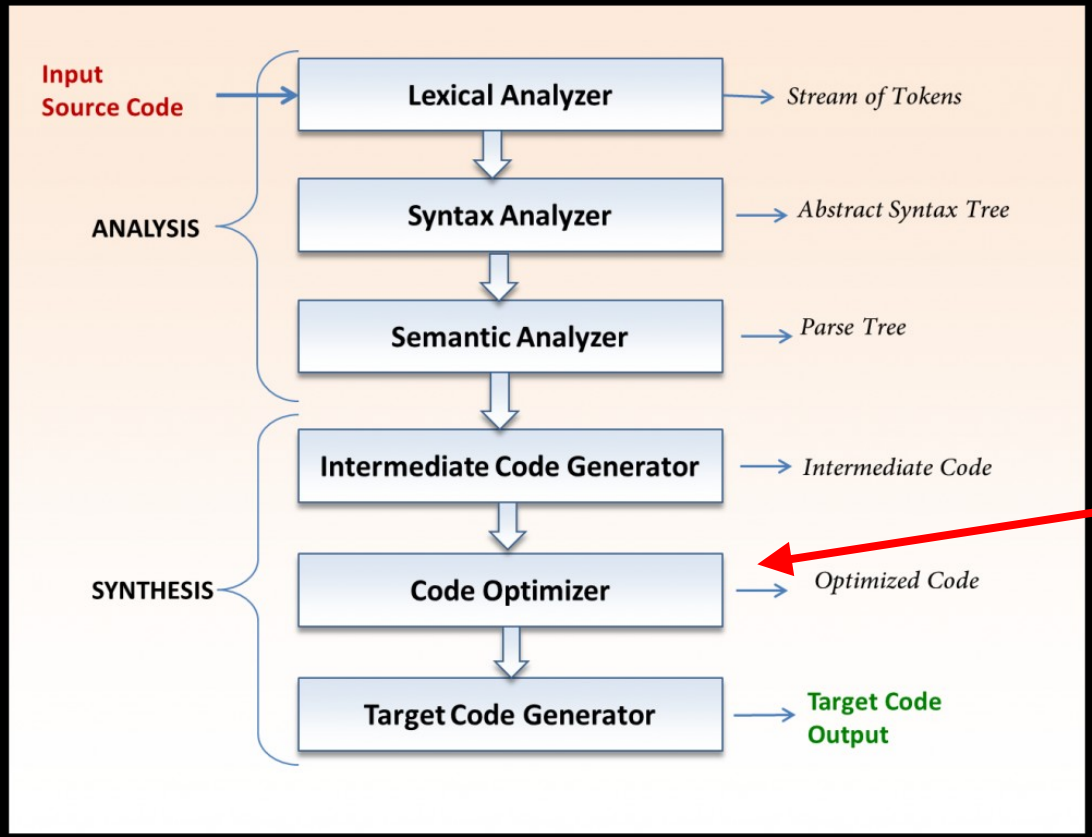
We need a way of obfuscating binary code without creating other suspicious indicators.

GOALS

- No self-modifying code! (no RWE)
- Produce common instruction sequences
- Make it look like compiler generated
- Include most instruction types
- Keep the enthropy low

~# The Solution

~# Code Optimization



~# Code Optimization

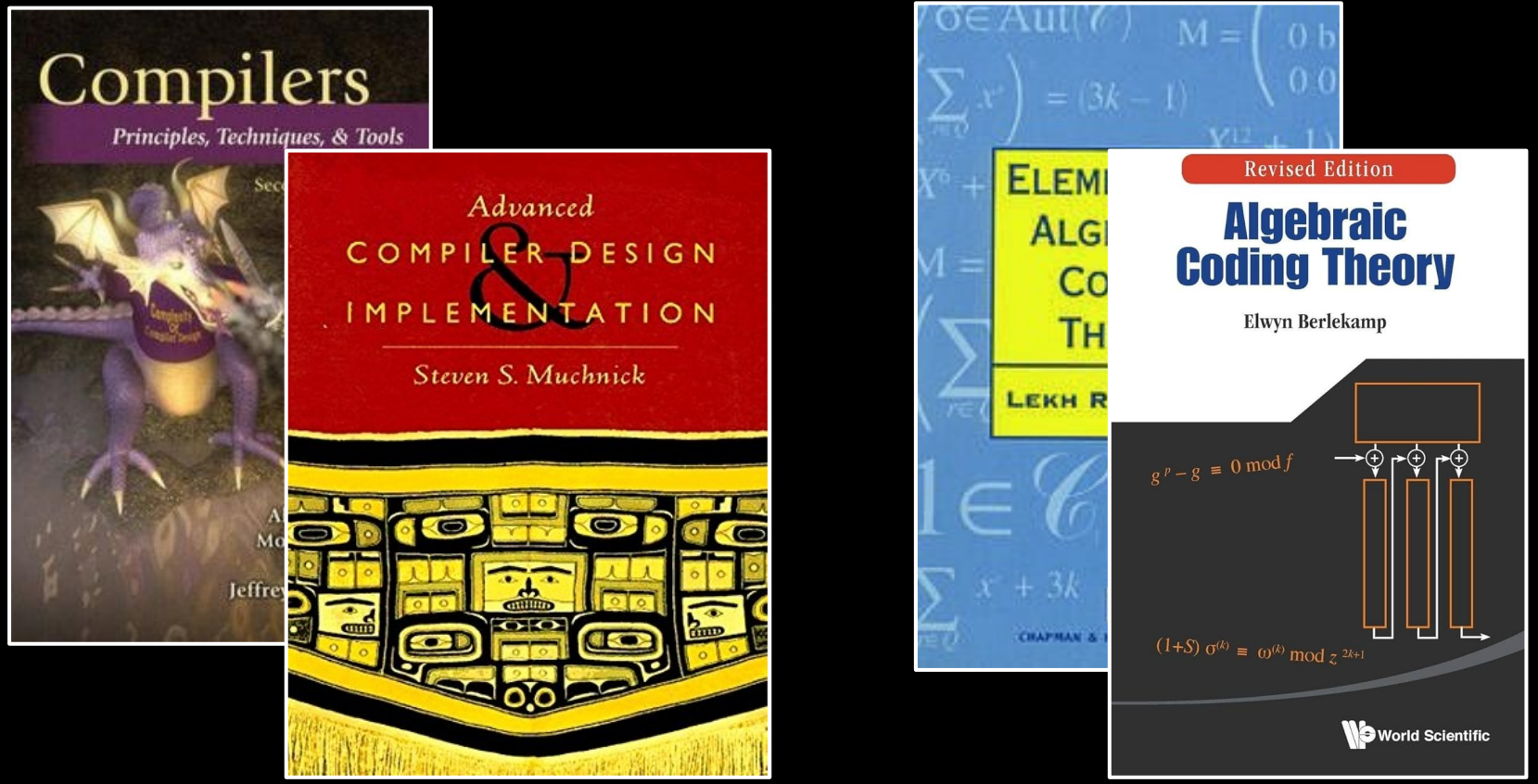
```
int sum = 0;
for (int i = 1; i <= n; i++) {
    if (i % 2 == 0) {
        sum += i;
    }
}
```

```
sum = (n / 2) * (n / 2 + 1);
```

