一、 前言

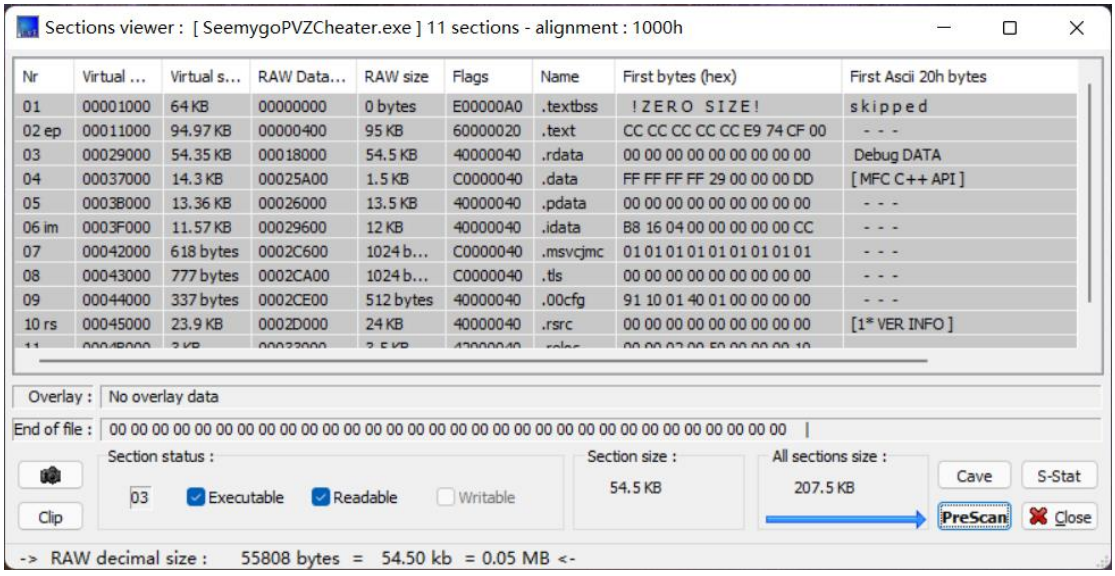
这篇文章主要目的是对 VMP 壳主要特性有初步的了解，掌握 VMProtect3.5 软件正确有效的使用方法，并以一个具体案例来演示，演示所使用的版本为 VMProtect3.5 已注册版本，授权已经过期但保护效果依旧没有过时，官网也才是 3.51 补丁版本，文章演示所用版本将会在文章末尾附上下载链接。

二、 VMProtect 浅析

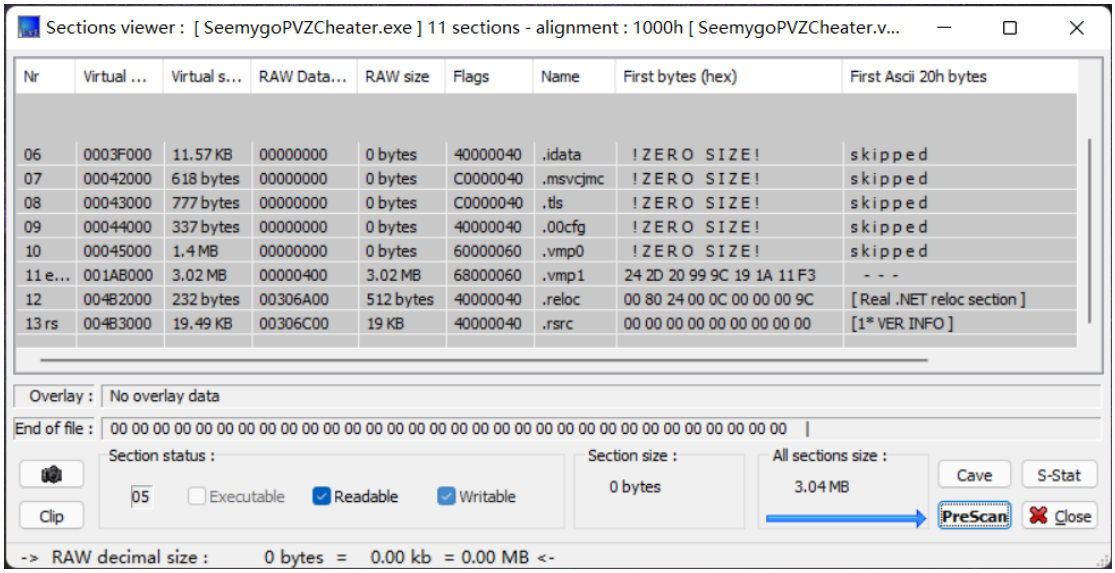
1.打包（压缩/加密）

压缩/加密可执行文件的代码部分可以防止被静态分析，这是很常见的保护手法。考虑到被压缩/加密代码在运行过程终究会被还原成正常的未加密的代码，所以会在执行期间某个时间点被转储，并导出修复，修复的程序跟源程序基本一样，所以可以说，打包只是增加了破解的门槛，并没有增加太多时间成本，起不到有效遏制破解的作用。

在未使用 VMProtect 打包之前可以看到：区段结构清楚，没有异常



使用 VMProtect 打包之后：对比明显，重要区段内容清空，并多出了两个区段 vmp0 与 vmp1，并清理可能对破解者有帮助的区段信息



而且在这种情况下，VMProtect 不会在 PE 头中存储真正的“RawFile”部分信息。但是虚拟地址和大小依旧存储在头部数据中，以便让系统可以为可执行文件分配正确大小内存空间，因此我们依旧可以读到区段的大小和可能的地址（不考虑 ASLR，也就是地址空间布局随机化）。

关于源文件的打包，你可以很容易发现区段.vmp1 占据了很大的空间，这是因为 VMProtect 将大部分程序内容加密整合写入了.vmp1 区段，其中包含已经被加密的代码，解包的程序以及其他信息（这里你会发现还有一个.vmp0 区段，这里我们只是演示打包操作，这个区段在虚拟化的时候会使用到）。

