Binary Exploitation

Pwning & Reversing

AG Rechnersicherheit

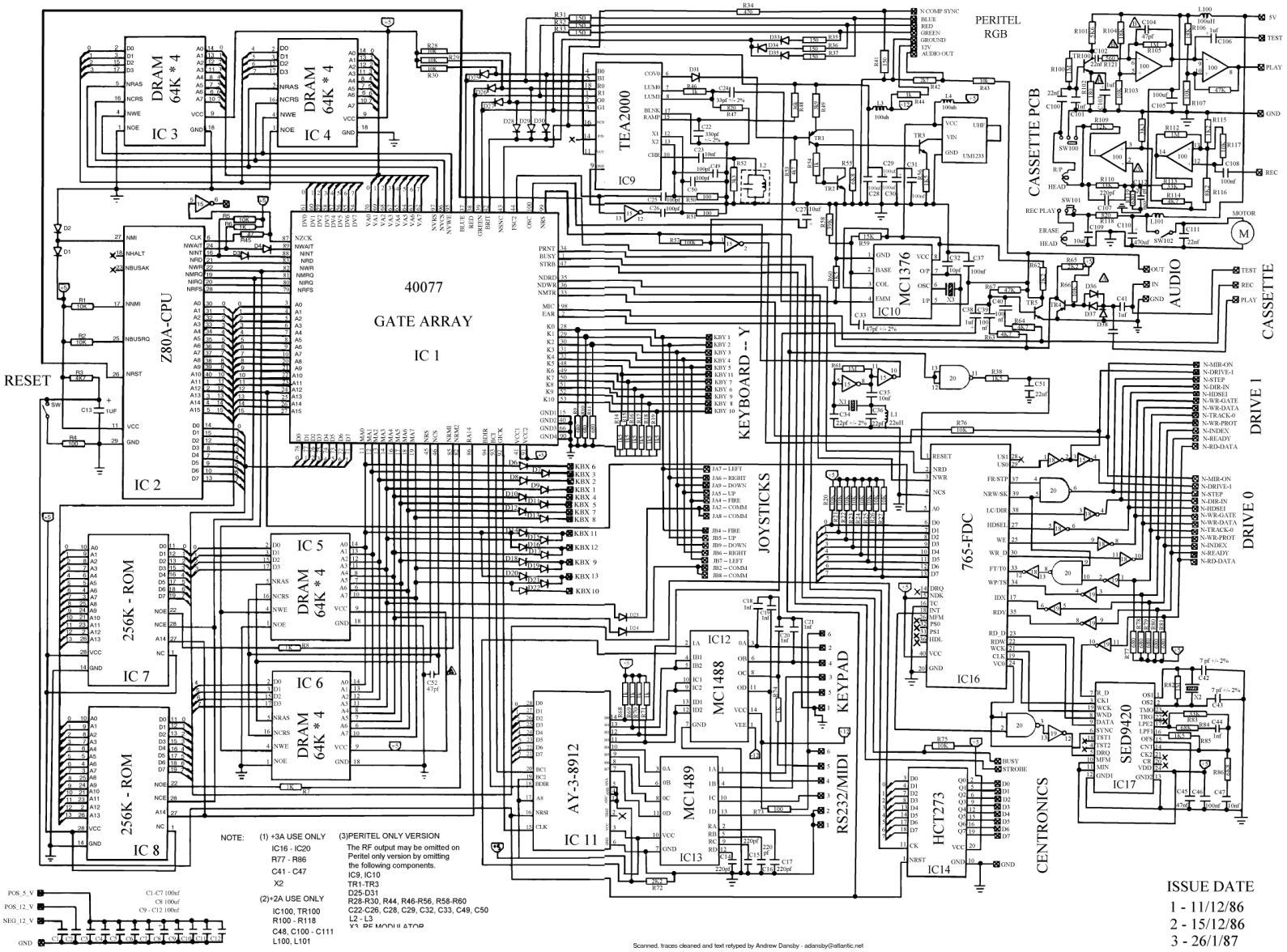
Team ENOFLAG / TU-Berlin

22.10.2019

Agenda

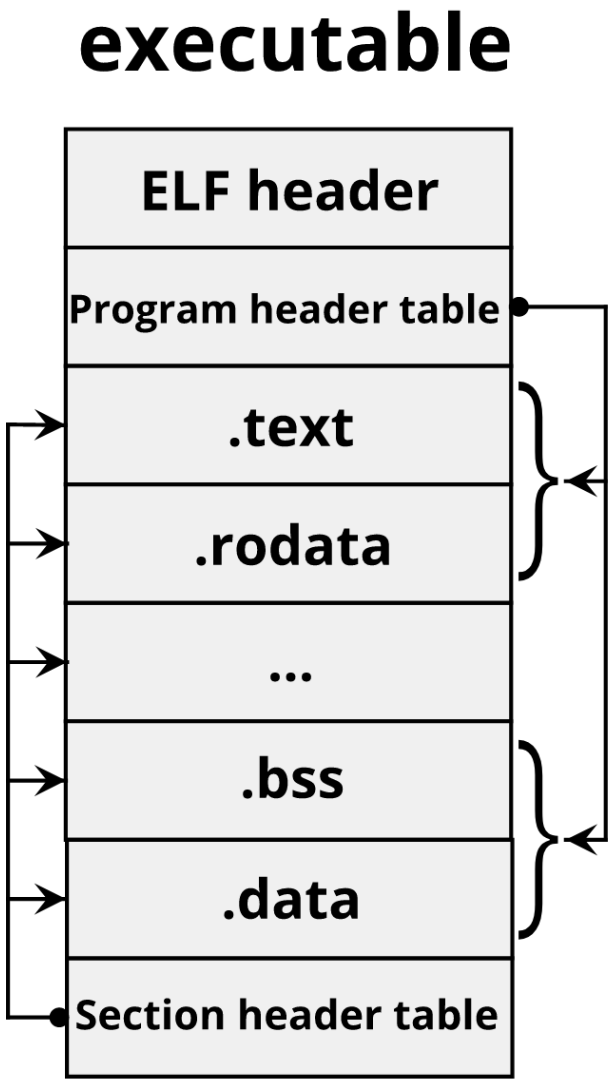
1. x86_64 Architecture
   - Sections, Segments and Registers/Calling Conventions
2. Tools
   - Disassembler, Debugger, Decompiler and other Tools
3. Vulnerabilities & Mitigations
   - Data Execution Prevention, Stack Canaries, ASLR. RelRO
   - Buffer Overflows, Ret2Libc, Return Oriented Programming
4. Exploitation Tools
   - pwntools, ropper, one_gadget
5. Exercises
   - Your Turn!

