

# 《竞学实训》报告



学 院：网络空间安全学院

专 业：密码科学与技术

学 生 学 号：20210046000065

学 生 姓 名： 李昕

二〇二三年七月

# 实验 Malware Analysis

## 一. 实验目的

掌握 Malware Analysis 的方法。

## 二. 实验内容

### 1. 样本准备

请编写一个可执行程序 Demo.exe，该程序使用 Windows API “WinExec”启动程序“calc.exe”。

编写 C++代码，利用 Windows API “WinExec”调用 win 命令行打开计算器，编译代码得到可执行程序 Demo.exe，代码如下：

```
#include<windows.h>
#include<iostream>
#include<cstdio>
int main()
{
    WinExec("calc", SW_SHOWNORMAL);
    //调用win 命令 calc 打开计算器
    return 0;
}
```

### 2. 静态分析

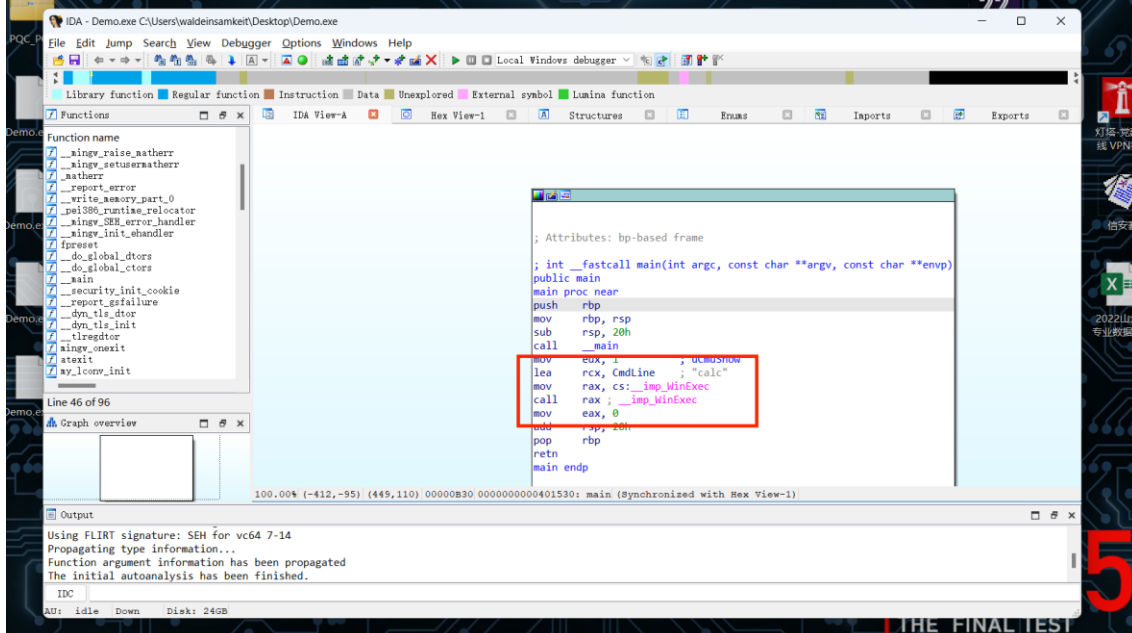
2.1 请使用反汇编工具 IDA<sup>1</sup>分析 Demo.exe 执行 calc.exe 的代码。

将编译好的 Demo 文件拖入反汇编工具 IDA 中，可以看到在 IDA View-A(反汇编窗口)中得到反编译好的汇编语言。

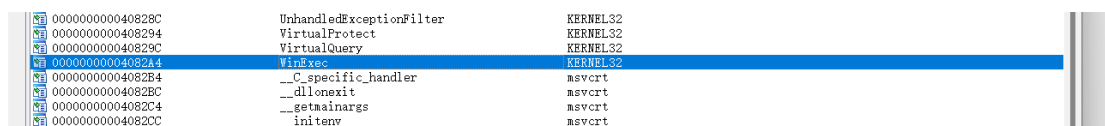
观察得到的汇编语言，可以轻松看出 Demo.exe 文件向 WinExec 的 API 传递了 calc 的参数，从而打开 windows 系统的计算器：