唯一不受“保护”的部分是 .rsrc 区段，因为.rsrc 区段是一个可执行程序的资源区段，你的程序相关图标作者语言等信息都存储在其中，若是被直接加密，那一定会显得很异常。 所以 VMProtect 有一个选项来保护图片作者等相关资源，它将资源分成两部分，一个可用于 Windows 操作系统提取使用，还有一个只包含其他信息，并在程序执行过程中被解密，以供程序运行时使用。

需要注意的是：如果选中 VMProtect 的压缩选项，.vmp1 区段将包含 VMProtect 的句柄和变异代码。反之，则所有内容都在 .vmp0 区段 中。

我在做相关案例时观察到，一般除.data 区段，其他大多数区段都是不可写入的。所以有个疑问，VMProtect 是如何进行写入的呢？所以 VMProtect 得调用 ZwProtectVirtualMemory 这个未公布函数来修改部分区段的属性以来修改区段中的内容，而相对应的从破解者的视角来看，这个函数就显得尤为重要，可以通过针对这个函数下断点的方式监控 VMProtect 解包过程中该函数的使用情况，并进行动态解包，通过转储工具导出，当然这一部分并不是我说的如此简略的而且这不是文章的主要目的，所以不再做具体叙述。

2.代码变异与虚拟

可以说 VMProtect 最核心最有威慑力的就是它的虚拟引擎了，但是大佬认为

它的虚拟引擎并不是最厉害的，但是摸清楚它的虚拟引擎太费时间，让人望而却步，所以在这里并不会特别的去关注它，在这里就简单了解一下虚拟引擎的原理就可以了，在这一段我们主要了解代码变异，来看看 VMProtect 的代码变异一些特性……

首先 VMProtect 的虚拟功能是基于虚拟机（VM）的代码虚拟化，成为了实现代码混淆的一种有效方法。基于 VM 的保护的基本原理是用攻击者不熟悉的虚拟指令替换程序指令。然后这些虚拟指令将在运行时转换为本机代码，在底层硬件平台上执行。使用基于虚拟机的方案，模糊代码的执行路径由虚拟指令调度器控制。一个典型的调度程序由两个组件组成：一个用于确定哪条指令准备好执行的调度程序，以及一组字节码处理程序，它们首先对字节码进行解码，然后将其转换为本机代码。这个过程用定制字节码替换了原始的程序指令，允许开发人员隐藏敏感代码区域的用途或逻辑。

简单介绍完了虚拟，来了解一下本段最主要的内容代码变异，代码变异是以用户标注的函数为组成单位，加密代码部分中的每个变异函数都跳转到包含下一个要执行的代码的 VMProtect 生成的部分。因为在不破坏整个代码结构的情况下修改（插入）编译器生成的原始代码段是不可行的。因此 VMProtect 将所有变异函数存储在一个新区段中以避免大小问题(请谷歌:在 PE 头中插入代码)。

VMProtect 变异是很多小技巧的组合使用，例如代码变形等义替换、垃圾代码（以前叫花指令）插入、控制流跳转和块不对齐。

在我还在高中的时候，研究 VMProtect 时就很清晰的发现，VMProtect 的代码变异并不会对原始指令进行过多的变化，貌似喜欢在标定的变异函数插入垃圾指令，倒是自从上了大学，就纯学 Java 了，所以到现在为止都没有什么较大发现，但我在写这篇文章的时候已经在重拾之前的研究了，现在是 VMProtect3.5x 版本了，相信应该会比从前的 VMProtect2.x 有明显改进。

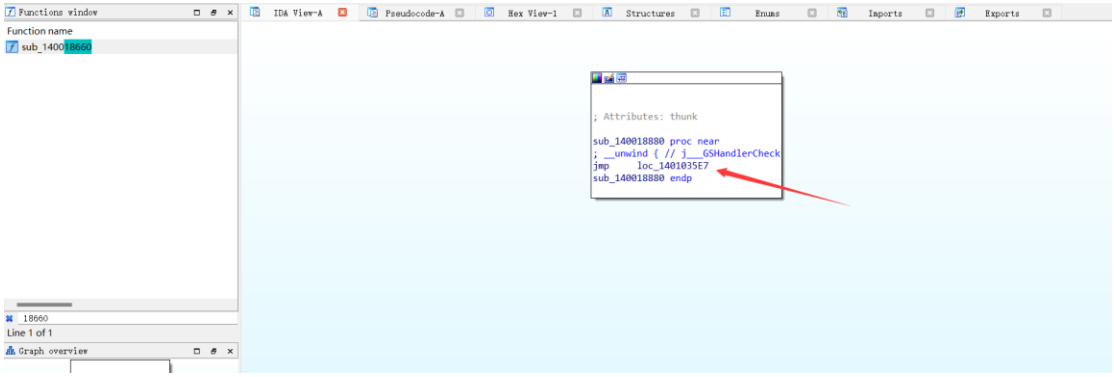
还有一些补充：VMProtect 在代码变异中使用控制流保护。它专注于所有基本块的跳转，而不是条件跳转，因此即使在变异之后原始控制流也可以看到（这也是为什么在 IDA 中单纯使用变异干扰效果不佳的原因，但与虚拟结合起来效果极佳），但 VMProtect 在变异代码上花费力气也很大，有很多有趣的细节可以探索，但在这里就不多说了。