~# Code Optimization

```
add al,10h --------------------> \x04\x10
add al,10h --------------------> \x80\xC0\x10

adc al,0DCh -------------------> \x14\xDC
adc al,0DCh -------------------> \x80\xD0\xDC

sub al,0A0h -------------------> \x2C\xA0
sub al,0A0h -------------------> \x80\xE8\xA0

sub eax,19930520h -------------> \x2D\x20\x05\x93\x19
sub eax,19930520h -------------> \x81\xE8\x20\x05\x93\x19

sbb al,0Ch --------------------> \x1C\x0C
sbb al,0Ch --------------------> \x80\xD8\x0C

sbb rax,221133h ---------------> \x48\x1D\x33\x11\x22\x00
sbb rax,221133h ---------------> \x48\x81\xD8\x33\x11\x22\x00
```

~# How Compiler Optimize Code?



~# Code Optimization

```
int sum = 0;
for (int i = 1; i <= n; i++) {
    if (i % 2 == 0) {
        sum += i;
    }
}
```

$$\frac{1}{4}(2n+1)^2 - \frac{1}{4}$$

$$n(n+1)$$

```
sum = (n / 2) * (n / 2 + 1);
```

~# Code De-Optimization

```
sum = (n / 2) * (n / 2 + 1);
```

$$n(n+1)$$

$$\frac{1}{4}(2n+1)^2 - \frac{1}{4}$$

```
int sum = 0;
for (int i = 1; i <= n; i++) {
    if (i % 2 == 0) {
        sum += i;
    }
}
```

~# Code Optimization

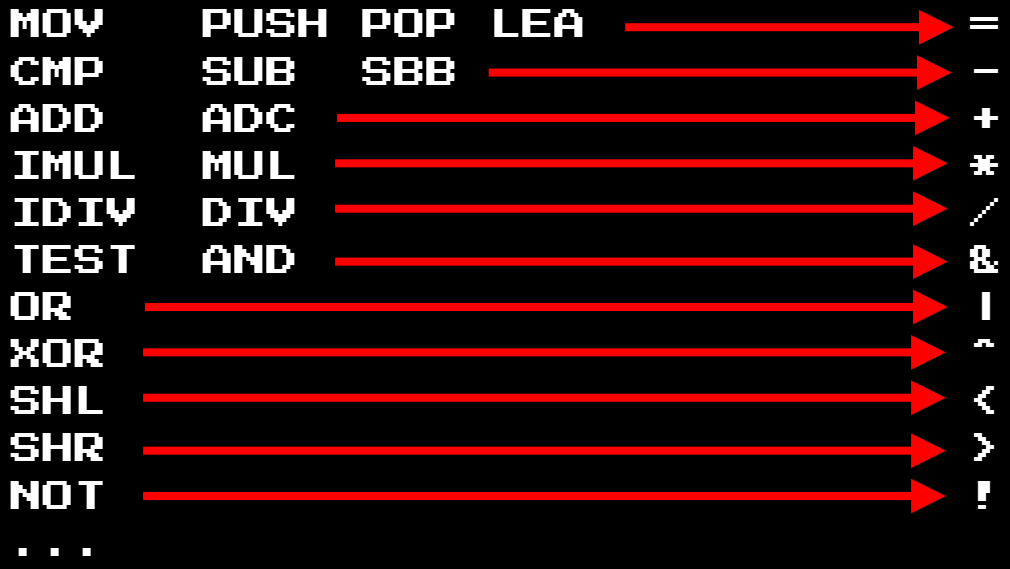
```
int sum = 0;
for (int i = 1; i <= n; i++) {
    if (i % 2 == 0) {
        sum += i;
    }
}
```

```
sum = (n / 2) * (n / 2 + 1);
```