---

<sup>1</sup>IDA: <https://hex-rays.com/ida-free/#download>

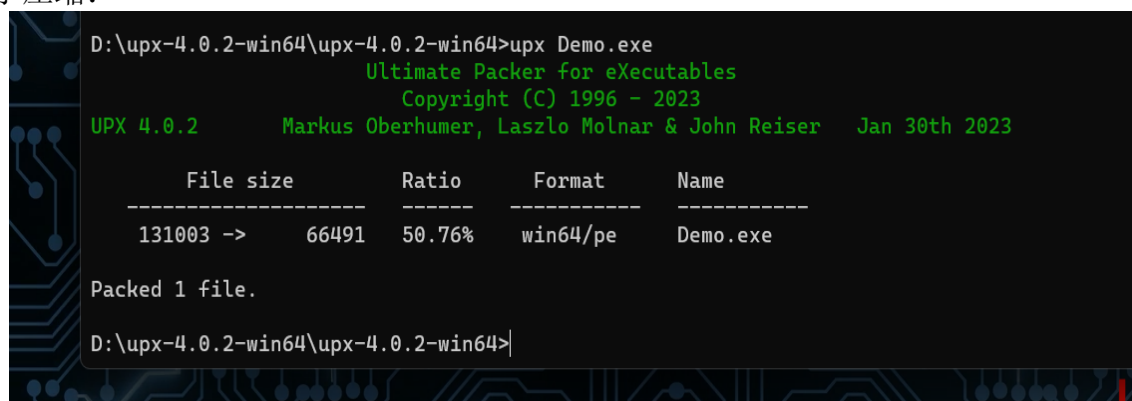


同时，在 import 模块窗口也可以看到，该代码调用了 WinExec 函数：



2.2 请先使用软件壳 UPX<sup>2</sup>对 Demo.exe 加壳，然后使用反汇编工具 IDA 分析加壳后 Demo.exe 的代码。本次的分析结果与 2.1 的分析结果相比，有什么不同？

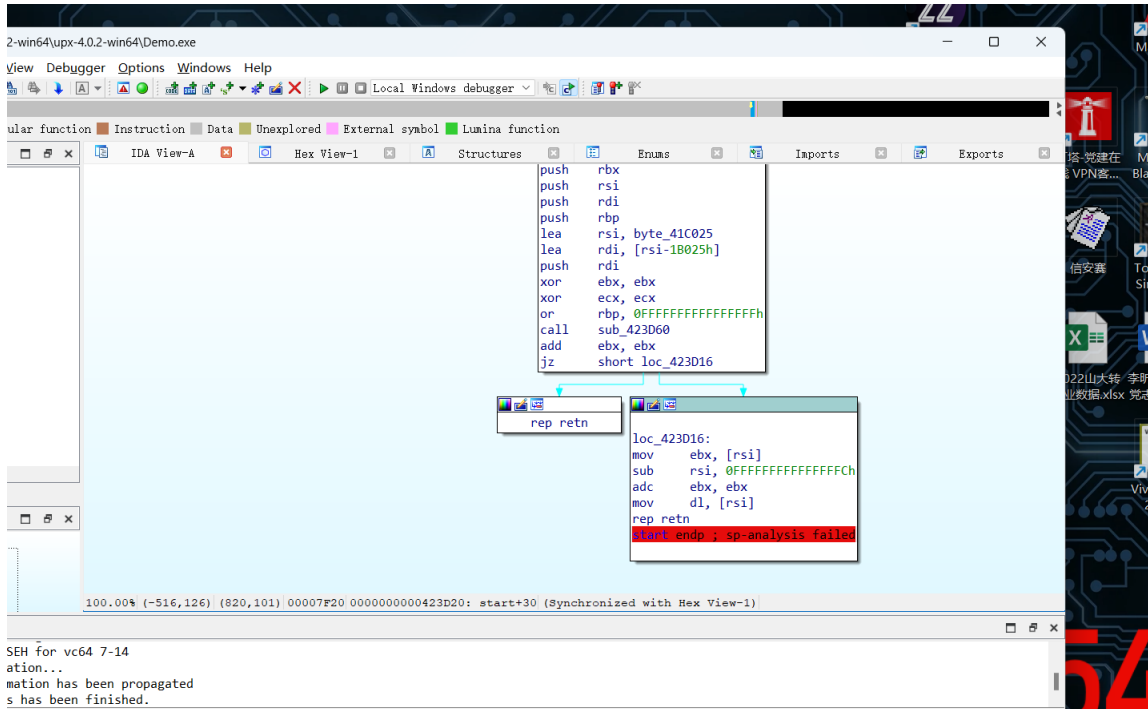
利用 UPX 对生成的 Demo 文件加壳，在加壳时可以看到 UPX 同时对其进行压缩：



