火绒安全
HUORONG SECURITY

火绒虚拟沙盒简介

“动若脱兔”

公　　司：北京火绒网络科技有限公司

地　　址：北京市朝阳区红军营南路 15 号瑞普大厦 D 座 4 层

网　　址：https://www.huorong.cn

电　　话：400-998-3555

版权声明

目录

虚拟机技术概述

仿真技术分类及应用

仿真技术			
分类	硬件仿真	指令集仿真	操作系统仿真
定义	通过软件仿真操作系统所需要的硬件环境，包括 CPU、内存、总线等等；	通过软件仿真特定（真实或虚拟）指令集的执行行为，例如通过仿真全部 x86 指令的行为来仿真 x86 CPU；	通过仿真操作系统环境，使为该操作系统环境编译的程序得以运行；
经典案例	VMware Workstation，QEMU，Bochs，……	Java VM，LLVM，VMProtect，Themida VM……	火绒虚拟沙盒，WINE，x86emu，……
安全领域应用	1. 搭建反病毒分析平台，如 ThreatExpert 等； 2. 搭建云安全集群；	1. 代码保护，如 VMProtect, Themida 等； 2. 代码逻辑序列化，如 ClamAV 支持把查毒代码编译成 LLVM 中间代码（IR），并在查毒时虚拟执行中间代码来执行查毒逻辑；	1. 实现相同平台、不同操作系统间的"跨界"执行，如 WINE 等； 2. 配合指令集仿真，实现虚拟沙盒，如火绒虚拟沙盒等；

虚拟执行技术分类及应用

虚拟执行技术				
分类	**模拟类虚拟执行技术**		**虚拟化类虚拟执行技术**	
定义	通过软件方法模拟 CPU 运行环境(如：寄存器等)和各种机制(如：内存寻址、异常处理、保护机制等)来实现实现虚拟执行；		通过软件或硬件方法，利用当前 CPU 环境，创建(虚拟)出可控的隔离环境，并在这个隔离环境中利用真实 CPU 资源进行受控的代码执行；	
实现技术	指令模拟	动态翻译	软件虚拟化	硬件虚拟化
典型应用	瑞星 v16+，……	火绒，MSE，……	火绒，……	未知

执行速度（相较于真实代码执行）	≈1%	≈30%	≈95%	≈99%
代码级控制粒度	细	一般	粗	粗
限制	无	无	任何支持分页机制的 CPU（例如：Intel Pentium 386+以上，但并不限于 x86 架构）	仅限支持硬件虚拟化技术的 CPU（例如：Intel VT-x、AMD-V）
引擎应用场景	感染型病毒查杀 简单代码分析 辅助脱壳	感染型病毒查杀 指导脱壳 行为分析	感染型病毒查杀 通用脱壳 行为分析	感染型病毒查杀 通用脱壳 行为分析

关于虚拟沙盒

套用一句广告词，“不是所有的虚拟机都叫虚拟沙盒”。反病毒虚拟沙盒应至少符合以下几个基本要求：

1、虚拟执行效率要足够高

反病毒引擎要求能够在相对较短的时间内（毫秒级）完成对待扫描对象的扫描，所以低下的虚拟执行效率是无法真正应用于反病毒虚拟沙盒的；

2、具有完备的操作系统环境仿真

如果仅仅能够虚拟执行指令，那么还无法称作虚拟沙盒。虚拟沙盒应具有完备的操作系统环境仿真，至少应包括：文件系统、注册表、进程、线程、调度逻辑、时钟、同步对象；

3、能够捕获并记录程序虚拟执行时的行为

虚拟沙盒应当可以捕获并记录程序在沙盒内虚拟执行时所产生的行为，此类记录可以是系统调用级别或更高级的抽象等。缺失捕获并记录行为的特性，行为分析便无从谈起；

火绒虚拟沙盒技术与特性

虚拟沙盒总体架构

虚拟执行引擎

虚拟机化执行引擎负责实现指令集仿真，火绒虚拟沙盒实现了两个执行引擎：

1、火绒虚拟化执行引擎

通过火绒虚拟化技术，为目标代码划分独立的地址空间，并通过接管中断和异常为目标代码分配私有时间片，从而使目标代码得以受控执行。执行效率几乎可以达到与真实机相当，仅执行环境切换带来微量开销；

