REVERSE ENGINEERING – CLASS 0x04

DYNAMIC ANALYSIS

Cristian Rusu

LAST TIME

- **static analysis**
  - ELF
  - PE
- **IDA**

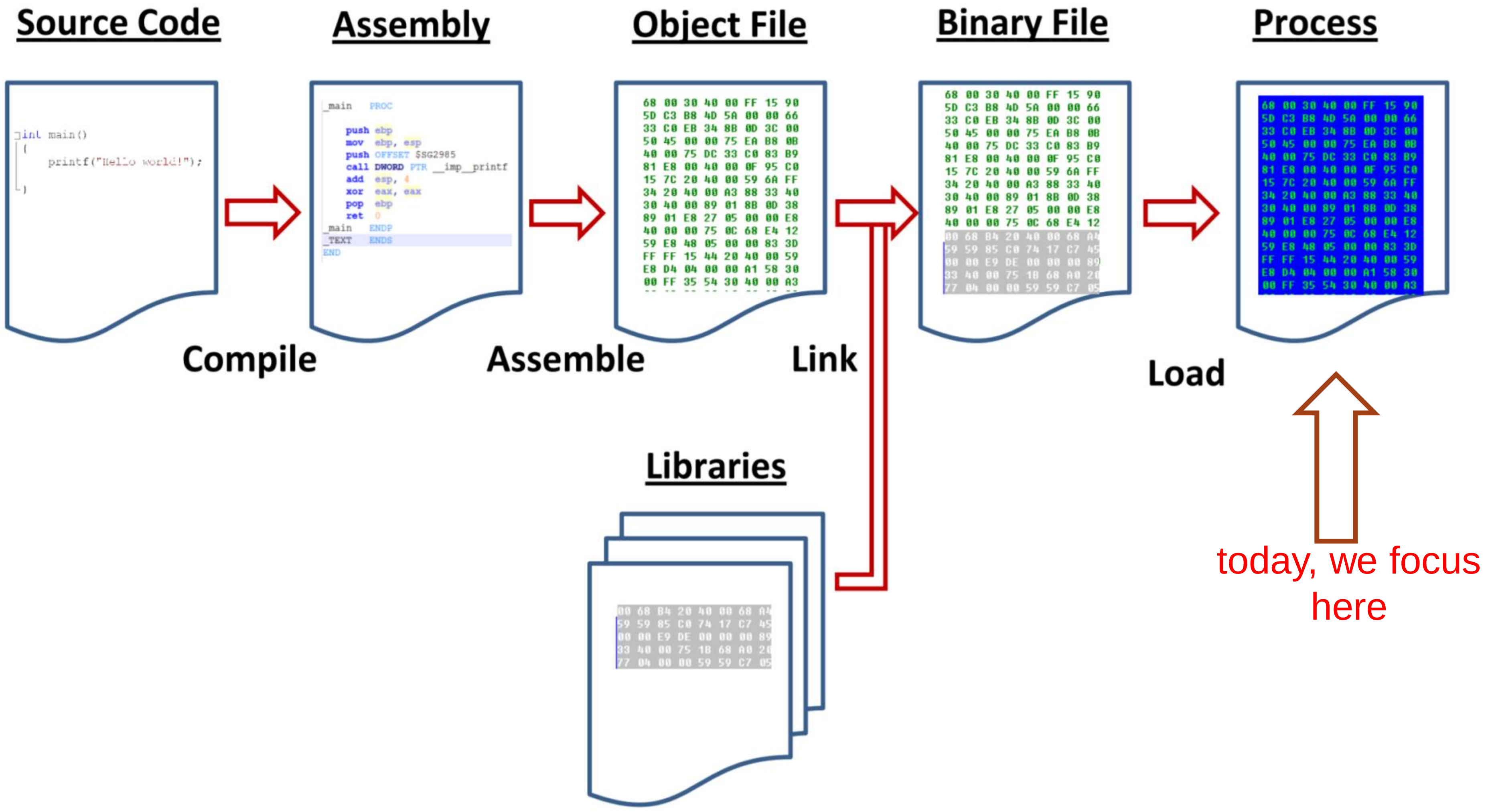
TODAY

- dynamic analysis

- debugging

.

