Binary Reverse Engineering And Analysis

Course 4: Dynamic Analysis

Caragea Radu

March 9, 2021

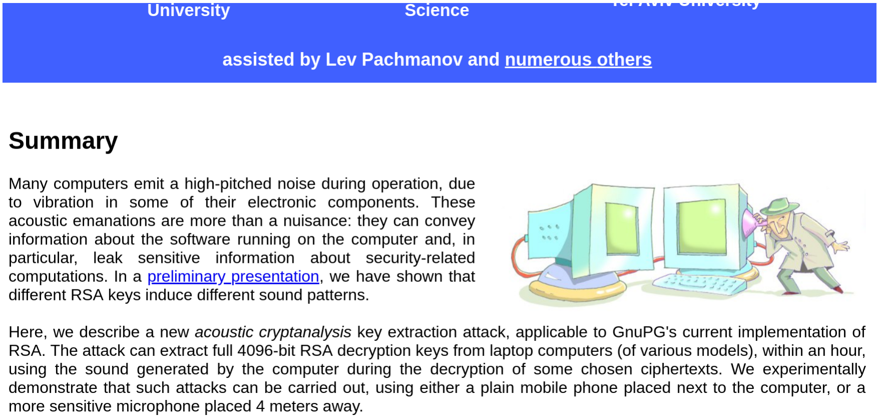
Recap

- Last time: dissecting executables
- Today we study "moving targets"
  - From executable to process
  - Tracing unknown binaries
  - Modifying control flow

But... why?

- Can drastically reduce static analysis time
- Can uncover subtle vulnerabilities inside/outside the code
- Can uncover vulnerabilities unrelated to the actual code (!)

Example 1: side-channels



University

Science

Tel Aviv University

assisted by Lev Pachmanov and numerous others

Summary

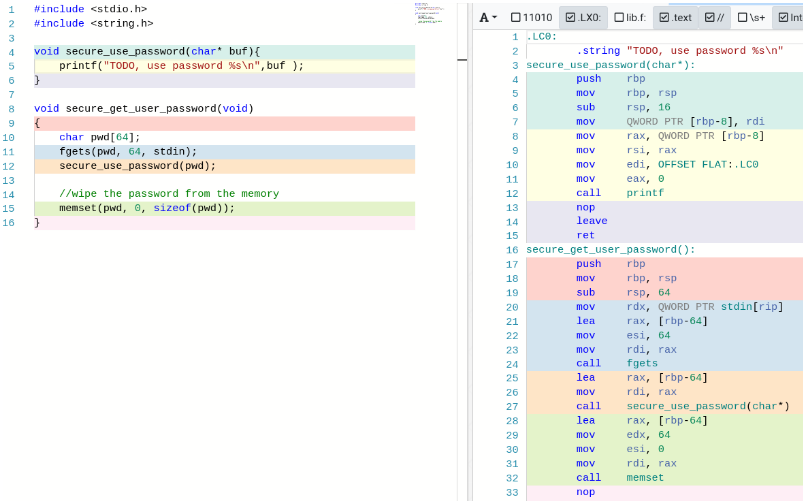
Many computers emit a high-pitched noise during operation, due to vibration in some of their electronic components. These acoustic emanations are more than a nuisance: they can convey information about the software running on the computer and, in particular, leak sensitive information about security-related computations. In a preliminary presentation, we have shown that different RSA keys induce different sound patterns.

Here, we describe a new *acoustic cryptanalysis* key extraction attack, applicable to GnuPG's current implementation of RSA. The attack can extract full 4096-bit RSA decryption keys from laptop computers (of various models), within an hour, using the sound generated by the computer during the decryption of some chosen ciphertexts. We experimentally demonstrate that such attacks can be carried out, using either a plain mobile phone placed next to the computer, or a more sensitive microphone placed 4 meters away.

