Recent Shellcode Developments

spoonm

ReCon, 2005

Part I

Introduction

Who am I?

- spoonm
- Metasploit developer since 2003
- University student
- Independent security researcher

What is this talk about?

- Recent shellcode research
- Older but lesser known tricks
- New tricks and techniques

Overview of shellcode structure

[payload]

- The payload does stuff

Part II

Shellcode

Making a connection

- Connect back to attacker (reverse)
- Connect to victim (bind)
- Reuse existing connection (find)
- Findsock only worth talking about

getpeername findsock

- Made popular by LSD
- Embed connection source port in shellcode
- Call getpeername in a loop
- Found the socket when matching source port

getpeername findsock

- Made popular by LSD
- Embed connection source port in shellcode
- Call getpeername in a loop
- Found the socket when matching source port
- Pros
  - Simple idea and implementation
  - Works well when it works

getpeername findsock

- Made popular by LSD
- Embed connection source port in shellcode
- Call getpeername in a loop
- Found the socket when matching source port
- Pros
  - Simple idea and implementation
  - Works well when it works
- Cons
  - Doesn’t work through a proxy
  - Doesn’t work through a NAT
  - You need to embed source port

Find tag findsock

- Metasploit moved to using this
- Embed tag in shellcode
- Call recv in a loop
- Use MSG_DONTWAIT on linux
- Found the socket when you find the tag

Find tag findsock

- Metasploit moved to using this
- Embed tag in shellcode
- Call recv in a loop
- Use MSG_DONTWAIT on linux
- Found the socket when you find the tag
- Pros
  - Will work through a proxy or NAT
  - Still fairly simple and small

Find tag findsock

- Metasploit moved to using this
- Embed tag in shellcode
- Call recv in a loop
- Use MSG_DONTWAIT on linux
- Found the socket when you find the tag
- Pros
  - Will work through a proxy or NAT
  - Still fairly simple and small
- Cons
  - Data may be left on some sockets
  - Can be timing sensitive
  - More complicated on windows (ioctlsocket, etc)

Part III

Building a Nop Sled

Nop sleds, what and why?

- Often (especially on unix) we don’t know exactly were our payload is
- Nop sleds makes the target we are trying to hit bigger
- The bigger the nop sled, the better the brute force

Improved shellcode structure

[nop sled][payload]

Improved shellcode structure

[nop sled][payload]

- The nop sled slides into the payload
- The payload does stuff

Multibyte Nop Sled Concept

- Optyx released multibyte nop generator at Interz0ne 1
- Generates instructions 1 to 6 bytes long, and uses 0x66 prefix
- Aligned to 1 byte, land anywhere, end up at the final target

Multibyte Nop Sled Concept

- Optyx released multibyte nop generator at Interz0ne 1
- Generates instructions 1 to 6 bytes long, and uses 0x66 prefix
- Aligned to 1 byte, land anywhere, end up at the final target

- Builds the sled from back to front
- Prepends to the sled 1 byte at a time
- Generates a random byte and checks against tables

Multibyte Nop Sled Concept

- Optyx released multibyte nop generator at Interz0ne 1
- Generates instructions 1 to 6 bytes long, and uses 0x66 prefix
- Aligned to 1 byte, land anywhere, end up at the final target

- Builds the sled from back to front
- Prepends to the sled 1 byte at a time
- Generates a random byte and checks against tables
  - Is the instruction length too long?
  - Is it a valid instruction?
  - Does it have any bad bytes?
  - Does it modify restricted registers?

Backwardz

```
bb b0 bf 2c b6 27 67 2F 4A 1b f9 -- shellcode
|  |  |  |  |  |  |  |  |  |  | ... stc
|  |  |  |  |  |  |  |  |  |____^ . sbb edi,ecx
|  |  |  |  |  |  |  |  | ......... dec edx
|  |  |  |  |  |  |  | ............ das
|  |  |  |  |  |  |____^ .......... a16 das
|  |  |  |  |  | .................. daa
|  |  |  |  |____^ ................ mov dh, 0x27
|  |  |  |____^ ................... sub al, 0xb6
|  |  |_____________^ ............. mov edi, 0x6727b62c
|  |____^ ......................... mov al, 0xbf
|_____________^ ................... mov ebx, 0xb62cbfb0
```

Backwardz

```
bb b0 bf 2c b6 27 67 2F 4A 1b f9 -- shellcode
|  |  |  |  |  |  |  |  |  |  | ... stc
|  |  |  |  |  |  |  |  |  |____^ . sbb edi,ecx
|  |  |  |  |  |  |  |  | ......... dec edx
|  |  |  |  |  |  |  | ............ das
|  |  |  |  |  |  |____^ .......... a16 das
|  |  |  |  |  | .................. daa
|  |  |  |  |____^ ................ mov dh, 0x27
|  |  |  |____^ ................... sub al, 0xb6
|  |  |_____________^ ............. mov edi, 0x6727b62c
|  |____^ ......................... mov al, 0xbf
|_____________^ ................... mov ebx, 0xb62cbfb0
```