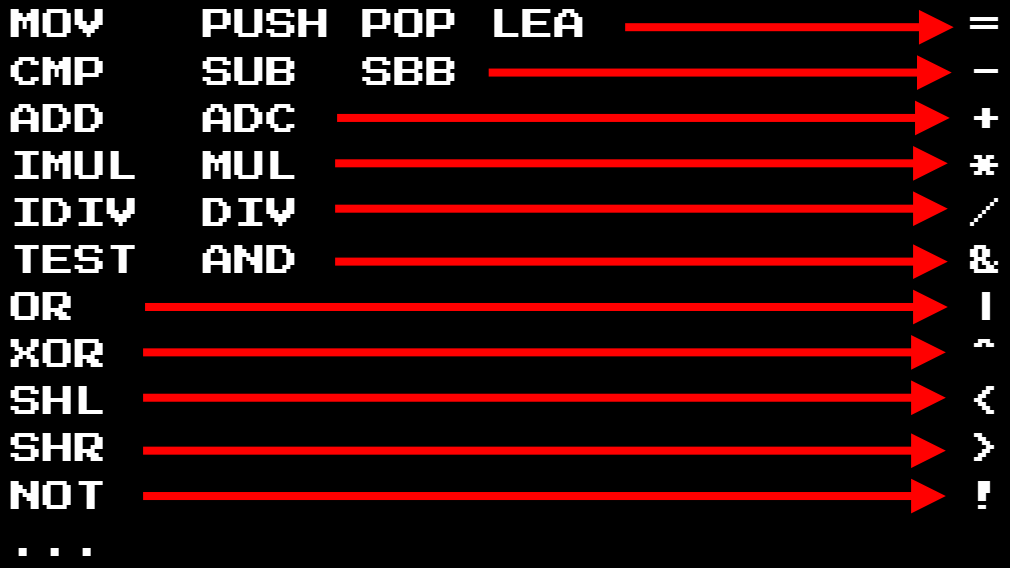
```
0x000000000000119f <+70>:    mov    DWORD PTR [rbp-0x10],0x0
0x00000000000011a6 <+77>:    mov    DWORD PTR [rbp-0xc],0x1
0x00000000000011ad <+84>:    jmp    0x11c3 <main+106>
0x00000000000011af <+86>:    mov    eax,DWORD PTR [rbp-0xc]
0x00000000000011b2 <+89>:    and    eax,0x1
0x00000000000011b5 <+92>:    test   eax,eax
0x00000000000011b7 <+94>:    jne    0x11bf <main+102>
0x00000000000011b9 <+96>:    mov    eax,DWORD PTR [rbp-0xc]
0x00000000000011bc <+99>:    add    DWORD PTR [rbp-0x10],eax
0x00000000000011bf <+102>:   add    DWORD PTR [rbp-0xc],0x1
0x00000000000011c3 <+106>:   mov    eax,DWORD PTR [rbp-0x14]
0x00000000000011c6 <+109>:   cmp    DWORD PTR [rbp-0xc],eax
0x00000000000011c9 <+112>:   jle    0x11af <main+86>
0x00000000000011cb <+114>:   mov    eax,DWORD PTR [rbp-0x14]
0x00000000000011ce <+117>:   mov    edx,DWORD PTR [rbp-0x10]
0x00000000000011d1 <+120>:   mov    esi,eax
0x00000000000011d3 <+122>:   lea    rax,[rip+0xe46]        # 0x2020
0x00000000000011da <+129>:   mov    rdi,rax
0x00000000000011dd <+132>:   mov    eax,0x0
```

```
0x00000000000011a2 <+73>:    mov    edx,eax
0x00000000000011a4 <+75>:    shr    edx,0x1f
0x00000000000011a7 <+78>:    add    eax,edx
0x00000000000011a9 <+80>:    sar    eax,1
0x00000000000011ab <+82>:    mov    ecx,eax
0x00000000000011ad <+84>:    mov    eax,DWORD PTR [rbp-0x10]
0x00000000000011b0 <+87>:    mov    edx,eax
0x00000000000011b2 <+89>:    shr    edx,0x1f
0x00000000000011b5 <+92>:    add    eax,edx
0x00000000000011b7 <+94>:    sar    eax,1
0x00000000000011b9 <+96>:    add    eax,0x1
0x00000000000011bc <+99>:    imul   eax,ecx
0x00000000000011bf <+102>:   mov    DWORD PTR [rbp-0xc],eax
0x00000000000011c2 <+105>:   mov    eax,DWORD PTR [rbp-0x10]
0x00000000000011c5 <+108>:   mov    edx,DWORD PTR [rbp-0xc]
0x00000000000011c8 <+111>:   mov    esi,eax
0x00000000000011ca <+113>:   lea    rax,[rip+0xe4f]        # 0x2020
0x00000000000011d1 <+120>:   mov    rdi,rax
0x00000000000011d4 <+123>:   mov    eax,0x0
```

~# Expressing Individual Instructions



~# Expressing Individual Instructions



Current x86-64 design contains 981 unique Mnemonics.

~# Expressing Individual Instructions

MOV PUSH POP LEA → =
CMP SUB SBB → -
ADD ADC → +
IMUL MUL → *
IDIV DIV → /
TEST AND → &
OR → |
XOR → ^
SHL → <
SHR → >
NOT → !
...

INSTRUCTION SET REFERENCE, A-Z

intel

VSM3RNDS2—Perform Two Rounds of SM3 Operation

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
VEX.128.66.0F3A.W0 DE /r /ib VSM3RNDS2 xmm1, xmm2, xmm3/m128, imm8	A	V/V	AVX SM3	Performs two rounds of SM3 operation using the initial SM3 states from xmm1 and xmm2, and pre-computed words from xmm3/m128, storing the result in xmm1.

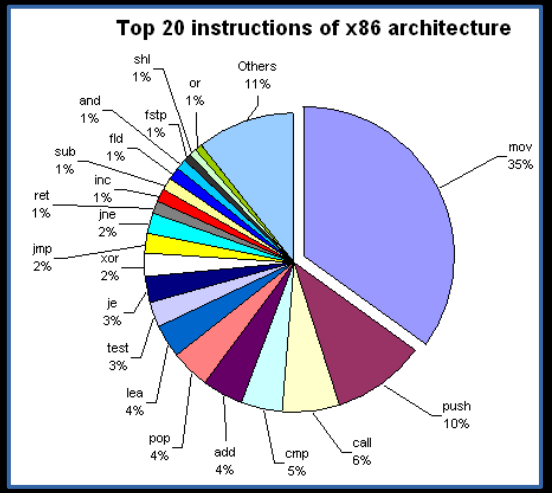
Current x86-64 design contains 981 unique Mnemonics.

~# Instruction Frequency Statistics

1.	%33.5	MOV
2.	%9.2	JCC (All conditional jumps)
3.	%6.4	CALL
4.	%5.5	LEA
5.	%4.9	CMP
6.	%3.9	ADD
7.	%3.7	TEST
8.	%3.5	JMP
9.	%3.3	PUSH
10.	%3.0	POP
11.	%2.7	NOP
12.	%2.2	XOR
13.	%1.7	SUB
14.	%1.5	INT3
15.	%1.1	MOVZX
16.	%1.0	AND
17.	%1.0	RET
18.	%0.6	SHL
19.	%0.5	OR
20.	%0.5	SHR
-	%11.3	<OTHER>

Based on ~300GB executable section sample pool. ~%95 transform gadget coverage.

Check below for similar instruction frequency studies on x86 instruction set.



~# Transform Gadgets

- Arithmetic Partitioning
- Logical Inverse
- Logical Partitioning
- Offset Mutation
- Register Swap

These transform gadgets are spesifically crafted for producing instructions that look like compiler generated sub-optimal code.

~# Arithmetic Partitioning

In number theory and combinatorics, a partition of a non-negative integer n, also called an integer partition, is a way of writing n as a sum of positive integers. Two sums that differ only in the order of their summands are considered the same partition.

ADD EAX,0x10 = (X + 16)

~# Arithmetic Partitioning

In number theory and combinatorics, a partition of a non-negative integer n, also called an integer partition, is a way of writing n as a sum of positive integers. Two sums that differ only in the order of their summands are considered the same partition.