2、动态翻译执行引擎

通过对目标代码进行分析并可控地翻译成本地代码执行的技术称为动态翻译，火绒虚拟沙盒通过动态翻译执行引擎实现对目标代码的细粒度控制；

火绒虚拟化引擎执行效率极高，但对目标代码的控制粒度粗，而动态翻译执行引擎虽然执行效率低，但对代码的控制粒度高。所以，火绒虚拟沙盒在运行时会根据虚拟执行的需要，自动对两个执行引擎进行切换以同时满足较高的执行效率和较细的代码控制粒度。

关于火绒虚拟化执行引擎和动态翻译执行引擎的效率比对，请参见“火绒虚拟沙盒应用演示”一节的相关演示。

操作系统环境仿真

我习惯将虚拟执行引擎比作楼房的地基，而操作系统环境仿真则是建立在牢固地基上的万丈高楼。

火绒虚拟沙盒设计了完备的操作系统环境仿真，模拟了超过 23000 个 Windows API，涵盖了绝大多数操作系统的核心机制，包括但不局限于：

1、文件系统

存储着核心系统文件，并可以跟踪虚拟沙盒内进程执行时创建和修改的文件，反病毒引擎在并发扫描时，不同虚拟进程拥有隔离的文件系统视图；

2、注册表系统

完备的注册表系统仿真，虚拟沙盒内进程的注册表操作可以被跟踪和记录，以便反病毒引擎进行分析；

3、进程、线程、同步对象、调度、时钟

火绒虚拟沙盒仿真了完备的进程、线程对象，并根据虚拟时钟及调度逻辑对线程进行调度，可以有效地“跑开”Themida 等多线程解码的壳；

4、窗口系统

火绒虚拟沙盒还实现了复杂、庞大的窗口系统，以及多中窗口控件，著名的 TrojanDownloader/Upatre 家族混淆器即通过 Edit/RichEdit 控件的各种消息和窗口特性来作为反虚拟机的手段；

5、……

跨平台特性

目前，火绒虚拟沙盒支持以下操作系统平台：

1、Windows x86/x64;

2、Linux x86/x64;

3、Mac OS X x86/x64；

火绒虚拟沙盒非常容易移植到 FreeBSD 等类 Unix 操作系统平台。

火绒虚拟沙盒应用

通用脱壳（Generic Unpacking）

仅实现指令集仿真就可以实现辅助脱壳的功能，而虚拟沙盒对于脱壳的真正意义在于通用脱壳。

通用脱壳是指在不需要识别样本是否加壳的情况下，通过将样本放入虚拟沙盒深度执行并通过启发式逻辑分析样本数据是否已被还原的技术。通用脱壳对于多层壳、虚拟机保护壳、自定义壳、高级包裹器（High-Level Language Wrapper，HLLW）等有着非常重要的意义。传统反病毒引擎的静态或动态指导脱壳，对于上述几类壳是毫无作用的。