x86_64 Architecture

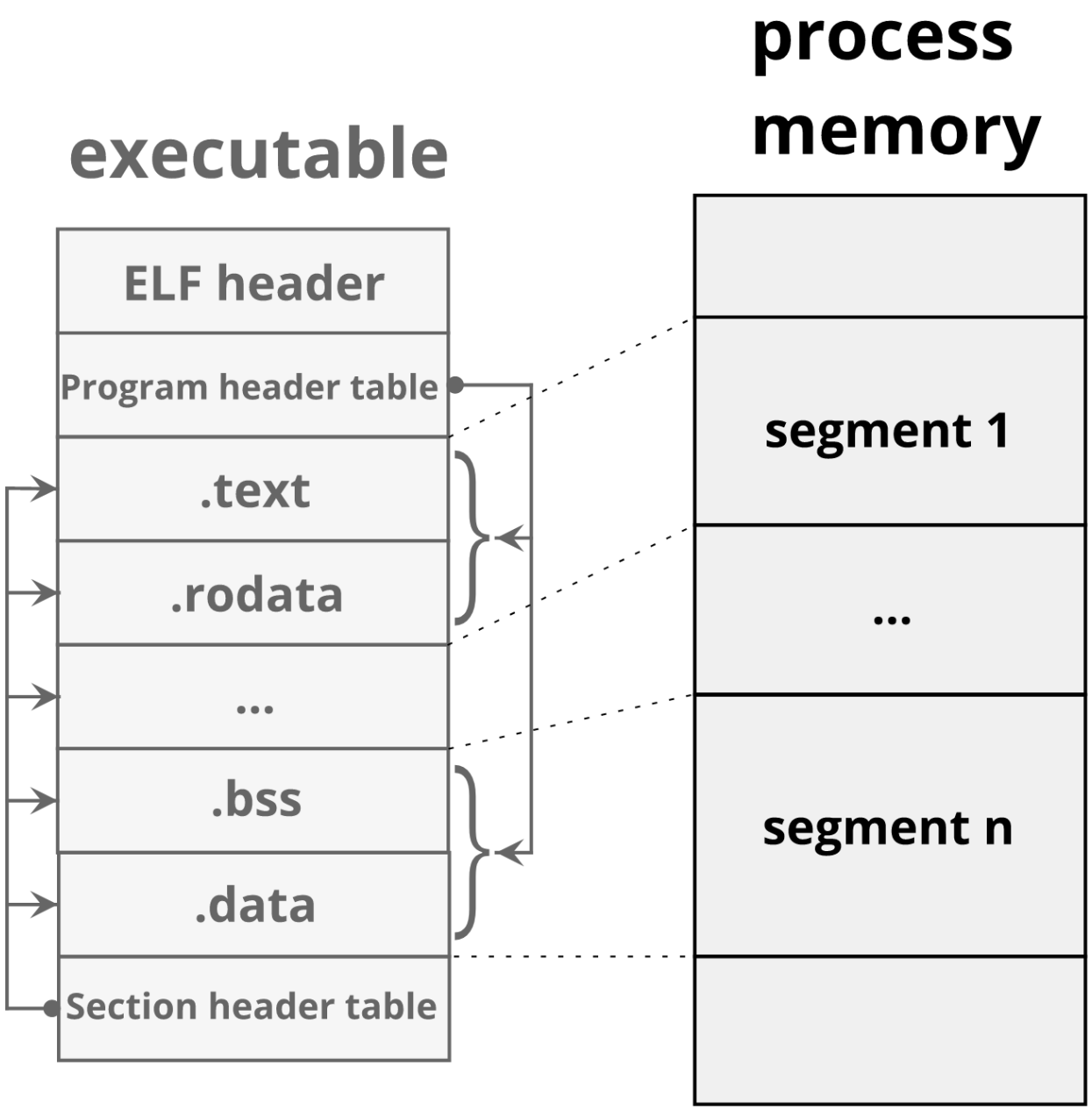


Scanned, traces cleaned and text retyped by Andrew Dansby - adansby@atlantic.net

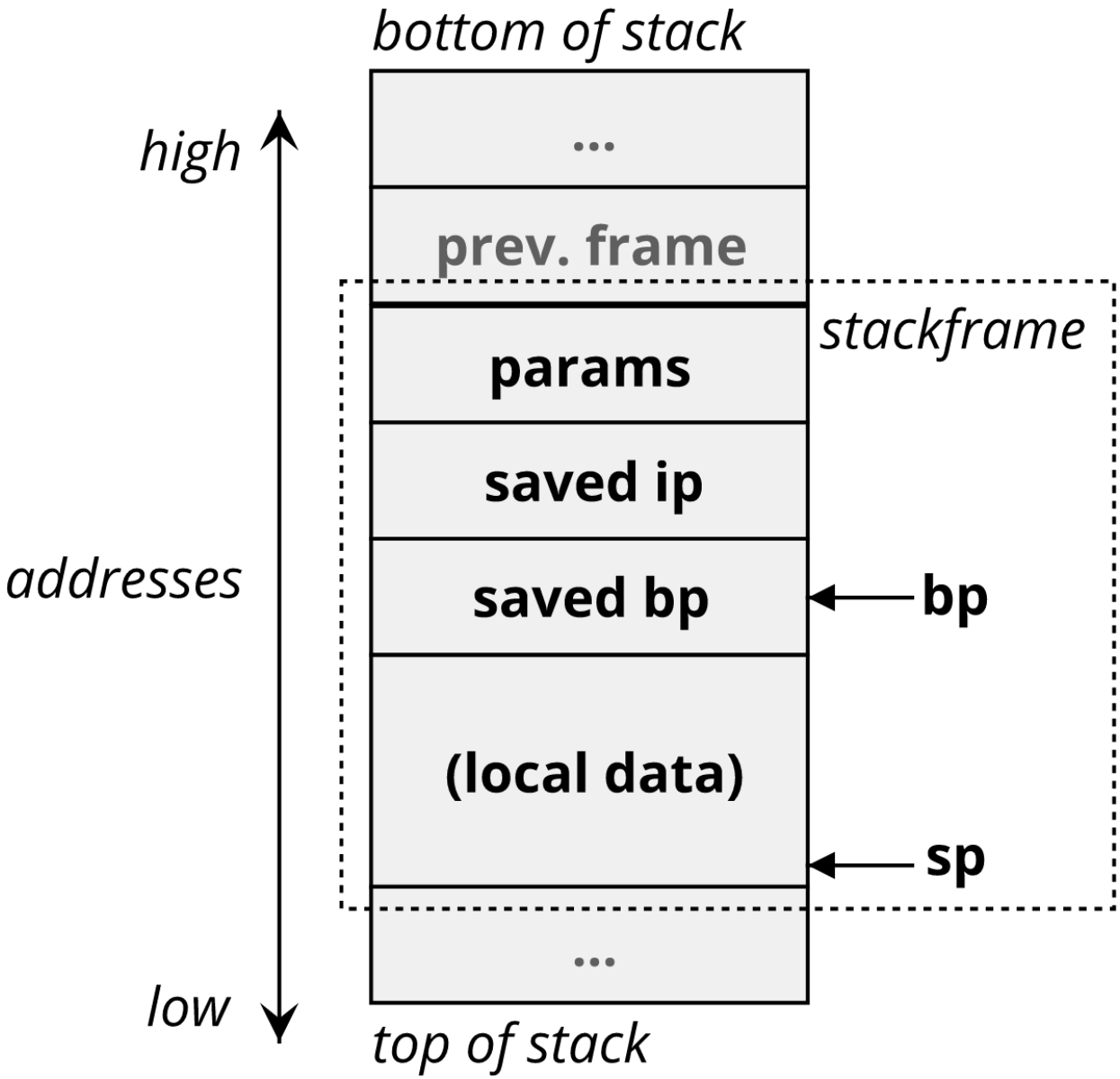
Sections & Segments



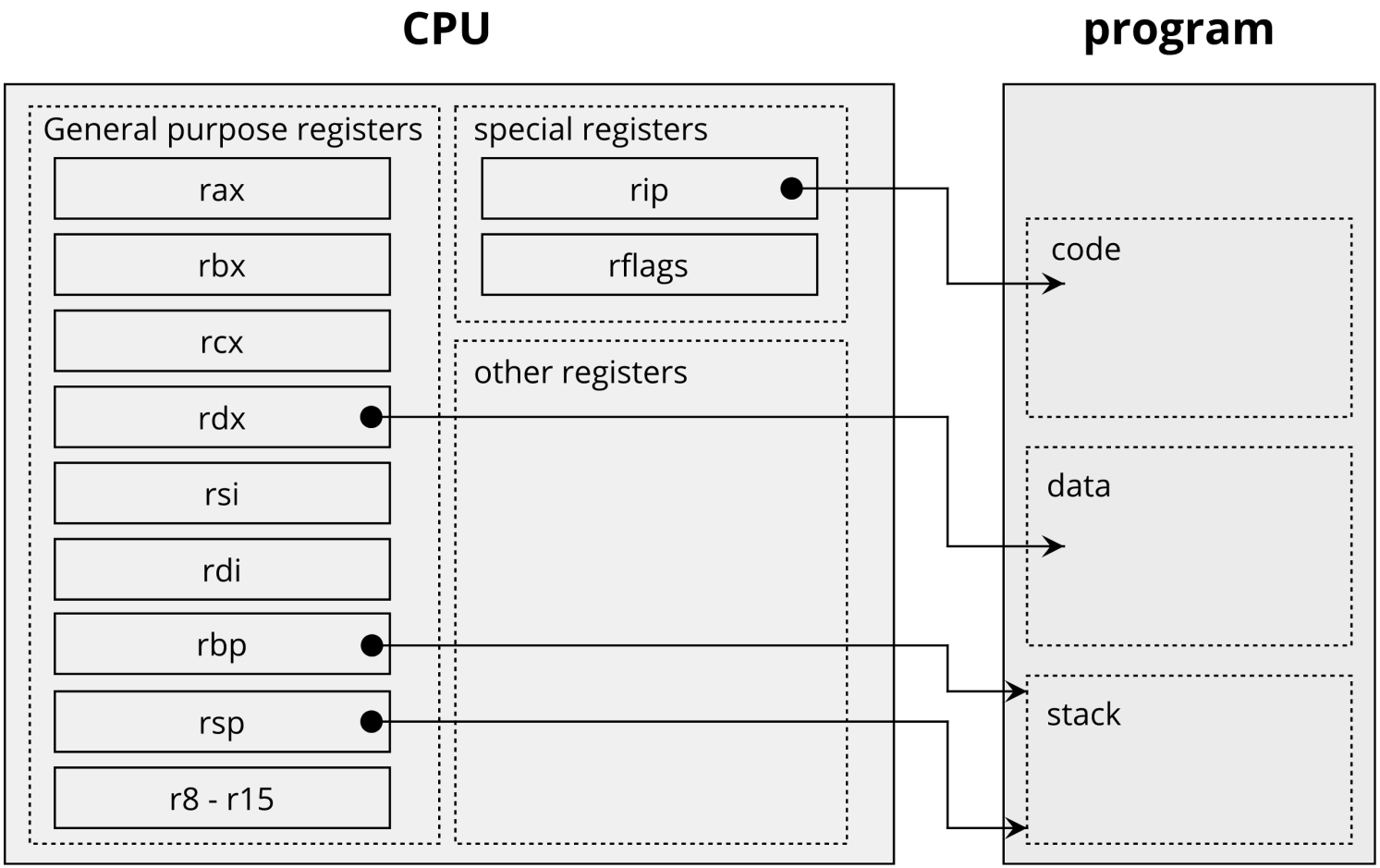
Sections & Segments



Stack


bottom of stack
high
...
prev. frame
stackframe
params
saved ip
addresses
saved bp
bp
(local data)
sp
...
low
top of stack

CPU Registers



Calling Conventions

- x86
  - **cdecl**:
    arguments are passed over the stack (default for c programs)
  - **others**: stdcall, fastcall, thiscall,...
- x86_64
  - **System V AMD64 ABI**:
    first six arguments are passed via registers (rdi, rsi, rdx, rcx, r8, r9) and only the rest over the stack (default on UNIX)
  - **Microsoft x64 calling convention**: (default on Windows)

SYSCALLS

- x86
  - **int 0x80**:
    syscall number passed via eax,
    up to 6 parameters via ebx, ecx, edx, esi, edi, ebp
    return value in eax
- x86_64
  - **int 0x80**
  - **syscall**:
    syscall number passed via rax,
    up to 6 parameters via rdi, rsi, rdx, r10, r8, r9
    return value in rax
- use the manual: `man syscall` and `man syscalls`

Tools

```
$ file ./babypwn

babypwn: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), dynamically linked,
     interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 3.2.0, BuildID[sha1]=
     aceea8523337cd304e3835a461e68c809d12fc01, not stripped
```

- **strings**: find strings in binary files
- **file**: metadata for files, e.g. architecture, endianess, linker
- **netcat**: swiss armyknife for networking
- **readelf**: inspect ELF metadata of file
- **Disassembler**
- **Decompiler**