ADD EAX,0x10 = (X + 16)

(X+5-4+2+13) = (X + 16)

~# Arithmetic Partitioning

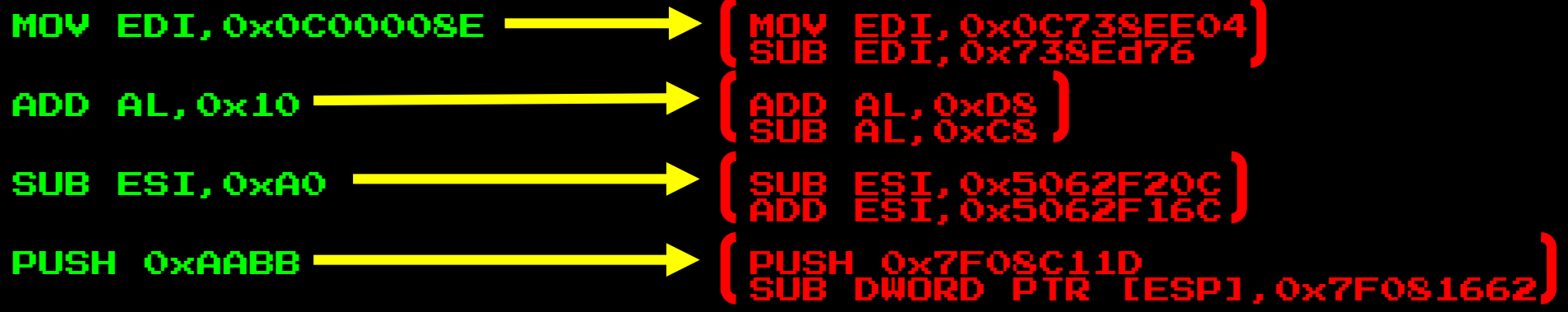
Targets:

· All arithmetic instructions with immediate operands.

GUIDE:

1)Randomize immediate value
2)Fix the immediate value with random arihmetic instruction

FREQUENCY: ~%38



~# Logical Inverse

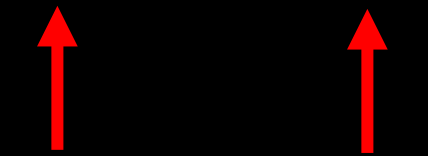
In logic, an inverse is a type of conditional sentence which is an immediate inference made from another conditional sentence. Given a conditional sentence of the form $P \rightarrow Q$, the inverse refers to the sentence $\neg P \rightarrow \neg Q$.

XOR R10, 0x10 = (X ^ 16)

~# Logical Inverse

In logic, an inverse is a type of conditional sentence which is an immediate inference made from another conditional sentence. Given a conditional sentence of the form $P \rightarrow Q$, the inverse refers to the sentence $\neg P \rightarrow \neg Q$.

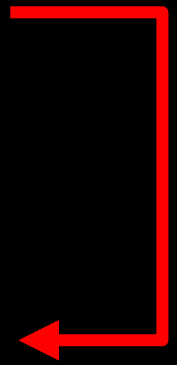
```
XOR R10, 0x10 = (X ^ 16)
```



~# Logical Inverse

In logic, an inverse is a type of conditional sentence which is an immediate inference made from another conditional sentence. Given a conditional sentence of the form P → Q, the inverse refers to the sentence ¬ P → ¬ Q.

```
XOR R10, 0x10 = (X ^ 16)
```



```
(X ^ 16) = (X' ^ '16) = (X' ^ -16)
```

~# Logical Inverse

Targets:

Some logical instructions. (AND/OR/XOR)

GUIDE:

1) Take the inverse of the first operand
2) Take the inverse of the second operand

FREQUENCY: ~%10 - ~%28 (depending on the third NOT op.)

EXAMPLES:

```
XOR R10,0x49656E69  ->  NOT R10
                        XOR R10,0xB69A9196

AND AL,0x01         ->  NOT AL
                        OR AL,0xFE

OR EDX,0x300        ->  NOT EDX
                        AND EDX,0xFFFFFCFF
                        NOT EDX
```

~# Logical Partitioning

Targets:

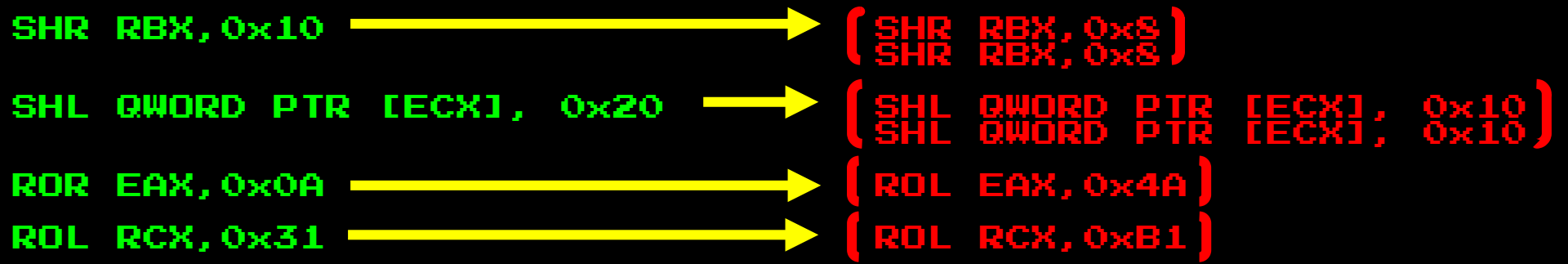
Some logical instructions. (SHR/SHL/ROL/ROR...)

GUIDE:

1)When shifting:
  1)Divide the immediate value into 2
  2)Shift twice with the new value
2)When Rolling:
  1)Add x times the size of the first operand to roll value

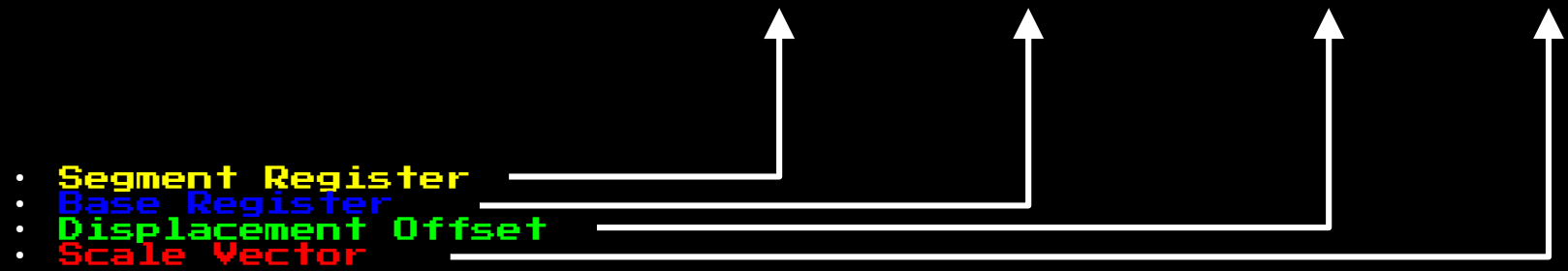
FREQUENCY: ~%59

EXAMPLES:



