Introduction

بسم الله الرحمن الرحيم السلام عليكم اليوم بتكلم عن ال direct syscalls

في البداية نظام الويندوز يتعامل مع أشياء كثير مثل الميموري و الاتصالات اللي تسويها كل بروسيس و أيضا عندنا الملفات و كيف يتم تغيررها و حذفها و كل هذي الأشياء ويندوز يسويها عنا باستخدام شي اسمة system calls

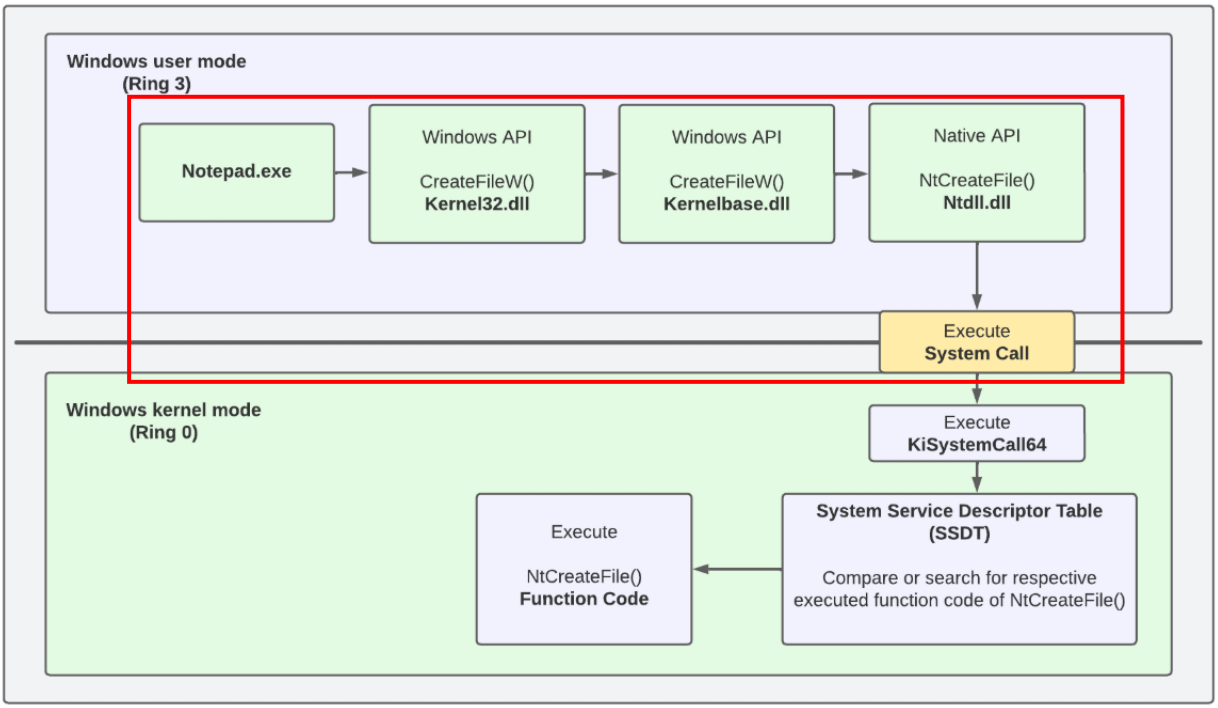
Simple code

عشان يكون اسهل للمبرمج و افضل لنظام بمعنى ان النظام بيسوي عنك أشياء كثير ف يصير افضل و اكثر كفاءة خلونا ناخذ مثال عشان توضح الفكرة

انا ابي اسوي ملف اسمه test الملف هذا لما ابي اسويه بينكتب على الdisk طيب كيف ينكتب؟ كم بياخذ بايت وشي ال attribute اللي بتكون بالملف هذي الأشياء كلها يسوي ويندوز حنا بس نقول له نبي نسوي مثل ناخذ مثال كود C

```c
#include <windows.h>
#include <stdio.h>
int main() {
    HANDLE hFile = CreateFile(
        "test.txt",           // File name
        GENERIC_WRITE,          // Write access
        0,                      // No sharing
        NULL,                   // Default security
        CREATE_ALWAYS,          // Overwrite if exists
        FILE_ATTRIBUTE_NORMAL,  // Normal attributes
        NULL                    // No template
    );
    if (hFile == INVALID_HANDLE_VALUE) {
        printf("Failed to create file. Error code: %lu\n", GetLastError());
        return 1;
    }
    printf("File created successfully!\n");
    // Write data to the file (optional)
    const char *data = "Hello, Windows!";
    DWORD bytesWritten;
    WriteFile(hFile, data, strlen(data), &bytesWritten, NULL);
    // Close the file handle
    CloseHandle(hFile);
    return 0;
}
```