Backwardz

```
bb b0 bf 2c b6 27 67 2F 4A 1b f9 -- shellcode
|  |  |  |  |  |  |  |  |  |  | ... stc
|  |  |  |  |  |  |  |  |  |____^ . sbb edi,ecx
|  |  |  |  |  |  |  |  | ......... dec edx
|  |  |  |  |  |  |  | ............ das
|  |  |  |  |  |  |____^ .......... a16 das
|  |  |  |  |  | .................. daa
|  |  |  |  |____^ ................ mov dh, 0x27
|  |  |  |____^ ................... sub al, 0xb6
|  |  |_____________^ ............. mov edi, 0x6727b62c
|  |____^ ......................... mov al, 0xbf
|_____________^ ................... mov ebx, 0xb62cbfb0
```

Backwardz

```
bb b0 bf 2c b6 27 67 2F 4A 1b f9 -- shellcode
|  |  |  |  |  |  |  |  |  |  | ... stc
|  |  |  |  |  |  |  |  |  |____^ . sbb edi,ecx
|  |  |  |  |  |  |  |  | ......... dec edx
|  |  |  |  |  |  |  | ............ das
|  |  |  |  |  |  |____^ .......... a16 das
|  |  |  |  |  | .................. daa
|  |  |  |  |____^ ................ mov dh, 0x27
|  |  |  |____^ ................... sub al, 0xb6
|  |  |_____________^ ............. mov edi, 0x6727b62c
|  |____^ ......................... mov al, 0xbf
|_____________^ ................... mov ebx, 0xb62cbfb0
```

Backwardz

```
bb b0 bf 2c b6 27 67 2F 4A 1b f9 -- shellcode
|  |  |  |  |  |  |  |  |  |  | ... stc
|  |  |  |  |  |  |  |  |  |____^ . sbb edi,ecx
|  |  |  |  |  |  |  |  | ......... dec edx
|  |  |  |  |  |  |  | ............ das
|  |  |  |  |  |  |____^ .......... a16 das
|  |  |  |  |  | .................. daa
|  |  |  |  |____^ ................ mov dh, 0x27
|  |  |  |____^ ................... sub al, 0xb6
|  |  |_____________^ ............. mov edi, 0x6727b62c
|  |____^ ......................... mov al, 0xbf
|_____________^ ................... mov ebx, 0xb62cbfb0
```

Backwardz

```
bb b0 bf 2c b6 27 67 2F 4A 1b f9 -- shellcode
|  |  |  |  |  |  |  |  |  |  | ... stc
|  |  |  |  |  |  |  |  |  |____^ . sbb edi,ecx
|  |  |  |  |  |  |  |  | ......... dec edx
|  |  |  |  |  |  |  | ............ das
|  |  |  |  |  |  |____^ .......... a16 das
|  |  |  |  |  | .................. daa
|  |  |  |  |____^ ................ mov dh, 0x27
|  |  |  |____^ ................... sub al, 0xb6
|  |  |_____________^ ............. mov edi, 0x6727b62c
|  |____^ ......................... mov al, 0xbf
|_____________^ ................... mov ebx, 0xb62cbfb0
```

Backwardz

```
bb b0 bf 2c b6 27 67 2F 4A 1b f9 -- shellcode
|  |  |  |  |  |  |  |  |  |  | ... stc
|  |  |  |  |  |  |  |  |  |____^ . sbb edi,ecx
|  |  |  |  |  |  |  |  | ......... dec edx
|  |  |  |  |  |  |  | ............ das
|  |  |  |  |  |  |____^ .......... a16 das
|  |  |  |  |  | .................. daa
|  |  |  |  |____^ ................ mov dh, 0x27
|  |  |  |____^ ................... sub al, 0xb6
|  |  |_____________^ ............. mov edi, 0x6727b62c
|  |____^ ......................... mov al, 0xbf
|_____________^ ................... mov ebx, 0xb62cbfb0
```

Backwardz

```
bb b0 bf 2c b6 27 67 2F 4A 1b f9 -- shellcode
|  |  |  |  |  |  |  |  |  |  | ... stc
|  |  |  |  |  |  |  |  |  |____^ . sbb edi,ecx
|  |  |  |  |  |  |  |  | ......... dec edx
|  |  |  |  |  |  |  | ............ das
|  |  |  |  |  |  |____^ .......... a16 das
|  |  |  |  |  | .................. daa
|  |  |  |  |____^ ................ mov dh, 0x27
|  |  |  |____^ ................... sub al, 0xb6
|  |  |_____________^ ............. mov edi, 0x6727b62c
|  |____^ ......................... mov al, 0xbf
|_____________^ ................... mov ebx, 0xb62cbfb0
```

Backwardz

```
bb b0 bf 2c b6 27 67 2F 4A 1b f9 -- shellcode
|  |  |  |  |  |  |  |  |  |  | ... stc
|  |  |  |  |  |  |  |  |  |____^ . sbb edi,ecx
|  |  |  |  |  |  |  |  | ......... dec edx
|  |  |  |  |  |  |  | ............ das
|  |  |  |  |  |  |____^ .......... a16 das
|  |  |  |  |  | .................. daa
|  |  |  |  |____^ ................ mov dh, 0x27
|  |  |  |____^ ................... sub al, 0xb6
|  |  |_____________^ ............. mov edi, 0x6727b62c
|  |____^ ......................... mov al, 0xbf
|_____________^ ................... mov ebx, 0xb62cbfb0
```