将得到的加壳 Demo 文件再次拖入 IDA 软件进行静态分析，反编译后，软件提示 “The imports segment seems to be destroyed. This MAY mean that the file was

<sup>2</sup>UPX: <https://github.com/upx/upx/releases/tag/v4.0.2>

packed or otherwise modified in order to make it more difficult to analyze”，即软件检测到程序可能被加壳，进入反编译界面后可以看到 IDA 无法得到我们期望看到的调用计算器指令的汇编语言：



同时，查看调用的 API，可以看到无法检测到使用 API 情况：

Address	Ordinal	Name	Library
000000000042403C		LoadLibraryA	KERNEL32
0000000000424044		ExitProcess	KERNEL32
000000000042404C		GetProcAddress	KERNEL32
0000000000424054		VirtualProtect	KERNEL32
0000000000424064		exit	msvcrt

说明加壳后的 Demo 代码不能通过静态分析(反编译)来进行恶意软件检测。其具体原理分析部分在最后总结中进行分析 and 思考。

### 3. 动态分析

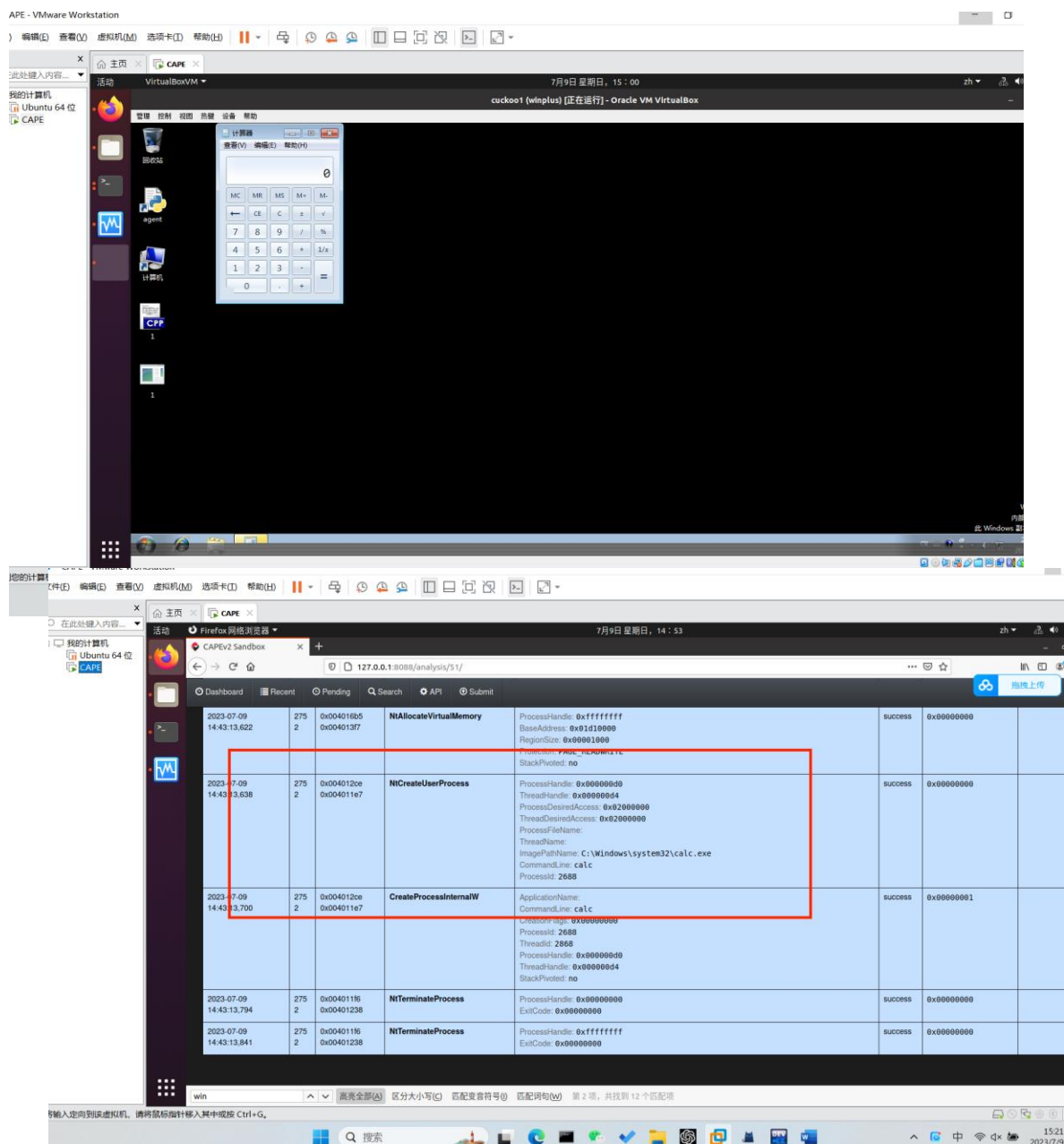
3.1 将加壳后的 Demo.exe 上传到在线 cape sandbox<sup>3</sup>，cape sandbox 是否能监控到 Demo.exe 的“WinExec”执行？