Disassembler: objdump

```
$ objdump -M intel -d ./babypwn
```

```
[...]
0000000000401169 <main>:
  401169: 55                              push   rbp
  40116a: 48 89 e5                        mov    rbp,rsp
  40116d: 48 8d 3d 9c 0e 00 00            lea    rdi,[rip+0xe9c]
  401174: e8 b7 fe ff ff                  call   401030 <puts@plt>
  401179: 48 8b 05 90 2e 00 00            mov    rax,QWORD PTR [rip+0x2e90]
  401180: 48 89 c7                        mov    rdi,rax
  401183: e8 b8 fe ff ff                  call   401040 <fflush@plt>
  401188: b8 00 00 00 00                  mov    eax,0x0
  40118d: e8 b4 ff ff ff                  call   401146 <copy>
  401192: b8 00 00 00 00                  mov    eax,0x0
  401197: 5d                              pop    rbp
  401198: c3                              ret
[...]
```

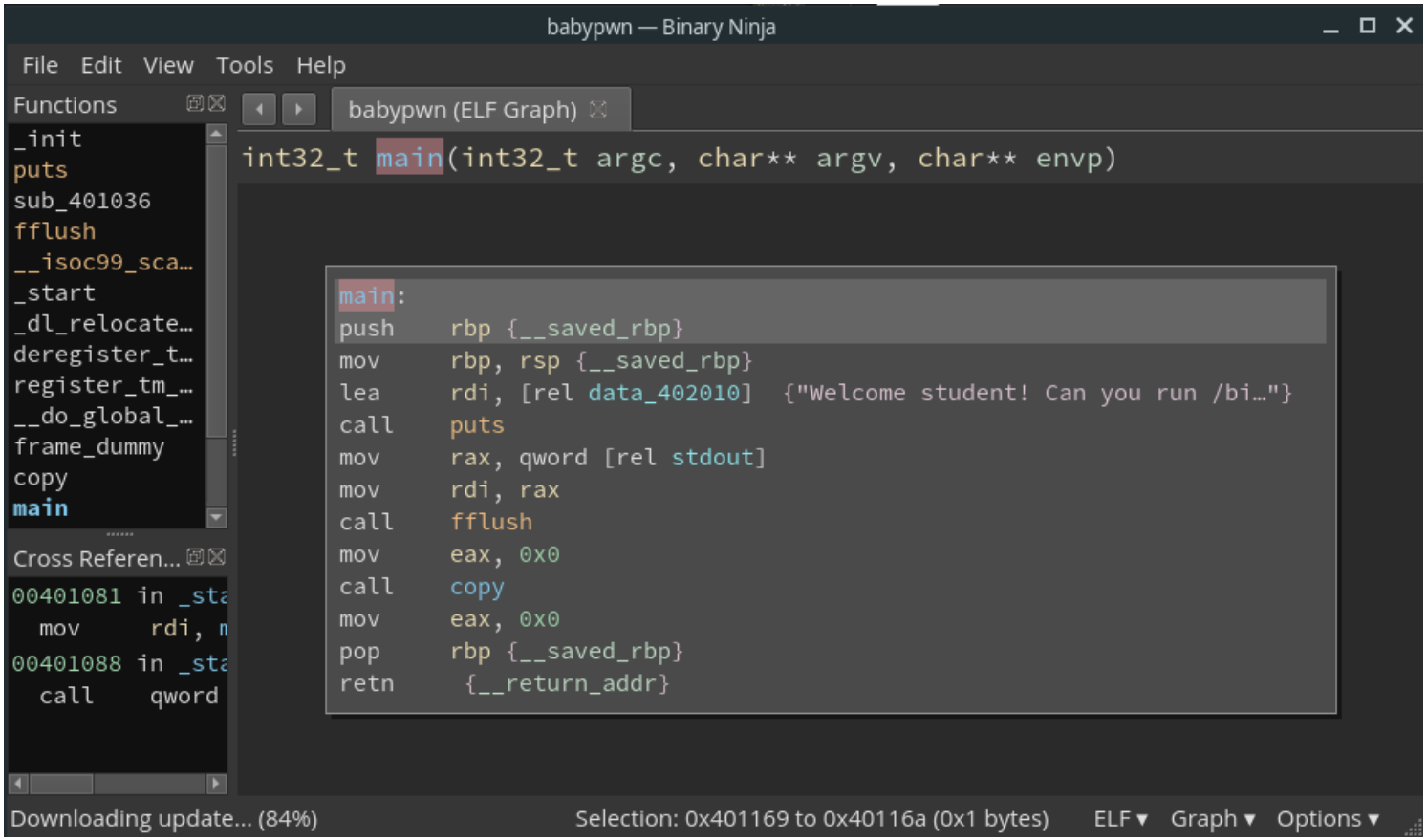
Disassembler: radare2

```
[0x00401169]> pdf

/(fcn) main 48
|     int main (int argc, char **argv, char **envp);
|       ; DATA XREF from entry0 @ 0x401081
|0x00401169   55              push rbp
|0x0040116a   4889e5          mov rbp, rsp
|0x0040116d   488d3d9c0e00.  lea rdi, str.Welcome...
|0x00401174   e8b7feffff      call sym.imp.puts              ; int puts(const char *s)
|0x00401179   488b05902e00.  mov rax, qword [obj.stdout]   ; rdi
|                                                           ; [0x404010:8]=0
|0x00401180   4889c7          mov rdi, rax                   ; FILE *stream
|0x00401183   e8b8feffff      call sym.imp.fflush            ; int fflush(FILE *stream)
|0x00401188   b800000000      mov eax, 0
|0x0040118d   e8b4ffffff      call sym.copy
|0x00401192   b800000000      mov eax, 0
|0x00401197   5d              pop rbp
\0x00401198   c3              ret
```