More info: `https://m.tau.ac.il/~tromer/acoustic`

Example 2: disappearing security measures (before)

https://godbolt.org/z/QMZxYe



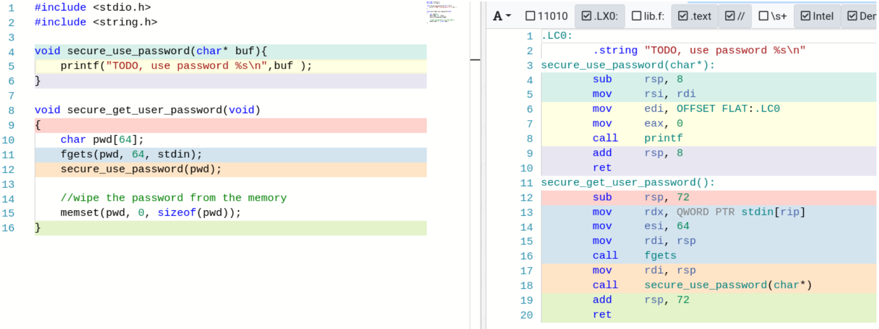
```c
#include <stdio.h>
#include <string.h>

void secure_use_password(char* buf){
    printf("TODO, use password %s\n",buf );
}

void secure_get_user_password(void)
{
    char pwd[64];
    fgets(pwd, 64, stdin);
    secure_use_password(pwd);

    //wipe the password from the memory
    memset(pwd, 0, sizeof(pwd));
}
```

A▾ ☐11010 ☑.LX0: ☐lib.f: ☑.text ☑// ☐\s+ ☑Int

```asm
.LC0:
        .string "TODO, use password %s\n"
secure_use_password(char*):
        push    rbp
        mov     rbp, rsp
        sub     rsp, 16
        mov     QWORD PTR [rbp-8], rdi
        mov     rax, QWORD PTR [rbp-8]
        mov     rsi, rax
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        nop
        leave
        ret
secure_get_user_password():
        push    rbp
        mov     rbp, rsp
        sub     rsp, 64
        mov     rdx, QWORD PTR stdin[rip]
        lea     rax, [rbp-64]
        mov     esi, 64
        mov     rdi, rax
        call    fgets
        lea     rax, [rbp-64]
        mov     rdi, rax
        call    secure_use_password(char*)
        lea     rax, [rbp-64]
        mov     edx, 64
        mov     esi, 0
        mov     rdi, rax
        call    memset
        nop
```

Example 2: disappearing security measures (after)

https://godbolt.org/z/3EyZXQ



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

void secure_use_password(char* buf){
    printf("TODO, use password %s\n",buf );
}

void secure_get_user_password(void)
{
    char pwd[64];
    fgets(pwd, 64, stdin);
    secure_use_password(pwd);

    //wipe the password from the memory
    memset(pwd, 0, sizeof(pwd));
}
```

A▾ □11010 ☑.LX0: □lib.f: ☑.text ☑// □\s+ ☑Intel ☑Der