FROM SOURCE CODE TO EXECUTION



```
int main()
{
    printf("Hello world!");
}
```

```
_main   PROC

    push ebp
    mov  ebp, esp
    push OFFSET $SG2985
    call DWORD PTR __imp__printf
    add  esp, 4
    xor  eax, eax
    pop  ebp
    ret  0
_main   ENDP
_TEXT   ENDS
END
```

```
68 00 30 40 00 FF 15 90
5D C3 B8 4D 5A 00 00 66
33 C0 EB 34 8B 0D 3C 00
50 45 00 00 75 EA B8 0B
40 00 75 DC 33 C0 83 B9
81 E8 00 40 00 0F 95 C0
15 7C 20 40 00 59 6A FF
34 20 40 00 A3 88 33 40
30 40 00 89 01 8B 0D 38
89 01 E8 27 05 00 00 E8
40 00 00 75 0C 68 E4 12
59 E8 48 05 00 00 83 3D
FF FF 15 44 20 40 00 59
E8 D4 04 00 00 A1 58 30
00 FF 35 54 30 40 00 A3
```

```
00 68 B4 20 40 00 68 A4
59 59 85 C0 74 17 C7 45
00 00 E9 DE 00 00 00 89
33 40 00 75 1B 68 A0 20
77 04 00 00 59 59 C7 05
```

today, we focus here

https://slideplayer.com/slide/4695781/

WHY DO DYNAMIC ANALYSIS?

- why isn’t static analysis enough?

.

WHY DO DYNAMIC ANALYSIS?

- **why isn't static analysis enough?**
  - dynamic analysis can complement static analysis (in practice, most likely, you will need to do both)
  - can detect subtle vulnerabilities
  - can detect new vulnerabilities
    - a new variable is added, time
  - can understand what the binary is doing when communicating
    - IPC
    - direct access

.

WHY DO DYNAMIC ANALYSIS?

- **why isn’t static analysis enough?**
  - dynamic analysis can complement static analysis (in practice, most likely, you will need to do both)
  - can detect subtle vulnerabilities
  - can detect new vulnerabilities
    - a new variable is added, time
  - can understand what the binary is doing when communicating
    - IPC (shared memory, pipes, sockets, messages queues, mutex)
    - direct access (debugging)

.

DYNAMIC ANALYSIS EXAMPLE 1

- **side-channel attacks**
  - in computer security, a side-channel attack is any attack based on extra information that can be gathered because of the fundamental way a computer protocol or algorithm is implemented, rather than flaws in the design of the protocol or algorithm itself
  - cache attacks
  - timing attacks
  - power-monitoring attacks
  - etc.

https://en.wikipedia.org/wiki/Side-channel_attack

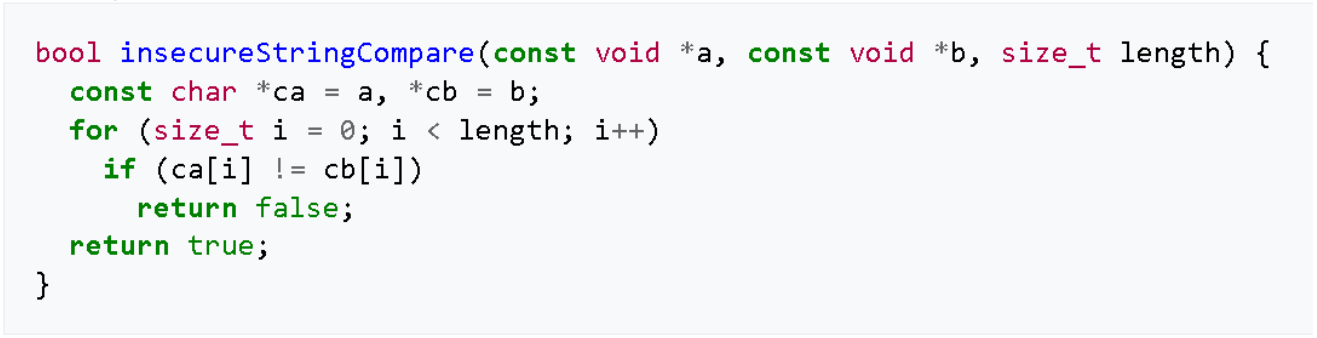
DYNAMIC ANALYSIS EXAMPLE 1

- **side-channel attacks**
  - in computer security, a side-channel attack is any attack based on extra information that can be gathered because of the fundamental way a computer protocol or algorithm is implemented, rather than flaws in the design of the protocol or algorithm itself
  - cache attacks
    - Meltdown, spectre