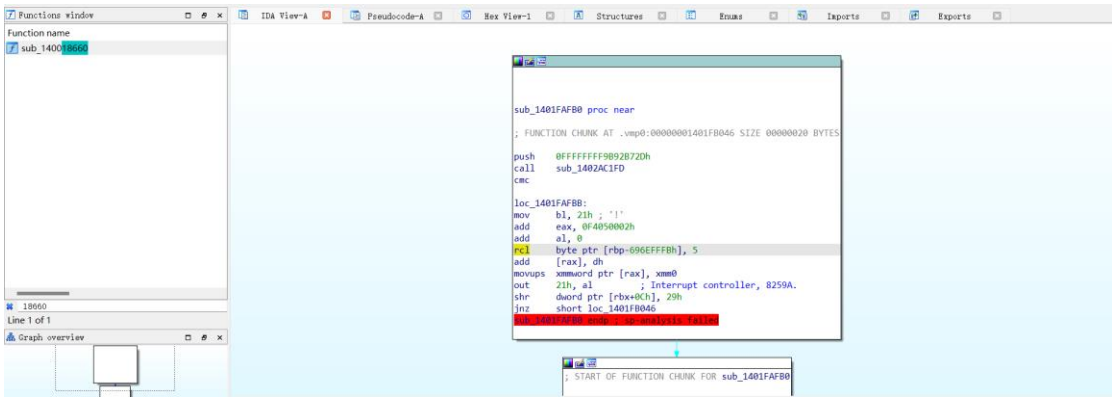
3.虚拟引擎（VM）

首先你要确定你使用的不是官网的演示 demo 版本，演示版的难度比不上付费版。演示版实际是一个阉割的虚拟机，有着一个带有类似偏移跳转到句柄开关的调度程序。可以轻松跟踪 vm 代码，没有跳转到下一个操作码的加密流程链，操作码的参数都没有加密。而且句柄中的改变很少。演示版生成的文件可以说是仅仅作为展示学习用，因为受到的保护没有付费版那么完整，而目前官网放出的 3.x 版本的 demo 演示实际上就是曾经的 2.x 老版本修改得来的。

下面简单检查（对源程序进行重要函数虚拟化处理，这一段会跟具体演示部分相关）

当我们对已经虚拟化的演示函数 sub_140018660 .text 进行查看，可以发现它跳转到了一个 sub_1401FAFB0 函数且在.vmp0 区段，可以想到 VMProtect 已经接管了原始函数，将原始函数的代码替换为了实际的 VMProtect 代码。





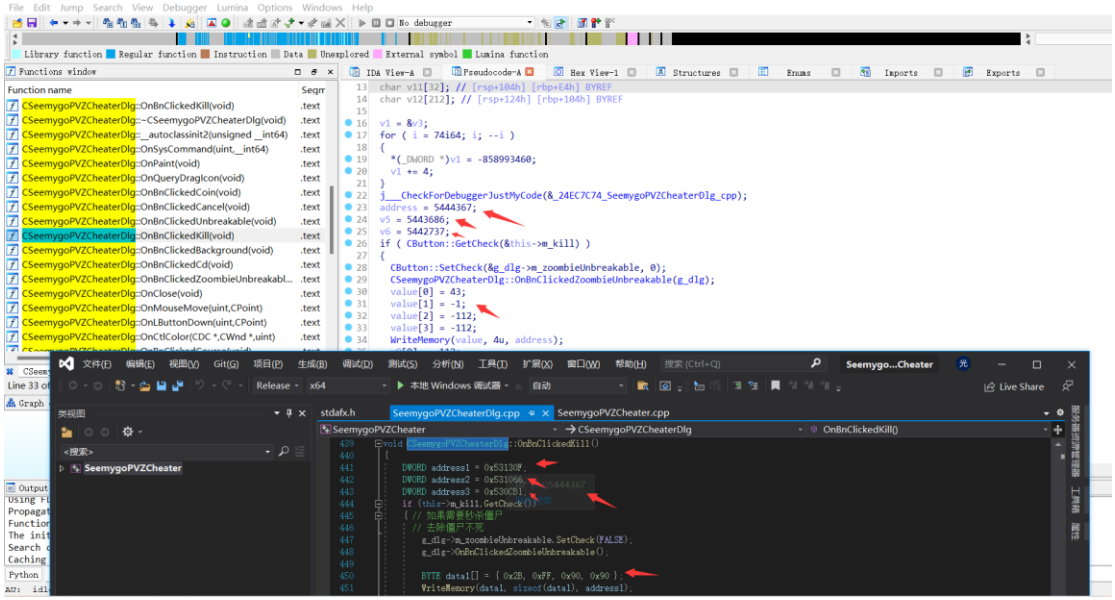
对我来说，比较敏感的指令代码就是 push 0FFFFFFFF9B92B72Dh 和 call sub_1402AC1FD，在这里就可以大胆的猜测，这可能就是 VMProtect 虚拟进程的开始，将 VMProtect 生成的虚拟代码在运行过程中转换为正常的程序代码，但是按照 VMProtect 的尿性，可能它只是一个永远都不返回的调用，而且它调用的数据是随机的，力图将调试者带入岔路，也可能它是一个自定义句柄调用一个 VENTER 指令（VENTER 意思是 VMProtect 实现的 VM 虚拟机中的类似 ENTER 指令用来在被调用过程自动创建堆栈帧，为之后的解密做准备），可能 push 传入的值可能就是加密操作码的开始，后续调用一个 VENTER 来解密它……。

这里讲的就显得尤为**重要**了！首先，想要让 VMProtect 真正有效的保护你的开发成果，你需要在源代码中或从 VMProtect 导入已编译的程序后（从地址或使用 PDB 文件手动标记）标记所有要变异（虚拟）的函数。如果你只是简单的将需要保护的程序导入 VMProtect 而不标记敏感重要函数，VMProtect 是没有体现它应该具备的保护效果的，就像我第一部分所说的那样，实质上是对源程序进行一次打包压缩，却没有有效保护自己的程序。