关于通用脱壳更具体的内容，可以参见《反病毒“芯”技术——火绒反病毒引擎简介》一文。

反病毒引擎深度扫描

凭借完备的操作系统环境模拟，火绒虚拟沙盒可以跟踪其中进程释放的文件、创建的进程等，火绒反病毒引擎在扫描过程中会对这些衍生物进行扫描，以此来实现对样本的深度扫描。

下面的扫描日志展示的是火绒反病毒引擎扫描一修改过的 CAB 自解压包。火绒反病毒引擎并不需要对这种修改过的 CAB 自解压包单独进行解码支持，而只需要在虚拟沙盒中虚拟运行该样本，样本“自己”便会完成解码并释放压缩包中的文件到火绒虚拟沙盒的文件系统中。火绒反病毒引擎可以提取沙盒跟踪到的虚拟文件，并对其进行扫描，进而发现威胁：

```
>> 1f9f805a839446db9b04fe66b9f9eb39a330a8b1
>> 1f9f805a839446db9b04fe66b9f9eb39a330a8b1 >> genpack
>> 1f9f805a839446db9b04fe66b9f9eb39a330a8b1 >> genpack >> subpe_000AB81A
>> 1f9f805a839446db9b04fe66b9f9eb39a330a8b1 >> vfs:[c:\Windows\temp\IXP000.TMP\CATALOGVIEWER.EXE]
```

```
>> 1f9f805a839446db9b04fe66b9f9eb39a330a8b1 >> vfs:[c:\Windows\temp\IXP000.TMP\wdfmgrs.exe]
[ENG-3] 1f9f805a839446db9b04fe66b9f9eb39a330a8b1 >> vfs:[c:\Windows\temp\IXP000.TMP\wdfmgrs.exe] ... infected: HVM:Trojan/MalBehav.gen!C [D736ACAC2F763626]
```

基于火绒虚拟沙盒的动态行为分析（行为沙盒）

火绒虚拟沙盒可以跟踪和记录运行在其中程序的行为，火绒反病毒引擎通过行为记录，可以通过启发式分析算法对程序的恶意性进行评估。我们将虚拟沙盒的此类应用称为行为沙盒。

火绒行为沙盒不仅仅被应用火绒反病毒引擎。在后台，火绒虚拟沙盒作为火绒反病毒自动处理平台的一部分，负责抽取样本的动态行为特征，与其他特征抽取模块组合，共同为样本的相似性聚类计算服务着。

更具体的关于火绒行为沙盒的相关内容，可以参见《反病毒“芯”技术——火绒反病毒引擎简介》一文。

火绒虚拟沙盒应用演示

以下演示通过 asciinema（https://asciinema.org/）或 GIF 录制。

火绒虚拟化执行引擎 vs. 动态翻译执行引擎

- 样本 SHA1：a8e20a4edbfc083fdeee18aa6da56d932ea35eb4
- 演示地址：https://asciinema.org/a/27411
- 演示说明

演示中，首先以常规模式，即开启火绒虚拟化执行引擎的模式，虚拟执行样本 a8e20a4edbfc083fdeee18aa6da56d932ea35eb4 直至程序运行完毕退出。运行结果显示，共执行了 58 条指令（只有自动切换到动态翻译执行引擎时才能够统计指令数，所以常规模式统计到的指令数非常少），执行了 806,940 个 API 调用，共耗时 301 毫秒。

接下来，仅开启动态翻译执行引擎，再次虚拟执行该样本直至程序运行完毕退出。运行结果显示，共执行了 907,528,493 条指令，同样是执行了 806,940 个 API 调用，但耗时高达 54,188 毫秒（接近 1 分钟），比常规模式慢了 180 倍。

从结果可以看出，火绒虚拟化执行引擎的执行效率，比动态翻译引擎至少快 100 倍以上。这为火绒反病毒引擎的深度扫描奠定了极为宝贵的基础，在同样的扫描时间下，火绒虚拟沙盒可以执行的深度更深，使得很多在传统反病毒虚拟机不可能“跑开”的样本，在火绒虚拟沙盒中成为了可能。

火绒行为沙盒检出典型恶意行为

- 样本 SHA1：4f52ac297e86bc595acfa64859a9c87385368e22

- 演示地址：https://asciinema.org/a/27387
- 演示说明

演示中展示了样本 4f52ac297e86bc595acfa64859a9c87385368e22 在火绒虚拟沙盒中执行时的跟踪记录。从跟踪记录可以看出，该样本具有很典型的恶意代码行为：将自身复制到系统目录并命名成类似系统文件的名称（C:\Windows\System32\SVOHOST.exe），运行这个自复制的副本，最后通过 WinExec("…cmd /c del…")的方式自删除。火绒反病毒引擎会在扫描时捕捉到上述行为，并将该样本检出为 HVM:Trojan/MalBehav.gen!C 威胁。

动态还原 Trojan/FakeAV 高级包裹器（HLLW）

- 样本 SHA1：d5cd4da969281905ed832a2587380c3968a2fb3f
- 演示地址：https://asciinema.org/a/27394
- 演示说明

