Polymorphic Shellcode at a Glance

Vlad Tsyrklevich

Agenda

- Introduction – What data to fingerprint
- Polymorphic encoders
- Developing fingerprints
- Looking to the future

Who is this guy?

- IAMF Vlad Tsyrklevich
  - vlad902@gmail.com
- Occasional Metasploit Framework contributor
- Assembler purist
- Ruby enthusiast

Overview

- What data can we fingerprint?
  - Protocol anomalies
  - Attack vector
  - Specific exploit
  - NOPs
  - Shellcode

Overview

- What data can we fingerprint?
  - Protocol anomalies
    - Advantages: Very generic, can catch 0day exploits.
    - Disadvantages: False positives, hard or impossible to do for some protocols
  - Attack vector
  - Specific exploit
  - NOPs
  - Shellcode

Overview

- What data can we fingerprint?
  - Protocol anomalies
  - Attack vector
    - Advantages: Can always catch one specific vulnerability exploitation attempt
    - Disadvantages: *Ex post facto*, can only catch old exploits
  - Specific exploit
  - NOPs
  - Shellcode

Overview

- What data can we fingerprint?
  - Protocol anomalies
  - Attack vector
  - Specific exploit
    - Advantages: Can always fingerprint one specific exploit when attacks vectors are hard to fingerprint
    - Disadvantages: *Ex post facto*, Different exploits abound, fingerprinting the attack vector is more useful when possible
  - NOPs
  - Shellcode

Overview

- What data can we fingerprint?
  - Protocol anomalies
  - Attack vector
  - Specific exploit
  - NOPs
    - Advantages: Very hard to evade
    - Disadvantages: False positives on large binary/ASCII transfers, can not fingerprint NOP-less exploits (Win32)
  - Shellcode

Overview

- What data can we fingerprint?
  - Protocol anomalies
  - Attack vector
  - Specific exploit
  - NOPs
  - Shellcode
    - Advantages: Shellcode is almost always necessary
    - Disadvantages: Many different types of shellcode, many different permutations (polymorphism), encoders, ret2libc

Overview

- Most IDS ignore NOPs and shellcode
  - Both can catch 0day
- NOPs false positive
  - AAAA...A is a valid NOP sled
- Shellcode can polymorph
  - Many different permutations are possible
  - Rarely polymorphed, often it’s encoded
  - So why not try to fingerprint the encoder?

Shellcode encoders

- Small stubs prepended to the shellcode used to evade character restrictions and hide common shellcode fingerprint characteristics.
- These are often polymorphed instead of the shellcode because it is easier to do rather than an entire shellcode
  - Example: ADMutate, CLET, Shikata Ga Nai

Three Rules of Polymorphism

- The instructions change to a large number of possible permutations
- The instructions are re-arranged randomly where possible
- Random no-operation instructions are added randomly in between instructions

```
00000000  EB38              jmp short 0x3a
00000002  5E                pop esi
00000003  8CC0              mov eax,es
00000005  31C9              xor ecx,ecx
00000007  68CA399D2D        push dword 0x2d9d39ca
0000000C  5B                pop ebx
0000000D  87D2              xchg edx,edx
0000000F  87D2              xchg edx,edx
00000011  8CE8              mov eax,gs
00000013  B101              mov cl,0x1
00000015  99                cdq
00000016  8B06              mov eax,[esi]
00000018  09D8              or eax,ebx
0000001A  211E              and [esi],ebx
0000001C  F716              not dword [esi]
0000001E  2106              and [esi],eax
00000020  48                dec eax
00000021  F8                clc
00000022  F5                cmc
00000023  4F                dec edi
00000024  83F08D            xor eax,byte -0x73
00000027  96                xchg eax,esi
00000028  40                inc eax
00000029  96                xchg eax,esi
0000002A  83C844            or eax,byte +0x44
0000002D  96                xchg eax,esi
0000002E  40                inc eax
0000002F  96                xchg eax,esi
00000030  46                inc esi
00000031  87DB              xchg ebx,ebx
00000033  46                inc esi
00000034  FC                cld
00000035  FC                cld
00000036  E2DE              loop 0x16
00000038  EB06              jmp short 0x40
0000003A  E8C3FFFFFF        call 0x2
```

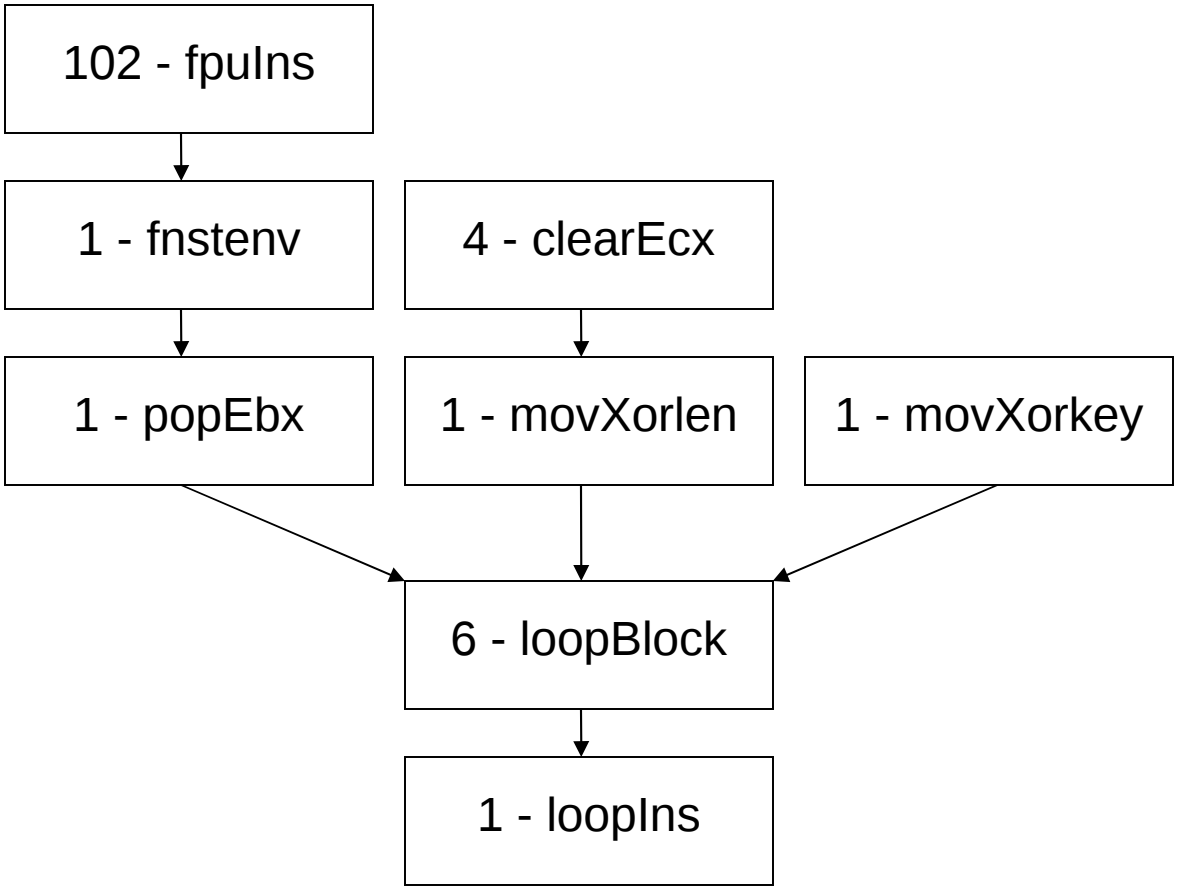
=

```
00000000  EB46              jmp short 0x48
00000002  F7D0              not eax
00000004  5E                pop esi
00000005  8CE0              mov eax,fs
00000007  9C                pushf
00000008  98                cwde
00000009  47                inc edi
0000000A  F8                clc
0000000B  31C9              xor ecx,ecx
0000000D  BB409E439C        mov ebx,0x9c439e40
00000012  2F                das
00000013  47                inc edi
00000014  83E027            and eax,byte +0x27
00000017  83F0E0            xor eax,byte -0x20
0000001A  83C0F7            add eax,byte -0x9
0000001D  6A01              push byte +0x1
0000001F  6659              pop cx
00000021  9F                lahf
00000022  98                cwde
00000023  9E                sahf
00000024  311E              xor [esi],ebx
00000026  83C601            add esi,byte +0x1
00000029  83C601            add esi,byte +0x1
0000002C  3F                aas
0000002D  83E0C1            and eax,byte -0x3f
00000030  8CC0              mov eax,es
00000032  9C                pushf
00000033  83C601            add esi,byte +0x1
00000036  85C0              test eax,eax
00000038  96                xchg eax,esi
00000039  40                inc eax
0000003A  96                xchg eax,esi
0000003B  90                nop
0000003C  33C0              xor eax,eax
0000003E  99                cdq
0000003F  F9                stc
00000040  92                xchg eax,edx
00000041  E2E1              loop 0x24
00000043  EB09              jmp short 0x4e
00000045  47                inc edi
00000046  4F                dec edi
00000047  F8                clc
00000048  E8B7FFFFFF        call 0x4
```