```
.LC0:
        .string "TODO, use password %s\n"
secure_use_password(char*):
        sub     rsp, 8
        mov     rsi, rdi
        mov     edi, OFFSET FLAT:.LC0
        mov     eax, 0
        call    printf
        add     rsp, 8
        ret
secure_get_user_password():
        sub     rsp, 72
        mov     rdx, QWORD PTR stdin[rip]
        mov     esi, 64
        mov     rdi, rsp
        call    fgets
        mov     rdi, rsp
        call    secure_use_password(char*)
        add     rsp, 72
        ret
```

Executables

- Start as files on the filesystem
- As seen last time, executables carry loading information
- But what happens when we `run` the executable?

OS Kernel

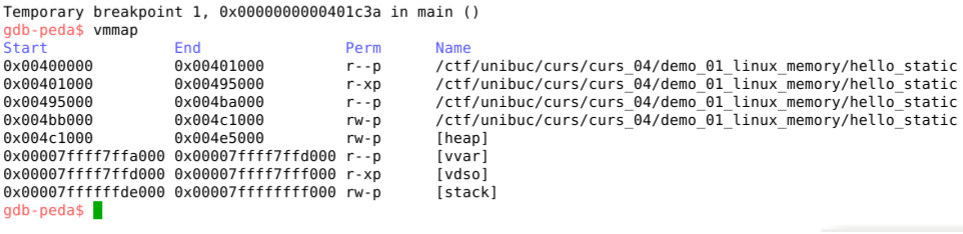
- Provides a separate address space from other processes
- Provides randomization where compatible (TBD)
- Provides expandable stack space, heap space
- Passes control to a suitable loader (interpreter)

Loaders

- Parse the file structure
- Copy segment contents into memory
- Expand sparse segments
- Set adequate permissions to each segment
- Do the same for any linked libraries needed
- Pass control to the address specified in the header

Linux Address Space Layout (1/2)

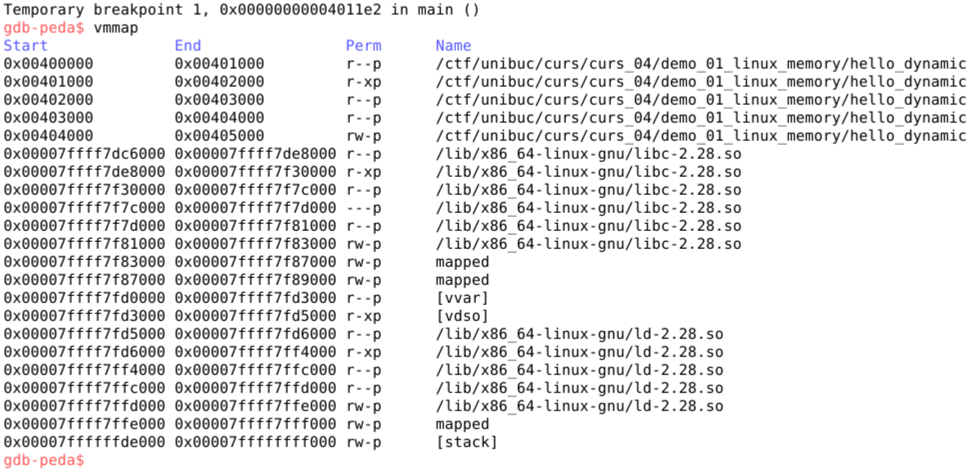
- Static Executable



```
Temporary breakpoint 1, 0x0000000000401c3a in main ()
gdb-peda$ vmmap
Start              End                Perm      Name
0x00400000         0x00401000         r--p      /ctf/unibuc/curs/curs_04/demo_01_linux_memory/hello_static
0x00401000         0x00495000         r-xp      /ctf/unibuc/curs/curs_04/demo_01_linux_memory/hello_static
0x00495000         0x004ba000         r--p      /ctf/unibuc/curs/curs_04/demo_01_linux_memory/hello_static
0x004bb000         0x004c1000         rw-p      /ctf/unibuc/curs/curs_04/demo_01_linux_memory/hello_static
0x004c1000         0x004e5000         rw-p      [heap]
0x00007ffff7ffa000 0x00007ffff7ffd000 r--p      [vvar]
0x00007ffff7ffd000 0x00007ffff7fff000 r-xp      [vdso]
0x00007ffffffde000 0x00007ffffffff000 rw-p      [stack]
gdb-peda$ 
```

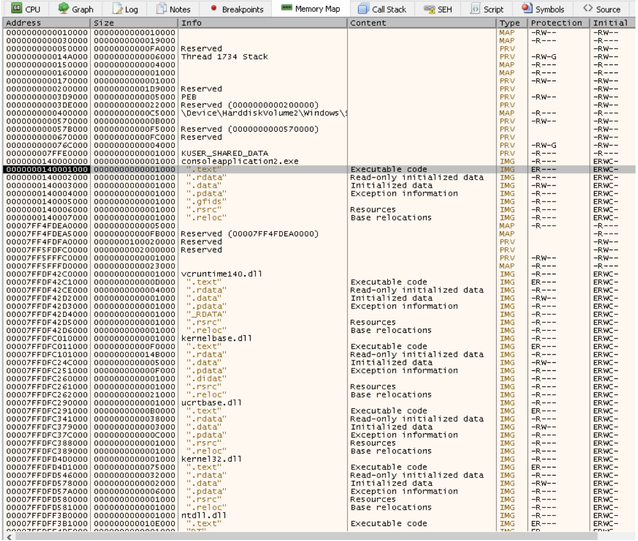
Linux Address Space Layout (2/2)

- Dynamic Executable