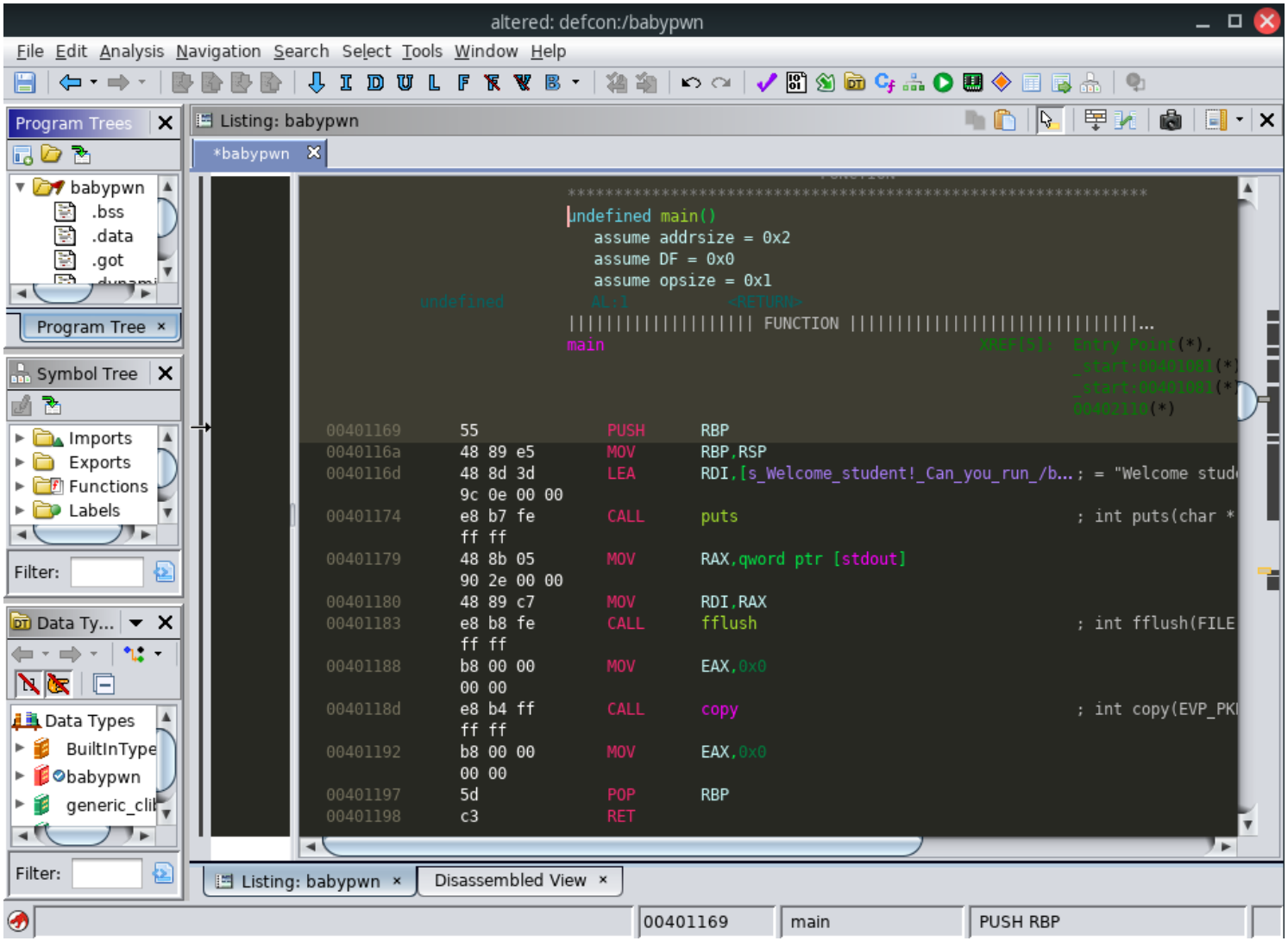
Disassembler: GUI

Binary Ninja



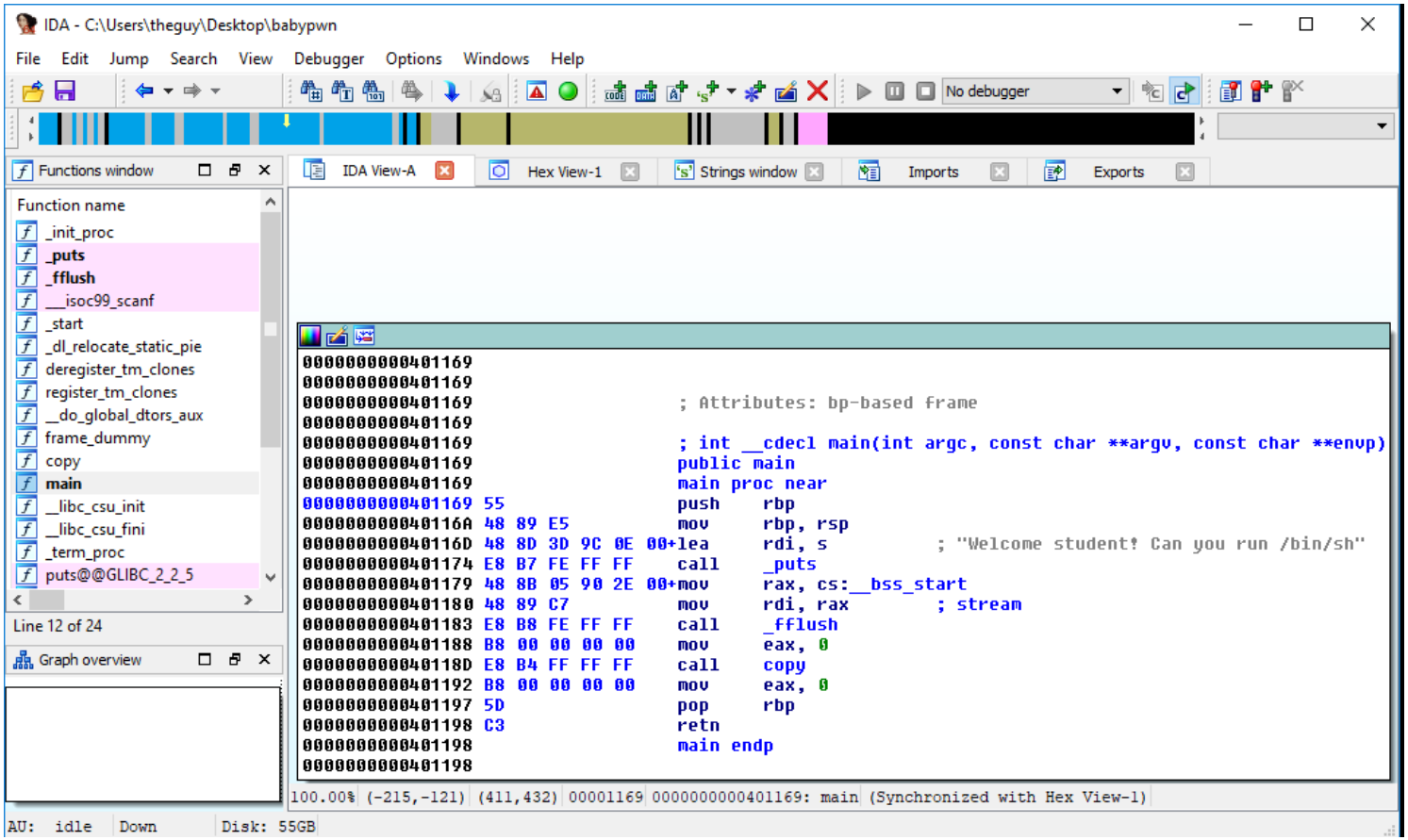
Disassembler: GUI

Ghidra



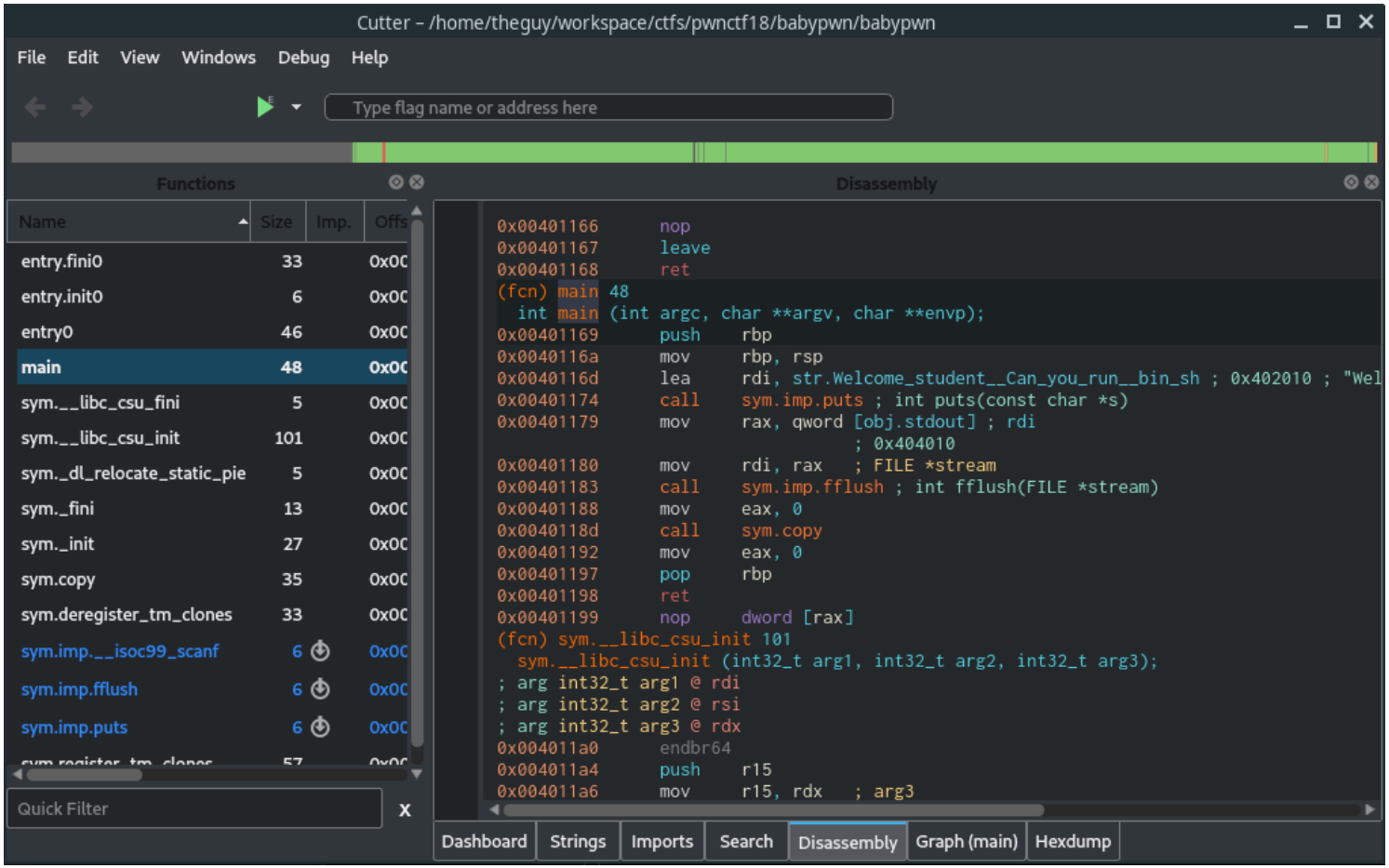
Disassembler: GUI

IDA Pro



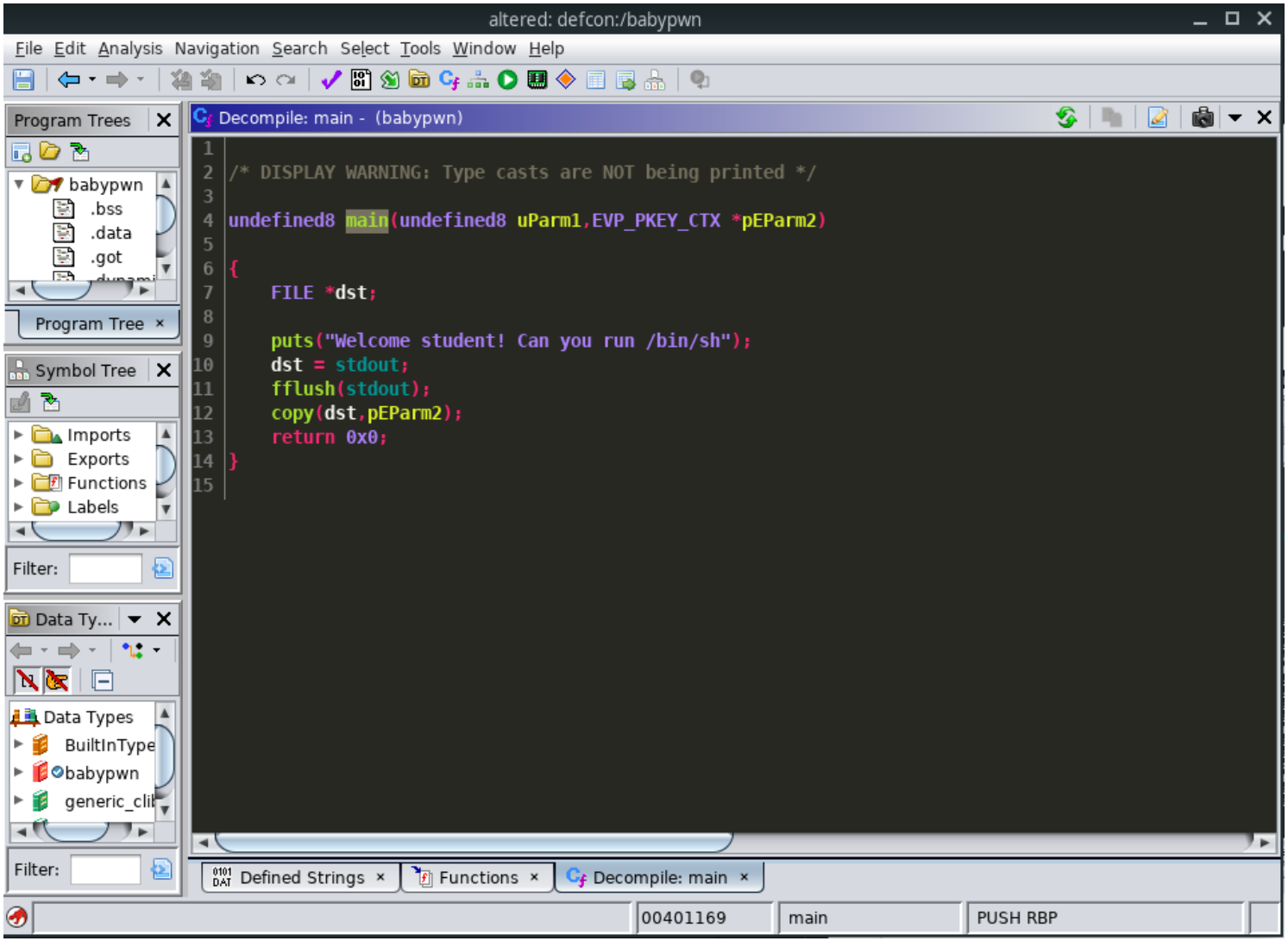
Disassembler: GUI

