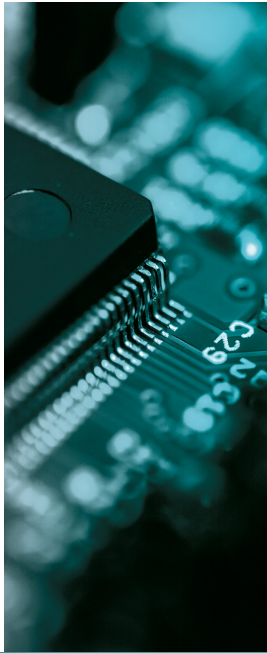
emproof
embedded security

Workshop: Firmware Reverse Engineering

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

Who are you?

Workshop Goals

- understanding fundamentals of reverse engineering
- exploring a wide variety of application scenarios
- learning of state-of-the-art tools and techniques

Workshop Goals

- understanding fundamentals of reverse engineering
- exploring a wide variety of application scenarios
- learning of state-of-the-art tools and techniques

Focus on hands-on sessions.

Outline

- basic file/ELF analysis
- software cracking and keygenning
- string decryption & malware triage
- embedded-Linux firmware unpacking
- bare-metal analysis
- crypto detection
- obfuscation/anti-analysis techniques and how to bypass them

Workshop Format

- 14 self-contained tasks teaching various skills
- solutions are mostly build-in (e.g., source code)
- task-driven with demonstrations, group discussions & trainer support
- AARCH64 Docker execution and analysis environment: `./docker_run.sh`
- native tools: Ghidra, Binary Ninja (Free)

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

Software Reverse Engineering

Software Reverse Engineering: Motivation

We want to *understand* what a given software does.

Software Reverse Engineering: Motivation

We want to *understand* what a given software does.

- program from an *unknown* source

Software Reverse Engineering: Motivation

We want to *understand* what a given software does.

- program from an *unknown* source
- malicious software analysis

Software Reverse Engineering: Motivation

We want to *understand* what a given software does.

- program from an *unknown* source
- malicious software analysis
- vulnerability analysis

Software Reverse Engineering: Motivation

We want to *understand* what a given software does.

- program from an *unknown* source
- malicious software analysis
- vulnerability analysis
- piracy and cracking

Software Reverse Engineering: Motivation

We want to *understand* what a given software does.

- program from an *unknown* source
- malicious software analysis
- vulnerability analysis
- piracy and cracking
- evaluation of software quality

Machine Code and Assembly Code

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

Machine Code and Assembly Code


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

Machine Code and Assembly Code


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

```
add
mul
```

Machine Code and Assembly Code


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

```
add R1
mul R2
```

Machine Code and Assembly Code


```
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
```

critical step in reverse engineering

```
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
```

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);
}
```

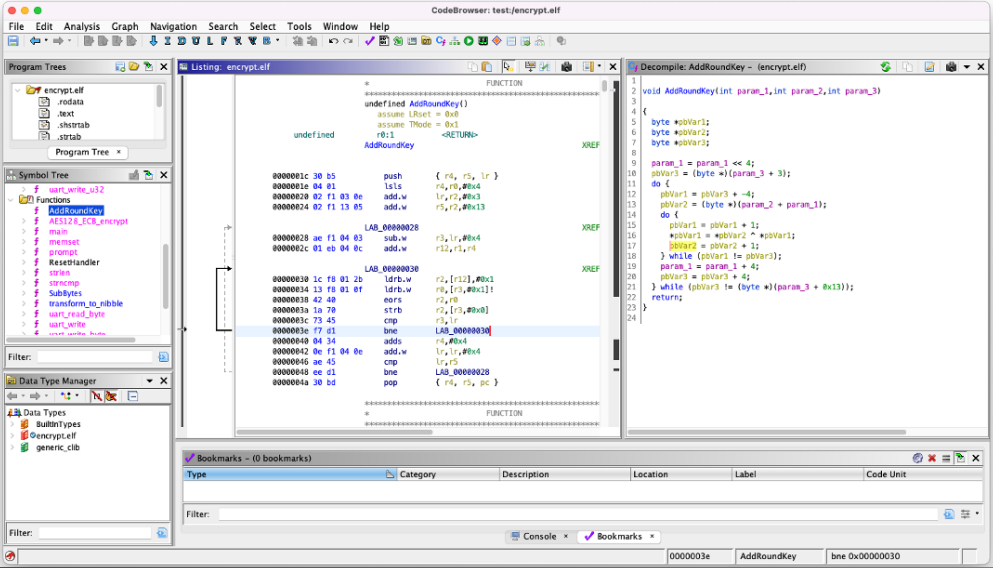
Decompiler: Reconstructs High-Level Code

```
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
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



Conclusion

https://github.com/emproof-com/workshop_firmware_reverse_engineering

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

Tim Blazytko

@mr_phrazer

https://www.emproof.com/

tblazytko@emproof.com