三、具体演示部分

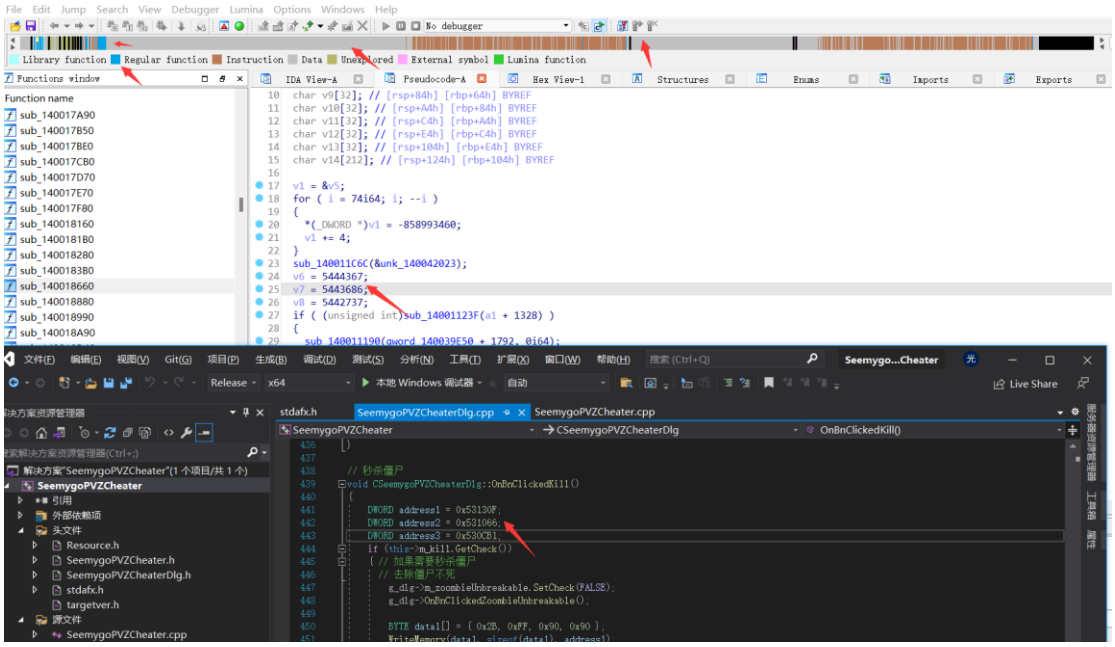
下面演示使用 PDB 文件标注重要函数，并对程序有效保护和无效保护操作的对比：

- 未处理的源程序（载入 PDB）IDA7.5 静态分析（图中演示函数静态地址 140018660）：



可以发现不做保护措施的程序，在反编译软件面前近乎都是裸奔，且 IDA 反编译出来的伪代码已经是可以阅读了，经验丰富的逆向人员都可以如常阅读伪代码了，就这样你程序的业务逻辑也随之暴露，想想这方面的后果吧。

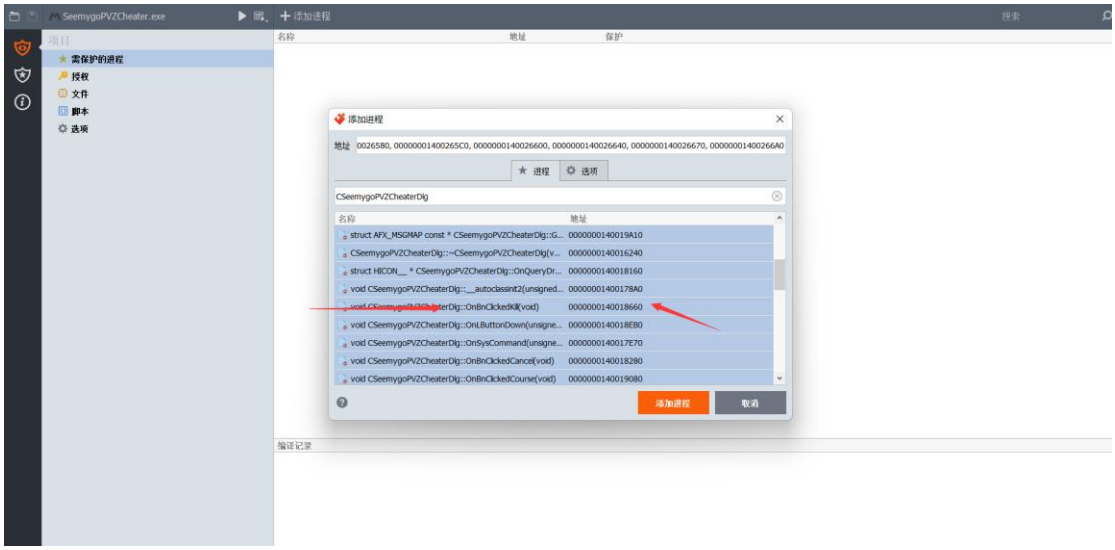
- 未有效加密的效果展示（只加入了入口函数，不载入 PDB，未压缩）：



这张图近乎模拟了破解者的角度，没有 PDB 文件，经过 VMProtect 处理可以看到与上方蓝色代码段部分大大减少，棕色数据段明显增多，因为没有 PDB 文件就没有正常的函数名注释，但破解者依旧可以通过很多办法找到他们想要找到的代码段，例如你在某一段代码调用的 API 都会让破解者得到线索，只是增加了时间的花销，而且破解者不仅仅只有静态分析的手段，还有动态调试。如图，不完善的保护，依旧让写入代码中的游戏数值地址暴露，代码逻辑暴露无遗。