```
Temporary breakpoint 1, 0x00000000004011e2 in main ()
gdb-peda$ vmmap
Start              End                Perm      Name
0x00400000         0x00401000         r--p      /ctf/unibuc/curs/curs_04/demo_01_linux_memory/hello_dynamic
0x00401000         0x00402000         r-xp      /ctf/unibuc/curs/curs_04/demo_01_linux_memory/hello_dynamic
0x00402000         0x00403000         r--p      /ctf/unibuc/curs/curs_04/demo_01_linux_memory/hello_dynamic
0x00403000         0x00404000         r--p      /ctf/unibuc/curs/curs_04/demo_01_linux_memory/hello_dynamic
0x00404000         0x00405000         rw-p      /ctf/unibuc/curs/curs_04/demo_01_linux_memory/hello_dynamic
0x00007ffff7dc6000 0x00007ffff7de8000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007ffff7de8000 0x00007ffff7f30000 r-xp      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007ffff7f30000 0x00007ffff7f7c000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007ffff7f7c000 0x00007ffff7f7d000 ---p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007ffff7f7d000 0x00007ffff7f81000 r--p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007ffff7f81000 0x00007ffff7f83000 rw-p      /lib/x86_64-linux-gnu/libc-2.28.so
0x00007ffff7f83000 0x00007ffff7f87000 rw-p      mapped
0x00007ffff7f87000 0x00007ffff7f89000 rw-p      mapped
0x00007ffff7fd0000 0x00007ffff7fd3000 r--p      [vvar]
0x00007ffff7fd3000 0x00007ffff7fd5000 r-xp      [vdso]
0x00007ffff7fd5000 0x00007ffff7fd6000 r--p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007ffff7fd6000 0x00007ffff7ff4000 r-xp      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007ffff7ff4000 0x00007ffff7ffc000 r--p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007ffff7ffc000 0x00007ffff7ffd000 r--p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007ffff7ffd000 0x00007ffff7ffe000 rw-p      /lib/x86_64-linux-gnu/ld-2.28.so
0x00007ffff7ffe000 0x00007ffff7fff000 rw-p      mapped
0x00007ffffffde000 0x00007ffffffff000 rw-p      [stack]
gdb-peda$
```

Windows Address Space Layout



Address	Size	Info	Content	Type	Protection	Initial
0000000000010000	0000000000010000			MAP	-RW--	-RW--
0000000000030000	0000000000019000			MAP	-R---	-R---
0000000000050000	00000000000FA000	Reserved		PRV		-RW--
000000000014A000	0000000000006000	Thread 1734 Stack		PRV	-RW-G	-RW--
0000000000150000	0000000000004000			MAP	-R---	-R---
0000000000160000	0000000000001000			MAP	-R---	-R---
0000000000170000	0000000000001000			PRV	-RW--	-RW--
0000000000200000	00000000001D9000	Reserved		PRV		-RW--
00000000003D9000	0000000000005000	PEB		PRV	-RW--	-RW--
00000000003DE000	0000000000022000	Reserved (0000000000200000)		PRV		-RW--
0000000000400000	00000000000C5000	\Device\HarddiskVolume2\Windows\S		MAP	-R---	-R---
0000000000570000	000000000000B000			PRV	-RW--	-RW--
000000000057B000	00000000000F5000	Reserved (0000000000570000)		PRV		-RW--
0000000000670000	00000000000FC000	Reserved		PRV		-RW--
000000000076C000	0000000000004000			PRV	-RW-G	-RW--
000000007FFE0000	0000000000001000	KUSER_SHARED_DATA		PRV	-R---	-R---
0000000140000000	0000000000001000	consoleapplication2.exe		IMG	-R---	ERWC-
0000000140001000	0000000000001000	".text"	Executable code	IMG	ER---	ERWC-