~# Offset Mutation

```
MOV EAX, CS:[EAX+0x100*8]
```



~# Offset Mutation

Targets:

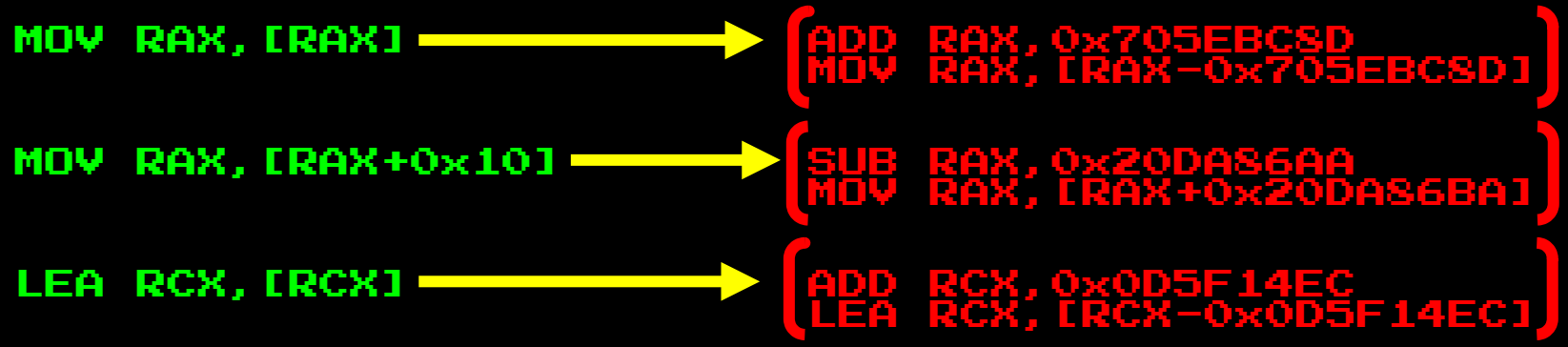
All instructions with a memory operand.

GUIDE:

1)Randomize the memory displacement offset
2)Fix the base register with a random arithmetic operation
  1)Restore the base register value depending on the target operand

FREQUENCY: ~%100

EXAMPLES:



~# Register Swap

Targets:

All instructions with register operand.

GUIDE:

1)Chose a random register operand
2)Swap register with a same size different register using XCHG
3)Modify the original instruction with new register
4)Swap the register value back using same XCHG instruction

FREQUENCY: ~%22

EXAMPLES:

```
XOR RAX,0x10 ──►  XCHG RAX,RCX
                  XOR RCX,0x10
                  XCHG RAX,RCX

ADD RBX,0x31 ──►  XCHG RBX,RSI
                  ADD RSI,0x31
                  XCHG RBX,RSI

MOV RDX,RAX  ──►  XCHG RDX,RDI
                  MOV RDI,RAX
                  XCHG RBX,RSI
```

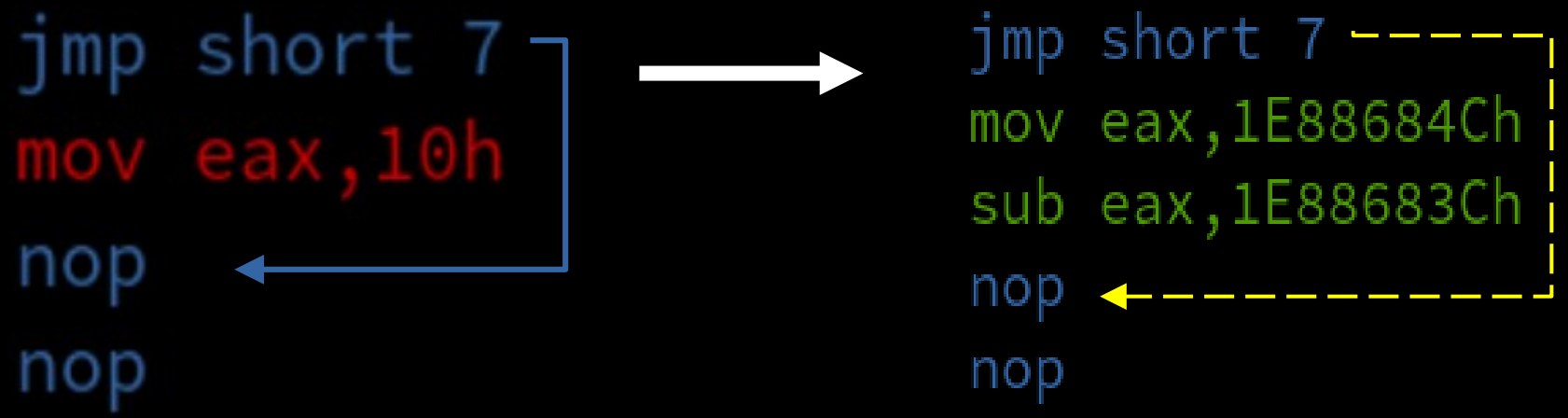
~# Memory Realignment

1.	%33.5	MOV
2.	%9.2	JCC (All conditional jumps)
3.	%6.4	CALL
4.	%5.5	LEA
5.	%4.9	CMP
6.	%3.9	ADD
7.	%3.7	TEST
8.	%3.5	JMP
9.	%3.3	PUSH
10.	%3.0	POP
11.	%2.7	NOP
12.	%2.2	XOR
13.	%1.7	SUB
14.	%1.5	INT3
15.	%1.1	MOVZX
16.	%1.0	AND
17.	%1.0	RET
18.	%0.6	SHL
19.	%0.5	OR
20.	%0.5	SHR
-	%11.3	<OTHER>