● 有效的加密效果展示（敏感函数标记并变异+虚拟，加入并利用 PDB 文件标记函数，未压缩）：

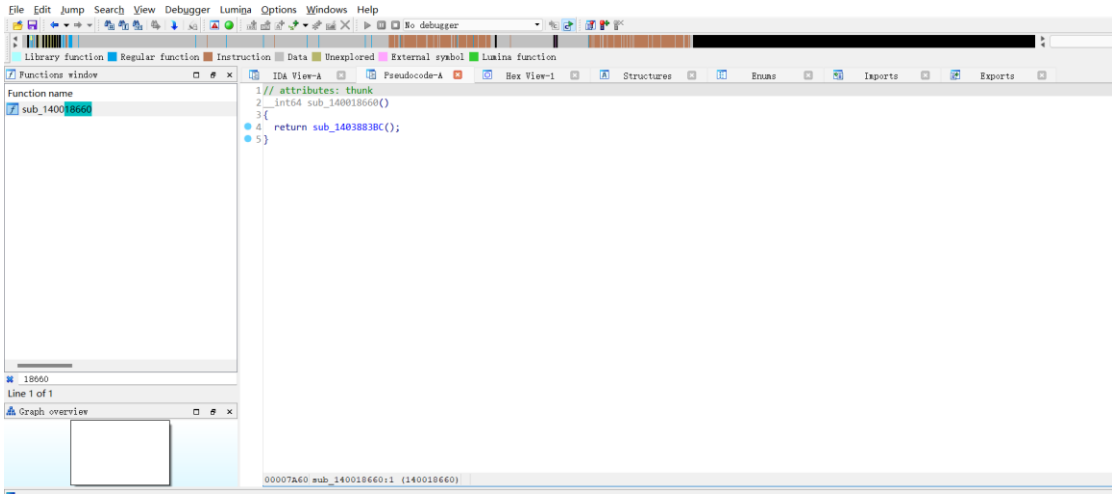
将编译生成的 PDB 文件跟被处理程序放置在同一文件目录下。



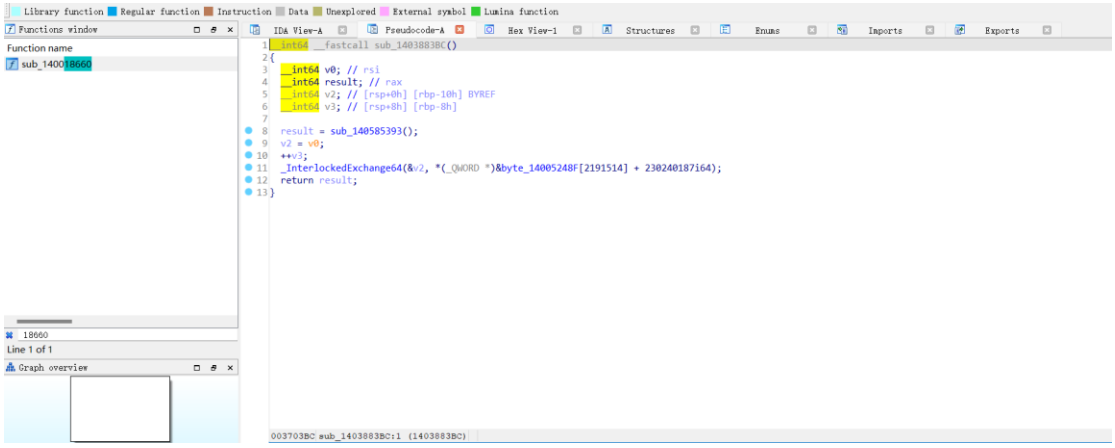
可以看到截图中标注是 CSeemygoPVZCheaterDlg::xxxxxxx 系列函数，

我们记住地址 140018660，这就是演示的 OnBnClickedKill 函数，这里我们全选这些函数并在选项中选择**变异+虚拟**直接拉满（**变异**对 IDA 来说干扰作用不大，两者结合效果极佳），然后添加进程，点击左上角三角加密即可。下面我们来看看效果……