Radare2/Cutter



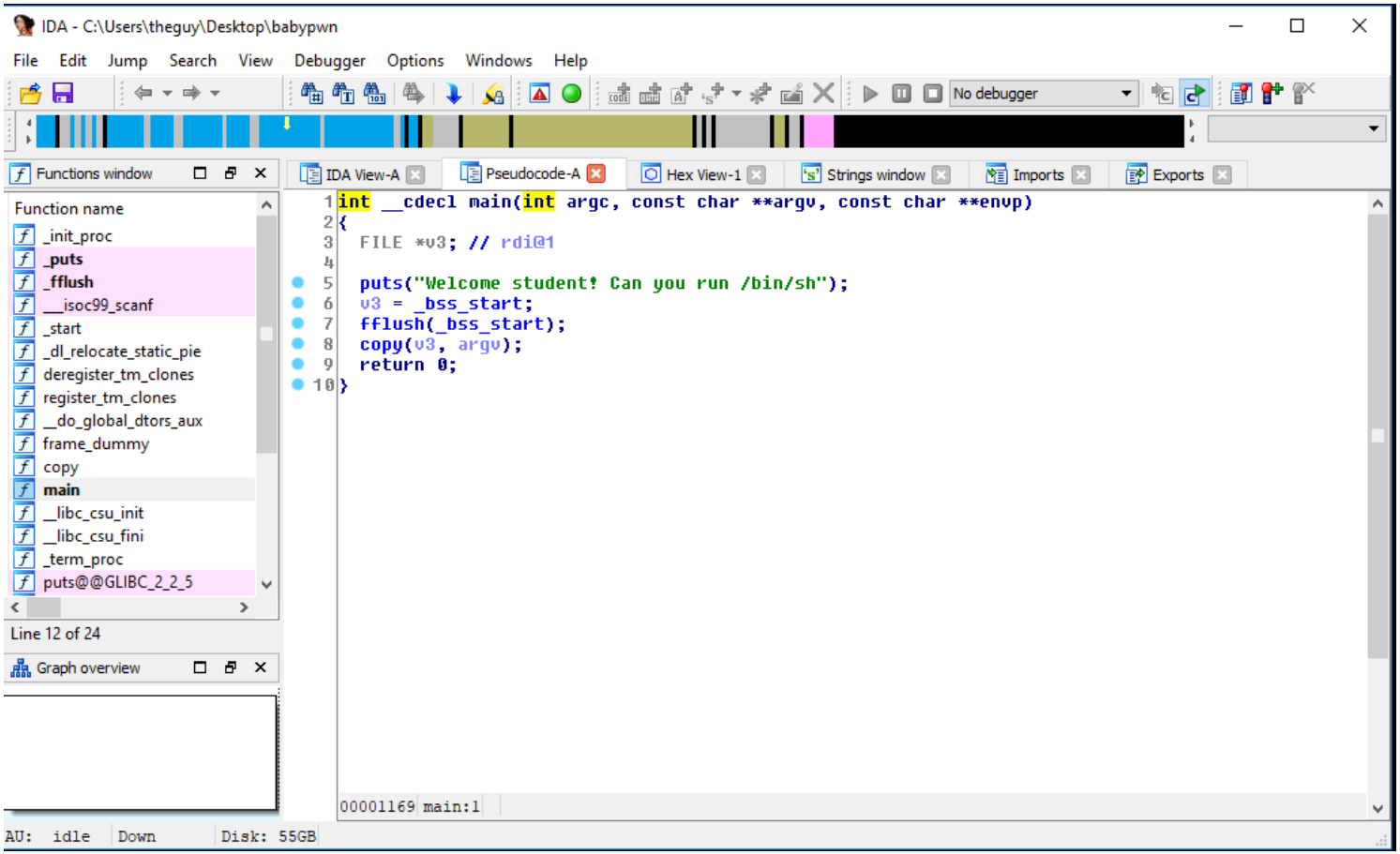
Decompiler: GUI

Ghidra



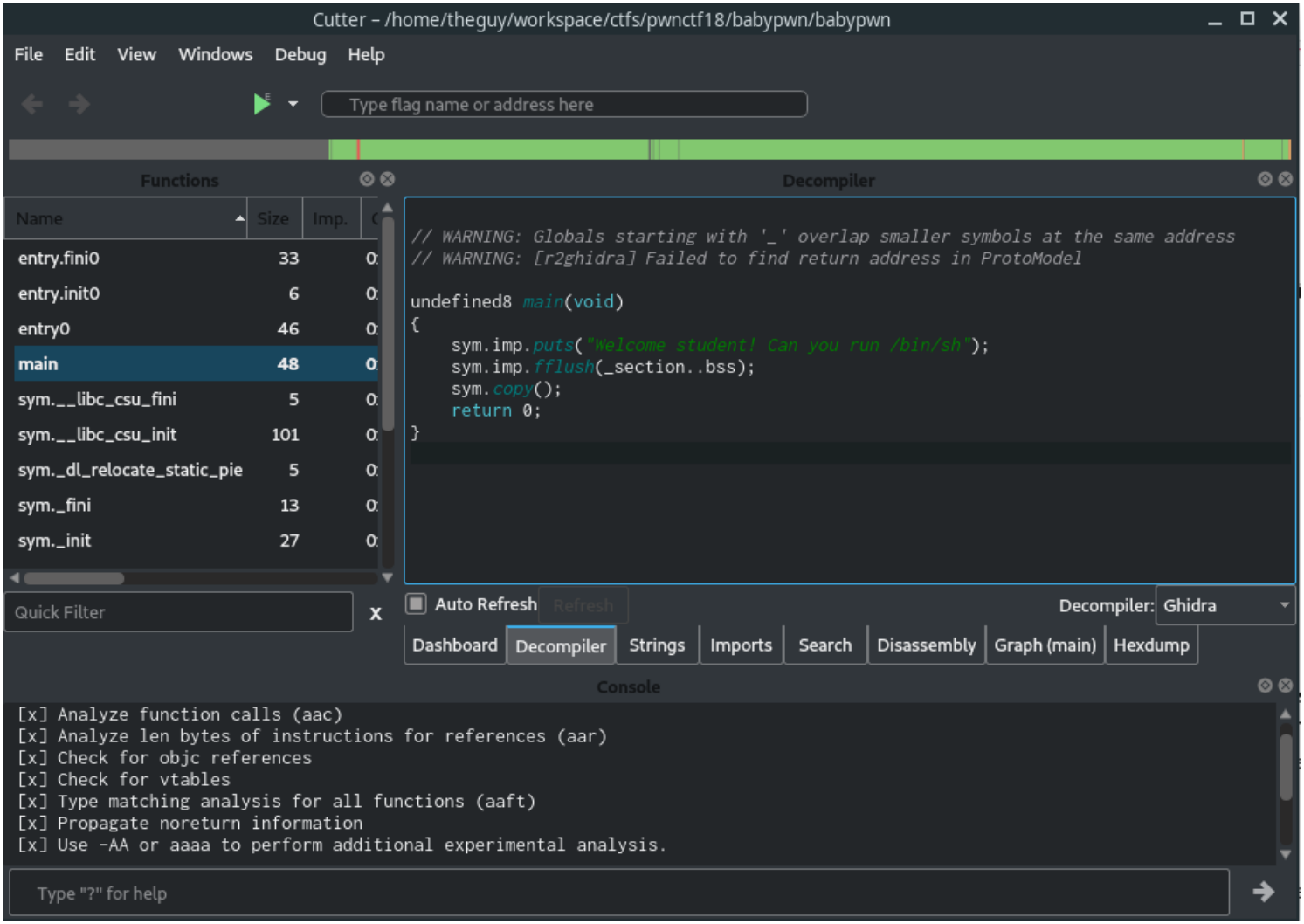
Decompiler: GUI

IDA Pro



Decompiler: GUI

Radare2/Cutter

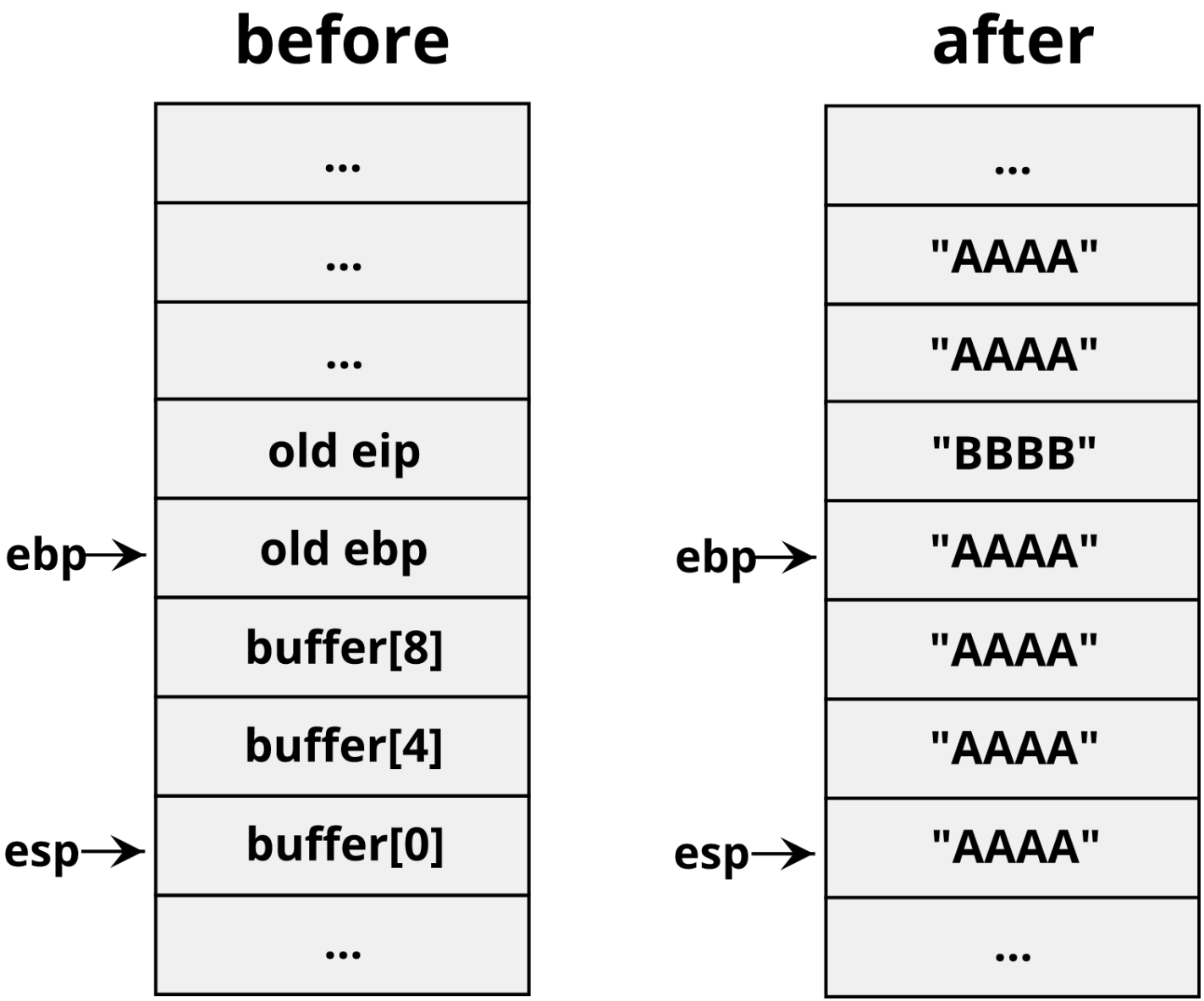


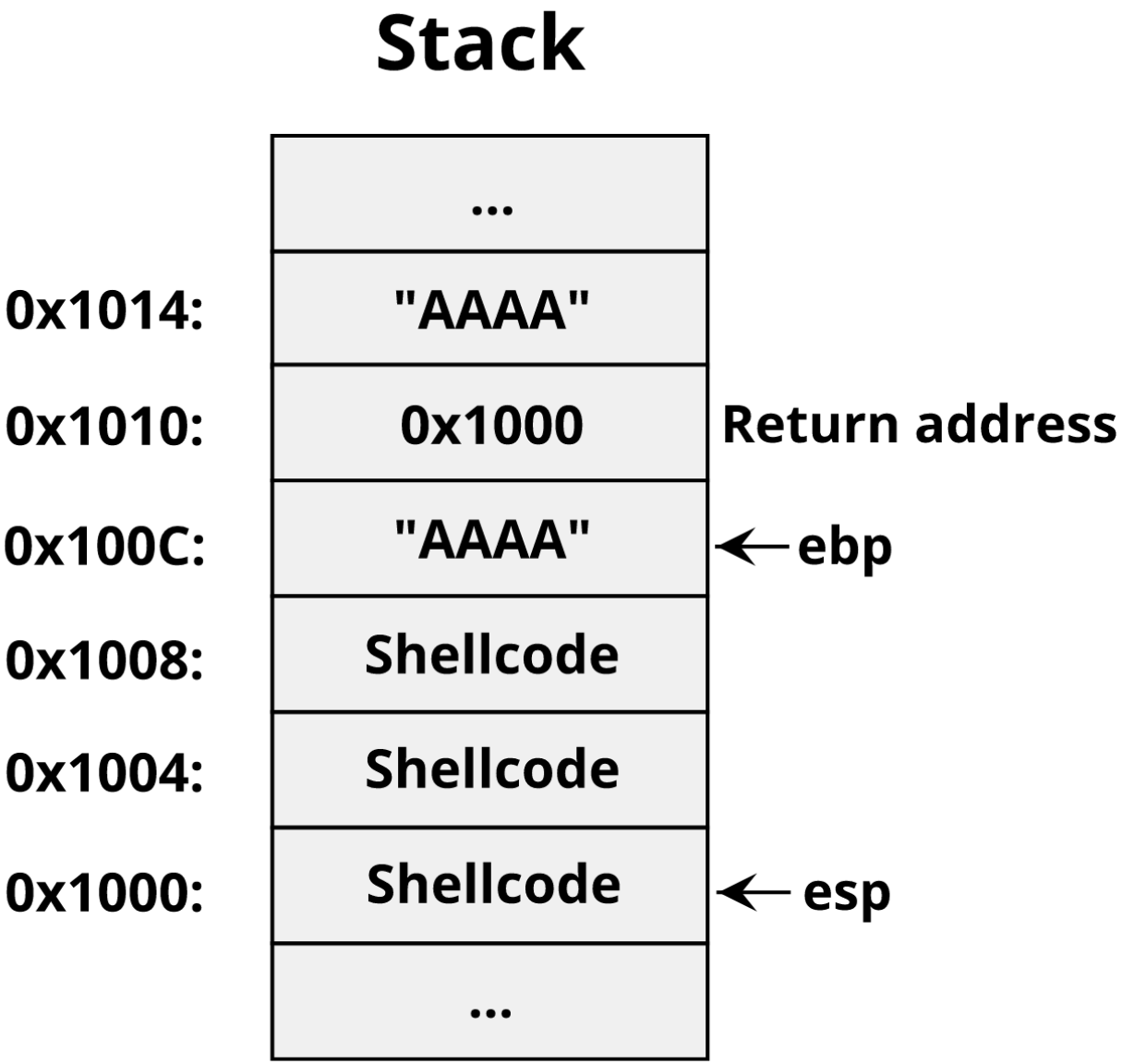
Debugger: gdb