Backwardz

```
bb b0 bf 2c b6 27 67 2F 4A 1b f9 -- shellcode
|  |  |  |  |  |  |  |  |  |  | ... stc
|  |  |  |  |  |  |  |  |  |____^ . sbb edi,ecx
|  |  |  |  |  |  |  |  | ......... dec edx
|  |  |  |  |  |  |  | ............ das
|  |  |  |  |  |  |____^ .......... a16 das
|  |  |  |  |  | .................. daa
|  |  |  |  |____^ ................ mov dh, 0x27
|  |  |  |____^ ................... sub al, 0xb6
|  |  |_____________^ ............. mov edi, 0x6727b62c
|  |____^ ......................... mov al, 0xbf
|_____________^ ................... mov ebx, 0xb62cbfb0
```

Backwardz

```
bb b0 bf 2c b6 27 67 2F 4A 1b f9 -- shellcode
|  |  |  |  |  |  |  |  |  |  | ... stc
|  |  |  |  |  |  |  |  |  |____^ . sbb edi,ecx
|  |  |  |  |  |  |  |  | ......... dec edx
|  |  |  |  |  |  |  | ............ das
|  |  |  |  |  |  |____^ .......... a16 das
|  |  |  |  |  | .................. daa
|  |  |  |  |____^ ................ mov dh, 0x27
|  |  |  |____^ ................... sub al, 0xb6
|  |  |_____________^ ............. mov edi, 0x6727b62c
|  |____^ ......................... mov al, 0xbf
|_____________^ ................... mov ebx, 0xb62cbfb0
```

OptyNop2 Implementation

- Generate random byte and check against tables
  - Inefficent, hard to get even distributions

OptyNop2 Implementation

- Generate random byte and check against tables
  - Inefficent, hard to get even distributions
- Generate random byte and check against disassembler
  - Need a good disassembler
  - Same problems as tables

OptyNop2 Implementation

- Generate random byte and check against tables
  - Inefficent, hard to get even distributions
- Generate random byte and check against disassembler
  - Need a good disassembler
  - Same problems as tables
- Precompiled state transition tables
  - Previous byte: 0x90 -> {0x04, 1, EAX} ... # add al,0x90

OptyNop2 Implementation

- Generate random byte and check against tables
  - Inefficent, hard to get even distributions
- Generate random byte and check against disassembler
  - Need a good disassembler
  - Same problems as tables
- Precompiled state transition tables
  - Previous byte: 0x90 -> {0x04, 1, EAX} ... # add al,0x90
  - Fairly language independent, C version 100 lines
  - Very fast, simple, deterministic
  - Allows for different scoring systems, recursion...

OptyNop2 Implementation

- Generate random byte and check against tables
  - Inefficent, hard to get even distributions
- Generate random byte and check against disassembler
  - Need a good disassembler
  - Same problems as tables
- Precompiled state transition tables
  - Previous byte: 0x90 -> {0x04, 1, EAX} ... # add al,0x90
  - Fairly language independent, C version 100 lines
  - Very fast, simple, deterministic
  - Allows for different scoring systems, recursion...
  - Can't support multibyte opcodes, escape groups, etc
  - Tables are pretty large, about 124k

OptyNop2 Output

```
$ ./waka 1000 4 5 | ndisasm -u - | head -700 | tail -20
000003B6  05419F40D4        add eax,0xd4409f41
000003BB  711C              jno 0x3d9
000003BD  9B                wait
000003BE  2C98              sub al,0x98
000003C0  37                aaa
000003C1  24A8              and al,0xa8
000003C3  27                daa
000003C4  E00D              loopne 0x3d3
000003C6  6692              xchg ax,dx
000003C8  2F                das
000003C9  49                dec ecx
000003CA  B34A              mov bl,0x4a
000003CC  F5                cmc
000003CD  BA4B257715        mov edx,0x1577254b
000003D2  700C              jo 0x3e0
000003D4  C0D6B0            rcl dh,0xb0
000003D7  A9FD469342        test eax,0x429346fd
000003DC  67BBB191B23D      a16 mov ebx,0x3db291b1
000003E2  1D9938FCB6        sbb eax,0xb6fc3899
000003E7  43                inc ebx
```

ADMmutate Distribution - 1

```
total: 6000
uniq:  52
   00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20 00 00 00 00 00 00 00 6e 00 00 00 00 00 00 00 76
30 00 00 00 00 00 00 00 87 00 00 00 00 00 00 00 6a
40 6b 72 6a 68 74 66 77 6f 6d 74 6c 77 70 74 58 72
50 6a 67 71 70 7b 74 76 7c 70 7c 6b 78 00 6e 56 64
60 71 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90 00 89 6c 78 00 74 72 df 7a 79 00 56 82 00 76 77
a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
f0 00 00 00 00 00 7c 00 00 71 7f 00 00 69 00 00 00
```

ADMmutate Distribution - 2

```
total: 6000
uniq:  52
   00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
20 00 00 00 00 00 00 00 64 00 00 00 00 00 00 00 6f
30 00 00 00 00 00 00 00 78 00 00 00 00 00 00 00 74
40 7f 6b 6f 7b 79 72 75 73 76 58 6f 7a 6c 78 7a 7e
50 71 6d 65 75 7f 72 7b 72 71 77 6d 64 00 71 7c 64
60 73 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
70 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
80 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
90 00 6b 79 87 00 74 74 e8 6b 68 00 76 5b 00 6d 72
a0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
b0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
f0 00 00 00 00 00 75 00 00 57 6b 00 00 6f 00 00 00
```

