

Advanced Cybersecurity Topics

–

Reversing

19-20

Reverse Engineering

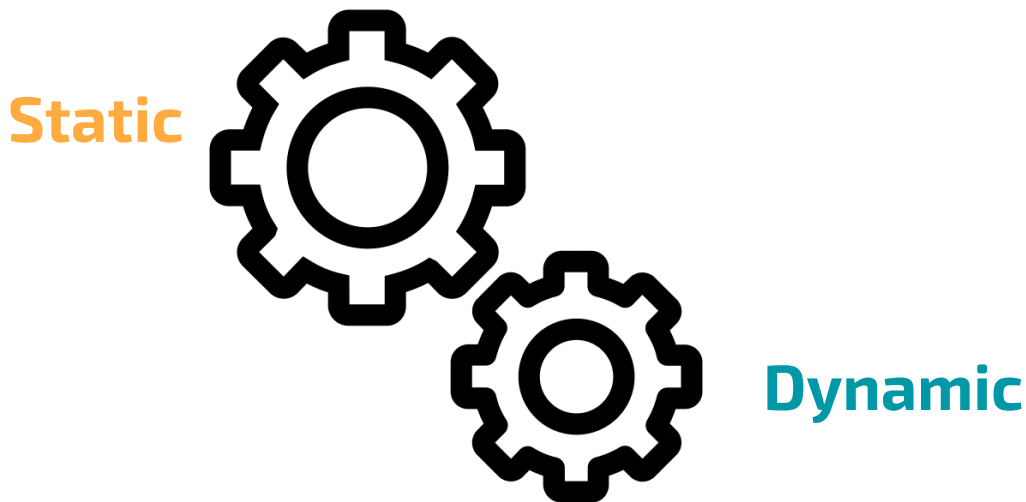
- “is the process by which a **man-made object** is **deconstructed** to reveal its designs, architecture, or to **extract knowledge** from the object **similar to scientific research.**”

Reverse Engineering - Software

- Probe the software to gain knowledge
 - **Statically**
 - Binary is not running
 - **Dynamically**
 - Running the binary

Static and Dynamic Analysis

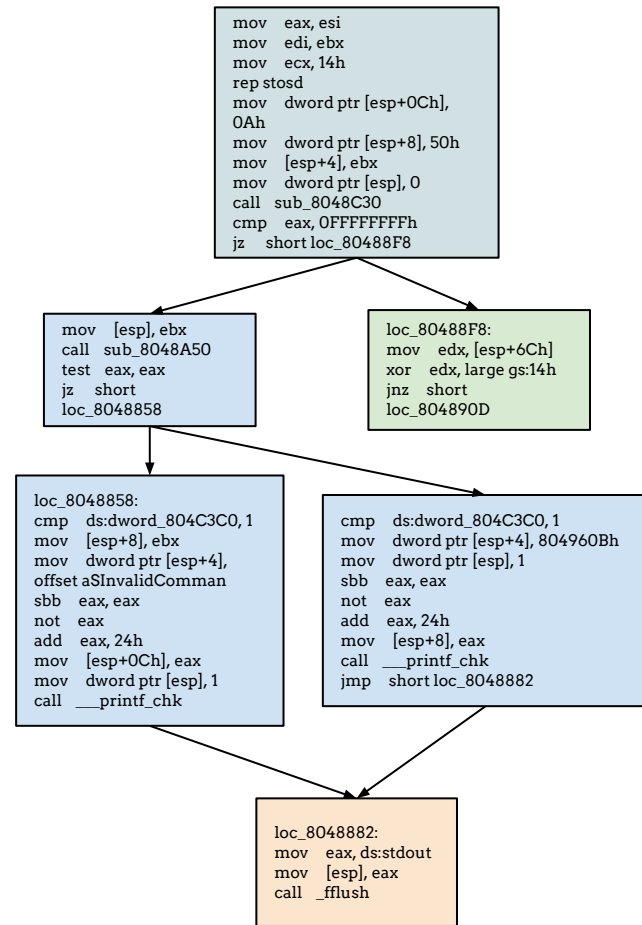
The reverse engineering process is a **sequence** of **static** and **dynamic** analysis that slowly **refine** your knowledge about the malware sample.