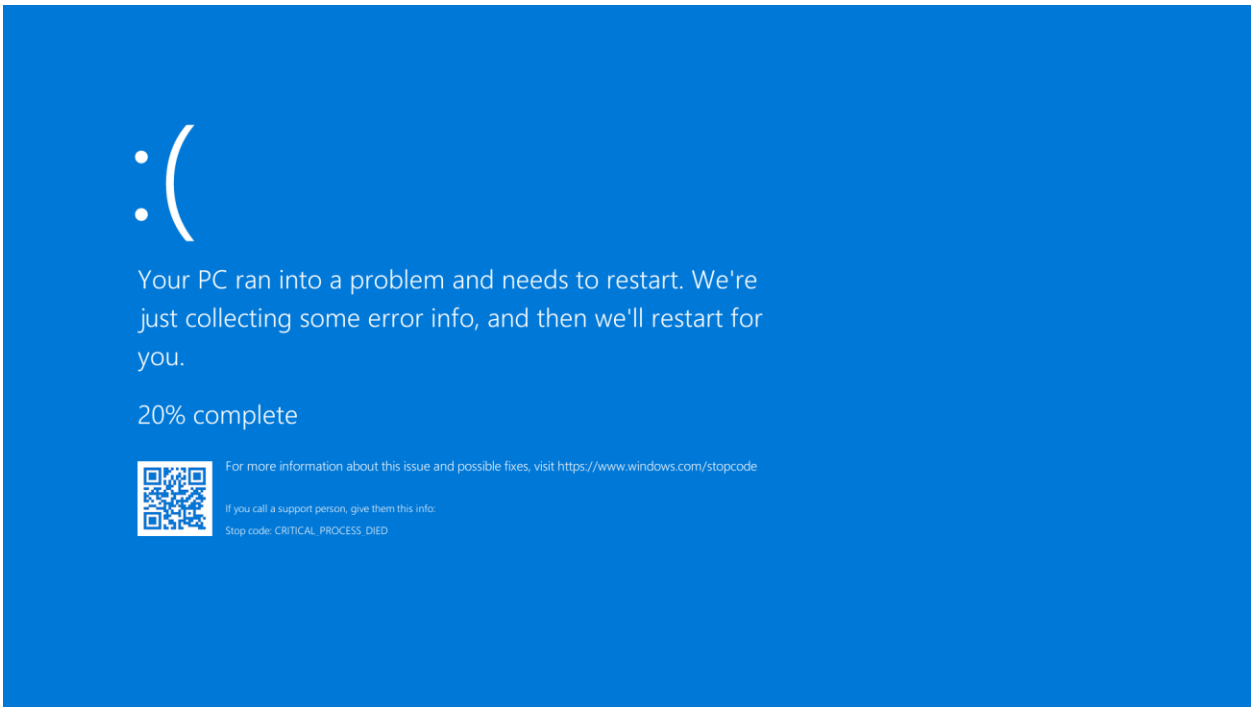
هذا الكود بيسوي ملف اسمة test.txt زي ما نشوف فيه أشياء كثير ياخذها غير اسم الملف هذي كلها مع اسم الملف تروح لنظام ياخذها و في شي اسمه kernel-mode و يكتب الملف بعدين يرجع لك قيمة حسب القيمة ذي تعرف الملف انكتب او صار خطأ الى الان ما وصلنا ل syscall هذا فقط كود بسيط مكتوب ب C يستخدم windows api اسمه CreateFile عشان يكتب الملف حقنا الفنكشن هذي CreateFile فيه أشياء تصير تحتها ناخذها حبه حبه الحين