Encoder details

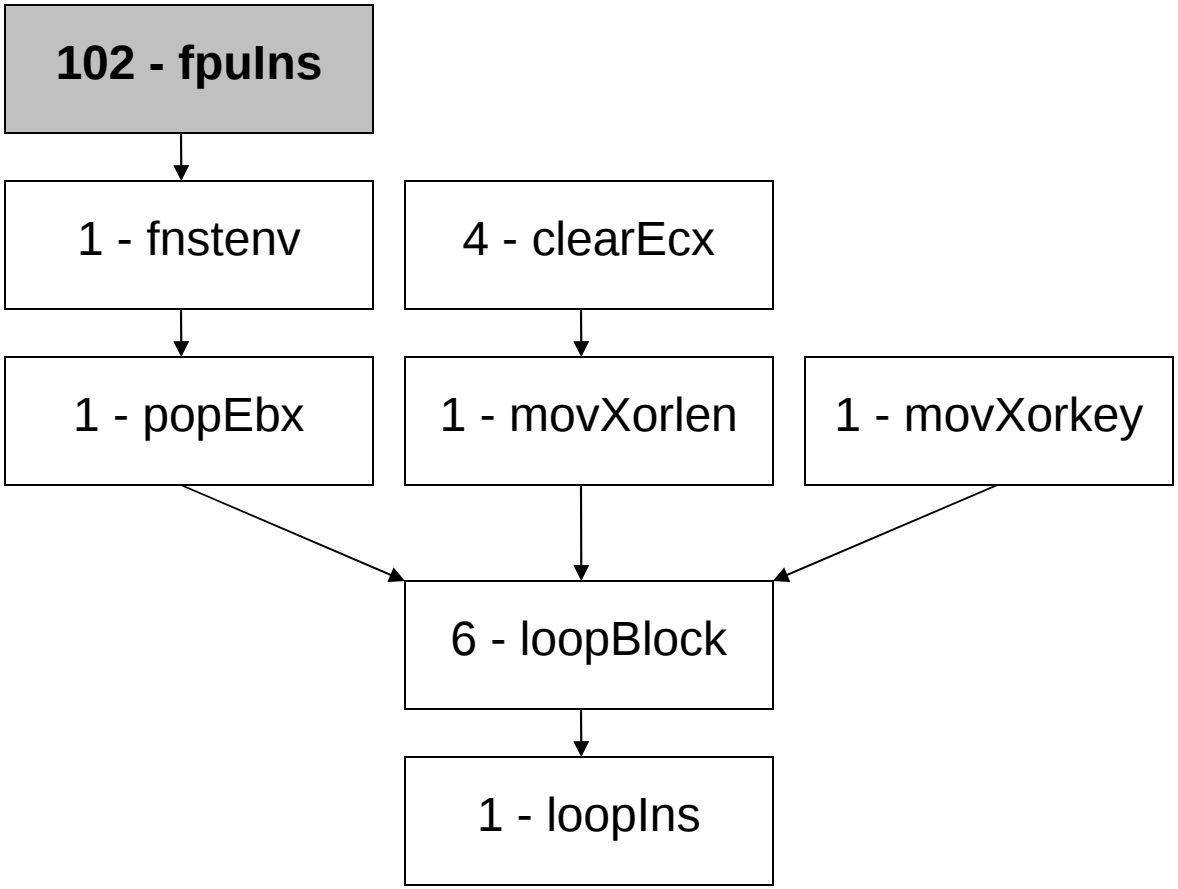
- **Shikata Ga Nai**
  - Used in the Metasploit Framework
  - Approximately 1.3 million permutations
  - Uses XOR
- **CLET**
  - Built-in support against spectrum analysis
  - Phrack 61-0x09
  - Variable number of reversible instructions (rol/xor/add)
- **ADMutate**
  - Multi-architecture + lots of NOP opcodes included
  - Uses XOR