由于在线 cape sandbox 网站需要管理员开发访问权限，故略过此步。

<sup>3</sup> 在线版 cape sandbox: <https://capesandbox.com/>

3.2 在 Ubuntu 系统上安装本地的 cape sandbox, 安装方法参考官网<sup>4</sup>或百度的帖子。

参考 CAPE2 SANDBOX 官方文档搭建本地环境和虚拟沙盒, 通过安装依赖, 安装 cape, 配置沙盒虚拟机三个基本步骤后, 成功配置 capev2 sandbox 环境, 可以看到程序允许 Deom.exe 打开沙盒并在网页端得到报告:

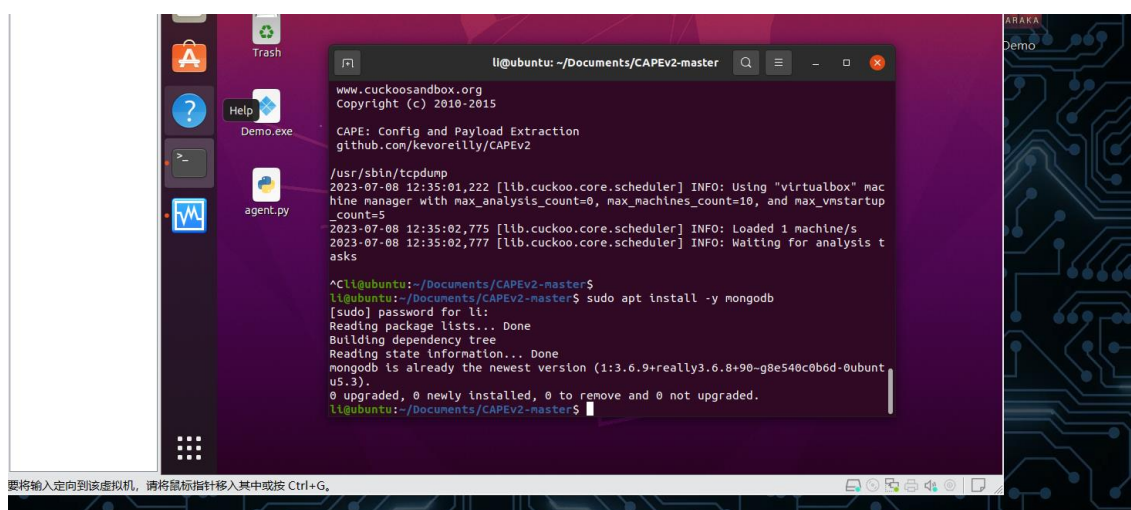


<sup>4</sup> 本地版 cape sandbox 的安装: <https://capev2.readthedocs.io/en/latest/installation/index.html>