0000000140002000	0000000000001000	".rdata"	Read-only initialized data	IMG	-R---	ERWC-
0000000140003000	0000000000001000	".data"	Initialized data	IMG	-RW--	ERWC-
0000000140004000	0000000000001000	".pdata"	Exception information	IMG	-R---	ERWC-
0000000140005000	0000000000001000	".gfids"		IMG	-R---	ERWC-
0000000140006000	0000000000001000	".rsrc"	Resources	IMG	-R---	ERWC-
0000000140007000	0000000000001000	".reloc"	Base relocations	IMG	-R---	ERWC-
00007FF4FDEA0000	0000000000005000			MAP	-R---	-R---
00007FF4FDEA5000	00000000000FB000	Reserved (00007FF4FDEA0000)		MAP		-R---
00007FF4FDFA0000	0000000100020000	Reserved		PRV		-RW--
00007FF5FDFC0000	0000000002000000	Reserved		PRV		-RW--
00007FF5FFFC0000	0000000000001000			PRV	-RW--	-RW--
00007FF5FFFD0000	0000000000023000			MAP	-R---	-R---
00007FFDF42C0000	0000000000001000	vcruntime140.dll		IMG	-R---	ERWC-
00007FFDF42C1000	000000000000D000	".text"	Executable code	IMG	ER---	ERWC-
00007FFDF42CE000	0000000000004000	".rdata"	Read-only initialized data	IMG	-R---	ERWC-
00007FFDF42D2000	0000000000001000	".data"	Initialized data	IMG	-RW--	ERWC-
00007FFDF42D3000	0000000000001000	".pdata"	Exception information	IMG	-R---	ERWC-
00007FFDF42D4000	0000000000001000	"_RDATA"		IMG	-R---	ERWC-
00007FFDF42D5000	0000000000001000	".rsrc"	Resources	IMG	-R---	ERWC-
00007FFDF42D6000	0000000000001000	".reloc"	Base relocations	IMG	-R---	ERWC-
00007FFDFC010000	0000000000001000	kernelbase.dll		IMG	-R---	ERWC-
00007FFDFC011000	00000000000F0000	".text"	Executable code	IMG	ER---	ERWC-
00007FFDFC101000	000000000014B000	".rdata"	Read-only initialized data	IMG	-R---	ERWC-
00007FFDFC24C000	0000000000005000	".data"	Initialized data	IMG	-RW--	ERWC-
00007FFDFC251000	000000000000F000	".pdata"	Exception information	IMG	-R---	ERWC-
00007FFDFC260000	0000000000001000	".didat"		IMG	-R---	ERWC-
00007FFDFC261000	0000000000001000	".rsrc"	Resources	IMG	-R---	ERWC-
00007FFDFC262000	0000000000021000	".reloc"	Base relocations	IMG	-R---	ERWC-
00007FFDFC290000	0000000000001000	ucrtbase.dll		IMG	-R---	ERWC-
00007FFDFC291000	00000000000B0000	".text"	Executable code	IMG	ER---	ERWC-
00007FFDFC341000	0000000000038000	".rdata"	Read-only initialized data	IMG	-R---	ERWC-
00007FFDFC379000	0000000000003000	".data"	Initialized data	IMG	-RW--	ERWC-
00007FFDFC37C000	000000000000C000	".pdata"	Exception information	IMG	-R---	ERWC-
00007FFDFC388000	0000000000001000	".rsrc"	Resources	IMG	-R---	ERWC-
00007FFDFC389000	0000000000001000	".reloc"	Base relocations	IMG	-R---	ERWC-
00007FFDFD4D0000	0000000000001000	kernel32.dll		IMG	-R---	ERWC-
00007FFDFD4D1000	0000000000075000	".text"	Executable code	IMG	ER---	ERWC-
00007FFDFD546000	0000000000032000	".rdata"	Read-only initialized data	IMG	-R---	ERWC-
00007FFDFD578000	0000000000002000	".data"	Initialized data	IMG	-RW--	ERWC-
00007FFDFD57A000	0000000000006000	".pdata"	Exception information	IMG	-R---	ERWC-
00007FFDFD580000	0000000000001000	".rsrc"	Resources	IMG	-R---	ERWC-
00007FFDFD581000	0000000000001000	".reloc"	Base relocations	IMG	-R---	ERWC-
00007FFDFF3B0000	0000000000001000	ntdll.dll		IMG	-R---	ERWC-
00007FFDFF3B1000	000000000010E000	".text"	Executable code	IMG	ER---	ERWC-

How do processes inter-communicate?