BRANCH INSTRUCTIONS

```
0x00001b00      488d3d892500.  lea rdi, obj.__TMC_END__     ; 0x4090
0x00001b07      488d35822500.  lea rsi, obj.__TMC_END__     ; 0x4090
0x00001b0e      4829fe         sub rsi, rdi
0x00001b11      4889f0         mov rax, rsi
0x00001b14      48c1ee3f       shr rsi, 0x3f
0x00001b18      48c1f803       sar rax, 3
0x00001b1c      4801c6         add rsi, rax
0x00001b1f      48d1fe         sar rsi, 1
0x00001b22      7414           je 0x1b38
0x00001b24      488b05ad2400.  mov rax, qword [reloc._ITM_registerTMCloneTable] ; [0x3fd8:8]=0
0x00001b2b      4885c0         test rax, rax
0x00001b2e      7408           je 0x1b38
0x00001b30      ffe0           jmp rax
0x00001b32      660f1f440000   nop word [rax + rax]
0x00001b38      c3             ret
0x00001b39      0f1f80000000.  nop dword [rax]
;-- entry.fini0:
;-- __do_global_dtors_aux:
0x00001b40      f30f1efa       endbr64
0x00001b44      803d9d250000.  cmp byte [obj.completed.0], 0 ; [0x40e8:1]=0
0x00001b4b      7533           jne 0x1b80
0x00001b4d      55             push rbp
0x00001b4e      48833d8a2400.  cmp qword [reloc.__cxa_finalize], 0 ; [0x3fe0:8]=0
0x00001b56      4889e5         mov rbp, rsp
0x00001b59      740d           je 0x1b68
0x00001b5b      488b3d1e2500.  mov rdi, qword [obj.__dso_handle] ; [0x4080:8]=0x4080 obj.__dso_handle
0x00001b62      ff157824000    call qword [reloc.__cxa_finalize] ; [0x3fe0:8]=0
0x00001b68      e863ffffff     call sym.deregister_tm_clones
0x00001b6d      c60574250000.  mov byte [obj.completed.0], 1 ; [0x40e8:1]=0
0x00001b74      5d             pop rbp
0x00001b75      c3             ret
0x00001b76      662e0f1f8400.  nop word cs:[rax + rax]
0x00001b80      c3             ret
0x00001b81      66662e0f1f84.  nop word cs:[rax + rax]
0x00001b8c      0f1f4000       nop dword [rax]
;-- entry.init0:
;-- frame_dummy:
0x00001b90      f30f1efa       endbr64
0x00001b94      e967ffffff     jmp sym.register_tm_clones
```

~# Memory Realignment



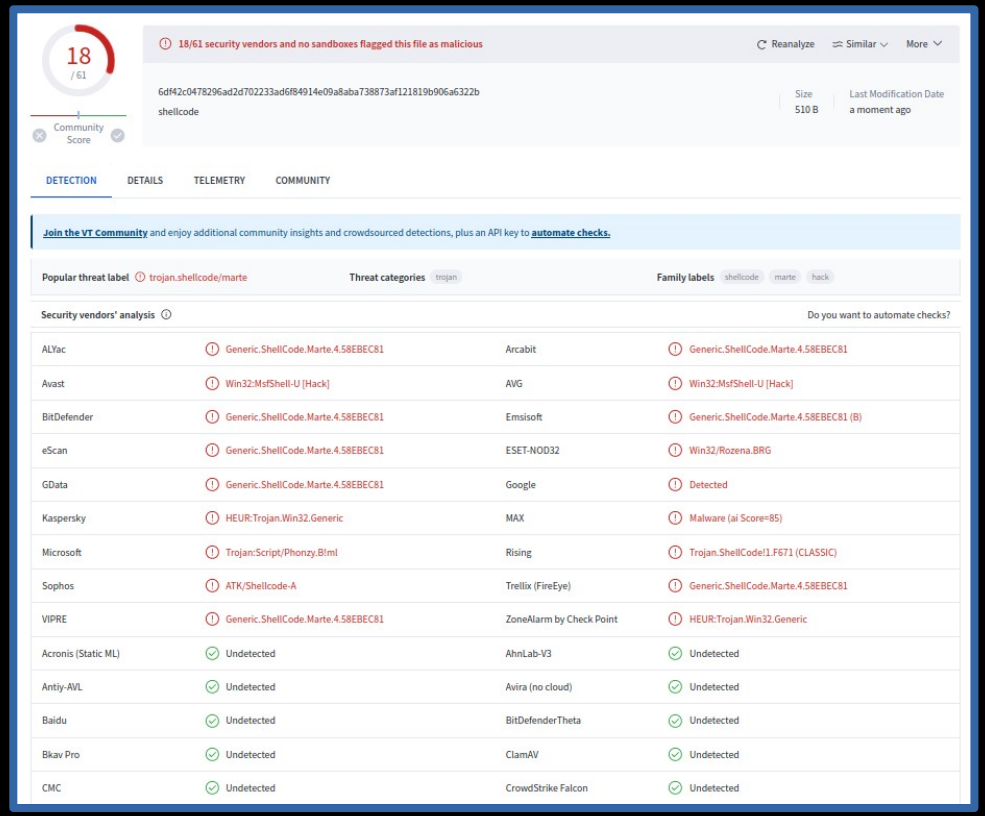
Demo Time!

/tmp/INS-DEMO

INT

msf6 >

~# Detection Rates



18 / 61

Community Score

18/61 security vendors and no sandboxes flagged this file as malicious

Reanalyze · Similar · More

6df42c0478296ad2d702233ad6f84914e09a8aba738873af121819b906a6322b

shellcode

Size 510 B

Last Modification Date a moment ago

DETECTION | DETAILS | TELEMETRY | COMMUNITY

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

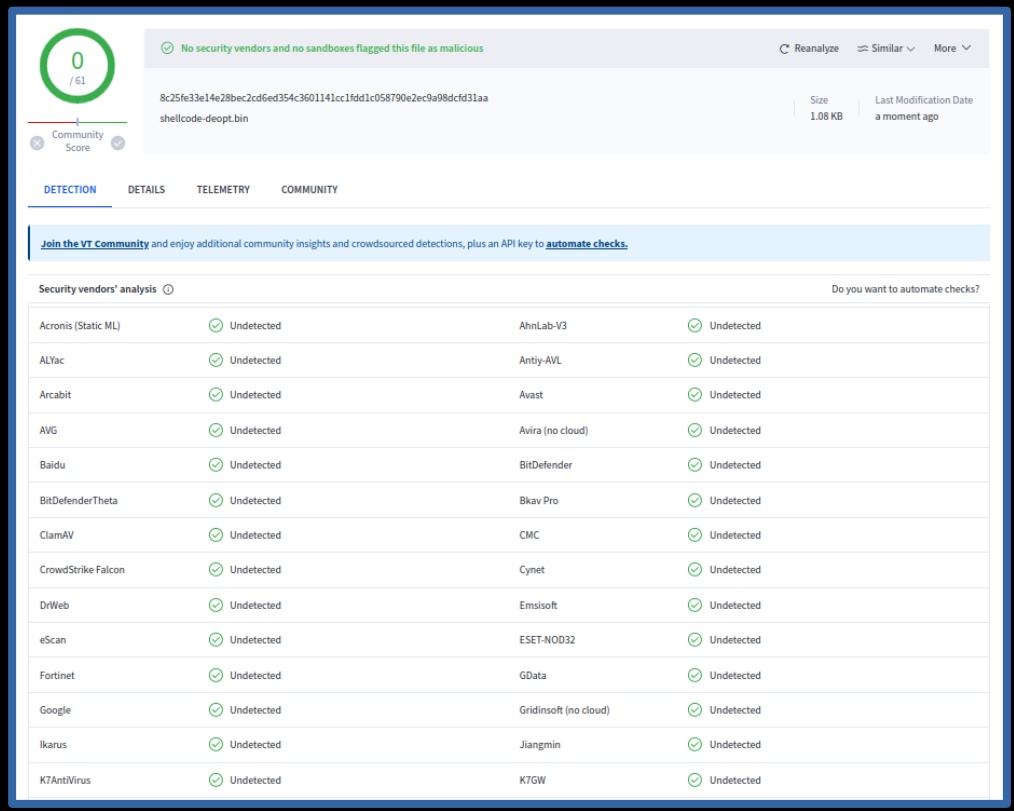
Popular threat label: trojan.shellcode/marte