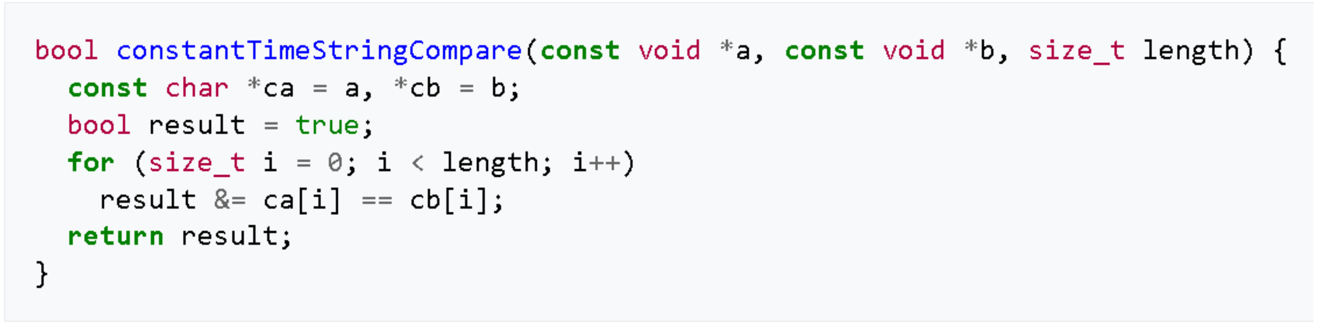
https://en.wikipedia.org/wiki/Side-channel_attack

DYNAMIC ANALYSIS EXAMPLE 1

- **side-channel attacks**
  - in computer security, a side-channel attack is any attack based on extra information that can be gathered because of the fundamental way a computer protocol or algorithm is implemented, rather than flaws in the design of the protocol or algorithm itself
  - timing attacks



```c
bool insecureStringCompare(const void *a, const void *b, size_t length) {
  const char *ca = a, *cb = b;
  for (size_t i = 0; i < length; i++)
    if (ca[i] != cb[i])
      return false;
  return true;
}
```



```c
bool constantTimeStringCompare(const void *a, const void *b, size_t length) {
  const char *ca = a, *cb = b;
  bool result = true;
  for (size_t i = 0; i < length; i++)
    result &= ca[i] == cb[i];
  return result;
}
```

https://en.wikipedia.org/wiki/Side-channel_attack

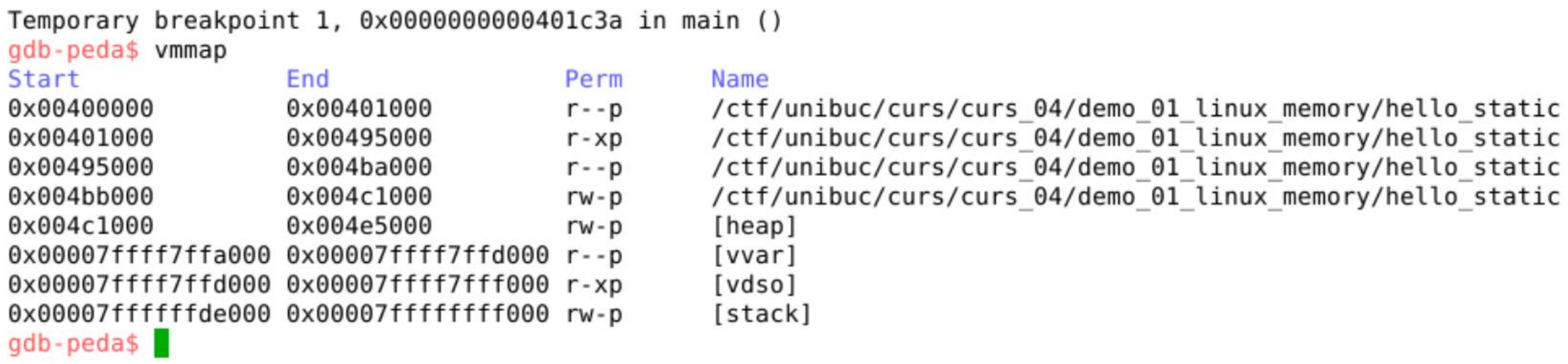
DYNAMIC ANALYSIS EXAMPLE 2

- **compiler eliminates security measures**
  - https://godbolt.org/z/QMZxYe
  - https://godbolt.org/z/3EyZXQ
- **same code, but with and without optimization flags**

RUNNING A PROCESS

- **OS kernel**
  - reads the binary
  - provides a separate address space for the process
    - *randomization can happen here*
  - provides expandable stack and heap spaces
  - passes control to the interpreter (loader)
    - parses the structure of the binary
    - copies segments into memory
    - sets appropriate permissions for each segment
    - checks for any linked libraries
    - passes control to the *_start* address written in the header