Shikata Ga Nai: Dependencies

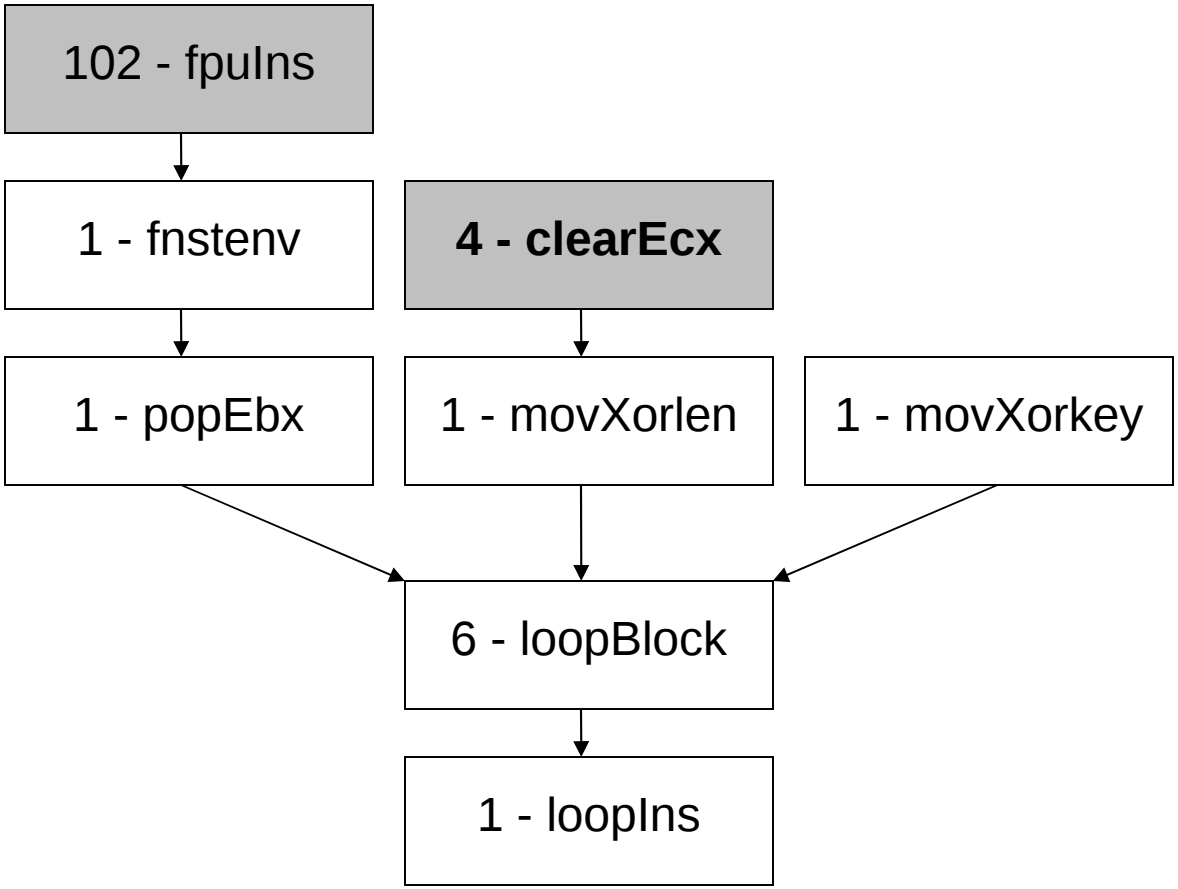


Shikata Ga Nai: Dependencies

DBC1 **fcmovnb st1**

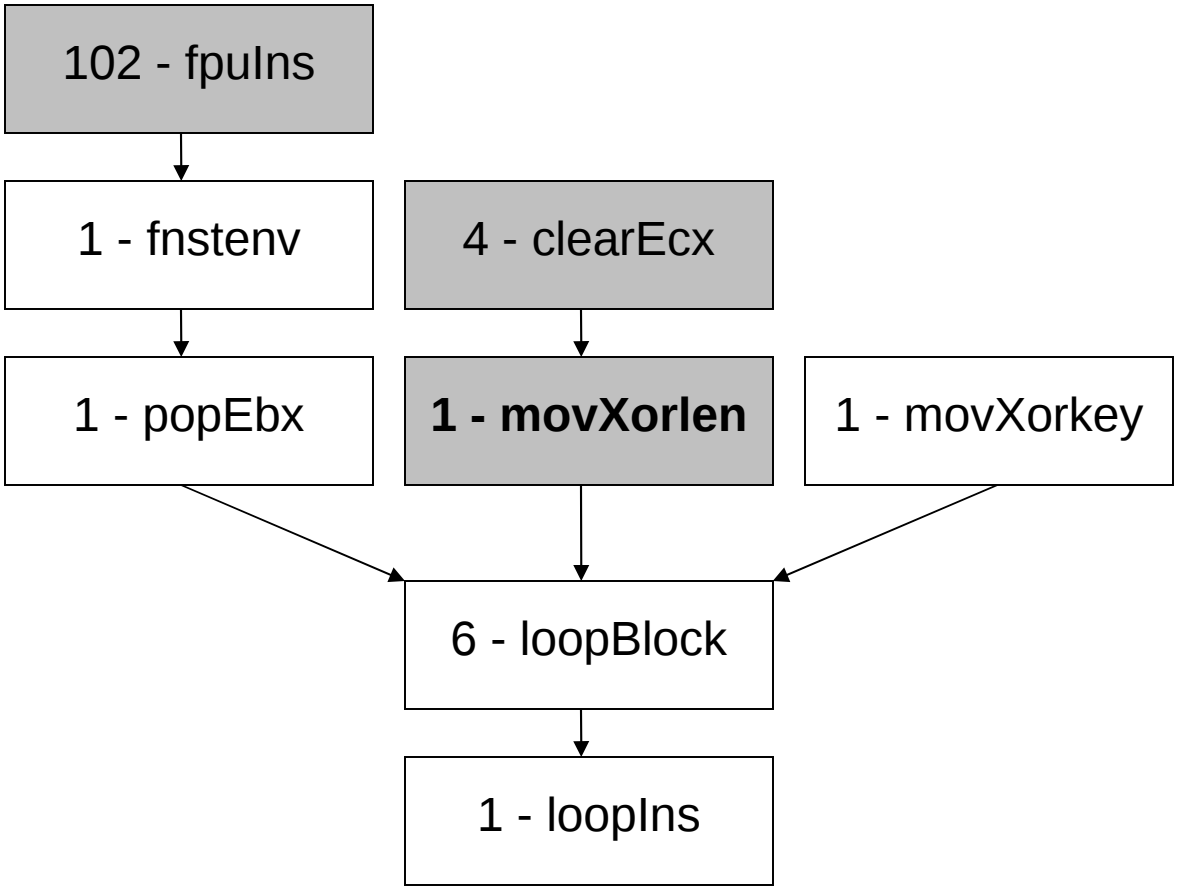


Shikata Ga Nai: Dependencies



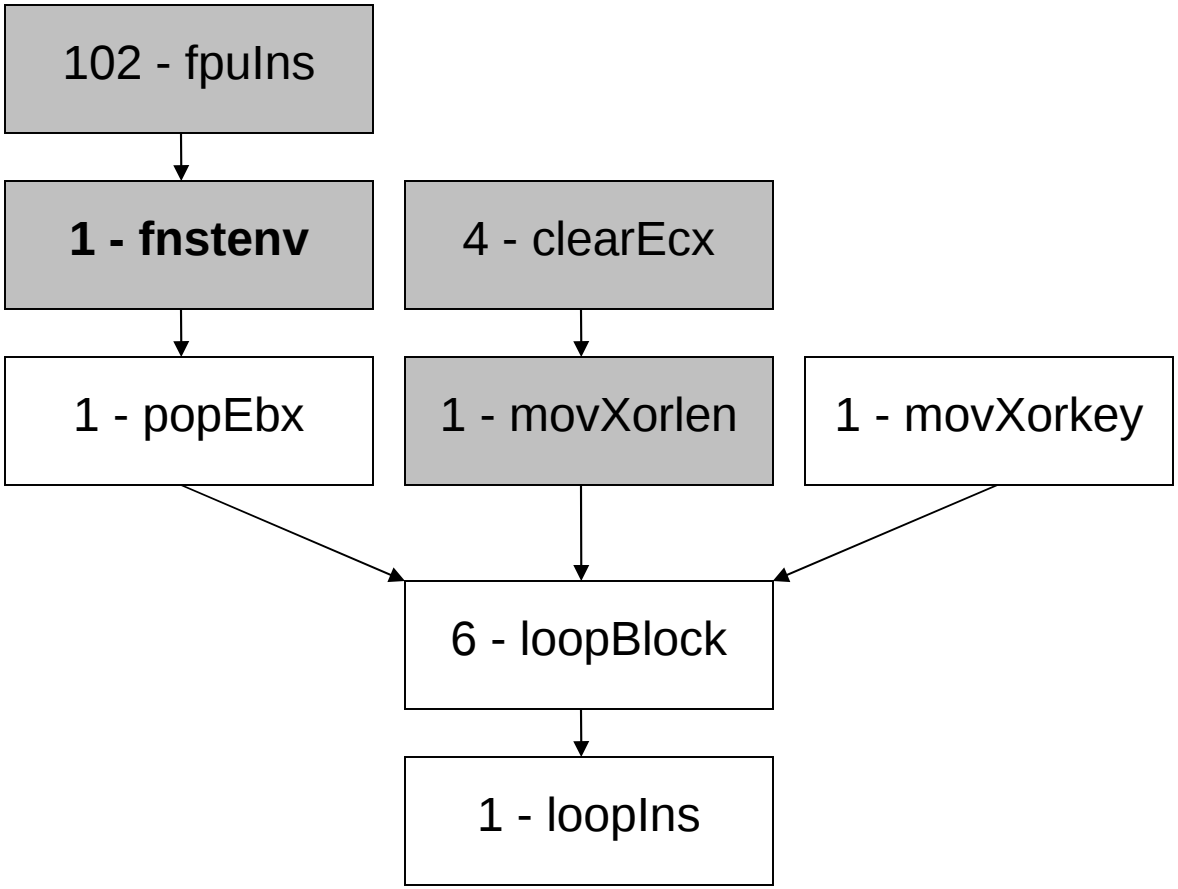
DBC1	fcmovnb st1
31C9	**xor ecx, ecx**