Debugger: gdb

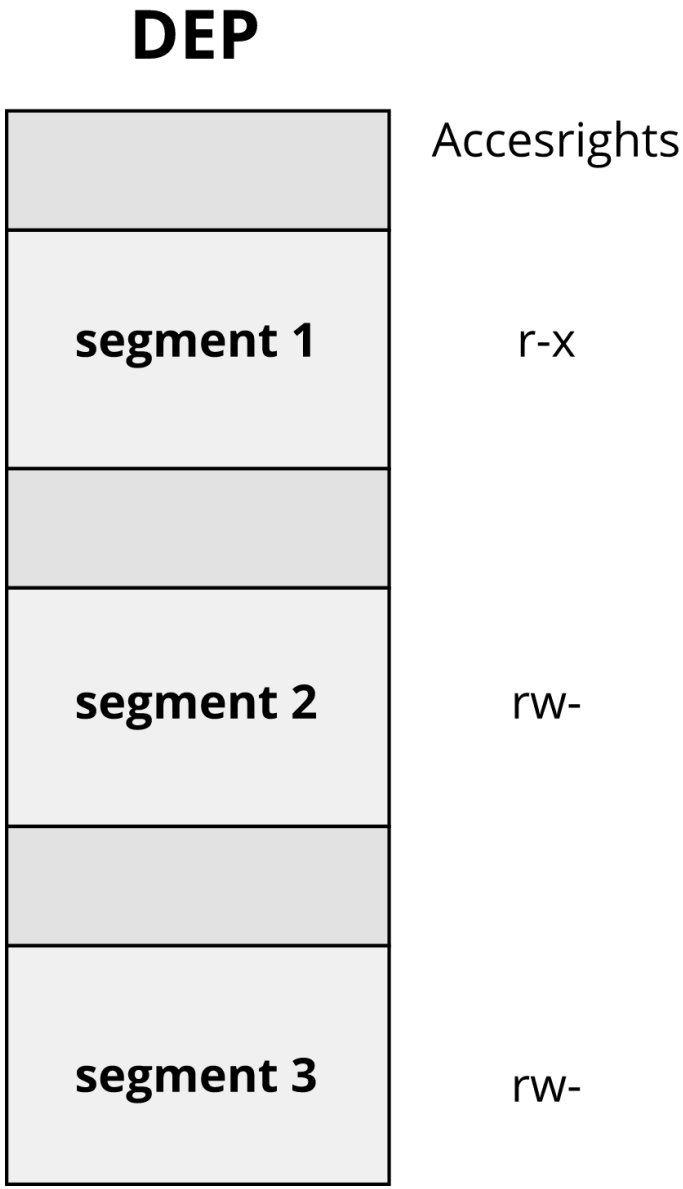
- Setting breakpoints: `break ADDR`
- Executing: `run`, `nexti`, `stepi`, `continue`, `finish`
- Examination: `disassemble`, `x/FMT`, `print EXPR`
- Metadata: `info EXPR`
- Help: `help CMD`
- GEF: `vmmap`, `xinfo`, `aslr`, `...`