同时，安装中我主要遇到以下五个问题：

### 问题一，安装 python 版本和某些模块版本时遇到了无法下载的问题：

安装 python 环境时，由于 python2 已经被基本弃用，许多指令无法运行，故修改文档要求的 pip 指令为 pip3 指令，安装 dpkt、jinja2 等必要依赖，特别注意的是，在安装数据库模块 MongoDB 时，提示找不到源，于是阅读 MongoDB 官方文档，发现目前即使 MongoDB 官方文档也不支持支持 22.04 ubuntu 的安装方式，经过尝试，按照官方文档安装也会报错，阅读 Github 问题讨论，发现这可能是由于较新的 Ubuntu 使用的是 openssl 3.0,而 mongodb 需要 1.1 版本导致了冲突，考虑到 capev2 sandbox 和其大部分依赖项的开发时间，决定改用适配性更高的 Ubuntu20.04 系统，成功安装 mongodb（图片为第二次安装，提示已安装）：



```
li@ubuntu: ~/Documents/CAPEv2-master
www.cuckoosandbox.org
Copyright (c) 2010-2015

CAPE: Config and Payload Extraction
github.com/kevoreilly/CAPEv2

/usr/sbin/tcpdump
2023-07-08 12:35:01,222 [lib.cuckoo.core.scheduler] INFO: Using "virtualbox" machine manager with max_analysis_count=0, max_machines_count=10, and max_vmstartup_count=5
2023-07-08 12:35:02,775 [lib.cuckoo.core.scheduler] INFO: Loaded 1 machine/s
2023-07-08 12:35:02,777 [lib.cuckoo.core.scheduler] INFO: Waiting for analysis tasks

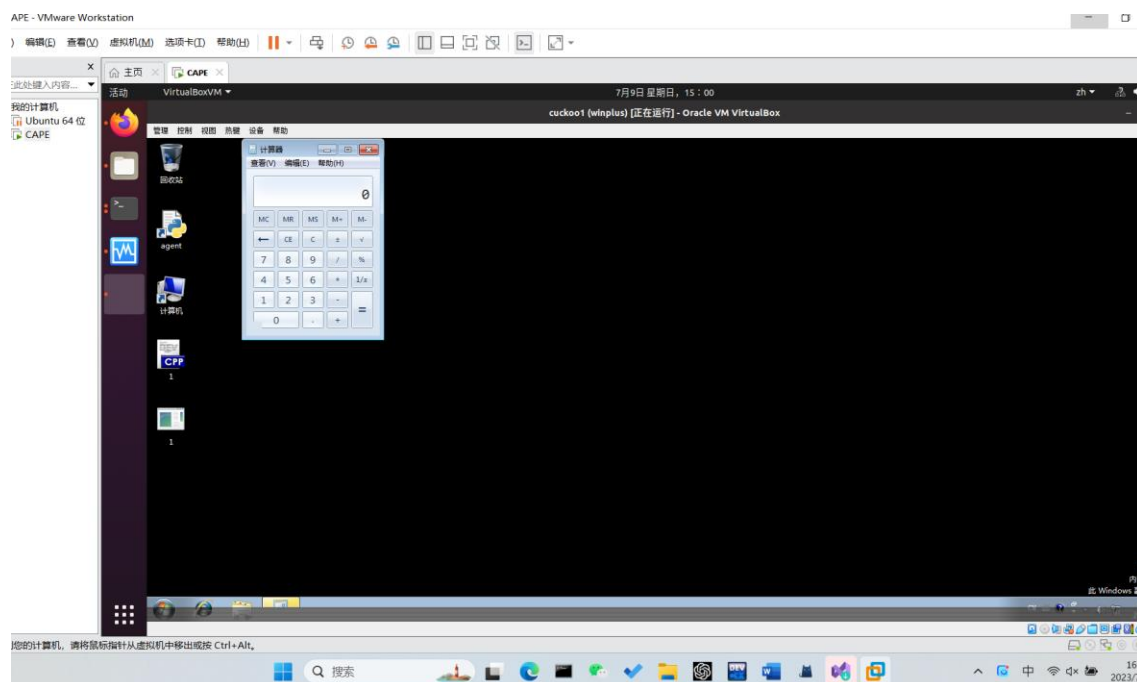
^C
li@ubuntu:~/Documents/CAPEv2-master$
li@ubuntu:~/Documents/CAPEv2-master$ sudo apt install -y mongodb
[sudo] password for li:
Reading package lists... Done
Building dependency tree
Reading state information... Done
mongodb is already the newest version (1:3.6.9+really3.6.8+90-g8e540c0b6d-0ubuntu5.3).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
li@ubuntu:~/Documents/CAPEv2-master$
```

之后依赖项 Virtualenv、pillow、tcpdump/pcap 等，在安装过程中，许多模块需要更换源才能顺利下载，步骤相对繁琐，同时，在此处，虽然官方推荐使用 Poetry 安装和管理 python 模块，考虑到对其命令并不熟悉，我在安装中均采取了基础的 pip3 命令。

### 问题二，虚拟沙盒 windows7 的环境配置问题：

当需要在客户机（win7）上启动 agent.py 文件进行监听，由于缺少环境，py 文件运行失败，故在 win7 系统中安装 python3 和 PIL（用于截屏分析恶意文件行为），在此处，英文文档提示可以安装部分软件，以提供完整的功能特性，而我没有理解其含义，故没有继续对虚拟机进行操作。

