

steg86: hiding messages in x86 binaries

rust munich meetup

william woodruff

august 25 2020

agenda

- ▶ yours truly
- ▶ steganography?
- ▶ steg on programs
- ▶ x86 instruction encoding
- ▶ steg86

yours truly

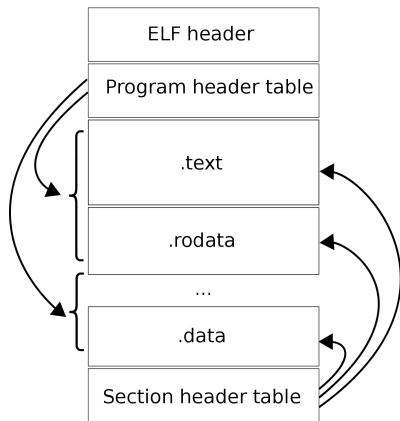
- ▶ william woodruff
 - ▶ [@8x5cIPW2](#) ▪ [yossarian.net](#) ▪ [blog.yossarian.net](#)
- ▶ senior security engineer @ [trail of bits](#)
- ▶ work: program analysis research, mostly in LLVM
 - ▶ disclaimer: independent talk, not representing employer
- ▶ open source: member of [homebrew](#), miscellaneous contributor

steganography?



- ▶ “hiding data within data”
- ▶ *not* cryptography
- ▶ different techniques for different data
- ▶ popular targets:
 - ▶ images
 - ▶ sound files
 - ▶ plain text
- ▶ what about programs?

steg on programs



programs are a natural choice for steg

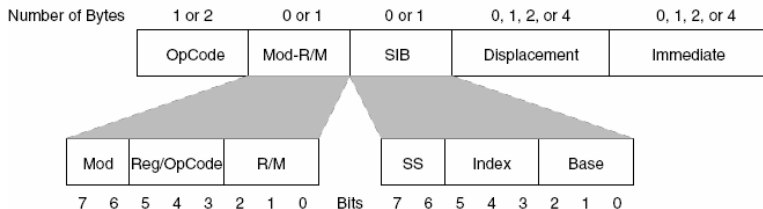
- ▶ can be very large (lots of info capacity)
- ▶ complex binary formats (PE, Mach-O, ELF)
- ▶ complex instruction encodings (x86/AMD64, ARM w/ Thumb)
- ▶ present on every computer, not inherently suspicious

steg on programs: approaches

- ▶ hide information in stack layout, register selection
 - ▶ problem: need the program's source
 - ▶ problem: need to maintain a compiler...
- ▶ hide information in the format itself (e.g. segment order)
 - ▶ problem: specific to a format, may not apply to others
- ▶ rewrite the program after compilation
 - ▶ ex: `add eax, -50` → `sub eax, 50`
 - ▶ problem: code/data disambiguation (difficult to solve)
 - ▶ problem: relocations, position independent code (`-fPIC`)
 - ▶ problem: CPU-level semantics (arithmetic, status flags)
- ▶ can we do better?

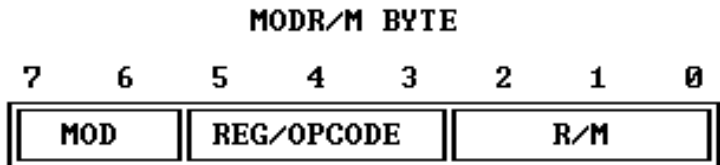
x86 instruction encoding

- ▶ variable length (up to 15 bytes)
- ▶ extremely complex (decades of compat, overloaded fields)
- ▶ rich source/sink combinations
 - ▶ register-to-register (`mov ebx, eax`)
 - ▶ register-to-memory (`mov dword [1337], eax`)
 - ▶ memory-to-register (`mov eax, dword [1337]`)
 - ▶ immediate-to-register (`mov eax, 1337`)
 - ▶ immediate-to-memory (`mov dword [1337], 1337`)



x86 instruction encoding: modr/m

- ▶ essentially an 8-bit lookup table of (some) operand encodings
 - ▶ doesn't cover all possible operands, for historical reasons. . .
- ▶ simplest case: encodes one or two operands
 - ▶ reg/opcode field: one register operand
 - ▶ r/m field: one register **or** memory operand
 - ▶ enables mem-to-reg, reg-to-mem, reg-to-reg operations



x86 instruction encoding: xor

opcode	instruction
31 /r	xor r/m32, r32
33 /r	xor r32, r/m32

- ▶ reg-to-mem, mem-to-reg, reg-to-reg, 🤔 ...
- ▶ there are **two** reg-to-reg encodings!
- ▶ 31 C0 → mov eax, eax
- ▶ 33 C0 → **also** mov eax, eax!
- ▶ they're even the same size!
- ▶ 64-bit variants (w/ REX prefix) work too!