演示中的样本由非常典型的高级包裹器（High-Level Language Wrapper，HLLW）包裹，这在 Trojan/FakeAV 等家族中相当常见。

在演示中，由于预先知道此样本应包含的关键字符串，所以当这些字符串无法被搜索到时，就怀疑可能是加壳保护了。之后，通过 Pyew 分析发现该样本入口符合 VC7 编译器生成的入口特征，在火绒虚拟沙盒调试器中手工查看入口也符合 VC7 入口特征，所以怀疑该样本由 HLLW 包裹。

接下来，在火绒虚拟沙盒调试器中调试执行该样本，发现经过代码解密后，代码停在 UPX 入口上，DUMP 当前位置的镜像并用 Pyew 分析也能够确定当前位置的确为 UPX 入口代码。

跟踪到 UPX 最后的跳转，发现解出的新入口明显属于 VC8 入口特征，并用 Pyew 验证。此时，对该样本的脱壳过程全部完成，DUMP 当前镜像并搜索关键字符串证明 Trojan/FakeAV 的原始数据已被还原。通过这样的还原，火绒反病毒引擎在扫描此类样本时就完全是顺水推舟了。

从对这个样本的脱壳过程可以看出，此样本最初由 VC8 编译，经过 UPX 加壳后（主要为了压缩体积），又用由 VC7 编写的高级包裹器再次包裹，可见此高级包裹器是独立存在的，很可能还会被其他恶意代码所使用。这种多个恶意代码家族使用同一款高级包裹器或混淆器的例子屡见不鲜。

另外，由于火绒虚拟沙盒会在虚拟执行时动态对目标代码做优化调整，所以该样本在火绒虚拟沙盒中执行时会比在真实系统下执行更快，有兴趣的朋友可以自行测试。但注意请在虚拟机环境中进行测试或调试，以免对真实系统产生不必要的危害。

疯狂利用窗口系统特性的 TrojanDownloader/Upatre 混淆器

- 样本 SHA1：51d17236fbf0236ccd2571e252a9f173f37702a4 等
- 演示地址：https://asciinema.org/a/27054
- 演示说明

火绒反病毒通过火绒虚拟沙盒动态还原 TrojanDownloader/Upatre 的明文代码及数据，并通过行为跟踪记录实现启发式检出 HVM:TrojanDownloader/Upatre.gen。演示中展示的是火绒反病毒引擎对 98 个不同变种的 TrojanDownloader/Upatre 进行扫描的过程。

下面，我以样本 51d17236fbf0236ccd2571e252a9f173f37702a4 为例，介绍 TrojanDownloader/Upatre 家族常用的反虚拟机技术。

1、首先，在程序入口该样本注册了一个名为 contents 的窗口类，并创建了一个窗口，句柄保存在 hwnd_contents 中；

```
.text:00401747          push    edi
.text:00401748          dec     ecx
.text:00401749          jnz     short loc_401747
.text:0040174B          lea     eax, [ebp+WndClass]
.text:0040174E          push    eax             ; lpWndClass
.text:0040174F          mov     edx, hInstance
.text:00401755          mov     [ebp+WndClass.hInstance], edx
.text:00401758          lea     ecx, aContents  ; "contents"
.text:0040175E          mov     [ebp+WndClass.lpszClassName], ecx
.text:00401761          lea     eax, wndproc_contents
.text:00401767          mov     [ebp+WndClass.lpfnWndProc], eax
.text:0040176A          call    RegisterClassA
.text:00401770          push    67h             ; lpBitmapName
.text:00401772          push    hInstance       ; hInstance
.text:00401778          call    LoadBitmapA
.text:0040177E          push    66h             ; lpBitmapName
.text:00401780          push    hInstance       ; hInstance
.text:00401786          call    LoadBitmapA
.text:0040178C          xor     eax, eax
.text:0040178E          push    eax             ; lpParam
.text:0040178F          push    hInstance       ; hInstance
.text:00401795          push    eax             ; hMenu
.text:00401796          push    eax             ; hWndParent
.text:00401797          push    500             ; nHeight
.text:0040179C          push    756             ; nWidth
.text:004017A1          push    1400            ; Y
.text:004017A6          push    3300            ; X
.text:004017AB          push    0               ; dwStyle
.text:004017AD          mov     ecx, offset aFrantically ; "frantically"
.text:004017B2          push    eax             ; lpWindowName
.text:004017B3          push    offset aContents ; "contents"
.text:004017B8          push    eax             ; dwExStyle
.text:004017B9          call    CreateWindowExA
.text:004017BF          mov     hwnd_contents, eax
```