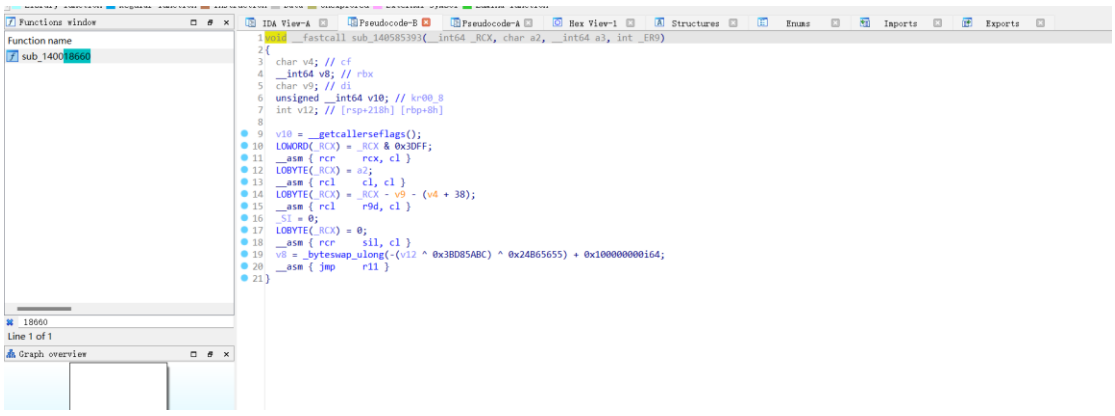
IDA 中搜索 140018660，可以看到：



点击伪代码中函数，查看：



再次点击进入 sub_140585393：

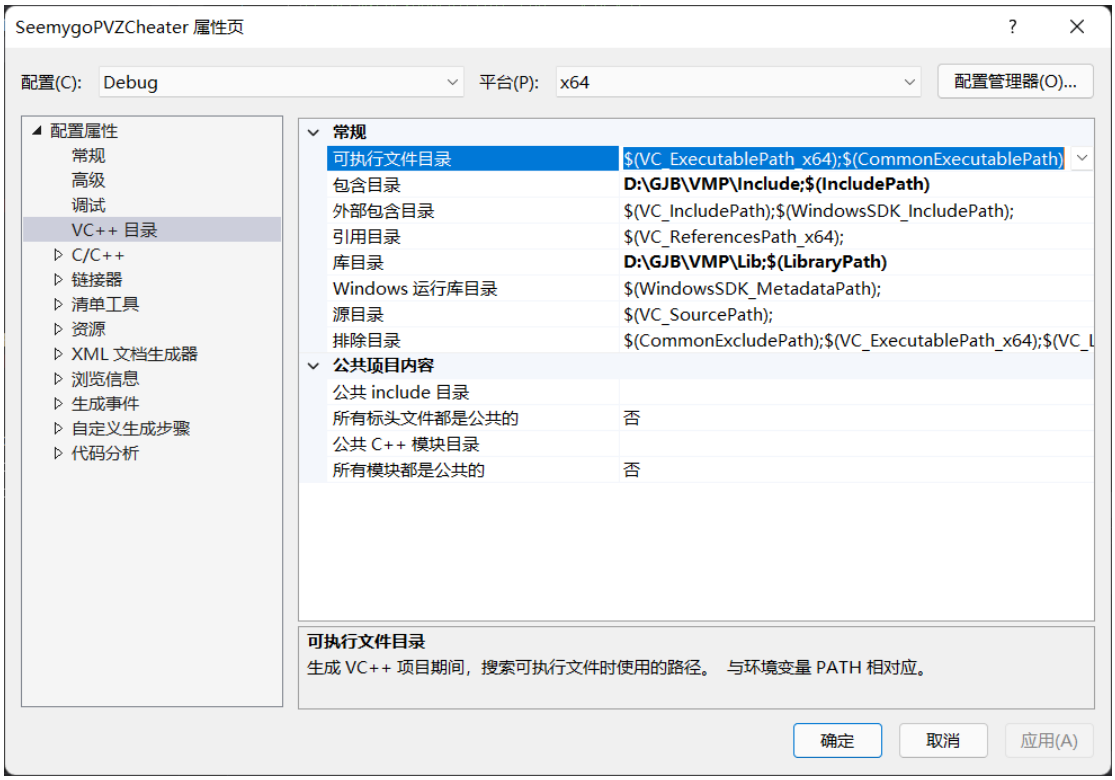


这是个什么玩意？在 IDA 给予的分析结果来看，确实看不出来啥玩意，

也没有之前那么清晰明了的伪代码了，动态调试也只会浪费大量时间跟虚拟机搏斗，这时候有了完整有效的加密流程，破解者的时间金钱成本都在呈指数增加，这样就很容易让破解者知难而退，但是呢，凡事没有绝对，永远没有不会被破解的程序，只能说我们增加了破解这个加密程序成本罢了，如果你的程序经济效益让人十分眼红，那尽量还是不要节省这些小钱了，直接上大公司优秀商业方案吧，不仅便捷还有满满的技术支持，总会适合你的。

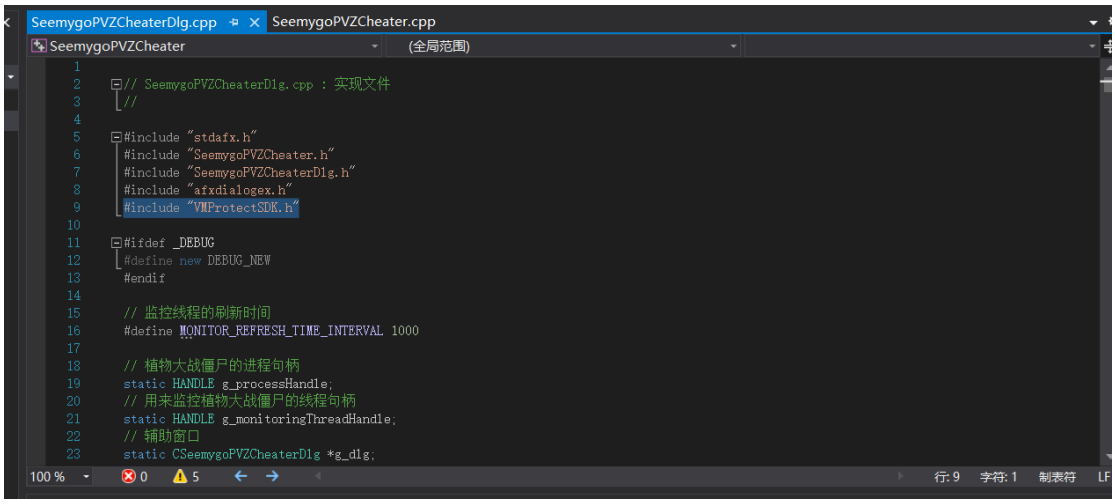
下面演示使用在项目中导入 VMProtect 提供的 SDK，在源码上标记需要保护的重要函数。

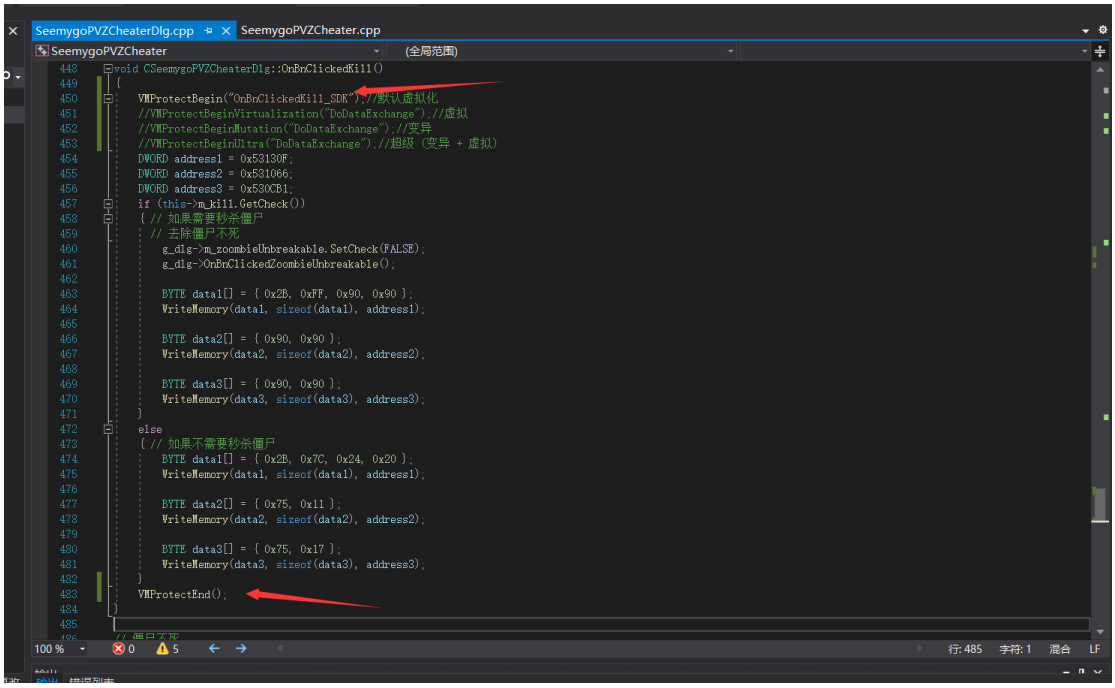
将 VMProtect 安装目录下的 include 和 lib 引入到你想保护的项目中：