steg86

- ▶ central conceit: each reg-to-reg pair represents one bit of information
 - ▶ with enough bits, we can hide messages!
- ▶ binary format independent
 - ▶ uses [goblin](#) to unpack PE/ELF/Mach-O binaries
- ▶ encodings are the same size, so PIC/relocations aren't broken
 - ▶ uses [iced](#) for decoding/encoding/semantics
- ▶ ~700 lines of rust total (much of it constants)
- ▶ CLI: `steg86 {profile,embed,extract}`

steg86: semantic duals

- ▶ it turns out there are a bunch of these
- ▶ 9 instructions (add, adc, sub, sbb, and, or, xor, mov, cmp)
 - ▶ 4 variants (8, 16, 32, 64-bit) each¹
- ▶ each dual gives us 1 bit of information
 - ▶ minus a little space for a header with metadata
- ▶ how common are these instructions?

```
$ steg86 profile /bin/bash
```

```
Summary for /bin/bash:
```

```
175828 total instructions
```

```
27957 potential semantic pairs
```

```
27925 bits of information capacity (3490 bytes)
```

- ▶ not bad!

¹actually 3 in any particular CPU mode...

steg86: semantic duals

each pair represents (false, true)...

```
static SEMANTIC_PAIRS: &[(Code, Code)] = &[  
    // ADD  
    (Code::Add_rm8_r8, Code::Add_r8_rm8),  
    (Code::Add_rm16_r16, Code::Add_r16_rm16),  
    (Code::Add_rm32_r32, Code::Add_r32_rm32),  
    (Code::Add_rm64_r64, Code::Add_r64_rm64),  
  
    // ... snip ...  
];
```

steg86: profiling

for every instruction in the program...

```
// skip instructions we don't support
if !SUPPORTED_OPCODES.contains(&instruction.code()) {
    continue;
}

// skip non reg-to-reg instructions
if instruction.op0_kind() != OpKind::Register
    || instruction.op1_kind() != OpKind::Register
{
    continue;
}

offsets.push(instruction.ip() as usize);
```

steg86: embedding

for each candidate instruction...

```
let new_code = {
  let tuple = SEMANTIC_PAIRS
    .iter()
    .find(|&&t| old_code == t.0 || old_code == t.1)
    .unwrap();

  match (bit, tuple.0 == old_code) {
    (false, true) | (true, false) => {
      // already correct!
      continue;
    }
    (false, false) => tuple.0,
    (true, true) => tuple.1,
  }
};
```

steg86: embedding

```
let new_instruction = Instruction::with_reg_reg(
    new_code,
    instruction.op0_register(),
    instruction.op1_register(),
);
let new_len = encoder
    .encode(&new_instruction, offset as u64)
    .map_err(|s| anyhow!(s))?;

// ... snip ...

text_copy
    .data
    .splice(
        offset..(offset + new_len), encoder.take_buffer());
```