2、在 hwnd_contents 窗口回调收到 WM_CREATE 消息时，该样本又创建了一个

Edit 控件，并在创建时通过窗口名指定控件内容为 7 行文字，句柄保存在 hwnd_edit_1 中；

```
.text:00401091 @@wm_create:                            ; CODE XREF: wndproc_contents+9j
...
.text:004010EE                 mov     ecx, hInstance
.text:004010F4                 lea     eax, ClassName  ; "edit"
.text:004010FA                                         mov     edi, offset WindowName ;
"sacdea\r\nebulu\r\nacasec\r\nlabai\r\nedufu\r\nmc"...
.text:004010FF                 push    0               ; lpParam
.text:00401101                 push    ecx             ; hInstance
.text:00401102                 push    0               ; hMenu
.text:00401104                 push    esi             ; hWndParent
.text:00401105                 mov     esi, 0
.text:0040110A                 push    30              ; nHeight
.text:0040110C                 push    240             ; nWidth
.text:00401111                 push    185             ; Y
.text:00401116                 push    25              ; X
.text:00401118                 push    ebx             ; dwStyle
.text:00401119                 push    edi             ; lpWindowName
.text:0040111A                 push    eax             ; lpClassName
.text:0040111B                 push    esi             ; dwExStyle
.text:0040111C                 call    CreateWindowExA
.text:00401122                 mov     hwnd_edit_1, eax
```

3、当 hwnd_contents 收到特定次数的 WM_PARENTNOTIFY 消息后（此样本是固定的 3 次），通过 EM_GETLINECOUNT 获取 hwnd_edit_1 窗口中文字的行数（此样本为 7 行），并调用以此行数与 0x40144a 之和为地址的解码函数；

```
.text:00401000 wndproc_contents proc near              ; DATA XREF: sub_401728+39o
.text:00401000
.text:00401000 hWndParent      = dword ptr  8
.text:00401000 Msg             = dword ptr  0Ch
.text:00401000 wParam          = dword ptr  10h
.text:00401000 lParam          = dword ptr  14h
.text:00401000
.text:00401000                 push    ebp
.text:00401001                 mov     ebp, esp
.text:00401003                 mov     eax, [ebp+Msg]
.text:00401006                 cmp     eax, WM_CREATE
.text:00401009                 jz      @@wm_create
.text:0040100F                 cmp     eax, WM_DESTROY
```

```
.text:00401012                  jz      short @@wm_destroy
.text:00401014                  cmp     eax, WM_PARENTNOTIFY
.text:00401019                  jz      short loc_401032
.text:0040101B                  push    [ebp+lParam]    ; lParam
.text:0040101E                  push    [ebp+wParam]    ; wParam
.text:00401021                  push    [ebp+Msg]       ; Msg
.text:00401024                  push    [ebp+hWndParent] ; hWnd
.text:00401027                  call    DefWindowProcA
.text:0040102D                  jmp     loc_401187
.text:00401032 ; ---------------------------------------------------------------------------
.text:00401032
.text:00401032 loc_401032:                              ; CODE XREF: wndproc_contents+19j
.text:00401032                  mov     eax, [ebp+lParam]
.text:00401035                  mov     eax, parent_notify_count
.text:0040103A                  dec     eax
.text:0040103B                  jz      short @@count_reached
.text:0040103D                  mov     parent_notify_count, eax
.text:00401042                  jmp     loc_401187
.text:00401047 ; ---------------------------------------------------------------------------
.text:00401047
.text:00401047 @@count_reached:                         ; CODE XREF: wndproc_contents+3Bj
.text:00401047                  push    offset aInstability ; "instability"
.text:0040104C                  push    0               ; wParam
.text:0040104E                  push    WM_SETTEXT      ; Msg
.text:00401050                  push    hwnd_edit_0     ; hWnd
.text:00401056                  call    SendMessageA
.text:0040105C                  xor     ecx, ecx
.text:0040105E                  push    ecx             ; lParam
.text:0040105F                  push    ecx             ; wParam
.text:00401060                  push    EM_GETLINECOUNT ; Msg
.text:00401065                  push    hwnd_edit_1     ; hWnd
.text:0040106B                  call    SendMessageA
.text:00401071                  shl     eax, 3          ; should return 7
.text:00401074                  mov     ecx, eax
.text:00401076                  add     ecx, offset unk_40144A
.text:0040107C                  call    ecx             ; => real decryptor
```

