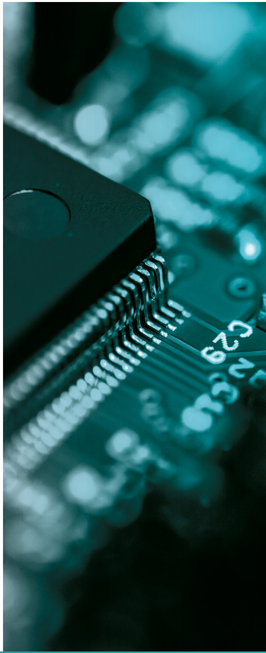
emproof
embedded security

Reverse Engineering: How Attackers Uncover Secrets in Binaries

Tim Blazytko



About Tim

- Chief Scientist, Head of Engineering & Co-Founder of Emproof
- focused on advancing embedded security solutions
- PhD in binary program analysis & reverse engineering
- training and lectures at industry conferences & universities

Setting the Scene

- Reverse Engineering
- Extracting Secrets
- Demonstration

Attacker Motivation

- sabotage
- competitor analysis & espionage
- piracy and feature unlocking
- financial gain

Today

- finding **hardcoded secrets** in binaries
  - cryptographic keys
  - serial numbers
  - passwords
- cracking software to **unlock premium features**

Machine Code and Assembly Code

```
0a 01 0a 00 0b 02 de ad
```

Machine Code and Assembly Code

opcode	register	constant

```
0a 01 0a 00 0b 02 de ad
```

Machine Code and Assembly Code

opcode	register	constant

```
0a 01 0a 00 0b 02 de ad
```

```
add
mul
```

Machine Code and Assembly Code

opcode	register	constant

```
0a 01 0a 00 0b 02 de ad
```

```
add R1
mul R2
```

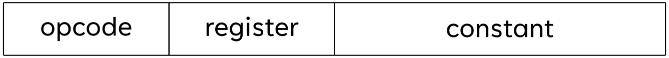
Machine Code and Assembly Code

opcode	register	constant

```
0a 01 0a 00 0b 02 de ad
```

```
add R1, 0x0a00
mul R2, 0xdead
```

Machine Code and Assembly Code



```
0a 01 0a 00 0b 02 de ad
```

```
add R1, 0x0a00
mul R2, 0xdead
```

The decoded machine code is called assembly code.

Disassembler: Decodes Machine Code

```
55 48 89 e5 89
7d fc 89 75 f8
8b 55 fc 8b 45
f8 01 d0 c1 e0
02 5d c3 00 00
```

Disassembler: Decodes Machine Code

```
55 48 89 e5 89
7d fc 89 75 f8
8b 55 fc 8b 45
f8 01 d0 c1 e0
02 5d c3 00 00
```

```
push    rbp
mov     rbp, rsp
mov     [rbp+var_4], edi
mov     [rbp+var_8], esi
mov     edx, [rbp+var_4]
mov     eax, [rbp+var_8]
add     eax, edx
shl     eax, 2
pop     rbp
retn
```

Disassembler: Decodes Machine Code

```
55 48 89 e5 89
7d fc 89 75 f8
8b 55 fc 8b 45
f8
02
```

```
push     rbp
mov      rbp, rsp
mov      [rbp+var_4], edi
mov      [rbp+var_8], esi
mov      edx, [rbp+var_4]
mov      eax, [rbp+var_8]

pop      rbp
retn
```

critical step in reverse engineering

Decompiler: Reconstructs High-Level Code

```
push    rbp
mov     rbp, rsp
mov     [rbp+var_4], edi
mov     [rbp+var_8], esi
mov     edx, [rbp+var_4]
mov     eax, [rbp+var_8]
add     eax, edx
shl     eax, 2
pop     rbp
retn
```

Decompiler: Reconstructs High-Level Code

```
push    rbp
mov     rbp, rsp
mov     [rbp+var_4], edi
mov     [rbp+var_8], esi
mov     edx, [rbp+var_4]
mov     eax, [rbp+var_8]
add     eax, edx
shl     eax, 2
pop     rbp
retn
```

```
ulong calculate(int param_1,int param_2)
{
    return (ulong)(uint)((param_2 + param_1) * 4);
}
```