Vulnerabilities: Buffer Overflows

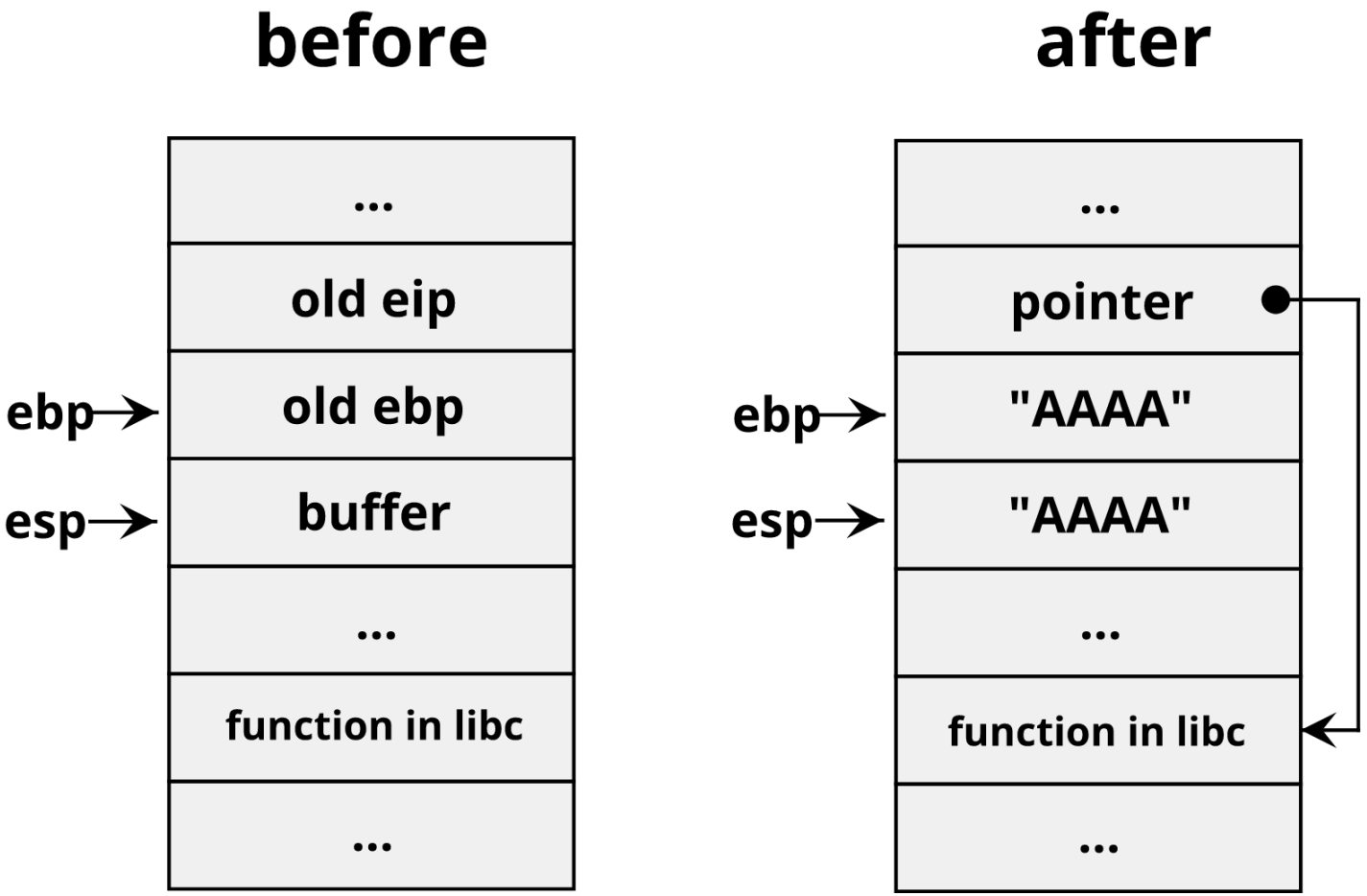



Stack
...
0x1014:
"AAAA"
0x1010:
0x1000
Return address
0x100C:
"AAAA"
ebp
0x1008:
Shellcode
0x1004:
Shellcode
0x1000:
Shellcode
esp
...

Mitigation: Data Execution Prevention



Vulnerabilities: Ret2Libc

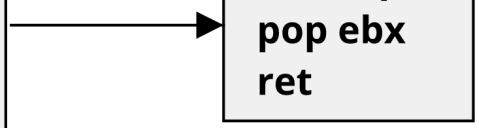


on **ret** the cpu executes the code at the specified libc function

Vulnerabilities: ROP Gadgets

complete function

```
080486f4 <_fini>:
 80486f4:   push   ebx
 80486f5:   sub    esp, 0x8
 80486f8:   call   80484b0
 80486fd:   add    ebx, 0x1903
 8048703:   add    esp, 0x8
 8048706:   pop    ebx
 8048707:   ret
```