Static Analysis

Understand the functionalities of a binary looking at its code.

- Disassemble instructions
- Recover CFG
- Recover Functions
- Recover Types
- Decompile
- Fingerprints



Reverse Engineering - Static

Developer

```
#include <stdio.h>
#include <stdlib.h>

int foo(int a, int b) {
    int c = 14;
    c = (a + b) * c;
    return c;
}

int main(int argc, char * argv[]) {
    int avar;
    int bvar;
    int cvar;
    char * str;
    avar = atoi(argv[1]);
    bvar = atoi(argv[2]);
    cvar = foo(avar, bvar);
    gets(str);
    puts(str);
    printf("foo(%d, %d) = %d\n", avar, bvar, cvar);
    return 0;
}
```

Compiler

```
.cfi_startproc
pushl   %ebp
.cfi_def_cfa_offset 8
.cfi_offset 5, -8
movl    %esp, %ebp
.cfi_def_cfa_register 5
andl    $-16, %esp
subl    $32, %esp
movl    12(%ebp), %eax
addl    $4, %eax
movl    (%eax), %eax
```

Assembler

```
00000000: 01111111 01000101 01001100 01000110 00000001 00000001
00000006: 00000001 00000000 00000000 00000000 00000000 00000000
0000000c: 00000000 00000000 00000000 00000000 00000010 00000000
00000012: 00000011 00000000 00000001 00000000 00000000 00000000
: 11000000 10000011 00000100 00001000 00101100 00000000
00000018: 00000000 00000000 10110100 00001100 00000000 00000000
00000024: 00000000 00000000 00000000 00000000 00110100 00000000
0000002a: 00100000 00000000 00001000 00000000 00101000 00000000
```

Machine

≠

≠

```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

int32_t foo(int32_t a, int32_t b);

// From module: layout.c
// Address range: 0x80484ac - 0x80484cd
// Line range: 5 - 10
int32_t foo(int32_t a, int32_t b) {
    int32_t c = 14 * (b + a); // 0x80484c4
    return c;
}

// From module: layout.c
// Address range: 0x80484cf - 0x8048559
// Line range: 13 - 30
int main(int argc, char **argv) {
    int32_t apple = (int32_t)argv; // 0x80484d8
    int32_t str_as_i = atoi((int8_t *) (int32_t *) (apple + 4));
    int32_t str_as_i2 = atoi((int8_t *) (int32_t *) (apple + 8));
    int32_t banana = foo(str_as_i, str_as_i2); // 0x804850f
    gets(NULL);
    puts(NULL);
    printf("foo(%d, %d) = %d\n", str_as_i, str_as_i2, banana);
    return 0;
}
```

Decompiler

```
push    %ebp
mov     %esp,%ebp
and     $0xffffffff0,%esp
sub     $0x20,%esp
mov     0xc(%ebp),%eax
add     $0x4,%eax
mov     (%eax),%eax
mov     %eax,(&esp)
call    80483b0<atoi@plt>
mov     %eax,0xc(%esp)
mov     0xc(%ebp),%eax
```

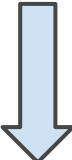
Disassembler

Static Analysis - Disassembler

- Linear Sweep

```
0000000: 01111111 01000101 01001100 01000110 00000001 00000001
0000006: 00000001 00000000 00000000 00000000 00000000 00000000
000000c: 00000000 00000000 00000000 00000000 00000010 00000000
0000012: 00000011 00000000 00000001 00000000 00000000 00000000
0000018: 11000000 10000011 00000100 00001000 00110100 00000000
000001e: 00000000 00000000 10110100 00001100 00000000 00000000
0000024: 00000000 00000000 00000000 00000000 00110100 00000000
000002a: 00100000 00000000 00001000 00000000 00101000 00000000
```

- Recursive



```
push    %ebp
mov     %esp,%ebp
and     $0xffffffff0,%esp
sub     $0x20,%esp
mov     0xc(%ebp),%eax
add     $0x4,%eax
mov     (%eax),%eax
mov     %eax,(%esp)
call    80483b0 <atoi@plt>
mov     %eax,0x1c(%esp)
mov     0xc(%ebp),%eax
```