Threat categories: trojan

Family labels: shellcode | marte | hack

Security vendors' analysis — Do you want to automate checks?

ALYac	Generic.ShellCode.Marte.4.58EBEC81	Arcabit	Generic.ShellCode.Marte.4.58EBEC81
Avast	Win32:MsfShell-U [Hack]	AVG	Win32:MsfShell-U [Hack]
BitDefender	Generic.ShellCode.Marte.4.58EBEC81	Emsisoft	Generic.ShellCode.Marte.4.58EBEC81 (B)
eScan	Generic.ShellCode.Marte.4.58EBEC81	ESET-NOD32	Win32/Rozena.BRG
GData	Generic.ShellCode.Marte.4.58EBEC81	Google	Detected
Kaspersky	HEUR:Trojan.Win32.Generic	MAX	Malware (ai Score=85)
Microsoft	Trojan:Script/Phonzy.B!ml	Rising	Trojan.ShellCode!1.F671 (CLASSIC)
Sophos	ATK/Shellcode-A	Trellix (FireEye)	Generic.ShellCode.Marte.4.58EBEC81
VIPRE	Generic.ShellCode.Marte.4.58EBEC81	ZoneAlarm by Check Point	HEUR:Trojan.Win32.Generic
Acronis (Static ML)	Undetected	AhnLab-V3	Undetected
Antiy-AVL	Undetected	Avira (no cloud)	Undetected
Baidu	Undetected	BitDefenderTheta	Undetected
Bkav Pro	Undetected	ClamAV	Undetected
CMC	Undetected	CrowdStrike Falcon	Undetected



0 / 61

Community Score

No security vendors and no sandboxes flagged this file as malicious

Reanalyze · Similar · More

8c25fe33e14e28bec2cd6ed354c3601141cc1fdd1c058790e2ec9a98dcfd31aa

shellcode-deopt.bin

Size 1.08 KB

Last Modification Date a moment ago

DETECTION | DETAILS | TELEMETRY | COMMUNITY

Join the VT Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Security vendors' analysis — Do you want to automate checks?

Acronis (Static ML)	Undetected	AhnLab-V3	Undetected
ALYac	Undetected	Antiy-AVL	Undetected
Arcabit	Undetected	Avast	Undetected
AVG	Undetected	Avira (no cloud)	Undetected
Baidu	Undetected	BitDefender	Undetected
BitDefenderTheta	Undetected	Bkav Pro	Undetected
ClamAV	Undetected	CMC	Undetected
CrowdStrike Falcon	Undetected	Cynet	Undetected
DrWeb	Undetected	Emsisoft	Undetected
eScan	Undetected	ESET-NOD32	Undetected
Fortinet	Undetected	GData	Undetected
Google	Undetected	Gridinsoft (no cloud)	Undetected
Ikarus	Undetected	Jiangmin	Undetected
K7AntiVirus	Undetected	K7GW	Undetected

~# Known Limitations

- Scope
- Self-Modifing Code
- Code With Data
- Overlapping Instructions

~# Known Limitations

- Scope
- Self-Modifying Code
- Code With Data
- Overlapping Instructions

INSTRUCTIONS WITH 0 OPERANDS

CLD—Clear Direction Flag

Opcode	Instruction	Op/ En	64-bit Mode	Compat/ Leg Mode	Description
FC	CLD	ZO	Valid	Valid	Clear DF flag.

Instruction Operand Encoding

Op/En	Operand 1	Operand 2	Operand 3	Operand 4
ZO	N/A	N/A	N/A	N/A

NOP—No Operation

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode	Description

SYSCALL—Fast System Call

Opcode	Instruction	Op/	64-Bit	Compat/	Descripti

SYSENTER—Fast System Call

Opcode	Instruction	Op/	64-Bit	Compat/	Description

CLD—Clear Direction Flag

Opcode	Instruction	Op/	64-bit	Compat/	Descri

AAA—ASCII Adjust After Addition

Opcode	Instruction	Op/	64-bit	Compat/

PUSHF/PUSHFD/PUSHFQ—Push EFLAGS Register Onto the Stack

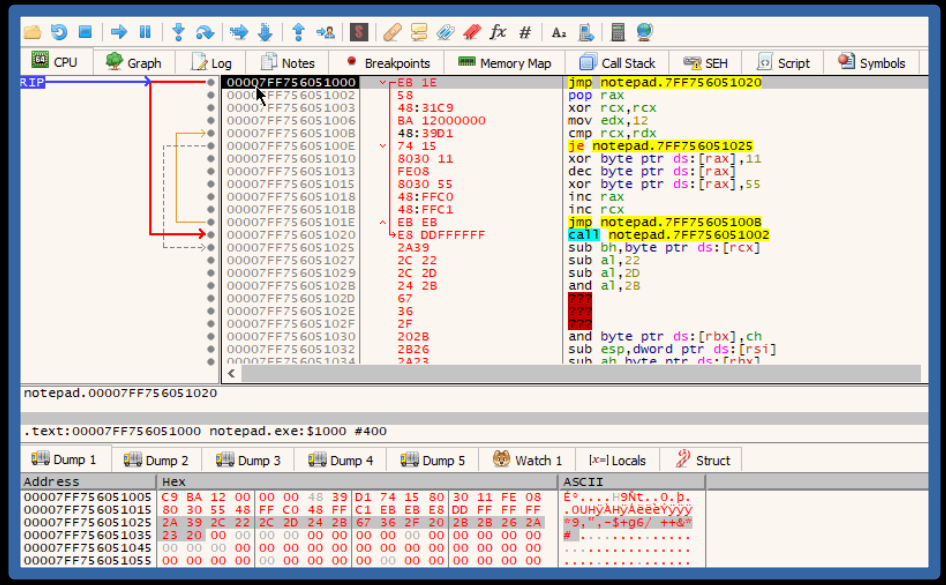
Opcode	Instruction	Op/	64-Bit	Compat/	Descr