4、前面的代码均为反虚拟机所用，上述代码在 call ecx 后，方才真正执行解码函数。

64 位 Trojan/Emotet 混淆器

- 样本 SHA1：02915b207d667729f003b510eb65c9de9bd9a589 等

- 演示地址： 演示 GIF 链接
- 演示说明

与 32 位混淆器相似，64 位 Emotet 变种中所使用的混淆器也会通过冗余代码、无用的拖慢循环代码等方式对抗安全软件查杀。现阶段，64 位混淆器所使用的混淆手段相较于 32 位混淆器而言较为简单。以示例样本为例，其中只使用无用的拖慢循环代码对抗安全软件检测，后续则直接执行解密逻辑。混淆器代码，如下图所示：

```
.text:0000000010045F70                  sub     rsp, 48h
.text:0000000010045F74                  mov     eax, cs:value_5F5E100h
.text:0000000010045F7A                  mov     rcx, rax        ; cb
.text:0000000010045F7D                  call    cs:CoTaskMemAlloc
.text:0000000010045F83                  mov     [rsp+48h+pv], rax
.text:0000000010045F88                  cmp     [rsp+48h+pv], 0
.text:0000000010045F8E                  jnz     short loc_10045F94
.text:0000000010045F90                  xor     eax, eax
.text:0000000010045F92                  jmp     short loc_1004600B
.text:0000000010045F94 ; ---------------------------------------------------------------------------
.text:0000000010045F94
.text:0000000010045F94 loc_10045F94:            ; CODE XREF: large_loop_useless+1E↑j
.text:0000000010045F94                  mov     eax, cs:value_5F5E100h
.text:0000000010045F9A                  mov     [rsp+48h+large_integer], rax
.text:0000000010045F9F                  mov     [rsp+48h+index], 0
.text:0000000010045FA8                  mov     rax, [rsp+48h+pv]
.text:0000000010045FAD                  mov     [rsp+48h+buffer_ptr], rax
.text:0000000010045FB2                  jmp     short loc_10045FD0
.text:0000000010045FB4 ; ---------------------------------------------------------------------------
.text:0000000010045FB4
.text:0000000010045FB4 loc_10045FB4:            ; CODE XREF: large_loop_useless+79↓j
.text:0000000010045FB4                  mov     rax, [rsp+48h+index]
.text:0000000010045FB9                  add     rax, 1
.text:0000000010045FBD                  mov     [rsp+48h+index], rax
.text:0000000010045FC2                  mov     rax, [rsp+48h+buffer_ptr]
.text:0000000010045FC7                  add     rax, 1
.text:0000000010045FCB                  mov     [rsp+48h+buffer_ptr], rax
.text:0000000010045FD0
.text:0000000010045FD0 loc_10045FD0:            ; CODE XREF: large_loop_useless+42↑j
.text:0000000010045FD0                  mov     eax, cs:value_5F5E100h
```

```
.text:0000000010045FD6                  cmp     [rsp+48h+index], rax
.text:0000000010045FDB                  jnb     short loc_10045FEB
.text:0000000010045FDD                  mov     rax, [rsp+48h+buffer_ptr]
.text:0000000010045FE2                  movzx   ecx, byte ptr [rsp+48h+index]
.text:0000000010045FE7                  mov     [rax], cl
.text:0000000010045FE9                  jmp     short loc_10045FB4
.text:0000000010045FEB ; ---------------------------------------------------------------------------
.text:0000000010045FEB
.text:0000000010045FEB loc_10045FEB:            ; CODE XREF: large_loop_useless+6B↑j
.text:0000000010045FEB                  mov     rcx, [rsp+48h+pv] ; pv
.text:0000000010045FF0                  call    cs:CoTaskMemFree
.text:0000000010045FF6                  mov     rax, [rsp+48h+large_integer]
.text:0000000010045FFB                  cmp     [rsp+48h+index], rax
.text:0000000010046000                  jnz     short loc_10046006
.text:0000000010046002                  xor     eax, eax
.text:0000000010046004                  jmp     short loc_1004600B
.text:0000000010046006 ; ---------------------------------------------------------------------------
.text:0000000010046006
.text:0000000010046006 loc_10046006:            ; CODE XREF: large_loop_useless+90↑j
.text:0000000010046006                  mov     eax, 3
.text:000000001004600B
.text:000000001004600B loc_1004600B:            ; CODE XREF: large_loop_useless+22↑j
.text:000000001004600B                                          ; large_loop_useless+94↑j
.text:000000001004600B                  add     rsp, 48h
.text:000000001004600F                  retn
```