Static Analysis - Disassembler Tools

objdump - Disasm

radare2 - Disasm

capstone - **Programmable** Disasm

Binary Ninja - Disasm + Primitive Decompiler

GHIDRA - Disasm + Decompiler (<https://ghidra-sre.org/>)

IDA Pro - Disasm + Decompiler (de facto standard)

Static Analysis - Decompile

You can go back to pseudo code:

```
push    %ebp
mov     %esp,%ebp
and     $0xffffffff0,%esp
sub     $0x20,%esp
mov     0xc(%ebp),%eax
add     $0x4,%eax
mov     (%eax),%eax
mov     %eax,(%esp)
call    80483b0 <atoi@plt>
mov     %eax,0x1c(%esp)
mov     0xc(%ebp),%eax
```



```
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

int32_t foo(int32_t a, int32_t b);

// From module: layout.c
// Address range: 0x80484ac - 0x80484cd
// Line range: 5 - 10
int32_t foo(int32_t a, int32_t b) {
    int32_t c = 14 * (b + a); // 0x80484c4
    return c;
}

// From module: layout.c
// Address range: 0x80484cf - 0x8048559
// Line range: 13 - 30
int main(int argc, char **argv) {
    int32_t apple = (int32_t)argv; // 0x80484d8
    int32_t str_as_i = atoi((int8_t *)*(int32_t *) (apple + 4));
    int32_t str_as_i2 = atoi((int8_t *)*(int32_t *) (apple + 8));
    int32_t banana = foo(str_as_i, str_as_i2); // 0x804850f
    gets(NULL);
    puts(NULL);
    printf("foo(%d, %d) = %d\n", str_as_i, str_as_i2, banana);
    return 0;
}
```

Static Analysis - Decompile

Binary Ninja - Disasm + Primitive Decompiler

GHIDRA - Disasm + Decompiler (<https://ghidra-sre.org/>)

IDA Pro - Disasm + Decompiler (de facto standard)

rev.ng - not available yet

Static Analysis - Struct

Building correct types make decompilation more readable.

```
void __fastcall sub_1413(__int64 a1)
{
    signed int i; // [rsp+10h] [rbp-10h]
    signed int v2; // [rsp+14h] [rbp-Ch]
    void *v3; // [rsp+18h] [rbp-8h]

    for ( i = 0; i <= 15; ++i )
    {
        if ( !*( _DWORD *) (24LL * i + a1) )
        {
            printf("Size: ");
            v2 = sub_1AD5();
            if ( v2 > 0 && v2 <= 88 )
            {
                v3 = calloc(v2, 1uLL);
                if ( !v3 )
                    exit(-1);
                *( _DWORD *) (24LL * i + a1) = 1;
                *( _QWORD *) (a1 + 24LL * i + 8) = v2;
                *( _QWORD *) (a1 + 24LL * i + 16) = v3;
                printf("Chunk %d Allocated\n", (unsigned int)i);
            }
            else
            {
                puts("Invalid Size");
            }
        }
        return;
    }
}
```

```
void __fastcall allocate(nota *notes)
{
    signed int i; // [rsp+10h] [rbp-10h]
    signed int sz; // [rsp+14h] [rbp-Ch]
    void *chunk; // [rsp+18h] [rbp-8h]

    for ( i = 0; i <= 15; ++i )
    {
        if ( !notes[i].state )
        {
            printf("Size: ");
            sz = get_long();
            if ( sz > 0 && sz <= 0x58 )
            {
                chunk = calloc(sz, 1uLL);
                if ( !chunk )
                    exit(-1);
                notes[i].state = 1;
                notes[i].size = sz;
                notes[i].data = (__int64)chunk;
                printf("Chunk %d Allocated\n", (unsigned int)i);
            }
            else
            {
                puts("Invalid Size");
            }
        }
        return;
    }
}
```

```
signed int i; // [rsp+10h] [rbp-10h]
signed int v2; // [rsp+14h] [rbp-Ch]
void *v3; // [rsp+18h] [rbp-8h]
```

```
for ( i = 0; i <= 15; ++i )
{
    if ( !*( _DWORD *) (24LL * i + a1) )
```

```
    {
        printf("Size: ");
        v2 = sub_1AD5();
        if ( v2 > 0 && v2 <= 88 )
        {
            v3 = calloc(v2, 1uLL);
            if ( !v3 )
                exit(-1);
```

```
            *( _DWORD *) (24LL * i + a1) = 1;
            *( _QWORD *) (a1 + 24LL * i + 8) = v2;
            *( _QWORD *) (a1 + 24LL * i + 16) = v3;
            printf("Chunk %d Allocated\n", (unsigned
```

```
int)i);
        }
        else
        {
            puts("Invalid Size");
        }
        return;
    }
}
```

```
signed int i; // [rsp+10h] [rbp-10h]
signed int sz; // [rsp+14h] [rbp-Ch]
void *chunk; // [rsp+18h] [rbp-8h]
```

```
for ( i = 0; i <= 15; ++i )
{
    if ( !notes[i].state )
```

```
    {
        printf("Size: ");
        sz = get_long();
        if ( sz > 0 && sz <= 0x58 )
        {
            chunk = calloc(sz, 1uLL);
            if ( !chunk )
                exit(-1);
```

```
            notes[i].state = 1;
            notes[i].size = sz;
            notes[i].data = (__int64)chunk;
            printf("Chunk %d Allocated\n",
```

```
(unsigned int)i);
        }
        else
        {
            puts("Invalid Size");
        }
        return;
    }
}
```

