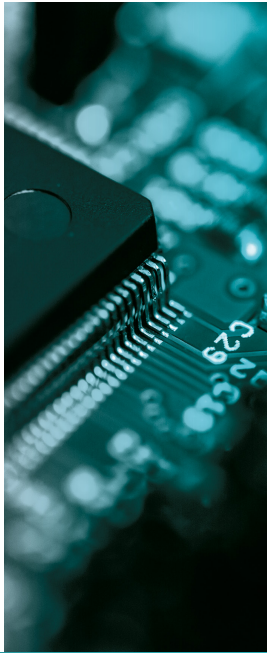
emproof
embedded security

Software Protection – Safeguarding Code Against 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

Setting the Scene

Reverse Engineering Threats

Software Protection

Demonstration

Previous Webinars

- “Reverse Engineering: How Attackers Uncover Secrets in Binaries”
- “Secrets Unveiled: What Attackers Find in Embedded Systems”

Check out slides and recordings: `https://github.com/emproof-com/webinars`

Recap: Reverse Engineering

Attacker Motivation

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

What do find in binaries

- finding **hardcoded secrets** in binaries
  - cryptographic keys
  - serial numbers
  - passwords
- algorithms and other sensitive IP

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);
}
```

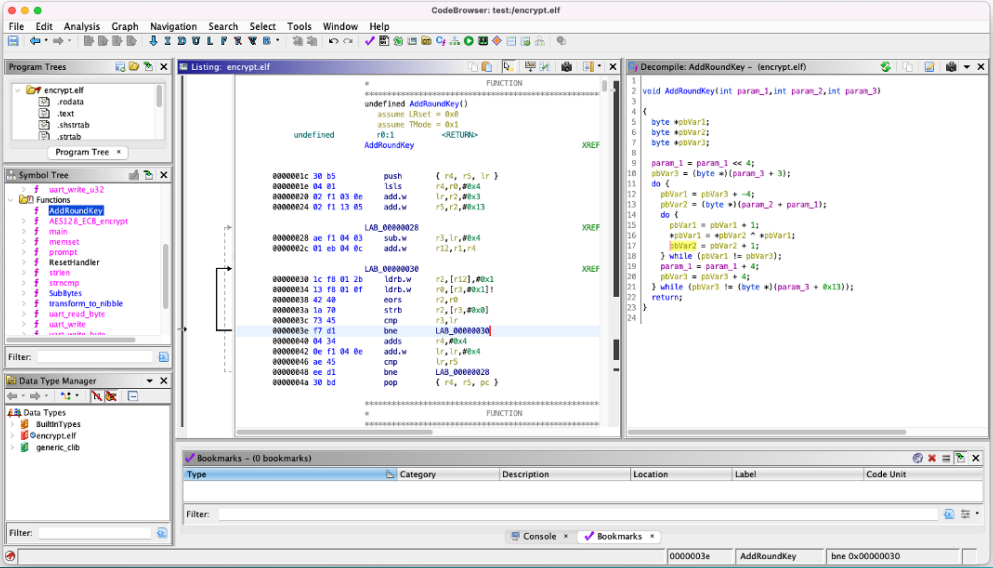
Decompiler: Reconstructs High-Level Code

```
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



How to protect?

Software Protection

Goal: Complicate reverse engineering attempts

Software Protection

Goal: Complicate reverse engineering attempts

- passive protections
  - code obfuscation
  - data encoding

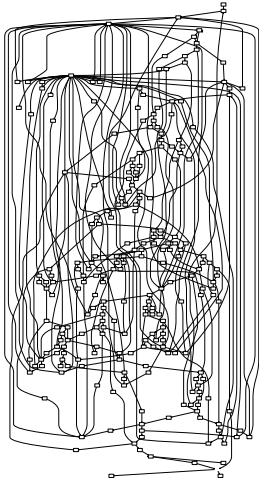
Software Protection

Goal: Complicate reverse engineering attempts

- passive protections
  - code obfuscation
  - data encoding
- active protections
  - anti-debug
  - anti-tamper
  - anti-emulation

Code Obfuscation & Data Encoding

- increase **code complexity** to impede **reverse engineering** (code obfuscation)
- **hide keys and credentials** and decode them at **runtime**

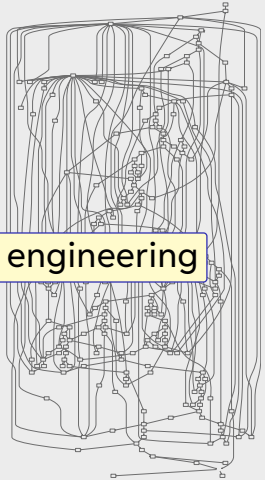


Code Obfuscation & Data Encoding

- increase **code complexity** to impede **reverse engineering** (code obfuscation)

passive protections impede reverse engineering

- **hide keys and credentials** and decode them at **runtime**



DEMO

Anti-Debug & Anti-Tamper

- observe execution environment for debuggers (anti-debug)

```
if debugger_detected() {
    terminate()
}
```

- detect code modifications (patching) by code checksumming (anti-tamper)

```
if checksum(code) != 0xd75648 {
    terminate()
}
```

Anti-Debug & Anti-Tamper

- observe execution environment for debuggers (anti-debug)

```
if debugger_detected() {
    terminate()
}
```

runtime protections to prevent analysis & modifications

- detect code modifications (patching) by code checksumming (anti-tamper)

```
if checksum(code) != 0xd75648 {
    terminate()
}
```

Emproof Nyx: Online Demo

https://demo.emproof.com/ip_protection

Conclusion

- common reverse engineering threats
- passive & active methods to prevent reverse engineering
- online demo: `https://demo.emproof.com/ip_protection`

Try it yourself:

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

Tim Blazytko
@mr_phrazer
https://www.emproof.com/
tblazytko@emproof.com

emproof