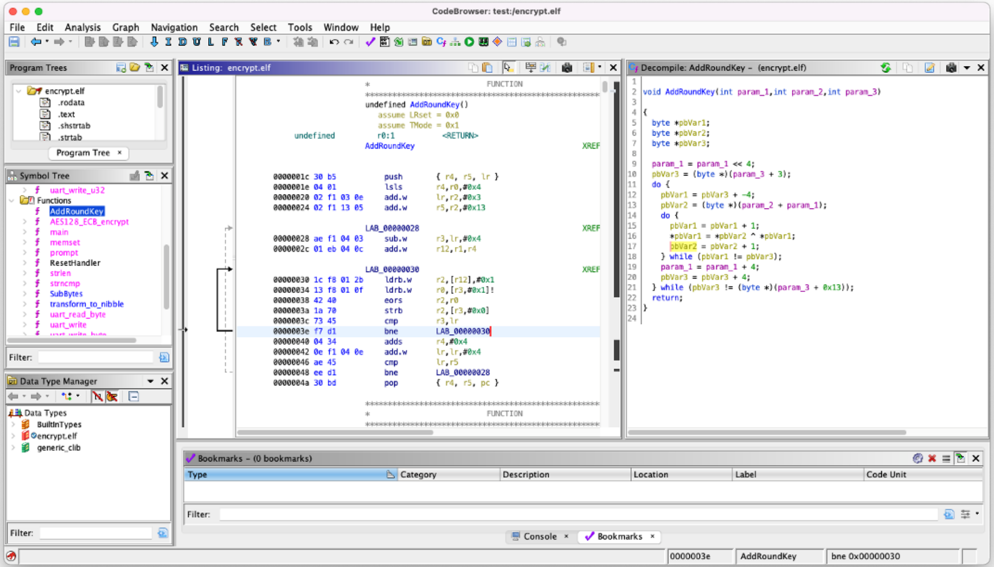
Decompiler: Reconstructs High-Level Code

```
push    rbp
mov     rbp, rsp
mov     [rbp+var_4], edi
mov     [rbp+var_8], esi
mov     edx, [rbp+var_4]
mov     eax, [rbp+var_8]
add     eax, edx
shl
pop
retn
```

```
ulong calculate(int param_1,int param_2)
{
    return (ulong)(uint)((param_2 + param_1) * 4);
}
```

eases reverse engineering significantly

Ghidra: Open Source Reverse Engineering Framework



```
                                   *                          FUNCTION
                                   ************************************************
                                   undefined AddRoundKey()
                                     assume LRset = 0x0
                                     assume TMode = 0x1
             undefined               r0:1           <RETURN>
                                   AddRoundKey                                  XREF


        0000001c 30 b5          push       { r4, r5, lr }
        0000001e 04 01          lsls       r4,r0,#0x4
        00000020 02 f1 03 0e    add.w      lr,r2,#0x3
        00000024 02 f1 13 05    add.w      r5,r2,#0x13

                                   LAB_00000028                                 XREF
        00000028 ae f1 04 03    sub.w      r3,lr,#0x4
        0000002c 01 eb 04 0c    add.w      r12,r1,r4

                                   LAB_00000030                                 XREF
        00000030 1c f8 01 2b    ldrb.w     r2,[r12],#0x1
        00000034 13 f8 01 0f    ldrb.w     r0,[r3,#0x1]!
        00000038 42 40          eors       r2,r0
        0000003a 1a 70          strb       r2,[r3,#0x0]
        0000003c 73 45          cmp        r3,lr
        0000003e f7 d1          bne        LAB_00000030
        00000040 04 34          adds       r4,#0x4
        00000042 0e f1 04 0e    add.w      lr,lr,#0x4
        00000046 ae 45          cmp        lr,r5
        00000048 ee d1          bne        LAB_00000028
        0000004a 30 bd          pop        { r4, r5, pc }

                                   ************************************************
                                   *                          FUNCTION
                                   ************************************************
```

```c

void AddRoundKey(int param_1,int param_2,int param_3)

{
  byte *pbVar1;
  byte *pbVar2;
  byte *pbVar3;

  param_1 = param_1 << 4;
  pbVar3 = (byte *)(param_3 + 3);
  do {
    pbVar1 = pbVar3 + -4;
    pbVar2 = (byte *)(param_2 + param_1);
    do {
      pbVar1 = pbVar1 + 1;
      *pbVar1 = *pbVar2 ^ *pbVar1;
      pbVar2 = pbVar2 + 1;
    } while (pbVar1 != pbVar3);
    param_1 = param_1 + 4;
    pbVar3 = pbVar3 + 4;
  } while (pbVar3 != (byte *)(param_3 + 0x13));
  return;
}

```

DEMO

Conclusion

- reverse engineering to extract secrets from binaries
- powerful tools are freely available

Try it yourself:

`https://github.com/emproof-com/webinars`

Tim Blazytko

@mr_phrazer

https://www.emproof.com/

tblazytko@emproof.com

emproof