OptyNop2 Distribution - 1

total: 6000
uniq: 141

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	00	12	12	12	39	39	00	00	12	11	11	11	39	39	00	00
10	12	12	12	11	39	39	00	00	12	12	12	12	39	39	00	00
20	12	11	12	12	39	39	00	39	12	12	11	12	39	39	00	39
30	11	11	12	12	39	39	00	39	11	11	12	11	39	39	00	39
40	39	39	39	3a	00	00	39	39	39	39	39	39	00	00	39	3a
50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	39	39	00	12	00	11	00	00	00	00
70	3a	39	39	39	39	39	39	39	39	39	39	39	3a	39	39	39
80	12	12	00	12	12	11	11	12	12	12	00	00	00	00	00	00
90	39	39	39	3a	00	00	39	39	39	39	00	39	00	00	00	39
a0	00	00	00	00	00	00	00	00	3a	39	00	00	00	00	00	00
b0	3a	39	39	39	39	3a	39	39	39	39	39	39	00	00	3a	39
c0	12	12	00	00	00	00	00	00	00	00	00	00	00	00	00	00
d0	12	12	12	11	39	39	39	00	00	00	00	00	00	00	00	00
e0	39	39	39	39	00	00	00	00	00	00	00	39	00	00	00	00
f0	00	00	00	00	00	39	11	11	3a	39	00	00	39	39	11	11

OptyNop2 Distribution - 2

total: 6000
uniq: 141

	00	01	02	03	04	05	06	07	08	09	0a	0b	0c	0d	0e	0f
00	00	12	11	11	39	3a	00	00	11	12	12	12	39	39	00	00
10	11	11	11	11	39	39	00	00	11	12	11	11	39	39	00	00
20	12	12	12	12	39	3a	00	3a	12	11	12	12	39	39	00	39
30	11	12	12	11	39	3a	00	3a	12	12	12	12	39	39	00	39
40	39	3a	3a	39	00	00	39	39	39	39	39	3a	00	00	39	39
50	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
60	00	00	00	00	00	00	39	39	00	12	00	11	00	00	00	00
70	39	39	39	39	3a	39	39	39	39	39	39	39	39	3a	39	39
80	11	12	00	12	11	12	11	12	12	12	00	00	00	00	00	00
90	39	39	39	3a	00	00	39	3a	3a	3a	00	39	00	00	00	39
a0	00	00	00	00	00	00	00	00	39	39	00	00	00	00	00	00
b0	39	39	39	39	39	39	39	39	39	3a	39	39	00	00	39	39
c0	11	11	00	00	00	00	00	00	00	00	00	00	00	00	00	00
d0	12	12	11	11	39	39	3a	00	00	00	00	00	00	00	00	00
e0	3a	39	39	39	00	00	00	00	00	00	00	39	00	00	00	00
f0	00	00	00	00	00	39	11	12	39	39	00	00	39	39	10	10

ADMmutate and optyx-mutate Gzip'd

```
# ADMmutate

$ time ./nops 1000000| gzip -v >/dev/null
 27.3%
real    0m0.241s

# optyx's interz0ne mutate

$ time ./driver nop 1000000 | gzip -v >/dev/null
 29.7%
real    0m0.467s
```

OptyNop2 Gzip'd

```
# C version, save ESP and EBP

$ time ./waka 1000000 4 5 | gzip -v >/dev/null
 12.2%
real    0m11.900s

# save just ESP

$ time ./waka 1000000 4 | gzip -v >/dev/null
 11.7%
real    0m11.277s

# save nothing (good way to crash process)

$ time ./waka 1000000 | gzip -v >/dev/null
  8.3%
real    0m12.404s
```

Conclusion

- Benefits
  - Handles restricted bytes and registers
  - More versatile sled generation (nop stuffing, etc)
  - Implementation and theory are simple

Conclusion

- Benefits
  - Handles restricted bytes and registers
  - More versatile sled generation (nop stuffing, etc)
  - Implementation and theory are simple
- Possible Improvements
  - Support processor flags (nop stuffing)
  - Support 2-byte opcodes and escape groups
  - Improved byte scoring systems and look-ahead
  - Output according to a given byte distribution
  - Reduce the table sizes, memory usage

Part IV

Encoders

Encoder, what and why?

- We need to avoid bad characters
- Now we don't need to worry about this in the payload

Encoder, what and why?

- We need to avoid bad characters
- Now we don't need to worry about this in the payload

- Encodes the payload to a different set of characters
- Common methods: byte/word/dword xor, add, etc
- Prepends a decoder before the decoded data
- Decoder loops and decodes the payload

Improved shellcode structure

[nop sled][encoder (payload)]

- The nop sled slides into the decoder
- The decoder decodes the payload
- The payload does stuff

The simplest, call, 6 bytes

```
00000000  E800000000        call 0x5
00000005  58                pop eax
```

The simplest, call, 6 bytes

```
00000000  E800000000          call 0x5
00000005  58                  pop eax
```

- Call pushes EIP of the pop instruction on the stack
- pop puts it in a register

jmp / call, 8 bytes

- You often need to avoid having 0x00

jmp / call, 8 bytes

- You often need to avoid having 0x00

```
00000000  EB02              jmp short 0x4
00000002  58                pop eax
00000003  90                nop
00000004  E8F9FFFFFF        call 0x2
```

jmp / call, 8 bytes

- You often need to avoid having 0x00

```
00000000  EB02              jmp short 0x4
00000002  58                pop eax
00000003  90                nop
00000004  E8F9FFFFFF        call 0x2
```