Shikata Ga Nai: Dependencies



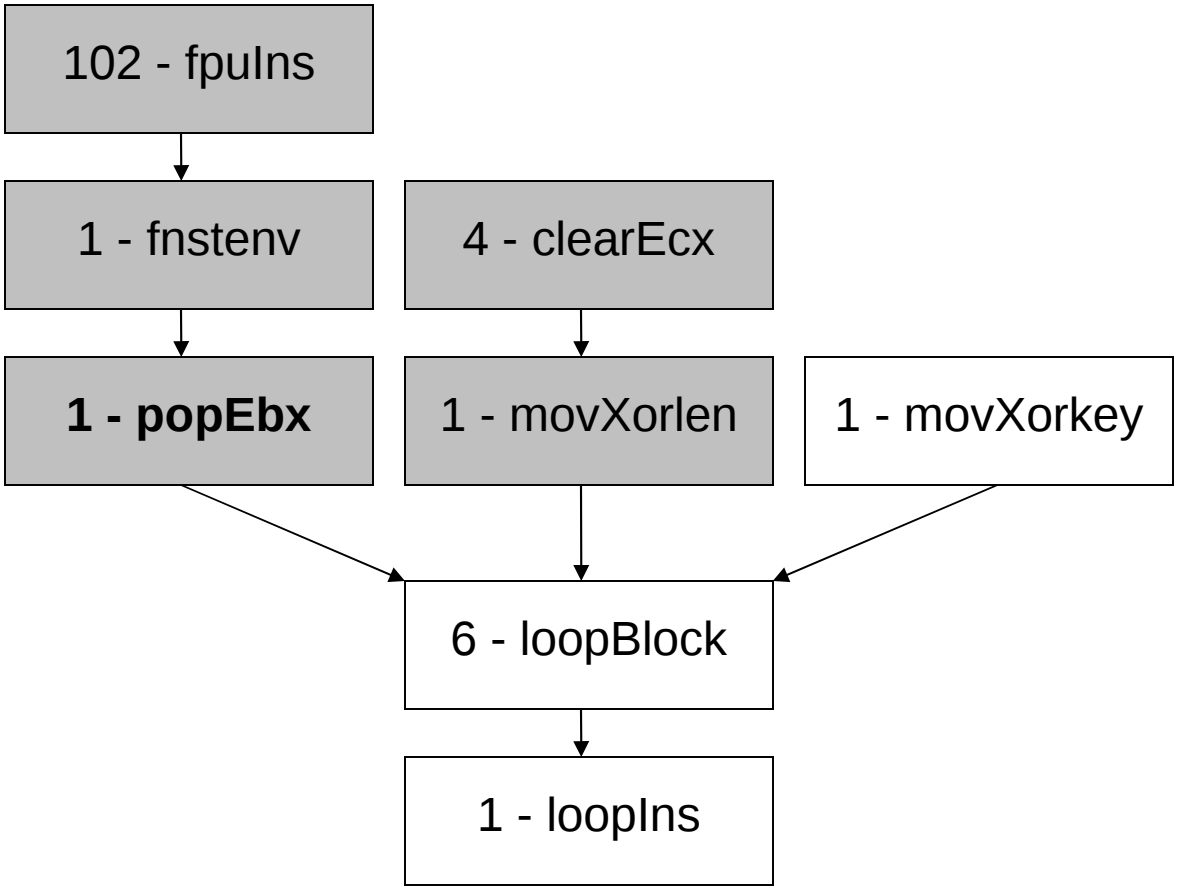
DBC1	fcmovnb st1
31C9	xor ecx, ecx
B101	**mov cl, 1**

Shikata Ga Nai: Dependencies



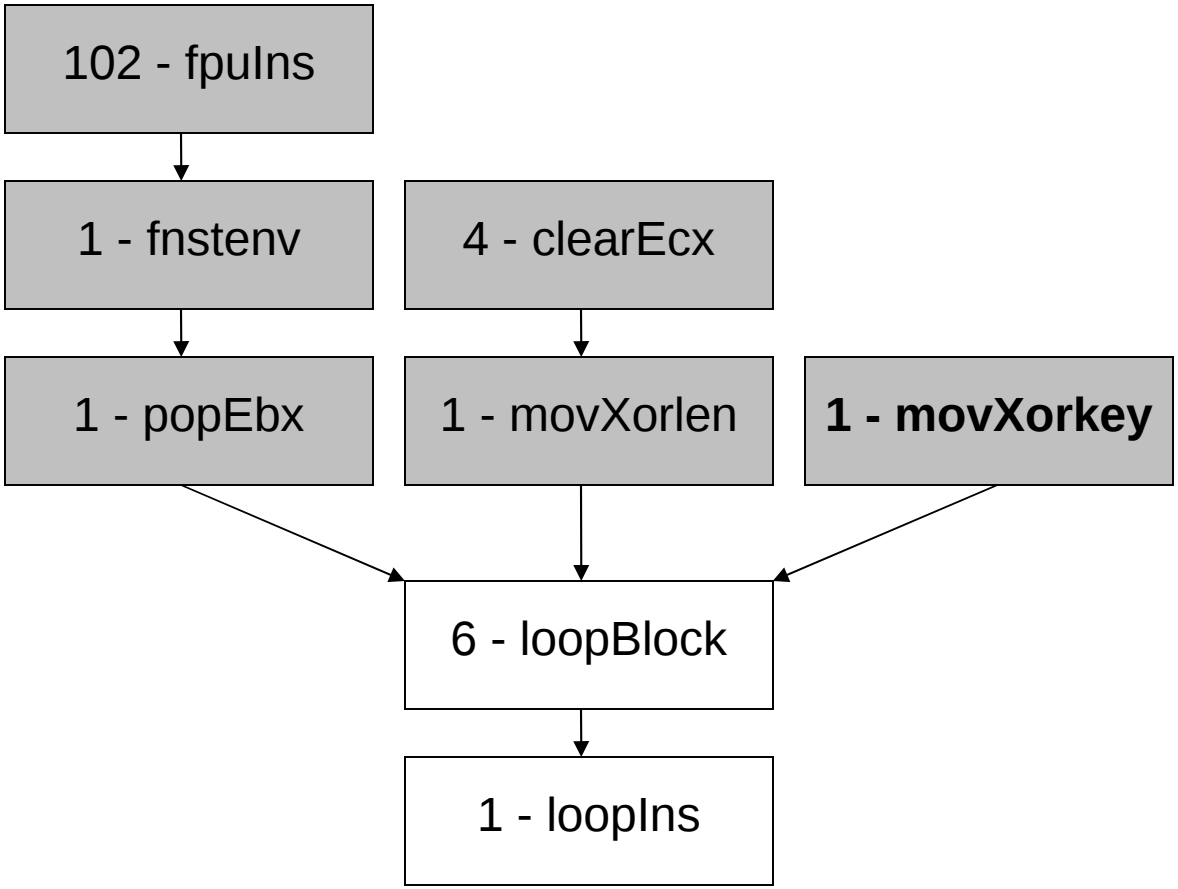
DBC1	fcmovnb st1
31C9	xor ecx, ecx
B101	mov cl, 1
D97424F4	**fnstenv [esp-12]**