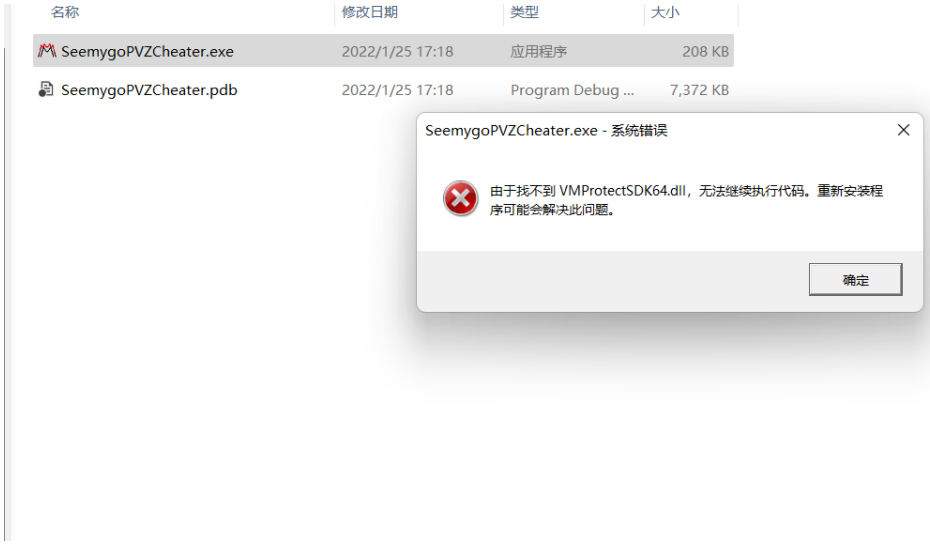
并在项目源码中引入#include "VMProtectSDK.h"，使用方法如下：

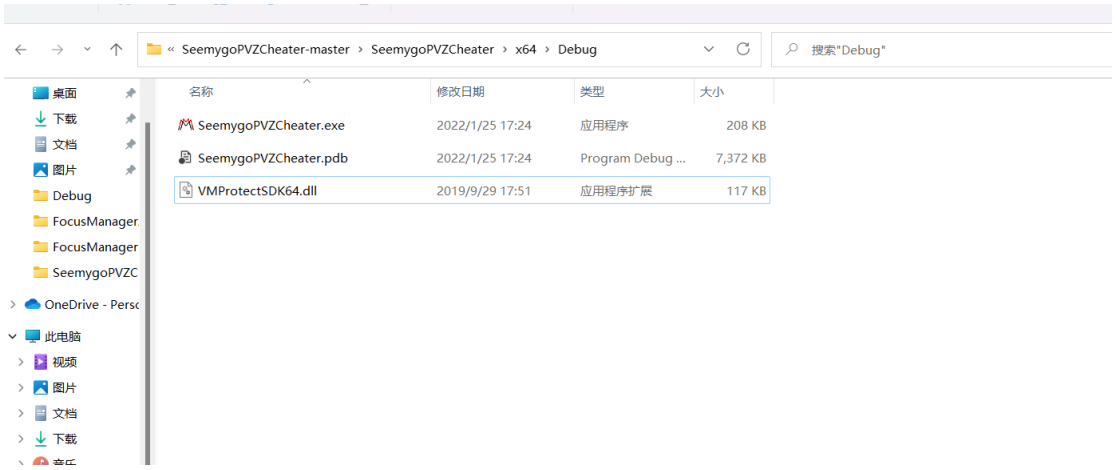
```
VMProtectBegin("DoDataExchange");//默认虚拟化
 //VMProtectBeginVirtualization("DoDataExchange");//虚拟
 //VMProtectBeginMutation("DoDataExchange");//变异
 //VMProtectBeginUltra("DoDataExchange");//超级 (变异 + 虚拟)
VMProtectEnd();
```