- Jmp to a call instruction
- The call is now backwards (negative)

FPU Get EIP, 7 bytes

- Sometimes you want to avoid 0xff
- Noir’s Get EIP

FPU Get EIP, 7 bytes

- Sometimes you want to avoid 0xff
- Noir’s Get EIP

```
00000000  D9EE                  fldz
00000002  D97424F4              fnstenv [esp-0xc]
00000006  58                    pop eax
```

FPU Get EIP, 7 bytes

- Sometimes you want to avoid 0xff
- Noir's Get EIP

```
00000000  D9EE            fldz
00000002  D97424F4        fnstenv [esp-0xc]
00000006  58              pop eax
```

- fnsetenv will get EIP of last fpu instruction
- It allows for much more permutations
- Also smaller than jmp/call

Call $+4, 7 bytes

- The coolest of them all
- Gera & CoreST

Call $+4, 7 bytes

- The coolest of them all
- Gera & CoreST

```
00000000  E8FFFFFFFF          call 0x4
00000005  C3                  ret
00000006  58                  pop eax

00000004  FFC3                inc ebx
00000006  58                  pop eax
```

Call $+4, 7 bytes

- The coolest of them all
- Gera & CoreST

```
00000000  E8FFFFFFFF          call 0x4
00000005  C3                  ret
00000006  58                  pop eax

00000004  FFC3                inc ebx
00000006  58                  pop eax
```

- Call is relative to the end of it’s instruction
- Call jmps into itself, decodes an 0xff instruction
- can inc, dec, or push reg

SEH, bigger

- People really want a alpha numeric get eip

SEH, bigger

- People really want a alpha numeric get eip
- Well, so far, only found a way for windows
- Link in exception handler, cause exception
- Get EIP from the CONTEXT record

CLET

- Generates permutations of decoder stubs
- Inserts reversing instructions, nop equivalents
- All decoders are C code to generate themselves

CLET

- Generates permutations of decoder stubs
- Inserts reversing instructions, nop equivalents
- All decoders are C code to generate themselves
- Pros:
  - Well thought out - analyzed attacks against NIDS
  - Mathematica files output, mathy backing
  - Specturm analysis - push sled to byte distribution

CLET

- Generates permutations of decoder stubs
- Inserts reversing instructions, nop equivalents
- All decoders are C code to generate themselves
- Pros:
  - Well thought out - analyzed attacks against NIDS
  - Mathematica files output, mathy backing
  - Specturm analysis - push sled to byte distribution
- Cons:
  - Complicated system, really hard to build upon
  - Decoder generation isn’t that great
  - Making compromises for size/robustness

Metasploit Pex::Poly

- "Conservative Polymorphism"
- Uses the inherit variability in shellcode

Metasploit Pex::Poly

- "Conservative Polymorphism"
- Uses the inherit variability in shellcode
- Pros:
  - Polymorphizing code is pretty easy
  - No size or functionality compromises
  - Bad character and register avoidence

Metasploit Pex::Poly

- "Conservative Polymorphism"
- Uses the inherit variability in shellcode
- Pros:
  - Polymorphizing code is pretty easy
  - No size or functionality compromises
  - Bad character and register avoidence
- Cons:
  - Less thought out, NIDS attacks not deeply analyzed
  - Hard to push to arbitrary byte distribution
  - Less "polymorphism", more restrictions

Implementation - Pex::Poly

- "Blocks" are dependency graph nodes
- "Blocks" consist of 0 or more possibilities
- Register pool assignment (mov reg1, reg2)
- Gained robustness as a nice effect

Implementation - Pex::Poly

- "Blocks" are dependency graph nodes
- "Blocks" consist of 0 or more possibilities
- Register pool assignment (mov reg1, reg2)
- Gained robustness as a nice effect
- Current implementation
  - Current system is a bit ugly
  - Hard without writing a real assembler
  - Want it to be fairly fast

Implementation - Pex::Poly

- "Blocks" are dependency graph nodes
- "Blocks" consist of 0 or more possibilities
- Register pool assignment (mov reg1, reg2)
- Gained robustness as a nice effect
- Current implementation
  - Current system is a bit ugly
  - Hard without writing a real assembler
  - Want it to be fairly fast
  - Pex::Poly has 3 phases
  - Dependency iteration and block selection
  - Instruction offset calculations
  - Instruction register assignment

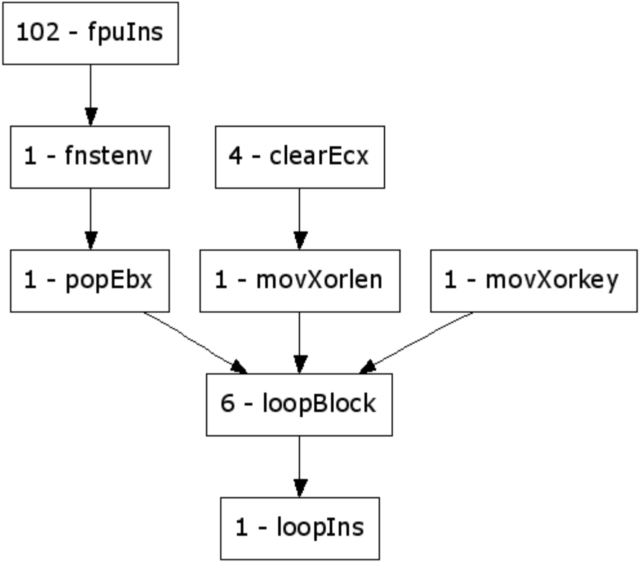
Shikata Ga Nai

- It’s too much work to polyize every payload
- Created one decent "polymorphic" encoder