Shikata Ga Nai: Dependencies



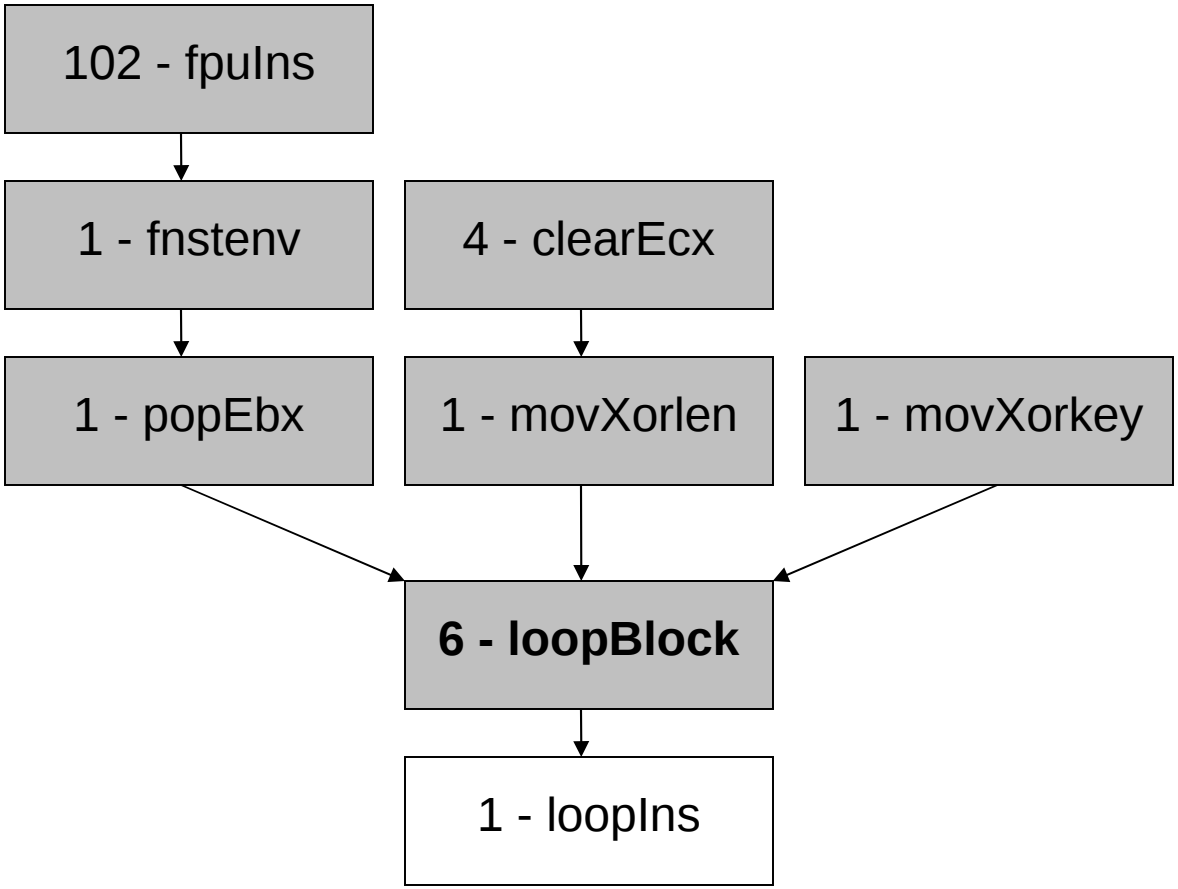
DBC1	fcmovnb st1
31C9	xor ecx, ecx
B101	mov cl, 1
D97424F4	fnstenv [esp-12]
5B	**pop ebx**

Shikata Ga Nai: Dependencies



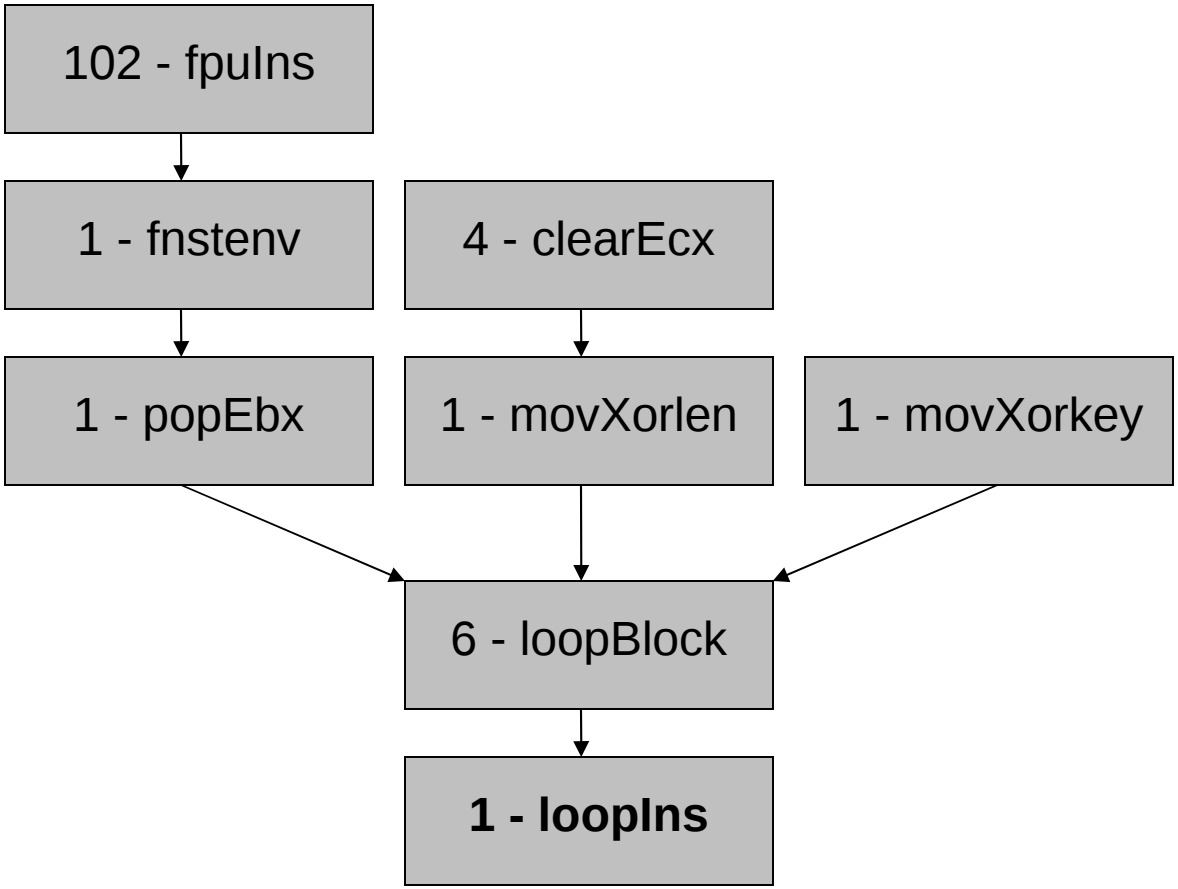
DBC1	fcmovnb st1
31C9	xor ecx, ecx
B101	mov cl, 1
D97424F4	fnstenv [esp-12]
5B	pop ebx
BAC8E2C8F8	**mov edx, XORKEY**

Shikata Ga Nai: Dependencies



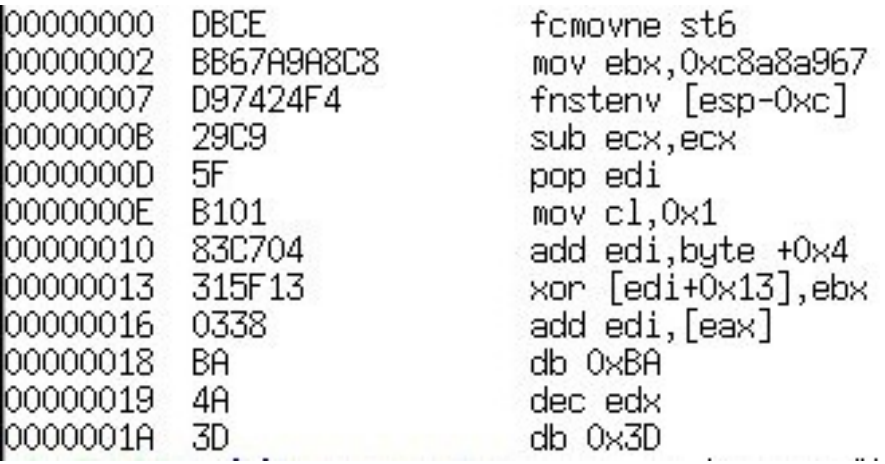
DBC1	fcmovnb st1
31C9	xor ecx, ecx
B101	mov cl, 1
D97424F4	fnstenv [esp-12]
5B	pop ebx
BAC8E2C8F8	mov edx, XORKEY
_LOOP:	
83C304	**add ebx, 4**
315313	**xor [ebx+19], edx**
035313	**add edx, [ebx+19]**