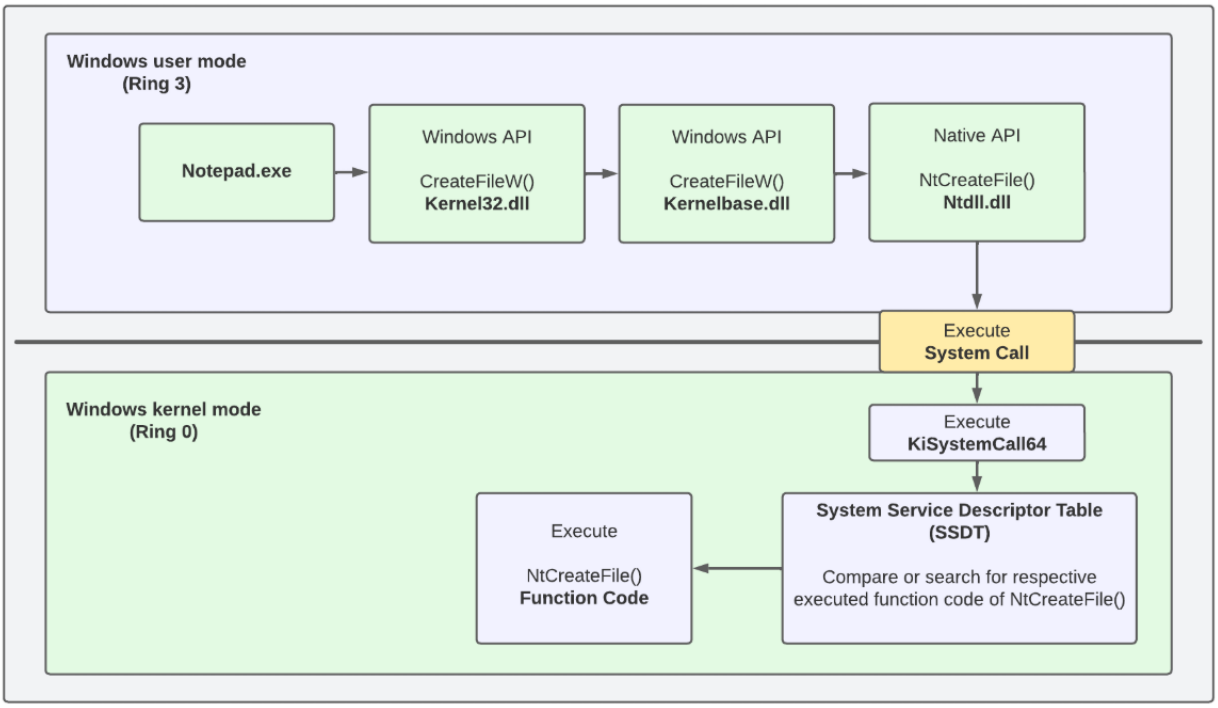
لما شغلنا فنكشن CreateFile الفنكشن هذي استدعت فنكشن ثانية بنفس الاسم بس في dll اخر اسمها CreateFile الفنكشن هذي استدعت فنكشن أخرى اسمها NtCreateFile اختصار ل Native Create File بعدين بنلاحظ فيه System call باقي الصورة ما نحتاجة حاليا بس اللي يهمنا ال path اللي اخذتة الفنكشن لين وصلت ل system call نلاحظ انها مرت على 2 فنكشن غير حقتنا اللي سويناها ب C

Why system call

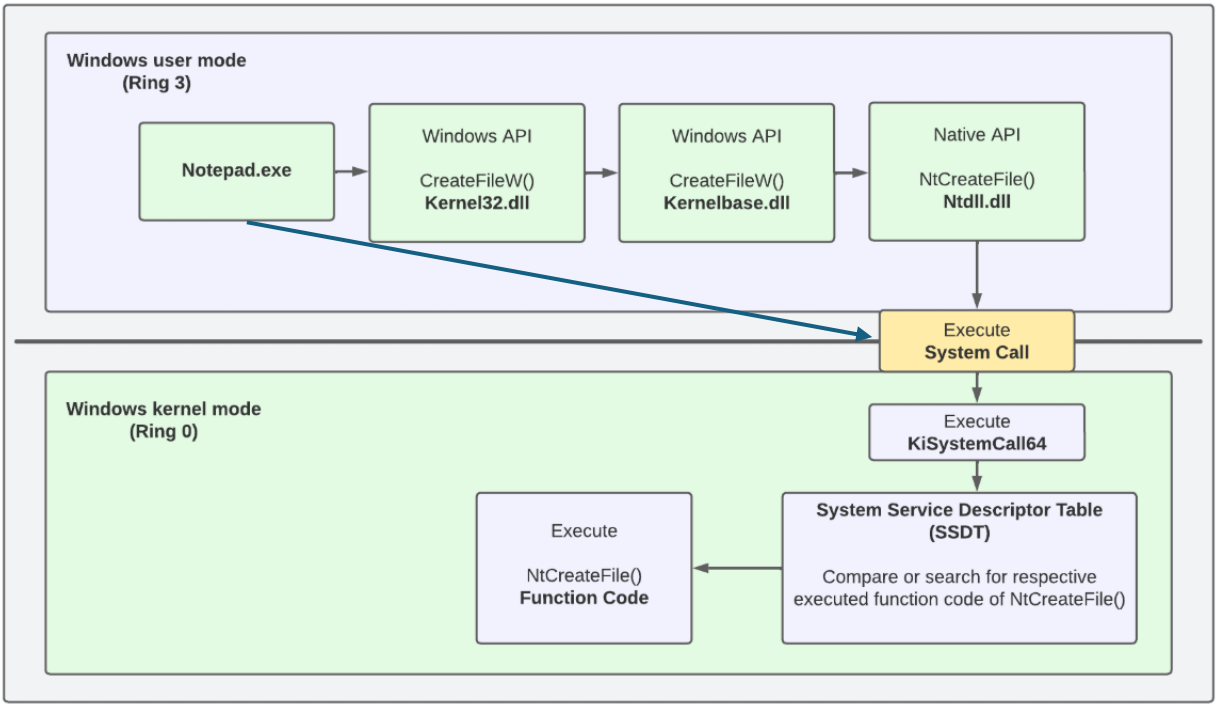
عندنا بالنظام شي اسمة user-mode و kernel-mode أي خطأ يصير داخل user-mode يعتبر شي عادي و مثل أي bug عادي مثلا بيقول لك الملف هذا موجود مقدر اسوي ثاني و لكن اذا صار غلط في kernel-mode ف يصير blue-screen لنظام