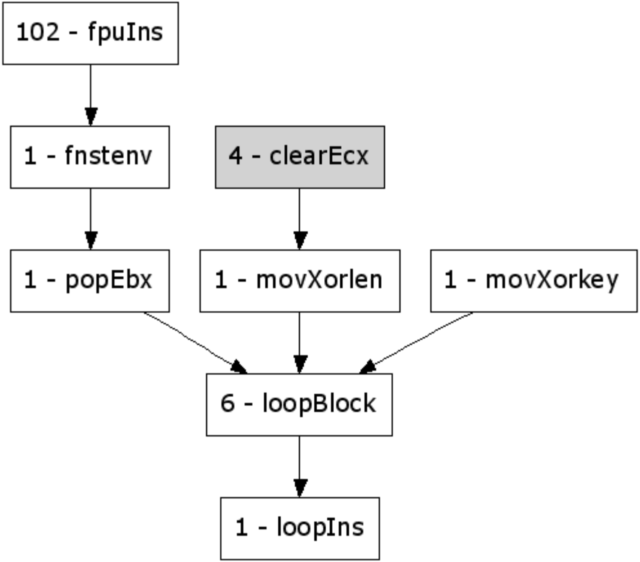
Shikata Ga Nai

- It’s too much work to polyize every payload
- Created one decent "polymorphic" encoder
- Noir’s FPU geteip technique
- Approximately 1.3 million permutations
- Additive feedback xor, encodes it’s own end
- 27 bytes for the stub, 4 key, 4 encoded

