InTELL
BY FOX IT

ReAsm

Who am I

Jesus.olmos@fox-it.com

@sha0coder

Concept

Source → Compilation → Binary

Binary → Source → Compilation

Binary → Assembly → Compilation

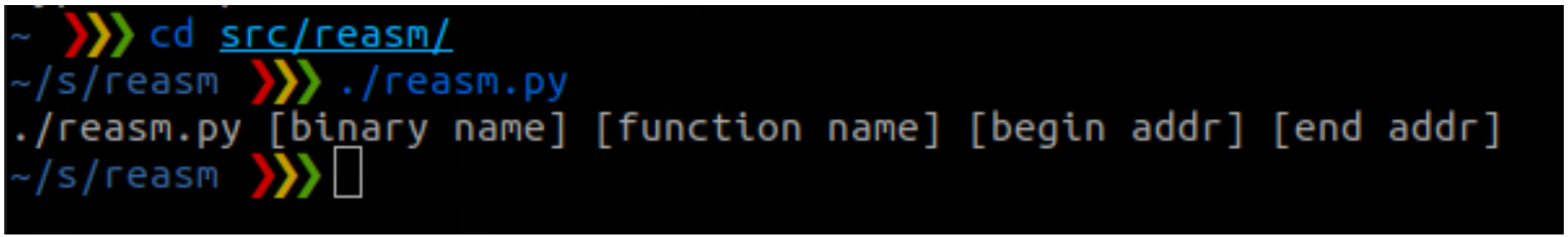
```
movdqa xmm6, xmm3
por xmm1, xmm7
pand xmm6, xmm1
psrld xmm1, 1
pcmpeqd xmm6, xmm5
pxor xmm6, xmm0
movdqa xmm7, xmm6
pandn xmm7, xmm1
pxor xmm1, xmm2
pand xmm1, xmm6
movdqa xmm6, xmm3
por xmm1, xmm7
pand xmm6, xmm1
psrld xmm1, 1
pcmpeqd xmm6, xmm5
pxor xmm6, xmm0
movdqa xmm7, xmm6
pandn xmm7, xmm1
pxor xmm1, xmm2
pand xmm1, xmm6
movdqa xmm6, xmm3
por xmm1, xmm7
pand xmm6, xmm1
psrld xmm1, 1
pcmpeqd xmm6, xmm5
pxor xmm6, xmm0
movdqa xmm7, xmm6
pandn xmm7, xmm1
pxor xmm1, xmm2
pand xmm1, xmm6
movdqa xmm6, xmm3
por xmm1, xmm7
pand xmm6, xmm1
psrld xmm1, 1
pcmpeqd xmm6, xmm5
pxor xmm6, xmm0
movdqa xmm7, xmm6
pandn xmm7, xmm1
pxor xmm1, xmm2
pand xmm1, xmm6
movdqa xmm6, xmm3
```

Dridex hashing function

1. What is this function?

2. How can we use it?

ReAsm Tool



Powered by R2Pipe

Generated Files

function_name.asm

main.c

Makefile

main.c

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

#define ASM __asm__ __volatile__


extern unsigned long function_name();

char *alloc(unsigned long sz) {
  printf("allocating %ld\n", sz);
  return (char *)malloc(sz);
}



int main(void) {
        printf("%x\n", function_name());
}
```

function_name.asm

```
; generated with reasm
; nasm -felf32 -Fdwarf xmm_crc32.asm -o xmm_crc32.o

bits 32
global xmm_crc32
extern alloc

xmm_crc32:
        mov ecx, [esp+4]
        push ebp
        mov ebp, esp
        and esp, 0xfffffff0
        push esi
        push edi
        push ebx
        sub esp, 0x404
        mov esi, edx
        mov eax, 0
        movd xmm4, eax
        ;movdqu xmm4, 0 ;[0x43a2f0]
        pxor xmm5, xmm5
        mov eax, 0x4
        movd xmm1, eax ;movdqu xmm1, 0x4 ; [0x43a2e0]
        mov ebx, ecx
        ;mov ecx, 0x016D1160 ;dword [0x43b228]
        mov eax, 1
        movd xmm3,  eax ;[0x43a300]
        mov eax, 0x20
        movd xmm2, eax ;[0x43a310]
        movdqu  [esp], xmm4
addr_0042d662:
        xor eax, eax
        cmp byte eax, 0
        jne addr_0042daa4
        cmp ecx, 0xe18e2f19
        jne addr_0042da2b
        push 0x996e050f
        push 0xe5ab9b45
```

With stack

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

#define ASM __asm__ __volatile__

extern void decoder();

unsigned long currentStack;
char *stack;


char *Alloc(size_t size) {
    char *currFakeStack;
    asm("mov %%esp, %0" :"=r"(currFakeStack));
    asm("mov %0, %%esp" ::"r"(currentStack));
    printf("alocatando %d bytes\n", size);
    char *buff = (char *)malloc(size);
    asm("mov %0, %%esp" ::"r"(currFakeStack));
}


int main(void) {
    asm("mov %%esp, %0" :"=r"(currentStack));
    stack = (char *)malloc(1024*10);
    asm("mov %0, %%esp" ::"r"(stack));

    // launch the decoder
    ASM("push %ebx");
    decoder();
    ASM("pop %ebx");



    free(stack);
    return 0;
}
```