steg86: results

binary diff:

```
/bin/bash
0002 E880: F3 0F 1E FA F2 FF 25 4D A3 0E 00 0F 1F 44 00 00 .....XM...D
0002 E890: F3 0F 1E FA F2 FF 25 4D A3 0E 00 0F 1F 44 00 00 .....XE...D
0002 E8A0: 48 89 84 24 C0 00 00 00 48 8B 04 25 08 00 00 00 H.$...H.X
0002 E8B0: 0F 08 8B 04 25 28 00 00 00 0F 08 0F B6 04 25 00 .....X(C...X
0002 E8C0: 00 00 00 0F 08 0F B6 04 25 00 00 00 0F 08 90 .....X
0002 E8D0: F3 0F 1E FA 41 57 41 56 41 55 41 54 55 53 48 81 .....AWAY ANATUSH
0002 E8E0: EC 38 01 00 00 89 7C 24 04 48 8D 3D 70 C0 0F 00 8...tH-p
0002 E8F0: 48 89 74 24 08 31 F6 48 89 54 24 18 64 48 8B 04 H.tH.H.TH.dH
0002 EC00: 25 28 00 00 00 48 89 84 24 28 01 00 00 8B C0 E8 X.t.H.$(C...t
0002 EC10: 9C FE FF FF F3 0F 1E FA 85 C0 75 60 E8 3F 37 01 .....u'..T
0002 EC20: 00 E8 7A 03 01 00 83 3D 93 2A 0F 00 00 74 57 BF .....z.....*..tV
0002 EC30: 03 00 00 00 E8 87 FE FF FF EB E8 83 7C 24 28 00 .....l(C...
0002 EC40: 45 8B 8D 4C 8B 6C 24 20 0F 84 00 05 00 00 BA 05 E.L.lH
0002 EC50: 00 00 00 48 8D 35 86 F4 0A 00 8B FF EB CF F4 FF .....H.5...t
0002 EC60: FF 4C 89 E8 48 8B C7 81 C0 E8 D2 5F 02 00 48 88 L.H.H...H
0002 EC70: 3D 08 2A 0F 00 8B F6 E8 04 19 00 00 BF 02 00 00 .....t...
0002 EC80: 00 E8 3A FD FF FF E8 E5 AC 05 00 E8 20 F5 FF FF .....tH-
0002 EC90: 38 05 AA A3 0E 00 89 C3 74 54 48 8B 3D AF A3 0E H.t...H-
0002 ECA0: 00 48 85 FF 74 05 E8 A5 F1 FF FF 48 8B 3D A6 A3 H.t...H-
0002 ECB0: 0E 00 48 85 FF 74 05 E8 94 F1 FF FF 48 8B 3D 9D H.t...H-
0002 ECC0: A3 0E 00 48 85 FF 74 05 E8 83 F1 FF FF 48 C7 05 H.t...H-
0002 ECD0: 88 A3 0E 00 00 00 00 48 C7 05 75 A3 0E 00 00 H.u...
0002 ECE0: 00 00 00 48 C7 05 62 A3 0E 00 00 00 00 89 1D H.b.....
```

```
test.steg
0002 E880: F3 0F 1E FA F2 FF 25 4D A3 0E 00 0F 1F 44 00 00 .....XM...D
0002 E890: F3 0F 1E FA F2 FF 25 4D A3 0E 00 0F 1F 44 00 00 .....XE...D
0002 E8A0: 48 89 84 24 C0 00 00 00 48 8B 04 25 08 00 00 00 H.$...H.X
0002 E8B0: 0F 08 8B 04 25 28 00 00 00 0F 08 0F B6 04 25 00 .....X(C...X
0002 E8C0: 00 00 00 0F 08 0F B6 04 25 00 00 00 0F 08 90 .....X
0002 E8D0: F3 0F 1E FA 41 57 41 56 41 55 41 54 55 53 48 81 .....AWAY ANATUSH
0002 E8E0: EC 38 01 00 00 89 7C 24 04 48 8D 3D 70 C0 0F 00 8...tH-p
0002 E8F0: 48 89 74 24 08 31 F6 48 89 54 24 18 64 48 8B 04 H.tH.H.TH.dH
0002 EC00: 25 28 00 00 00 48 89 84 24 28 01 00 00 8B C0 E8 X.t.H.$(C...t
0002 EC10: 9C FE FF FF F3 0F 1E FA 85 C0 75 60 E8 3F 37 01 .....u'..T
0002 EC20: 00 E8 7A 03 01 00 83 3D 93 2A 0F 00 00 74 57 BF .....z.....*..tV
0002 EC30: 03 00 00 00 E8 87 FE FF FF EB E8 83 7C 24 28 00 .....l(C...
0002 EC40: 45 8B 8D 4C 8B 6C 24 20 0F 84 00 05 00 00 BA 05 E.L.lH
0002 EC50: 00 00 00 48 8D 35 86 F4 0A 00 8B FF EB CF F4 FF .....H.5...t
0002 EC60: FF 4C 89 E8 48 8B C7 81 C0 E8 D2 5F 02 00 48 88 L.H.H...H
0002 EC70: 3D 08 2A 0F 00 8B F6 E8 04 19 00 00 BF 02 00 00 .....t...
0002 EC80: 00 E8 3A FD FF FF E8 E5 AC 05 00 E8 20 F5 FF FF .....tH-
0002 EC90: 38 05 AA A3 0E 00 89 C3 74 54 48 8B 3D AF A3 0E H.t...H-
0002 ECA0: 00 48 85 FF 74 05 E8 A5 F1 FF FF 48 8B 3D A6 A3 H.t...H-
0002 ECB0: 0E 00 48 85 FF 74 05 E8 94 F1 FF FF 48 8B 3D 9D H.t...H-
0002 ECC0: A3 0E 00 48 85 FF 74 05 E8 83 F1 FF FF 48 C7 05 H.t...H-
0002 ECD0: 88 A3 0E 00 00 00 00 48 C7 05 75 A3 0E 00 00 H.u...
0002 ECE0: 00 00 00 48 C7 05 62 A3 0E 00 00 00 00 89 1D H.b.....
```

```
$ cargo install steg86
```

```
$ echo "hello!" > message.txt
```

```
$ steg86 embed \  
/bin/bash test.steg \  
< message.txt
```

```
$ steg86 extract test.steg  
hello!
```


steg86: next steps

- ▶ other tricks
 - ▶ `test reg1, reg2` is the same as `test reg2, reg1`
 - ▶ same with `xchg`
 - ▶ multi-byte nops
- ▶ deficiencies
 - ▶ code/data disambiguation is impossible in the general case
 - ▶ many open problems in program analysis reduce to this
 - ▶ partial workarounds: CFG recovery, jump table identification
 - ▶ very easy to detect (real compilers stick to one encoding)

thank you!

slides: yossarian.net/publications#munich-rust-2020

github: [woodruffw/steg86](https://github.com/woodruffw/steg86)

blog post: [hiding messages in x86 binaries using semantic duals](#)

contact: william@yossarian.net / [@8x5clPW2](#)

links and prior work

- ▶ [A86 assembler](#) (1980s!)
- ▶ [HYDAN](#) (2004)
- ▶ [ARMaHYDAN](#) (2019, [PoC||GTFO](#))