Static Analysis - Struct IDA (local type)

Building correct types make decompilation more readable.

```
struct __attribute__((aligned(8))) nota
{
    int state;
    int unk1;
    __int64 size;
    __int64 data;
};
```

Static Analysis - Other Tools (Lifter)

You can Lift binary to perform program analysis tasks.

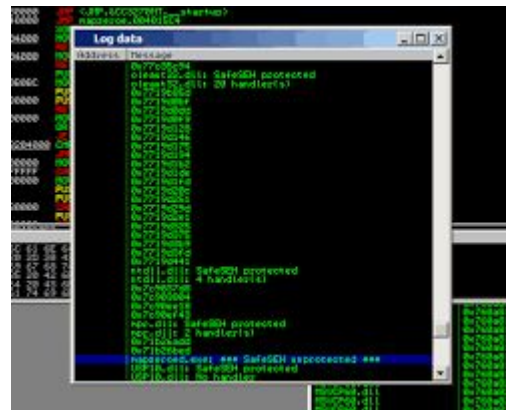
Angr - Binary Analysis (VEX IR) + Symbolic Execution (<http://angr.io/>)

rev.ng - Binary analysis with LLVM IR + Binary translation

BAP - Binary analysis with BIL IR

Dynamic Analysis - Debuggers

- Run the executable and see/modify process state live.
- Step by step Debugging
- BreakPoints
 - **Hardware breakpoints**
- **Watchpoints**
- syscall catch
- signal catch
- **Scriptable**
- Tools: windbg, ollydbg, Immunity dbg, IDA pro, GDB, etc.



Dynamic Analysis - GDB

disassemble

set disassembly-flavor intet	#sets syntax
disass *address	#disassemble

execution

step (s)	#exec nextline - enter fun
next(n)	#exec nectline - jump call
finish (f)	#exec til ret
continue (c)	#continue execution

examine

x/numF *address	#show num data of type F (useful Fs are bx, wx, gx, c, s)
printf "%c", \$reg	#print char from register

Dynamic Analysis - GDB

breakpoints

b *address	#set software breakp at addr
hb *address	#set hardware breakp at addr
b *address if \$reg==val	#set conditional breakp
del br_num	#remove breakpoint br_num

watchpoints

w *address	#set watch for write at addr
rw *address	#set watch for read at addr

Dynamic Analysis - GDB

Setting registers to a certain value:

```
set $reg = val
```

Change return value of a function, to jump into some memory are, etc.

You can also call a function with:

```
call address
```

Dynamic Analysis - GDB Automate

Create command that are runned **after** a break point

```
commands br_num
```

```
    command_list
```

```
end
```

Reversing in Adversarial Contest - Static

- Complex **CFG**
 - not aligned jmp
 - dead code
- **Packing**
- **Header** Corruption

Reversing in Adversarial Contest - Dynamic

- Debugging only Once
 - ptrace
- Check for Debugger
 - 0xcc
- Divert Execution
 - Signals / multithread