Shikata Ga Nai: Dependencies

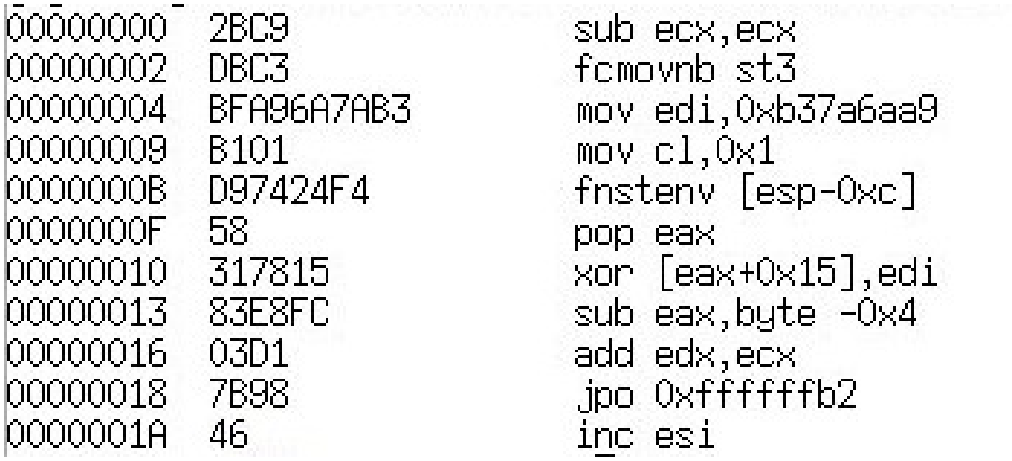


DBC1	fcmovnb st1
31C9	xor ecx, ecx
B101	mov cl, 1
D97424F4	fnstenv [esp-12]
5B	pop ebx
BAC8E2C8F8	mov edx, XORKEY
_LOOP:	
83C304	add ebx, 4
315313	xor [ebx+19], edx
035313	add edx, [ebx+19]
E2F5	**loop _LOOP**

Shikata Ga Nai: Diagram



```
00000000  DBCE              fcmovne st6
00000002  BB67A9A8C8        mov ebx,0xc8a8a967
00000007  D97424F4          fnstenv [esp-0xc]
0000000B  29C9              sub ecx,ecx
0000000D  5F                pop edi
0000000E  B101              mov cl,0x1
00000010  83C704            add edi,byte +0x4
00000013  315F13            xor [edi+0x13],ebx
00000016  0338              add edi,[eax]
00000018  BA                db 0xBA
00000019  4A                dec edx
0000001A  3D                db 0x3D
```



```
00000000  2BC9              sub ecx,ecx
00000002  DBC3              fcmovnb st3
00000004  BFA96A7AB3        mov edi,0xb37a6aa9
00000009  B101              mov cl,0x1
0000000B  D97424F4          fnstenv [esp-0xc]
0000000F  58                pop eax
00000010  317815            xor [eax+0x15],edi
00000013  83E8FC            sub eax,byte -0x4
00000016  03D1              add edx,ecx
00000018  7B98              jpo 0xffffffb2
0000001A  46                inc esi
```

Shikata Ga Nai: Weaknesses

- Rule #1: Random instructions
  - Some instructions have few or no permutations
    - Fnstenv [esp – 0x0c]
    - Sub/xor ecx, ecx + mov cl, Xorlen
  - ECX is always as the loop counter
  - GetEIP and looping methods can have more permutations
- Rule #2: Random ordering
  - Done where possible
- Rule #3: Random NOP padding
  - Not done at all

Shikata Ga Nai: Weaknesses

- Some static instructions
- Basic fingerprinting possible

```
content: "|d9 74 24 f4|"; offset: 2;
pcre: "/[\x29\x2b\x31\x33]\xc9/sm";
pcre:"/[\xd9-\xdb\xdd].{1,11}\xd9\x74\x24\xf4.{0,10}
   [\x58\x5a\x5b\x5d\x5f]/sm";
```

- Weak fingerprint
- Possible to be more accurate?

Byte Distribution

- Few permutations are possible for most instructions
  - Each offset has only a limited number of possibilities
- First byte can only be fpuIns, movXorlen, or clearEcx. All possible combinations for the first byte:
  - `[\x29\x2b\x31\x33\xb8\xba\xbb\xbd-\xbf\xd9-\xdb\xdd]`
- Might be an effective method of narrowing down false positives

Byte Distribution

- ./spectrum: Encoder analyzer
  - Input: Large set of permutations of a polymorphic encoder (10,000+)
  - Output: Static bytes, PCREs, Spectrum analysis

```
vlad@debian:~/presentation/spectrum$ ./spectrum -sbp shikata_msf2_l1024 2>/dev/null
Total hits: 20000
Size distribution:
        27 bytes: 20000 hits
Static sequences: \xd9\x74\x24\xf4
PCRE: [\x29\x2b\x31\x33\xb8\xba\xbb\xbe\xbf\xd9-\xdb\xdd][^\x00\xff][^\x00\xff][^\x00
][^\x00\xff][^\x00][^\x00\xff][^\x00\xff][^\x00][^\x00][^\x00][^\x00\xff][^\x00\xff][
^\x00\xff][^\x00][^\x00][\x31\x83][\x42\x43\x46\x47\x50\x53\x56-\x58\x5a\x5e\x5f\x70\
x72\x73\x77\x78\x7a\x7b\x7e\xc0\xc2\xc3\xc6\xc7\xe8\xea\xeb\xee\xef][\x04\x0e\x10\x12
\x13\x15\x17\xfc][\x03\x31\x83][\x42\x43\x46\x47\x50\x53\x56-\x58\x5a\x5e\x5f\x70\x72
\x73\x77\x78\x7a\x7b\x7e\xc0\xc2\xc3\xc6\xc7\xe8\xea\xeb\xee\xef][\x04\x0a\x0c\x0e-\x
13\x15\x17\xfc][\x03\x83]..[^\x1d\xe2][^\x0a\xf5]
vlad@debian:~/presentation/spectrum$
```

Shikata Ga Nai: Fingerprint

```
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"SHELLCODE
  Shikata Ga Nai polymorphic payload"; classtype:shellcode-
  detect;
dsize: >26;
content: "|d9 74 24 f4|"; offset: 2;
pcre: "/[\x29\x2b\x31\x33]\xc9/sm";
pcre: "/[\xd9-\xdb\xdd].{1,11}\xd9\x74\x24\xf4.{0,10}
  [\x58\x5a\x5b\x5d-\x5f]/sm";
pcre: "/[\x29\x2b\x31\x33\xb8\xba\xbb\xbe\xbf\xd9-\xdb\xdd]
  [^\x00\xff][^\x00\xff][^\x00][^\x00\xff][^\x00]
  [^\x00\xff][^\x00\xff][^\x00][^\x00][^\x00][^\x00\xff]
  [^\x00\xff][^\x00\xff][^\x00][^\x00][\x31\x83]
  [\x42\x43\x46\x47\x50\x53\x56-\x58\x5a\x5e\x5f\x70\x72\x7
  3\x77\x78\x7a\x7b\x7e\xc0\xc2\xc3\xc6\xc7\xe8\xea\xeb\xee
  \xef][\x04\x0e\x10\x12\x13\x15\x17\xfc][\x03\x31\x83]
  [\x42\x43\x46\x47\x50\x53\x56-\x58\x5a\x5e\x5f\x70\x72\x7
  3\x77\x78\x7a\x7b\x7e\xc0\xc2\xc3\xc6\xc7\xe8\xea\xeb\xee
  \xef][\x04\x0a\x0c\x0e-\x13\x15\x17\xfc][\x03\x83]..
  [^\x1d\xe2][^\x0a\xf5]/sm“;)
```

Shikata Ga Nai: Fingerprint

- Shikata Ga Nai signature changes between Metasploit 2.x and 3.x versions.
- It also changes if the payload is greater then 1024 bytes.
- Fingerprint on the last slide only for Metasploit Framework 2.x
- code.tgz file should have the full signature.