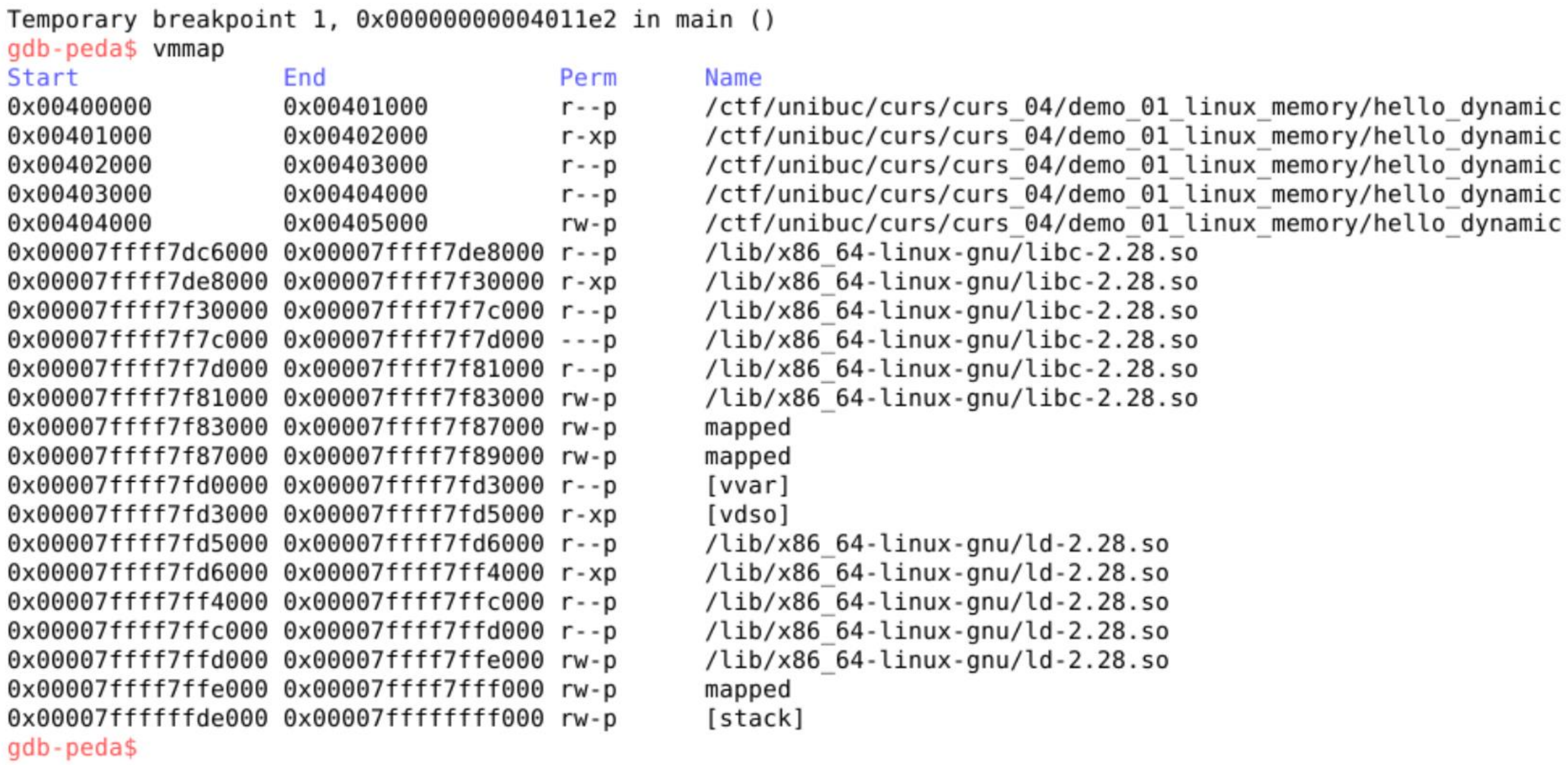
LINUX, STATIC BINARY/EXECUTABLE



```
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$ 
```

.