How do processes inter-communicate?

- Shared memory
- Message queues
- Pipes
- Sockets
- Synchronization

How do processes inter-communicate?

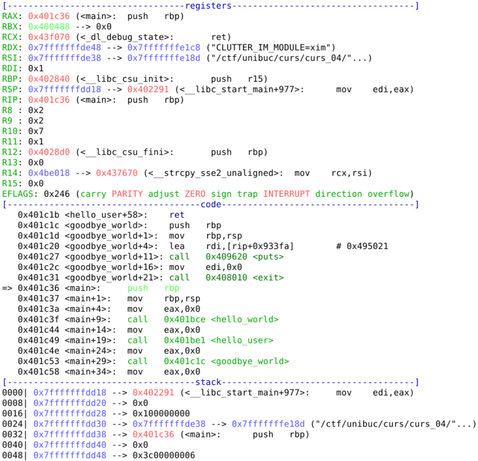
- Shared memory
- Message queues
- Pipes
- Sockets
- Synchronization
- Direct access (used by debugging processes)

Linux debug methods (ptrace syscall)

- Attach to a process (called tracee)
- Read/write memory from tracee
- Read/write CPU registers from tracee
- Single step (one CPU instruction at a time)
- Start/stop/continue execution
- Handle breakpoints

Linux low-level debugging

- Debuggers mainly use ptrace
- We study GDB plus a plugin (PEDA)



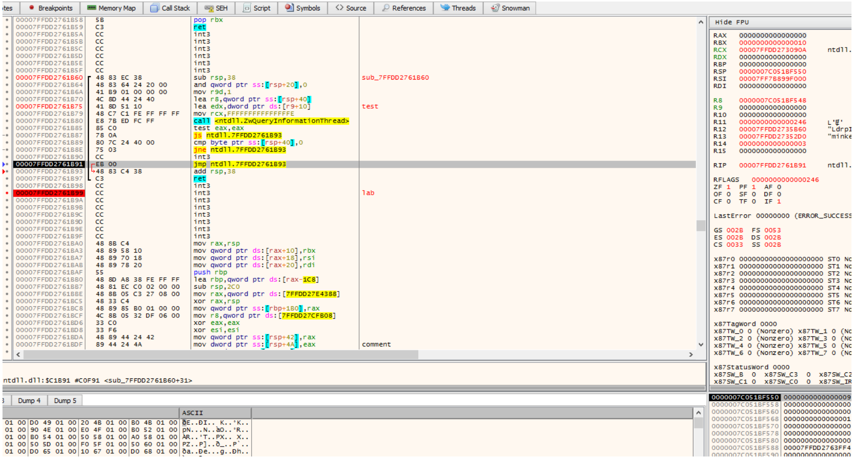
```
[----------------------------------registers-----------------------------------]
RAX: 0x401c36 (<main>:  push   rbp)
RBX: 0x400488 --> 0x0
RCX: 0x43f070 (<_dl_debug_state>:       ret)
RDX: 0x7fffffffde48 --> 0x7fffffffe1c8 ("CLUTTER_IM_MODULE=xim")
RSI: 0x7fffffffde38 --> 0x7fffffffe18d ("/ctf/unibuc/curs/curs_04/"...)
RDI: 0x1
RBP: 0x402840 (<__libc_csu_init>:       push   r15)
RSP: 0x7fffffffdd18 --> 0x402291 (<__libc_start_main+977>:      mov    edi,eax)
RIP: 0x401c36 (<main>:  push   rbp)
R8 : 0x2
R9 : 0x2
R10: 0x7
R11: 0x1
R12: 0x4028d0 (<__libc_csu_fini>:       push   rbp)
R13: 0x0
R14: 0x4be018 --> 0x437670 (<__strcpy_sse2_unaligned>:  mov    rcx,rsi)
R15: 0x0
EFLAGS: 0x246 (carry PARITY adjust ZERO sign trap INTERRUPT direction overflow)
[-------------------------------------code-------------------------------------]
   0x401c1b <hello_user+58>:    ret
   0x401c1c <goodbye_world>:    push   rbp
   0x401c1d <goodbye_world+1>:  mov    rbp,rsp
   0x401c20 <goodbye_world+4>:  lea    rdi,[rip+0x933fa]        # 0x495021
   0x401c27 <goodbye_world+11>: call   0x409620 <puts>
   0x401c2c <goodbye_world+16>: mov    edi,0x0
   0x401c31 <goodbye_world+21>: call   0x408010 <exit>
=> 0x401c36 <main>:     push   rbp
   0x401c37 <main+1>:   mov    rbp,rsp
   0x401c3a <main+4>:   mov    eax,0x0
   0x401c3f <main+9>:   call   0x401bce <hello_world>
   0x401c44 <main+14>:  mov    eax,0x0
   0x401c49 <main+19>:  call   0x401be1 <hello_user>
   0x401c4e <main+24>:  mov    eax,0x0
   0x401c53 <main+29>:  call   0x401c1c <goodbye_world>
   0x401c58 <main+34>:  mov    eax,0x0
[------------------------------------stack-------------------------------------]
0000| 0x7fffffffdd18 --> 0x402291 (<__libc_start_main+977>:     mov    edi,eax)
0008| 0x7fffffffdd20 --> 0x0
0016| 0x7fffffffdd28 --> 0x100000000
0024| 0x7fffffffdd30 --> 0x7fffffffde38 --> 0x7fffffffe18d ("/ctf/unibuc/curs/curs_04/"...)
0032| 0x7fffffffdd38 --> 0x401c36 (<main>:      push   rbp)
0040| 0x7fffffffdd40 --> 0x0
0048| 0x7fffffffdd48 --> 0x3c00000006
```

Windows debug methods (separate syscalls)

- Attach to a process (OpenProcess)
- Read/write memory from tracee (ReadProcessMemory/WriteProcessMemory)
- Read/write CPU registers from tracee (GetThreadContext)
- Start/stop/continue execution (DebugBreakProcess)
- Handle breakpoints (WaitForDebugEvent/ContinueDebugEvent)

Windows low-level debugging

- Windbg is the most powerful but hard to learn
- X64dbg is a decent debugger handling 32/64