CLET

- Encoding uses a variable amount of reversible instructions
  - Makes it harder to fingerprint
- Split into three parts
  - Start: Initialization
  - Middle: Ciphering
  - End: Looping

```
00000000  EB3B              jmp short 0x3d
00000002  58                pop eax
00000003  31D2              xor edx,edx
00000005  B220              mov dl,0x20
00000007  8B08              mov ecx,[eax]
00000009  C1C90F            ror ecx,0xf
0000000C  C1C10C            rol ecx,0xc
0000000F  81F1BD9A8391      xor ecx,0x91839abd
00000015  C1C919            ror ecx,0x19
00000018  81C1053B89F6      add ecx,0xf6893b05
0000001E  81C101E272D1      add ecx,0xd172e201
00000024  C1C90C            ror ecx,0xc
00000027  C1C91E            ror ecx,0x1e
0000002A  8908              mov [eax],ecx
0000002C  2DFEFFFFFF        sub eax,0xfffffffe
00000031  40                inc eax
00000032  40                inc eax
00000033  80EA01            sub dl,0x1
00000036  4A                dec edx
00000037  4A                dec edx
00000038  4A                dec edx
00000039  7407              jz 0x42
0000003B  EBCA              jmp short 0x7
0000003D  E8C0FFFFFF        call 0x2
```

CLET: Weaknesses

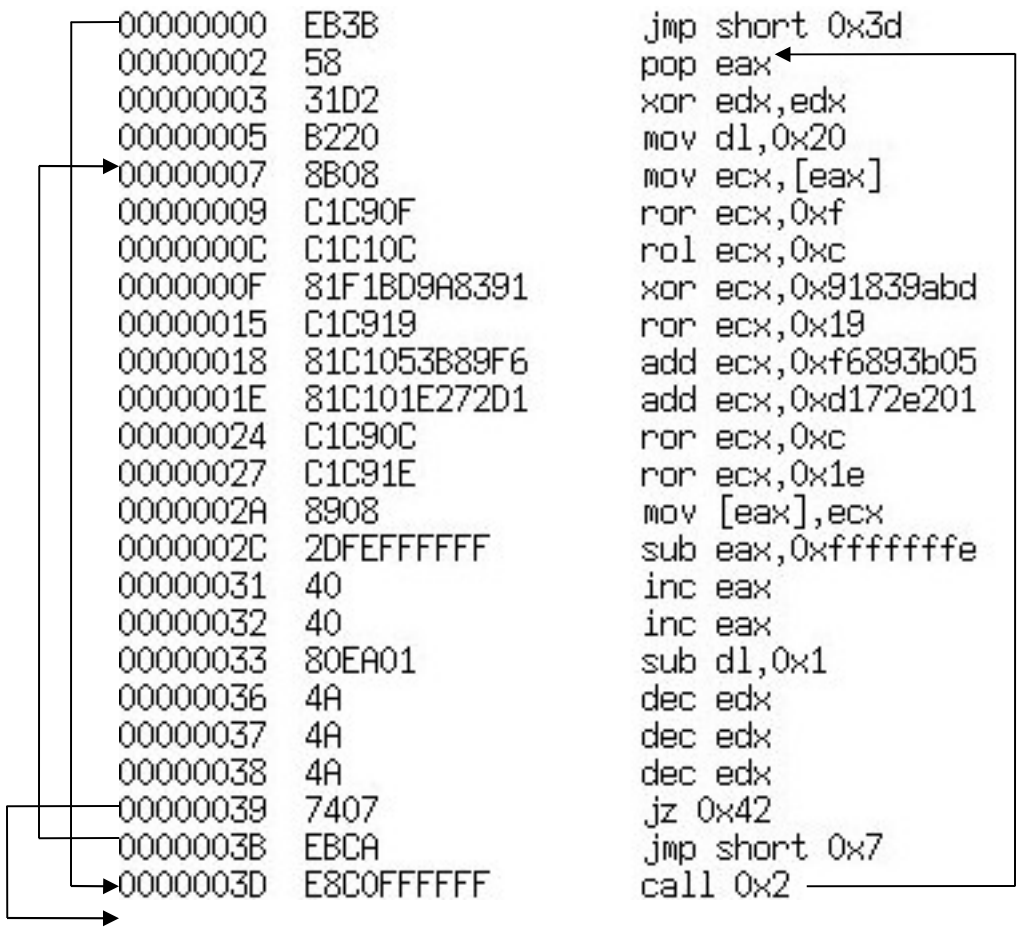
- Rule #1: Random instructions
  - Some instructions have few or no permutations
    - The starting and ending sections are predictable
- Rule #2: Random ordering
  - Only done in the middle section
- Rule #3: Random NOP padding
  - Not done

CLET: Byte distribution

- Middle section varies in length
  - Different CLET encoders differ in size
- Impossible to do effective byte distribution analysis
  - Unless we want to fingerprint just the start and the end
  - In that case we only need a large set of CLET encoders of the same size (that way the middle section is the same size, so we can just look at the start and end and ignore the middle)
    - Means some offsets may stay the same and only apply to that encoder length

CLET: Jump offsets

- The jmp/call offsets always go to the same instruction[s]
- We can use that as another fingerprint using snort's byte_jump
  - Annoying, I didn't implement it
  - You can if you want



CLET: Fingerprint

```
alert ip $EXTERNAL_NET any -> $HOME_NET any
  (msg:"SHELLCODE CLET polymorphic payload";
  classtype:shellcode-detect; \
dsize: >40; \
content: "|74 07 eb|"; offset: 10; \
content: "|e8|"; distance: 1; within: 1; \
content: "|ff ff ff|"; distance:1; within: 3; \
pcre: "/\xeb.[\x58-\x5b]\x31[\xc0\xc9\xd2\xdb]
  [\xb0-\xb3].\x8b/sm"; \
pcre: "/[\x40-\x43\xfd\xff][\x40-\x43\xff]
  [\x40-\x43\x80\xff][\x40-\x43\xe9-\xeb\xff\x80\x2c]
  [\x40-\x43\x48-\x4b\xe9-\xeb\x01\x2c\x80]
  [\x48-\x4c\xe9-\xeb\x02\x2c][\x03\x48-\x4b]
  [\x48-\x4b]\x74\x07\xeb.\xe8.[\xf0-\xff]\xff\xff/smR";)
```

```
00000000  EB38              jmp short 0x3a
00000002  5E                pop esi
00000003  8CC0              mov eax,es
00000005  31C9              xor ecx,ecx
00000007  68CA399D2D        push dword 0x2d9d39ca
0000000C  5B                pop ebx
0000000D  87D2              xchg edx,edx
0000000F  87D2              xchg edx,edx
00000011  8CE8              mov eax,gs
00000013  B101              mov cl,0x1
00000015  99                cdq
00000016  8B06              mov eax,[esi]
00000018  09D8              or eax,ebx
0000001A  211E              and [esi],ebx
0000001C  F716              not dword [esi]
0000001E  2106              and [esi],eax
00000020  48                dec eax
00000021  F8                clc
00000022  F5                cmc
00000023  4F                dec edi
00000024  83F08D            xor eax,byte -0x73
00000027  96                xchg eax,esi
00000028  40                inc eax
00000029  96                xchg eax,esi
0000002A  83C844            or eax,byte +0x44
0000002D  96                xchg eax,esi
0000002E  40                inc eax
0000002F  96                xchg eax,esi
00000030  46                inc esi
00000031  87DB              xchg ebx,ebx
00000033  46                inc esi
00000034  FC                cld
00000035  FC                cld
00000036  E2DE              loop 0x16
00000038  EB06              jmp short 0x40
0000003A  E8C3FFFFFF        call 0x2
```