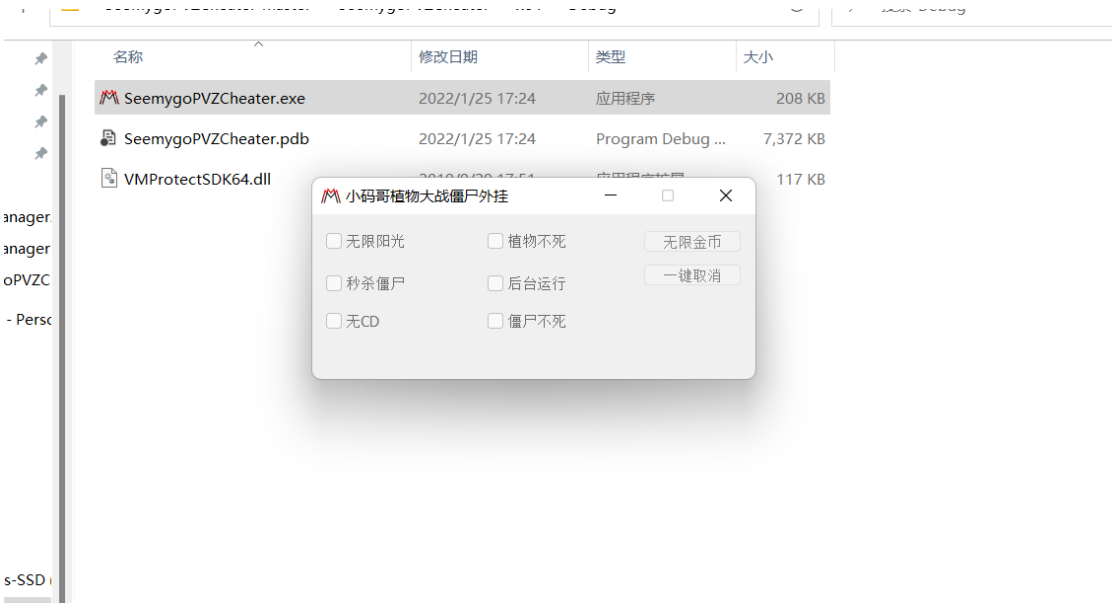




生成完毕时点击运行，会发现缺少 dll 错误，这个直接到 VMProtect 目录寻找寻找 lib 文件下的 VMProtectSDK64.dll 文件，将其放置到程序所在目录即可运行。

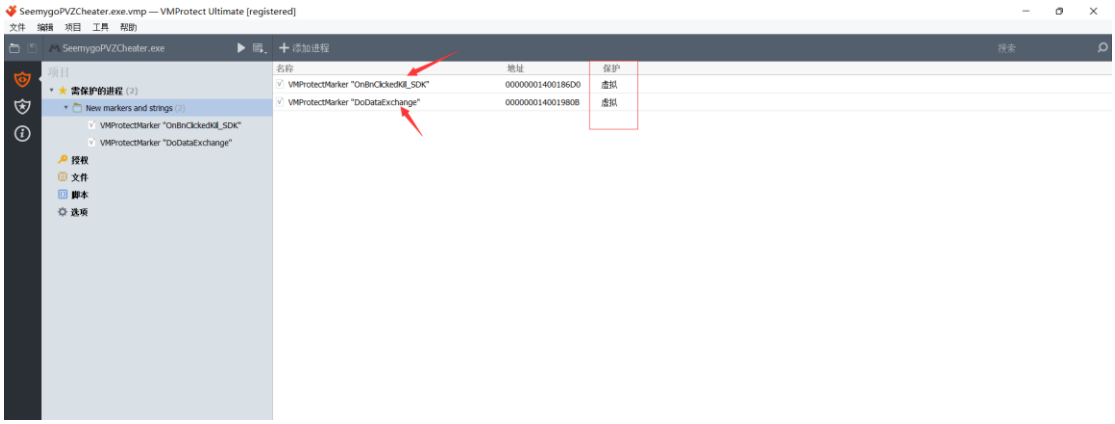




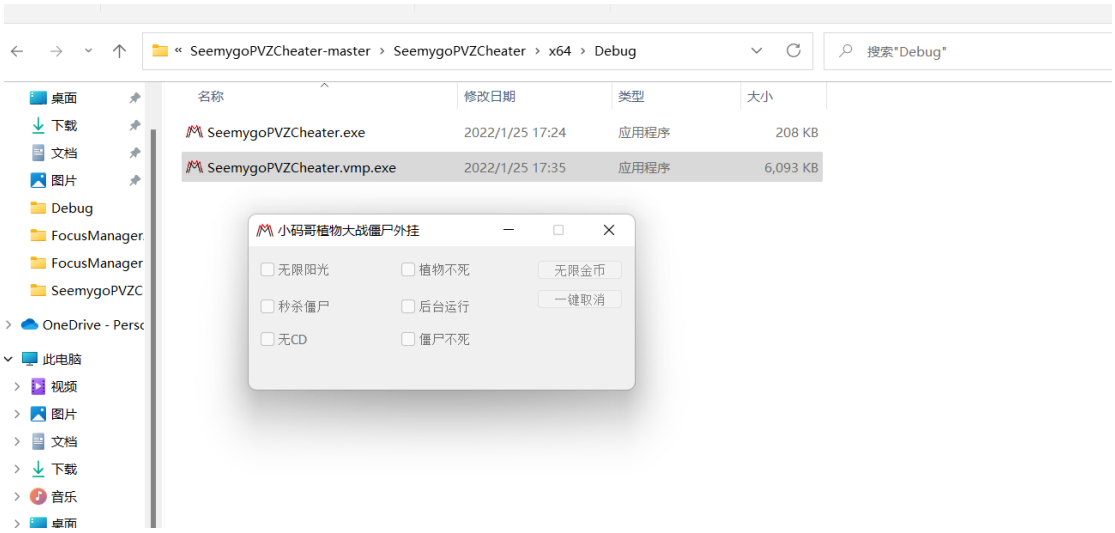


下面将程序载入 VMProtect，记得移除 PDB 文件，毕竟我们演示的是在源码中标

注重要函数。



看，可以看到在源码中标注的函数出现在了 VMProtect 添加进程界面中，而且经过 VMProtect 处理后，也不再需要 VMProtectSDK64.dll 支持了，直接可以运行。



还有最后一点小小的提醒：没有任何一款保护软件是万无一失的，只是提高了破解的门槛，增加了破解的成本。是的，这样的确会让人担心，如果觉得实在不安心，可以在 VMProtect 将反调试等等选项都选上，处理完毕的程序可以在 VMP 壳的基础上再加一层兼容性强的压缩/加密壳，尽可能的将安全性拉满，或者加钱上商业方案。。。拜拜咯

VMProtect3.5 下载链接（解压密码：Guerra）：

谷歌云盘（需梯子）：

https://drive.google.com/file/d/1qskt9tWDuDQw2m1_C8Kppn11kOB4MeAc/view?usp=sharing

天翼网盘：https://cloud.189.cn/web/share?code=YzE7bmVzqymu（访问码：iot9）