C++ calling conversion

```
// launch from c the algorithm on asm
#include <stdio.h>
#include <stdlib.h>

#define ASM __asm__ __volatile__

extern char *b64(char *, unsigned long);

char *Alloc(size_t size) {
    printf("allocating %d bytes\n", size);
    return (char *)malloc(size);
}

void Dealloc(char *ptr) {
    free(ptr);
}

int main(void) {
    char *this = Alloc(100);

    __asm__("mov %0, %%ecx" :: "r"(this));

    printf("%s\n", b64("hola", 4) );
    return 0;
}
```

Compensating registers

```c
// launch from c the algorithm on asm
#include "stdio.h"
#include <stdlib.h>

#define ASM __asm__ __volatile__

extern void decoder();

char *Alloc(size_t size) {
    ASM("pop %ebx");
    printf("alloc %d bytes\n", size);
    return (char *)malloc(size);
    ASM("push %ebx");
}

int main(void) {
    ASM("push %ebx");
    decoder();
    ASM("pop %ebx");
    return 0;
}
```

Formbook rc4 + b64

```c
// launch from c the algorithm on asm
#include "stdio.h"
#include <stdlib.h>
#include <string.h>

#define ASM __asm__ __volatile__

extern void formbook_b64();
extern void formbook_rc4();

int main(void) {
    char out[2049];
    char key[] = {0x43,0x6f,0xc5,0xa3,0x0e,0x4d,0x94,0x73,0xe8,0xcc,0x6e,0xb7,0x6c,0x32,0x17,0x0d,0x28,0xad,0xa1,0x8c};
    char plaintext[] = "FBNG:3B3049604.1:Windows 7 Professional x86:QWRtaW4=";
    size_t len = strlen(plaintext);

    memset(out, 0, 2049);
    formbook_rc4(plaintext, len, key);
    formbook_b64(out, plaintext, len);

    printf("%s\n", out);
    exit(1);
}
```

branches

```
branches = ['call', 'jmp', 'je', 'jl', 'jg', 'jge', 'gle', 'ja', 'jb', 'jbe', 'jae', 'jne', 'jo', 'jno', 'js', 'jns', 'jnz', 'jnb', 'jna',
            'jnae', 'jc', 'jnc', 'jnbe', 'jnge', 'jnl', 'jng', 'jle', 'jp', 'jpe','jnp','jpo','jcxz','jecxz']
```

```
        test edi, edi
        jle addr_04f6b715
        jmp addr_04f6b6b0
        lea ecx,  [ecx]
addr_04f6b6b0:
        movzx edx, byte [esi + ecx]
        shr edx, 2
        movzx edx, byte [ebp + edx - 0x40]
        mov byte [eax], dl
        movsx edx, byte [esi + ecx]
        movzx ebx, byte [esi + ecx + 1]
        and edx, 3
        shl edx, 4
        shr ebx, 4
        or edx, ebx
        movzx edx, byte [ebp + edx - 0x40]
        mov byte [eax + 1], dl
        movsx ebx, byte [esi + ecx + 1]
        movzx edx, byte [esi + ecx + 2]
        and ebx, 0xf
        shr edx, 6
        add ebx, ebx
        add ebx, ebx
        or edx, ebx
        movzx edx, byte [ebp + edx - 0x40]
        mov byte [eax + 2], dl
        movsx edx, byte [esi + ecx + 2]
        and edx, 0x3f
        movzx edx, byte [ebp + edx - 0x40]
        mov byte [eax + 3], dl
        add ecx, 3
        add eax, 4
        cmp ecx, edi
        jl addr_04f6b6b0
        mov edx, dword [ebp + 0x10]
addr_04f6b715:
        cmp ecx, edx
        jge addr_04f6b774
        movzx edi, byte [ecx + esi]
```

branches

R2Pipe

```
r2 = r2pipe.open(binaryname)
r2.cmd('s 0x%x' % addr_begin)
r2.cmd('e asm.bytes=0')
r2.cmd('e asm.lines=0')
asm = r2.cmd('pD %d' % (addr_end-addr_begin)).split('\n')
```

Github link

https://github.com/fox-it/reasm

Improvements

- recursive call extraction
- detect the end of the functions

Execution vs Emulation

- anti-emulation bypass
- speed
- replicate exact state of the malware

InTELL
BY FOX IT

That’s it