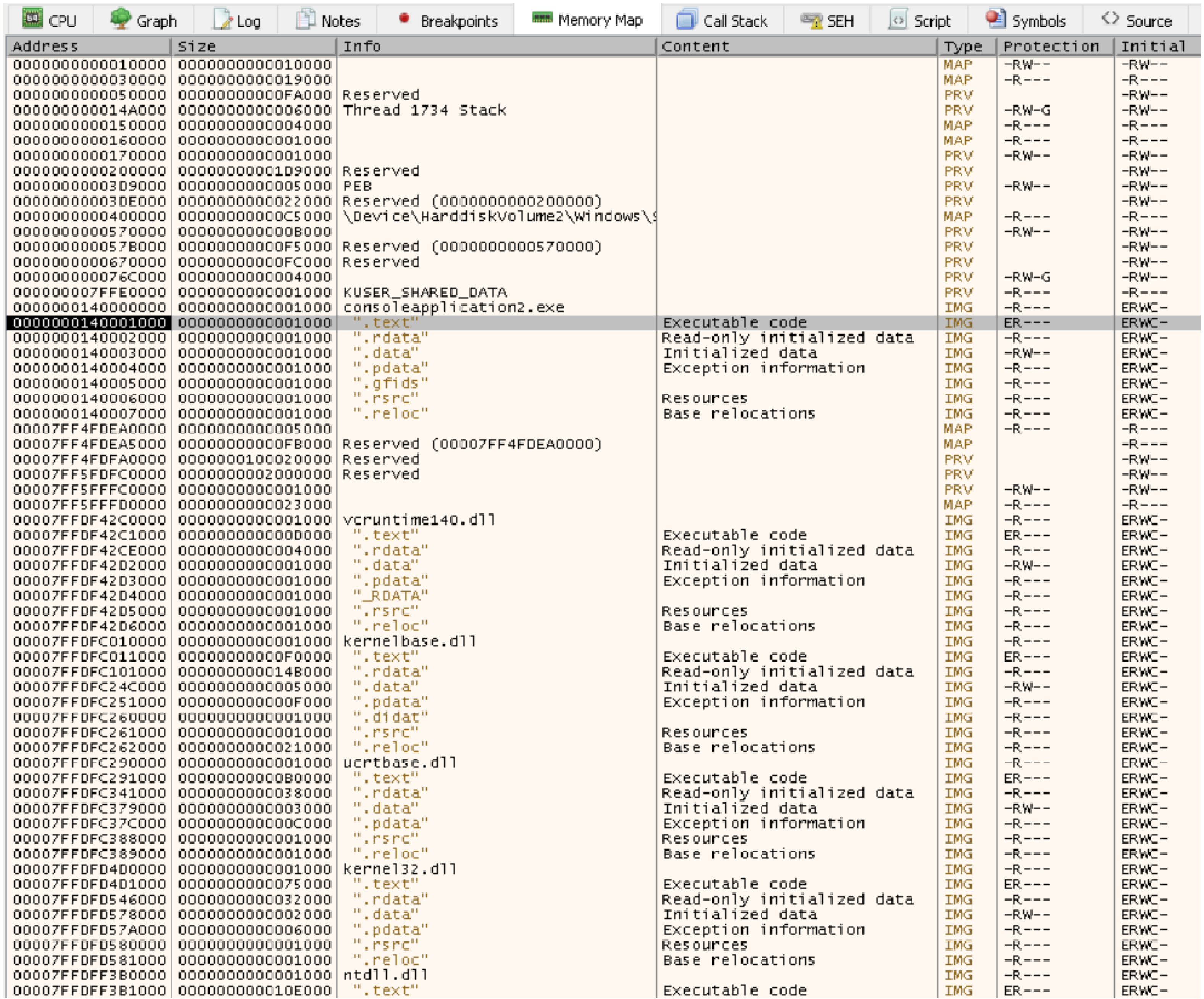
LINUX, DYNAMIC BINARY/EXECUTABLE



```
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\		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-

LINUX, DEBUGGING METHODS

- **ptrace syscalls**

- **you attach to a process (tracee): gdb –p PID**
  - read/write memory of the tracee
  - read/write CPU registers from tracee
  - single step (one CPU instruction at a time)
  - start/stop/continue execution
  - handle breakpoints

- **gdb + peda**

https://github.com/longld/peda

WINDOWS, DEBUGGING METHODS

- **special 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)

- **X64dbg and Windbg**

https://x64dbg.com/
https://learn.microsoft.com/en-us/windows-hardware/drivers/debugger/debugger-download-tools

DEBUGGING FOR 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

WHAT WE DID TODAY

- dynamic analysis

- debugging

NEXT TIME ...

- more on loading binaries
- obfuscation of binaries

.

REFERENCES

- **GDB,** https://www.youtube.com/watch?v=bWH-nL7v5F4
- **Windows debugging,** https://www.youtube.com/watch?v=2rGS5fYGtJ4
- **WinDBG,** https://www.youtube.com/watch?v=QuFJpH3My7A
- **Read a bluescreen using WinDBG,** https://www.youtube.com/watch?v=wUh592phlnQ

.