هنا تجي فايدة ال system calls هي مثل الجسر بين user-mode و kernel-mode بحيث تفصلهم عن بعض ف هذا الشي منع ال blue screen بشكل كبير



نرجع لصورتنا هل اقدر اخلي البرنامج حقي ما يروح للفنكشنز كلها هذي ابيه يسوي علطول system call؟ ايه نقدر و هنا جا شي اسمه direct system call



هنا البروسيس حقتنا علطول بتسوي direct system call و ما راح تمر الفنكشنز اللي فوق كلها و هنا نستفيد انه تتخطى ال hooking

الحين عرفنا وش يعني system call اقدر اني اسوي ملف او اسوي بروسيس او أي فنكشن تسوي system call عشان تروح للkernel تعالج الطلب اللي نرسلة بعدين تسوي الملف حقنا او تسوي البروسيس حقتنا طبعا مفهوم ال system call موجود باللينكس و قريب جدا من الويندوز

```
00007FFDB3F8D3B0    4C:8BD1          mov r10,rcx                    ZwAllocateVirtualMemory
00007FFDB3F8D3B3    B8 18000000      mov eax,18
00007FFDB3F8D3B8    F60425 0803FE7F 01 test byte ptr ds:[7FFE0308],1
00007FFDB3F8D3C0    75 03            jne ntdll.7FFDB3F8D3C5
00007FFDB3F8D3C2    0F05             syscall
00007FFDB3F8D3C4    C3               ret
00007FFDB3F8D3C5    CD 2E            int 2E
00007FFDB3F8D3C7    C3               ret
00007FFDB3F8D3C8    0F1F8400 00000000 nop dword ptr ds:[rax+rax],eax
```

هنا صورة تبين ال system call من dbg طبعا استخدم فنكشن AllocateVirtualMemory عشان يطلب من الكيرنل تعطيه مساحة من الرام زيادة عشان يحط الكود حقه سواء كان shellcode او صورة او أي شي

System calls vs win apis hooking

وش يعني hooking

باختصار هو كيف تغير الفنكشن هذي كيف تشتغل مثال بسيط:

```
const eval = (code)=>{
    console.log("Hooked!")
}

eval("Alert(1)")
```

هذا كود hookباستخدام js طبعا على مستوى الويندوز process الموضوع اعقد بكثير بس الحين عشان نفهم الفكرة سويت فنكشن تسوي hook ل eval بحيث أي استخدام ل eval يطبع لي Hooked!

```
PS C:\Users\Admin\Desktop> node .\main.js
Hooked!
```

لو اشغل الكود بيطبع لي Hooked! نفس الشي ال AV & EDRs يسوونه بس على مستوى windows api اذا شافك تسوي أي شي غلط يسوي له block و اذا استخدمنا direct system call بنسوي تخطي للفنكشنز هذي ف ماراح يعرف وش سوينا ناخذ مثال بسيط:

```
const print = (value)=>{
    console.log(value)
}
const check_evil = (code) =>{
    if(code == "Evil"){
        console.log("Blocked")
    }
    else{
        print(code)
    }
}
check_evil("kira") // kira
check_evil("Evil") // Blocked
print("Evil") // Evil
```

هنا كود أحاول فيه تبسيط فكرة ال system call عندنا check_evil بيشوف القيمة حقتنا قبل ما يرسلها ل print هذا مثل ال hook بيشوف وش نسوي و اذا شاف Evil بيعطينا بلوك بس لو كلمنا print علطول بيطبع لي أي شي ابيه مثل ال direct system call يصير ما فيه أي حماية تشوف وش اسوي غير جهه الkernel و هذا ببساطة كيف تقدر تسوي direct system callعشان تتخطى ال hook

Reference

https://redops.at/en/blog/direct-syscalls-a-journey-from-high-to-low

https://redops.at/en/blog/direct-syscalls-vs-indirect-syscalls

https://www.ired.team/offensive-security/defense-evasion/using-syscalls-directly-from-visual-studio-to-bypass-avs-edrs