POPF/POPFD/POPFQ—Pop Stack Into EFLAGS Register

Opcode	Instruction	Op/ En	64-Bit Mode	Compat/ Leg Mode
9D	POPF	ZO	Valid	Valid
9D	POPFD	ZO	N.E.	Valid
9D	POPFQ	ZO	Valid	N.E.

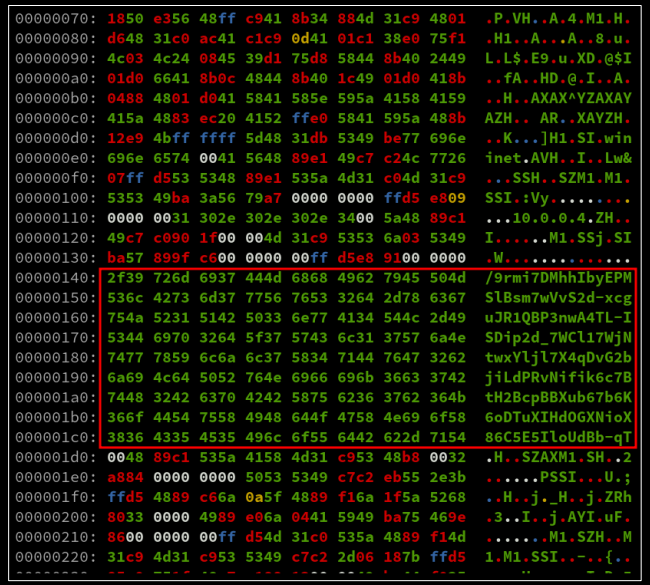
~# Known Limitations

- Scope
- **Self-modifing Code**
- Code With Data
- Overlapping Instructions



~# Known Limitations

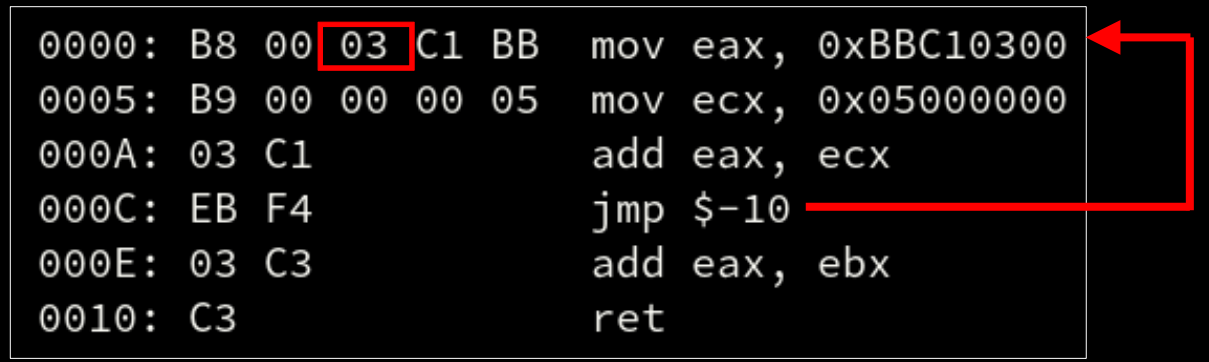
- Scope
- Self-modifing Code
- **Code With Data**
- Overlapping Instructions




```
106     jz failure
107     jmp short internetsetoption
108
109   dbl_get_server_host:
110     jmp get_server_host
111
112   get_server_uri:
113     call httpopenrequest
114
115   server_uri:
116     db "/12345", 0x00
117
118   failure:
119     mov r14, 0x56A2B5F0    ; hardcoded to exitprocess for size
120     call rbp
121
122   allocate_memory:
123     xor rcx, rcx           ; LPVOID lpAddress
124     mov rdx, 0x00400000    ; SIZE_T dwSize
125     mov r8, 0x1000         ; DWORD flAllocationType(MEM_COMMIT)
126     mov r9, 0x40           ; DWORD flProtect(PAGE_EXECUTE_READWRITE)
127     mov r10, 0xE553A458    ; hash( "kernel32.dll", "VirtualAlloc" )
128     call rbp
```

~# Known Limitations

- Scope
- Self-modifing Code
- Code With Data
- Overlapping Instructions



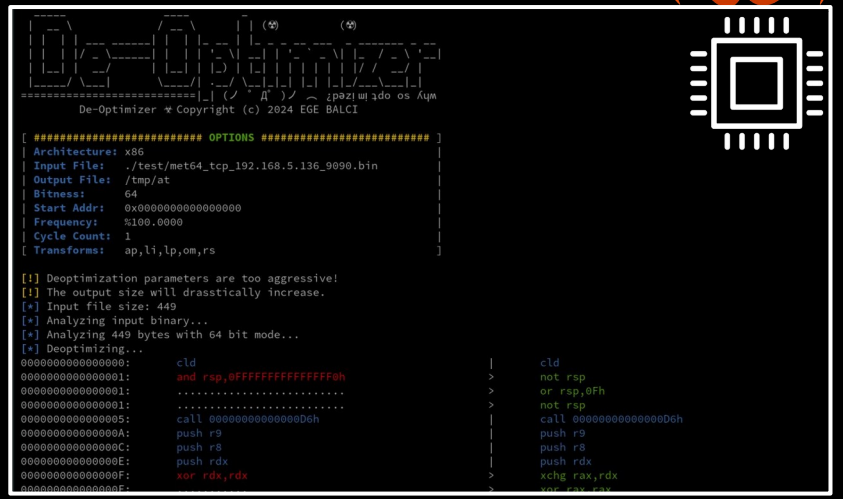
~# Deoptimizer Tool

https://github.com/EgeBalci/deoptimizer

- Fully written in Rust!
- No external dependencies
- Iced_x86 disassembler library

To-Do:

- Add PE support
- Add ELF support
- Add ARM architecture
- Add RISC architecture



Thanks!

Questions?