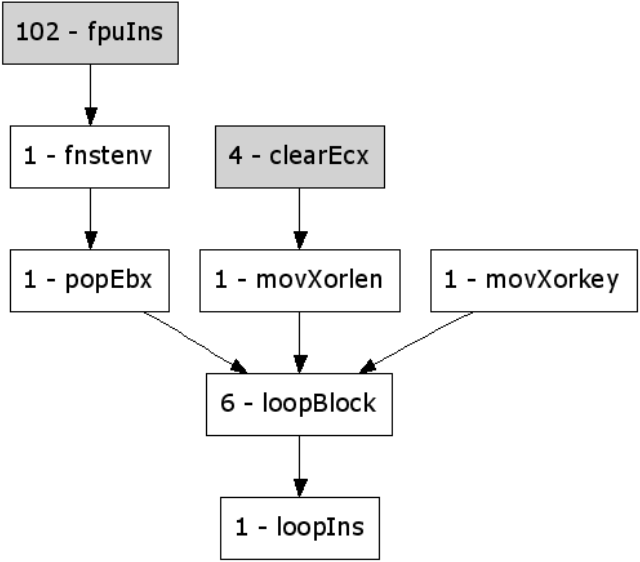
Shikata dependency iteration



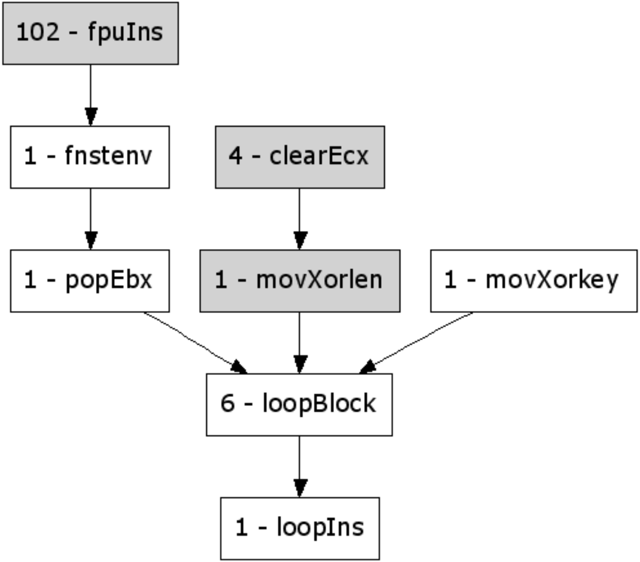
Shikata dependency iteration



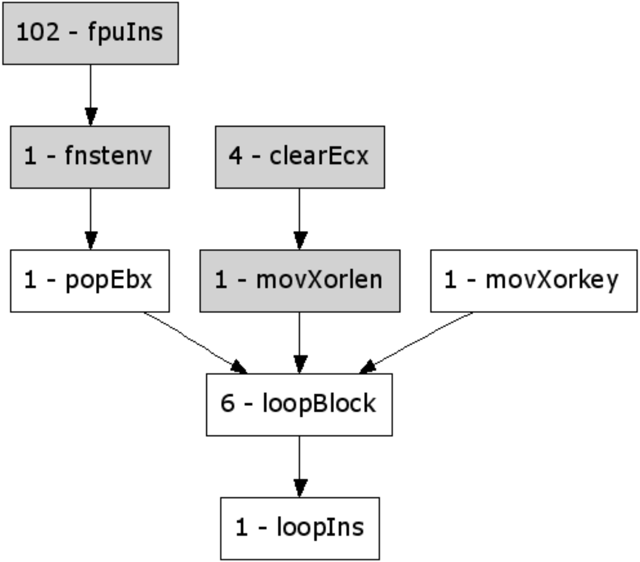
Shikata dependency iteration



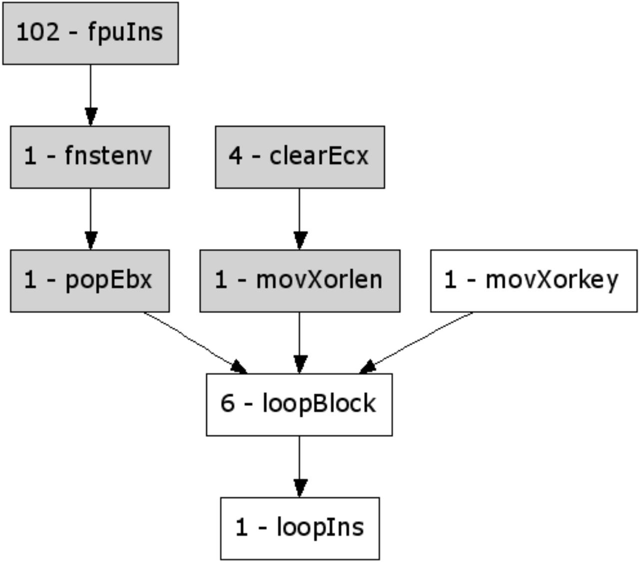
Shikata dependency iteration



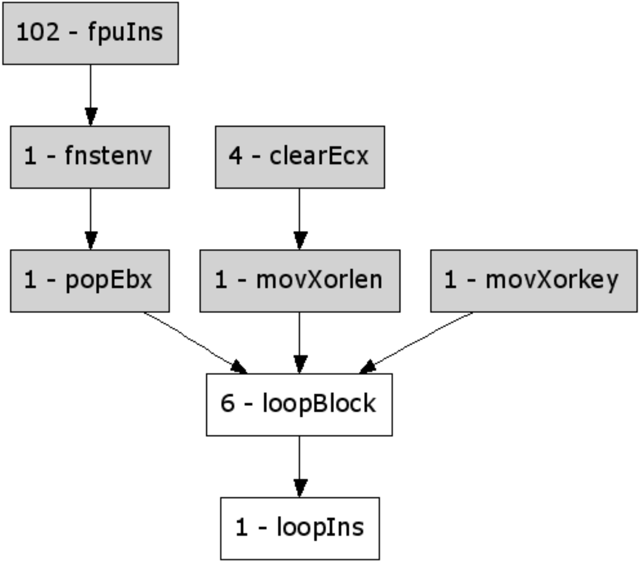
Shikata dependency iteration



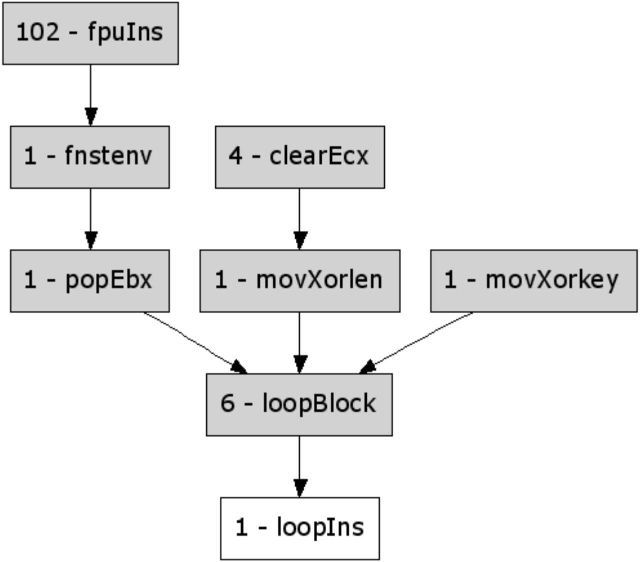
Shikata dependency iteration



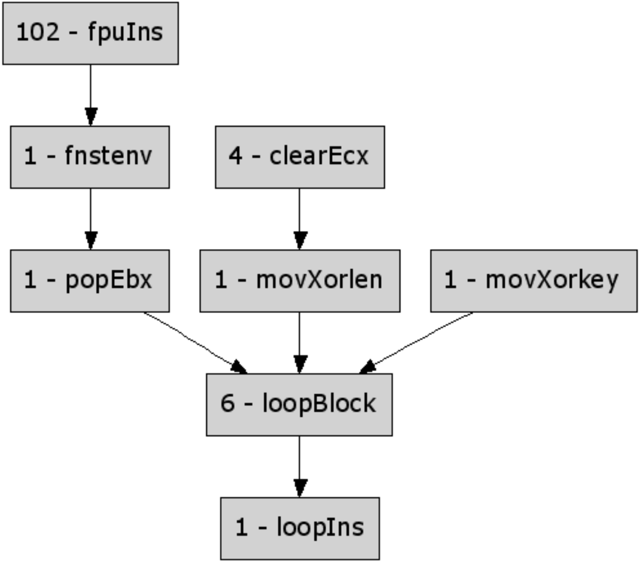
Shikata dependency iteration



Shikata dependency iteration



Shikata dependency iteration



Example output

```
00000000  BB6E887A69        mov ebx,0x697a886e
00000005  DDC4              ffree st4
00000007  D97424F4          fnstenv [esp-0xc]
0000000B  58                pop eax
0000000C  29C9              sub ecx,ecx
0000000E  B101              mov cl,0x1
00000010  83E8FC            sub eax,byte -0x4
00000013  31580E            xor [eax+0xe],ebx
00000016  03580E            add ebx,[eax+0xe]
00000019  E2F5              loop 0x10
```