上述代码首先会分配一块大小为 0x5F5E100 的内存，之后向内存中写入逐字节递增的无用数据（内容为 0~0xff）。写入完成后，比较局部变量 index 是否与逻辑循环次数相等（0x5F5E100），如果相等则继续执行解密逻辑，如果不相等则退出执行。后续解密执行 shellcode 代码，如下：

```
.text:000000001004D0D7                  add     esi, edx
.text:000000001004D0D9                  mov     edx, esi
.text:000000001004D0DB                  add     edi, edx
.text:000000001004D0DD                  mov     edx, edi
.text:000000001004D0DF                  or      ecx, edx
.text:000000001004D0E1                  movsxd  rdx, [rsp+0BE8h+var_BB8] ; dwSize
.text:000000001004D0E6                  mov     r9d, eax        ; flProtect
.text:000000001004D0E9                  mov     r8d, ecx        ; flAllocationType
.text:000000001004D0EC                  xor     ecx, ecx        ; lpAddress
.text:000000001004D0EE                  call    cs:VirtualAlloc
.text:000000001004D0F4                  mov     [rsp+0BE8h+shellcode_ptr], rax
```

```
.text:000000001004D0FC                movsxd    rax, [rsp+0BE8h+var_BB8]
.text:000000001004D101                mov       [rsp+0BE8h+var_BC8], 2Eh ; '.'
.text:000000001004D10A                lea       r9, [rsp+0BE8h+xor_key]
.text:000000001004D112                mov       r8, rax
.text:000000001004D115                mov       rdx, [rsp+0BE8h+shellcode_ptr]
.text:000000001004D11D                lea       rcx, [rsp+0BE8h+encrypted_code_data_ptr]
.text:000000001004D122                call      decrypt_data
.text:000000001004D127                mov       r9, [rsp+0BE8h+var_30]
.text:000000001004D12F                mov       r8d, 2Eh ; '.'
.text:000000001004D135                lea       rdx, [rsp+0BE8h+xor_key]
.text:000000001004D13D                mov       rcx, [rsp+0BE8h+arg_0]
.text:000000001004D145                call      [rsp+0BE8h+shellcode_ptr]
.text:000000001004D14C                mov       cs:qword_1006FEC0, rax
.text:000000001004D153
.text:000000001004D153 loc_1004D153:                                  ; CODE XREF: DllMain+41↑j
.text:000000001004D153                mov       eax, 1
.text:000000001004D158
.text:000000001004D158 loc_1004D158:                                  ; CODE XREF: DllMain+99↑j
.text:000000001004D158                mov       rcx, [rsp+0BE8h+var_20]
.text:000000001004D160                xor       rcx, rsp            ; StackCookie
.text:000000001004D163                call      __security_check_cookie
.text:000000001004D168                add       rsp, 0BD8h
.text:000000001004D16F                pop       rdi
.text:000000001004D170                pop       rsi
.text:000000001004D171                retn
```