```
00007FFDD2761B58  5B                     pop rbx
00007FFDD2761B59  C3                     ret
00007FFDD2761B5A  CC                     int3
00007FFDD2761B5B  CC                     int3
00007FFDD2761B5C  CC                     int3
00007FFDD2761B5D  CC                     int3
00007FFDD2761B5E  CC                     int3
00007FFDD2761B5F  CC                     int3
00007FFDD2761B60  48 83 EC 38            sub rsp,38                                sub_7FFDD2761B60
00007FFDD2761B64  48 83 64 24 20 00      and qword ptr ss:[rsp+20],0
00007FFDD2761B6A  41 B9 01 00 00 00      mov r9d,1
00007FFDD2761B70  4C 8D 44 24 40         lea r8,qword ptr ss:[rsp+40]
00007FFDD2761B75  41 8D 51 10            lea edx,dword ptr ds:[r9+10]              test
00007FFDD2761B79  48 C7 C1 FE FF FF FF   mov rcx,FFFFFFFFFFFFFFFE
00007FFDD2761B80  E8 7B ED FC FF         call <ntdll.ZwQueryInformationThread>
00007FFDD2761B85  85 C0                  test eax,eax
00007FFDD2761B87  78 0A                  js ntdll.7FFDD2761B93
00007FFDD2761B89  80 7C 24 40 00         cmp byte ptr ss:[rsp+40],0
00007FFDD2761B8E  75 03                  jne ntdll.7FFDD2761B93
00007FFDD2761B90  CC                     int3
00007FFDD2761B91  EB 00                  jmp ntdll.7FFDD2761B93
00007FFDD2761B93  48 83 C4 38            add rsp,38
00007FFDD2761B97  C3                     ret
00007FFDD2761B98  CC                     int3
00007FFDD2761B99  CC                     int3                                      lab
00007FFDD2761B9A  CC                     int3
00007FFDD2761B9B  CC                     int3
00007FFDD2761B9C  CC                     int3
00007FFDD2761B9D  CC                     int3
00007FFDD2761B9E  CC                     int3
00007FFDD2761B9F  CC                     int3
00007FFDD2761BA0  48 8B C4               mov rax,rsp
00007FFDD2761BA3  48 89 58 10            mov qword ptr ds:[rax+10],rbx
00007FFDD2761BA7  48 89 70 18            mov qword ptr ds:[rax+18],rsi
00007FFDD2761BAB  48 89 78 20            mov qword ptr ds:[rax+20],rdi
00007FFDD2761BAF  55                     push rbp
00007FFDD2761BB0  48 8D A8 38 FE FF FF   lea rbp,qword ptr ds:[rax-1C8]
00007FFDD2761BB7  48 81 EC C0 02 00 00   sub rsp,2C0
00007FFDD2761BBE  48 8B 05 C3 27 08 00   mov rax,qword ptr ds:[7FFDD27E4388]
00007FFDD2761BC5  48 33 C4               xor rax,rsp
00007FFDD2761BC8  48 89 85 B0 01 00 00   mov qword ptr ss:[rbp+1B0],rax
00007FFDD2761BCF  4C 8B 05 32 DF 06 00   mov r8,qword ptr ds:[7FFDD27CFB08]
00007FFDD2761BD6  33 C0                  xor eax,eax
00007FFDD2761BD8  33 F6                  xor esi,esi
00007FFDD2761BDA  48 89 44 24 42         mov qword ptr ss:[rsp+42],rax
00007FFDD2761BDF  89 44 24 4A            mov dword ptr ss:[rsp+4A],eax             comment
```