Example output

```
00000000  DBC1                  fcmovnb st1
00000002  31C9                  xor ecx,ecx
00000004  B101                  mov cl,0x1
00000006  D97424F4              fnstenv [esp-0xc]
0000000A  5B                    pop ebx
0000000B  BAC8E2C8F8            mov edx,0xf8c8e2c8
00000010  83C304                add ebx,byte +0x4
00000013  315313                xor [ebx+0x13],edx
00000016  035313                add edx,[ebx+0x13]
00000019  E2F5                  loop 0x10
```

Example output

```
00000000  BB7B833BB9        mov ebx,0xb93b837b
00000005  DAC0              fcmovb st0
00000007  D97424F4          fnstenv [esp-0xc]
0000000B  2BC9              sub ecx,ecx
0000000D  5E                pop esi
0000000E  B101              mov cl,0x1
00000010  315E12            xor [esi+0x12],ebx
00000013  83C604            add esi,byte +0x4
00000016  03                db 0x03
00000017  25                db 0x25
00000018  8D                db 0x8D
00000019  D9                db 0xD9
0000001A  4C                dec esp
```

Part V

Egg Hunters

Egg Hunting, what and why?

- Sometimes we have very small size constraints
- But we can often put data somewhere else

Egg Hunting, what and why?

- Sometimes we have very small size constraints
- But we can often put data somewhere else

- We somehow put our code in memory (tagged)
- We execute the egg hunter as our shellcode
- An egg hunter searches for and executes more code

Egg Hunting, what and why?

- Sometimes we have very small size constraints
- But we can often put data somewhere else

- We somehow put our code in memory (tagged)
- We execute the egg hunter as our shellcode
- An egg hunter searches for and executes more code

- We need to validate a memory region before we search it

Improved shellcode structure

[nop sled][egg hunter] - [encoder (payload)]

- The nop sled slides into the decoder
- The decoder decodes the payload
- The payload does stuff

Syscall Based Egghunting

- Linux
  - Linux is pretty easy
  - Use access() call to validate memory, search

Syscall Based Egghunting

- Linux
  - Linux is pretty easy
  - Use access() call to validate memory, search
- Windows
  - This is a harder
  - Try to use the Native API (syscalls)
  - Windows system calls are not meant to be static
  - Parse and build tables of all windows syscalls
  - Find a static system call, or collection that will work

Syscall Based Egghunting

- Linux
  - Linux is pretty easy
  - Use access() call to validate memory, search
- Windows
  - This is a harder
  - Try to use the Native API (syscalls)
  - Windows system calls are not meant to be static
  - Parse and build tables of all windows syscalls
  - Find a static system call, or collection that will work
  - NtAccessCheckAndAuditAlarm works
  - Need special privileges for to work normally
  - It validates the pointer before it validates rights

Windows egghunt example

```
// Skape’s syscall egghunter

                  // Address to check in edx
push  0x2         // Push NtAccessCheckAndAuditAlarm
pop   eax         // Pop into eax
int   0x2e        // Perform the syscall
cmp   al, 0x05    // Did we get 0xc0000005 (ACCESS_VIOLATION) ?
```

Part VI

Staging

Staging, what and why?

- We often have size constraints
- Staging abstracts the connection mechanism from the payload

Staging, what and why?

- We often have size constraints
- Staging abstracts the connection mechanism from the payload
- A stager establishes a connection to the attacker
- The stager reads in more code from the connection
- The stager executes the stage passing the connection

Improved shellcode structure

[nop sled][encoder (stager)] - [stage]

- The nop sled slides into the decoder
- The decoder decodes the payload
- The payload does stuff

Size issues

- Linux
  - Pretty easy, just use syscalls, etc

Size issues

- Linux
  - Pretty easy, just use syscalls, etc
- Windows
  - As you saw, it would be hard to use syscalls
  - We need to use the Windows APIs (ws2_32.dll...)
  - But function resolving takes a ton of code!

Size issues

- Linux
  - Pretty easy, just use syscalls, etc
- Windows
  - As you saw, it would be hard to use syscalls
  - We need to use the Windows APIs (ws2_32.dll...)
  - But function resolving takes a ton of code!
  - What in the world can we do!?

ws2_32.dll static ordinals

- ws2_32.dll is one of the few libraries with static ordinals

ws2_32.dll static ordinals

- ws2_32.dll is one of the few libraries with static ordinals
- However, not all functions have static ordinals
- Cannot call WSASocket() for example, must use socket()
- This means we need a pipe based shell stage :(

ws2_32.dll static ordinals

- ws2_32.dll is one of the few libraries with static ordinals
- However, not all functions have static ordinals
- Cannot call WSASocket() for example, must use socket()
- This means we need a pipe based shell stage :(

- Find ws2_32.dll base
- Resolve our functions by static ordinals
- 93 byte reverse connect shellcodez y0!

Part VII

Questions?