Gadget

```
add esp, 8
pop ebx
ret
```

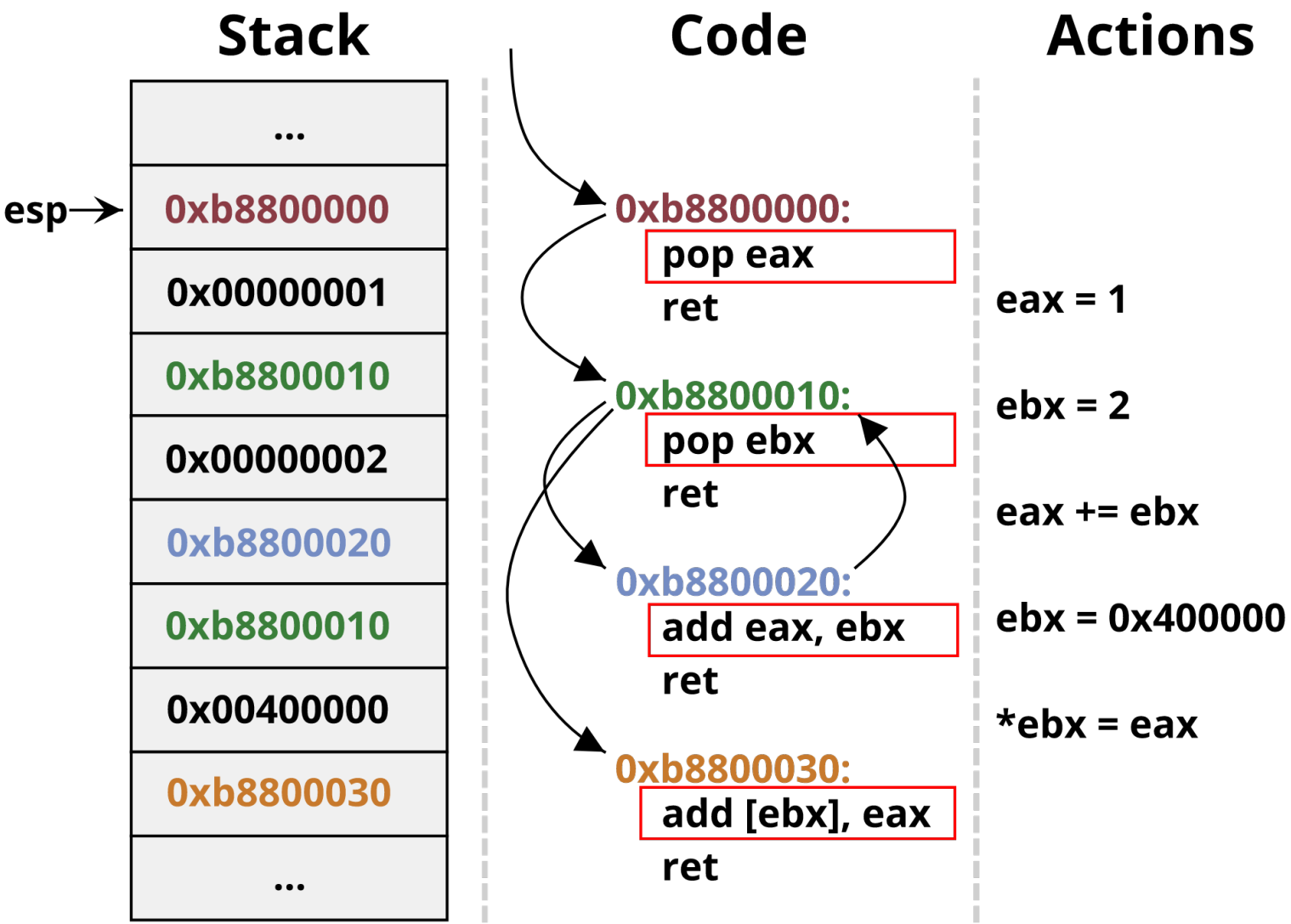
More Gadgets

```
inc ecx
ret
```

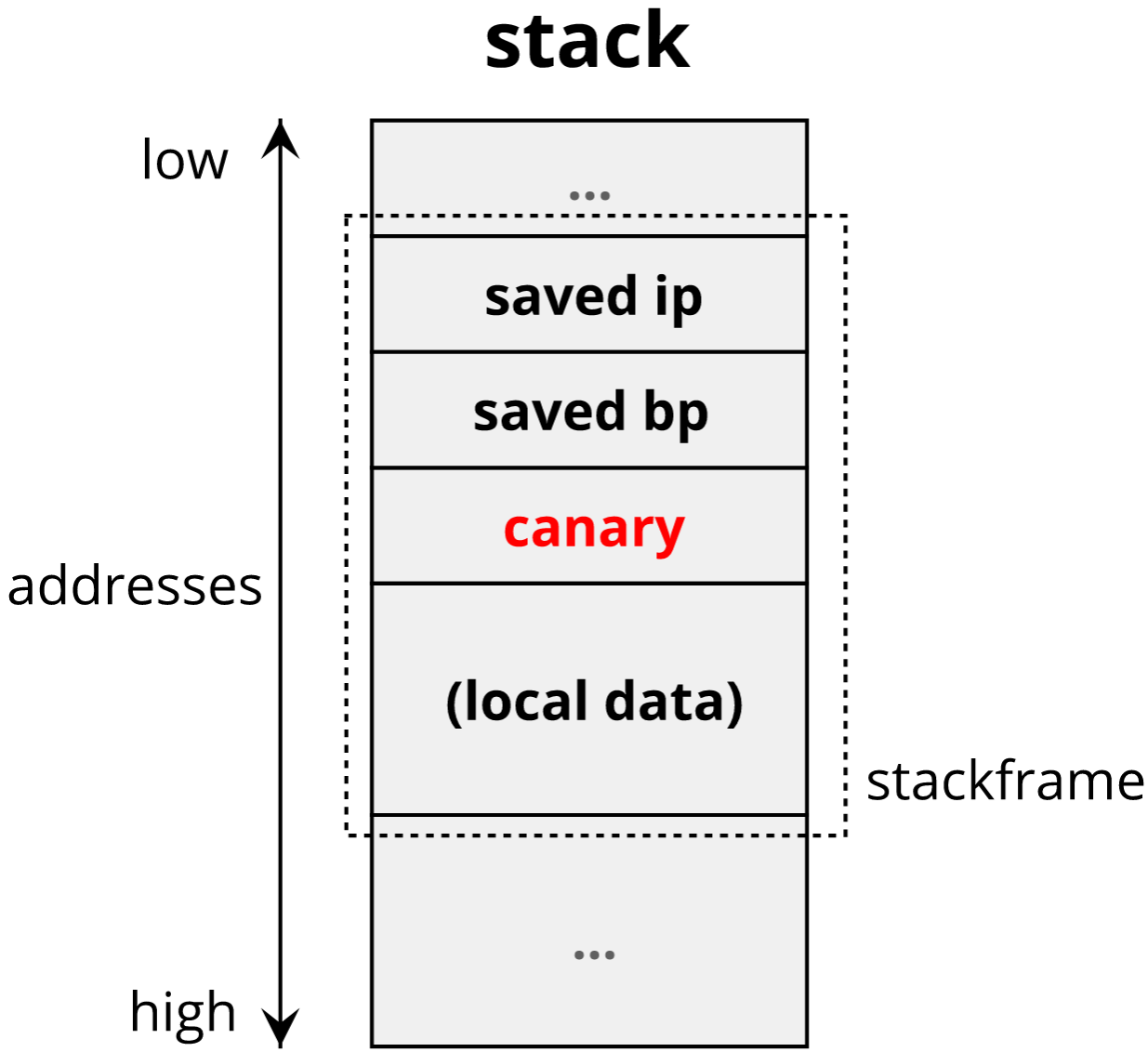
```
jmp [ebx]
ret
```

```
mov ebx, dword ptr [esp]
ret
```

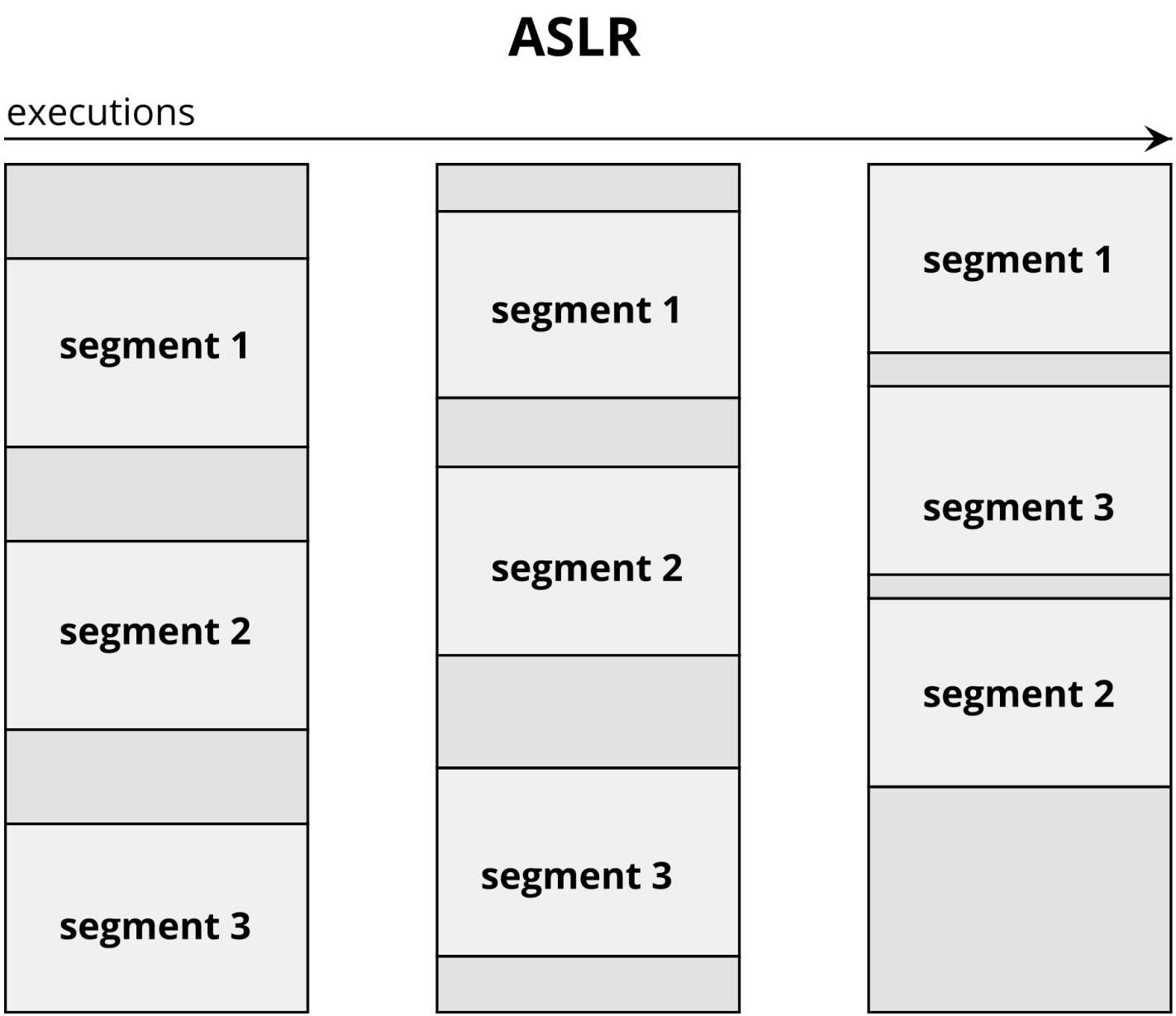
Vulnerabilities: ROP



Mitigations: Canaries



Mitigations: ASLR



More Tools

- Pwntools
- Ropper
- One_Gadget
- Checksec (included in Pwntools)

```
$ checksec ./babypwn

[*] '/absolute/path/to/babypwn'
    Arch:     amd64-64-little
    RELRO:    Full RELRO
    Stack:    No canary found
    NX:       NX enabled
    PIE:      No PIE (0x400000)
```

Pwntools

```
from pwn import *

# set up context of the executable
exe = context.binary = ELF('/path/to/executable')

io = process(exe.path)      # spawn process

io.recvline()               # receive one line from process
io.recvlines(2)             # receive two lines from process
io.recv(1024)               # receive up to 1024 bytes
io.recvuntil('keyword')     # receive until 'keyword'
io.recvall()                # receive everything

io.sendline('some input')   # send this line
io.send('some input')       # send this (no newline)
io.sendafter('wait for this',
    'then send this\n')     # combination of recvuntil and send

io.interactive()            # connect to process stdin, stderr and stdout

exe.sym['main']             # returns address of 'main' symbol
exe.got                     # returns GOT as list
exe.address                 # returns base address of executable

# examples for other useful functions
cyclic(100)                 # create de bruijn sequence
cyclic_find(sequence)       # find offset in de bruijn sequence
p64(value)                  # pack int as 64bit
u64(value)                  # unpack 64bit value to int
```

Ropper

`ropper -f /usr/lib/libc.so.6 --console`

```
(libc.so.6/ELF/x86_64)> type rop
[INFO] Load gadgets for section: LOAD
[LOAD] loading... 100%
[LOAD] removing double gadgets... 100%
(libc.so.6/ELF/x86_64)> search /1/ pop rax
[INFO] Searching for gadgets: pop rax

[INFO] File: /usr/lib/libc.so.6
0x0000000000122e94: pop rax; ret 0xfffe;
0x0000000000122e24: pop rax; ret 9;
0x000000000003f080: pop rax; ret;

(libc.so.6/ELF/x86_64)> search /1/ syscall
[INFO] Searching for gadgets: syscall

[INFO] File: /usr/lib/libc.so.6
0x000000000003f289: syscall; ret;

(libc.so.6/ELF/x86_64)>
```

OneGadget

`one_gadget /usr/lib/libc.so.6`

```
0xcd3aa execve("/bin/sh", r12, r13)
constraints:
  [r12] == NULL || r12 == NULL
  [r13] == NULL || r13 == NULL

0xcd3ad execve("/bin/sh", r12, rdx)
constraints:
  [r12] == NULL || r12 == NULL
  [rdx] == NULL || rdx == NULL

0xcd3b0 execve("/bin/sh", rsi, rdx)
constraints:
  [rsi] == NULL || rsi == NULL
  [rdx] == NULL || rdx == NULL

0xeafab execve("/bin/sh", rsp+0x60, environ)
constraints:
  [rsp+0x60] == NULL
```

Vielen Dank … **Fragen**?