```
ntdll.dll:$C1B91 #C0F91 <sub_7FFDD2761B60+31>
```

```
RAX     0000000000000000
RBX     0000000000000010
RCX     00007FFDD273090A     ntdll.
RDX     0000000000000000
RBP     0000000000000000
RSP     0000007C051BF550
RSI     00007FF7B899F000
RDI     0000000000000000

R8      0000007C051BF548
R9      0000000000000000
R10     0000000000000000
R11     0000000000000246     L'Ɇ'
R12     00007FFDD2735B60     "LdrpI
R13     00007FFDD27352D0     "minke
R14     0000000000000003
R15     0000000000000000

RIP     00007FFDD2761B91     ntdll.

RFLAGS    0000000000000246
ZF 1  PF 1  AF 0
OF 0  SF 0  DF 0
CF 0  TF 0  IF 1

LastError 00000000 (ERROR_SUCCESS

GS 002B  FS 0053
ES 002B  DS 002B
CS 0033  SS 002B

x87r0 00000000000000000000 ST0 Nc
x87r1 00000000000000000000 ST1 Nc
x87r2 00000000000000000000 ST2 Nc
x87r3 00000000000000000000 ST3 Nc
x87r4 00000000000000000000 ST4 Nc
x87r5 00000000000000000000 ST5 Nc
x87r6 00000000000000000000 ST6 Nc
x87r7 00000000000000000000 ST7 Nc

x87TagWord 0000
x87TW_0 0 (Nonzero) x87TW_1 0 (Nc
x87TW_2 0 (Nonzero) x87TW_3 0 (Nc
x87TW_4 0 (Nonzero) x87TW_5 0 (Nc
x87TW_6 0 (Nonzero) x87TW_7 0 (Nc

x87StatusWord 0000
x87SW_B  0  x87SW_C3  0  x87SW_C2
x87SW_C1 0  x87SW_C0  0  x87SW_IR
```

```
01 00 D0 49 01 00 20 4B 01 00 B0 4B 01 00   ðE..ÐI.. K..°K..
01 00 90 4E 01 00 E0 4F 01 00 B0 52 01 00   pN...N..àO..°R..
01 00 B0 54 01 00 50 58 01 00 A0 58 01 00   ÀR..°T..PX.. X..
01 00 50 5D 01 00 F0 5F 01 00 50 60 01 00   PZ..P]..ð_..P`..
01 00 D0 65 01 00 10 67 01 00 D0 68 01 00   ða..Ðe...g..Ðh..
```

```
0000007C051BF550  0000000000000009
0000007C051BF558  0000000000000000
0000007C051BF560  0000000000000000
0000007C051BF568  0000000000000001
0000007C051BF570  0000000000000000
0000007C051BF578  0000000000000000
0000007C051BF580  0000000000000000
0000007C051BF588  00007FFDD2763FF4
0000007C051BF590  0000000000000000
```

Fundamental tasks in a debugger wrt RE

- Interrupt (break) execution at a certain point in the code
- Inspect/modify virtual memory state/contents
- Inspect/modify CPU registers
- Analyze the call stack

Alternatives

- Processes can also be instrumented
- Intel PIN (Linux/Windows)
- Add extra code in the same address space
- More power, harder to detect, more complexity

Practice

- Any Questions?
- `http://pwnthybytes.ro/unibuc_re/04-lab.html`