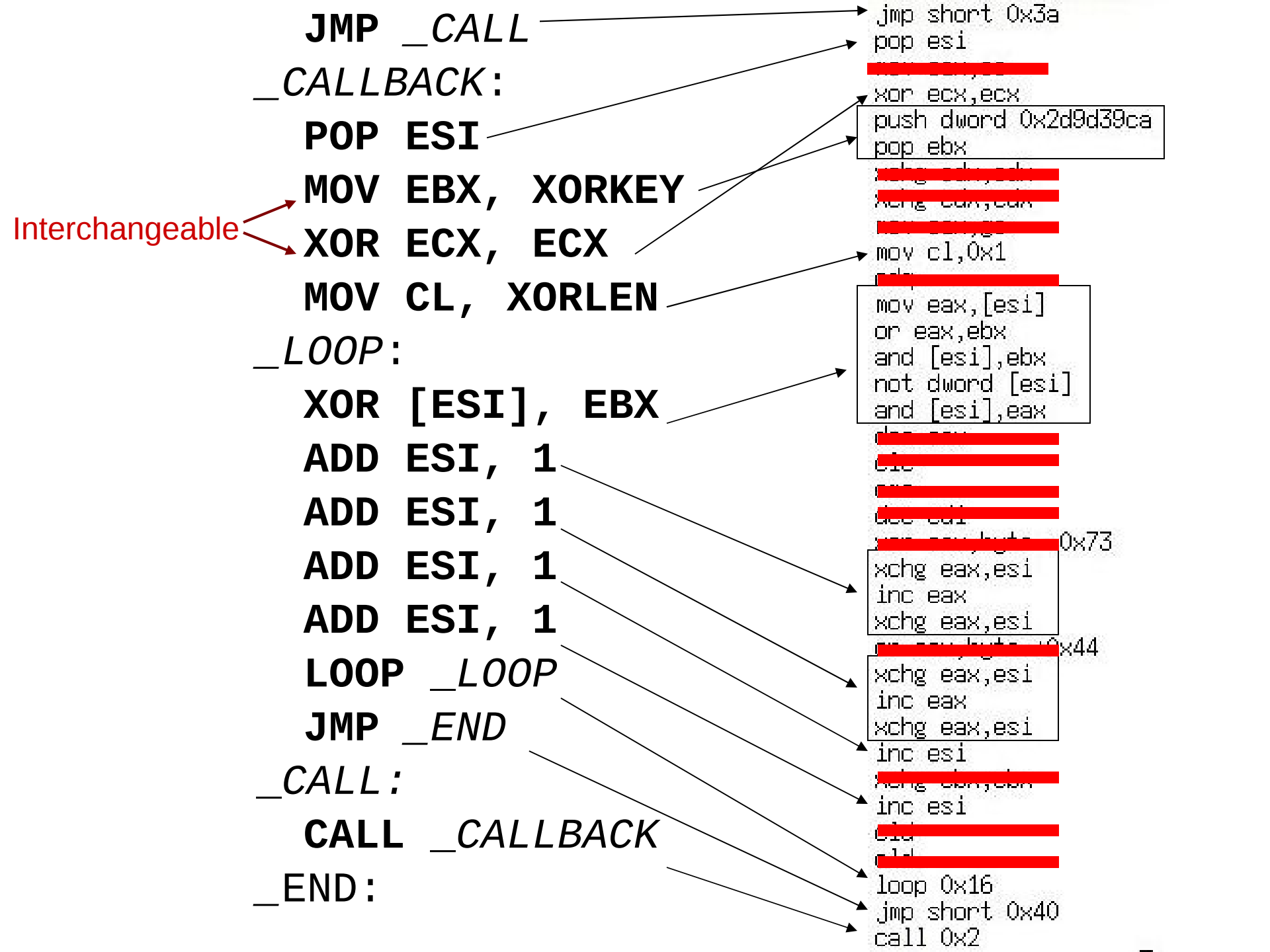
```
00000000  EB46              jmp short 0x48
00000002  F7D0              not eax
00000004  5E                pop esi
00000005  8CE0              mov eax,fs
00000007  9C                pushf
00000008  98                cwde
00000009  47                inc edi
0000000A  F8                clc
0000000B  31C9              xor ecx,ecx
0000000D  BB409E439C        mov ebx,0x9c439e40
00000012  2F                das
00000013  47                inc edi
00000014  83E027            and eax,byte +0x27
00000017  83F0E0            xor eax,byte -0x20
0000001A  83C0F7            add eax,byte -0x9
0000001D  6A01              push byte +0x1
0000001F  6659              pop cx
00000021  9F                lahf
00000022  98                cwde
00000023  9E                sahf
00000024  311E              xor [esi],ebx
00000026  83C601            add esi,byte +0x1
00000029  83C601            add esi,byte +0x1
0000002C  3F                aas
0000002D  83E0C1            and eax,byte -0x3f
00000030  8CC0              mov eax,es
00000032  9C                pushf
00000033  83C601            add esi,byte +0x1
00000036  85C0              test eax,eax
00000038  96                xchg eax,esi
00000039  40                inc eax
0000003A  96                xchg eax,esi
0000003B  90                nop
0000003C  33C0              xor eax,eax
0000003E  99                cdq
0000003F  F9                stc
00000040  92                xchg eax,edx
00000041  E2E1              loop 0x24
00000043  EB09              jmp short 0x4e
00000045  47                inc edi
00000046  4F                dec edi
00000047  F8                clc
00000048  E8B7FFFFFF        call 0x4
```

ADMutate: Weaknesses

- Rule #1: Random instructions
  - All instructions have few permutations
    - JMPs and CALLs always go to a specific instruction not NOP padding (makes it easier to fingerprint)
- Rule #2: Random ordering
  - Only done for two instructions
- Rule #3: Random NOP padding
  - Done everywhere
  - JMPs and CALLs always go to a specific instruction not NOP padding (makes it easier to fingerprint)

ADMutate: Byte distribution

- Byte distribution not possible, random NOP padding means that you have to find the combinations by hand or look at the source code (ADMmuteng.h)
- Small amount of variable instructions
- Jump offsets always go to the same instructions


JMP _CALL
_CALLBACK:
POP ESI
MOV EBX, XORKEY
XOR ECX, ECX
MOV CL, XORLEN
_LOOP:
XOR [ESI], EBX
ADD ESI, 1
ADD ESI, 1
ADD ESI, 1
ADD ESI, 1
LOOP _LOOP
JMP _END
_CALL:
CALL _CALLBACK
_END:
Interchangeable
jmp short 0x3a
pop esi
xor ecx,ecx
push dword 0x2d9d39ca
pop ebx
mov cl,0x1
mov eax,[esi]
or eax,ebx
and [esi],ebx
not dword [esi]
and [esi],eax
0x73
xchg eax,esi
inc eax
xchg eax,esi
x44
xchg eax,esi
inc eax
xchg eax,esi
inc esi
inc esi
loop 0x16
jmp short 0x40
call 0x2

ADMutate: Fingerprint

```
alert ip $EXTERNAL_NET any -> $HOME_NET any (msg:"SHELLCODE
  ADMutate polymorphic payload"; classtype:shellcode-detect; \
dsize: >45; \
content: "|e8|"; \
content: "|ff ff ff|"; distance: 1; within: 3; \
pcre: "/\xeb[\x26-\x7a].{0,20}
  (\x5e|\x58\x96|\x58\x89\xc6|\x8b\x34\x24\x83\xec\x04).{0,20}
  (((\xbb....|\x68....\x5b).{0,20}(\x31\xc9|\x31\xc0\x91))|
  ((\x31\xc9|\x31\xc0\x91).{0,20}(\xbb....|\x68....\x5b))).{0,20}
  (\xb1.|\x6a.\x58\x89\xc1|\x6a.\x66\x59).{0,20}
  (\x31\x1e|\x93\x31\x06\x93|\x8b\x06\x09\xd8\x21\x1e\xf7\x16\x21
  \x06).{0,20}(\x46|\x83\xc6\x01|\x96\x40\x96).{0,20}
  (\x46|\x83\xc6\x01|\x96\x40\x96).{0,20}
  (\x46|\x83\xc6\x01|\x96\x40\x96).{0,20}
  (\x46|\x83\xc6\x01|\x96\x40\x96).{0,20}\xe2[\xa0-\xf9].
  {0,20}\xeb[\x06-\x20].{0,20}\xe8[\x7f-\xff]\xff\xff\xff/sm";)
```

Future - Detection

- X-rays
  - Limited to XOR encoders with predictable shellcode byte sequences
- Dynamic translation vs. Emulation

- Scoring systems
  - X stream has 50 NOPs = .5
  - X stream protocol anomaly = .25
  - X stream looks like it may have shellcode = .5
  - Overall score: 1.25 **DROP**

Future - Evasion

- More in-depth polymorphism
- Polymorphism on shellcode vs. encoders
- Use low level OS details
  - Breaks dynamic translation
    - mov eax, 0x08048000 / mov ebx, [eax] # XORKEY
- Better exploit-side technique
  - Use egghunters
- Use instructor-specific or obscure instructions
  - Weak, just means some new instructions have to be added.