当之后 cpe 调用后端分析程序打开虚拟机时，出现虚拟机快速关闭或者闪出几秒缺失 VCRUNTIME140.dll 或者其他 dll 文件的错误窗口，经过检索得知这些.dll 文件是 win 系统运行的底层链接库，于是下载链接库修复补丁，修复 win7 系统缺失的 DLL 链接库，再次运行 Demo.exe，可以看到成功打开了虚拟机以及虚拟机中的计算器程序：



### 问题三：文件写权限缺失和 WEB 框架版本原因导致无法打开网页：

之前运行.sh 文件自动安装依赖时，大量依赖没有成功安装，导致此步频频报错，解决方法是反复运行 Cuckoo.py 文件和 Webserver.py 文件，针对其报错的缺失库进行安装(此方法针对 Cuckoo.py 缺失报错模块进行补充时较为有效，但是此处 Webserver.py 运行产生的报错均为网站框架 Django 的从属模块，分开安装导致版本无法匹配，后续界面显示异常)

运行 Webserver.py，终端显示缺失”web\_secret.key”模块，即在生成密钥的过程中失败，本来我认为是因为源码没有正确调用这个 secretkey 模块，try 失败，但是通过询问老师，老师提出这可能是 key 文件写权限缺失，因为运行本系统的用户为普通用户，原因是 cape 拒绝 root：

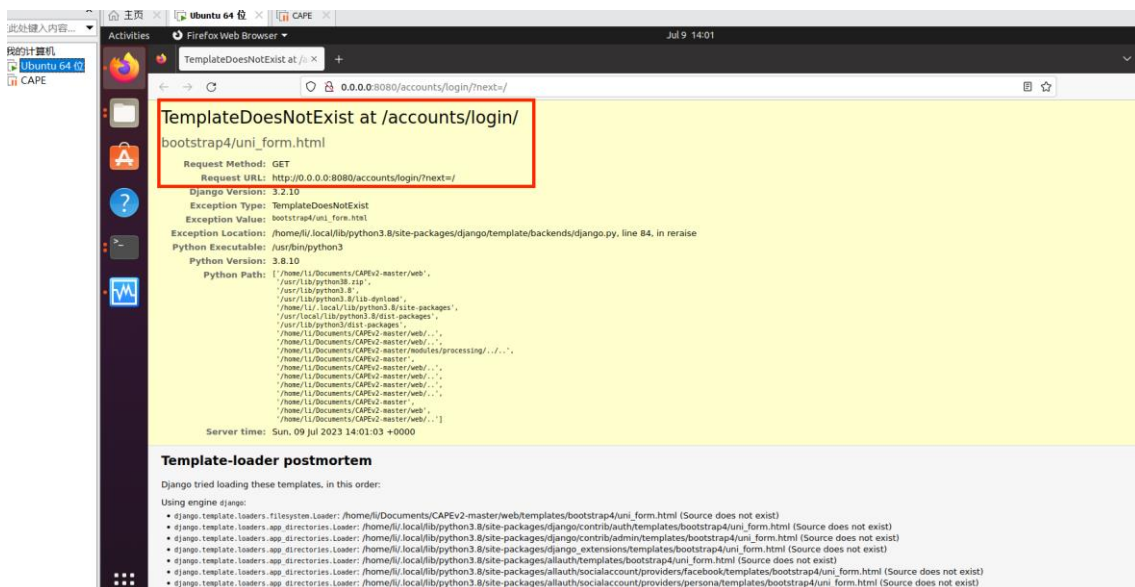


```
li@ubuntu:~/Documents/CAPEv2-master$ sudo python3 cuckoo.py
[sudo] password for li:
Root is not allowed. You gonna break permission and other parts of CAPE. RTM!
li@ubuntu:~/Documents/CAPEv2-master$
```

于是为其添加文档写权限，成功运行 WEB，通过端口 8080 进入 WEB 前端提交界面，但是新的问题是 WEB 界面显示缺失组件，无法进入 login 界面：

```
li@ubuntu:~/Documents/CAPEv2-master/web$ python3 manage.py runserver 0.0.0.0:8080
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (3 silenced).
July 09, 2023 - 13:59:46
Django version 3.2.10, using settings 'web.settings'
Starting development server at http://0.0.0.0:8080/
Quit the server with CONTROL-C.
```

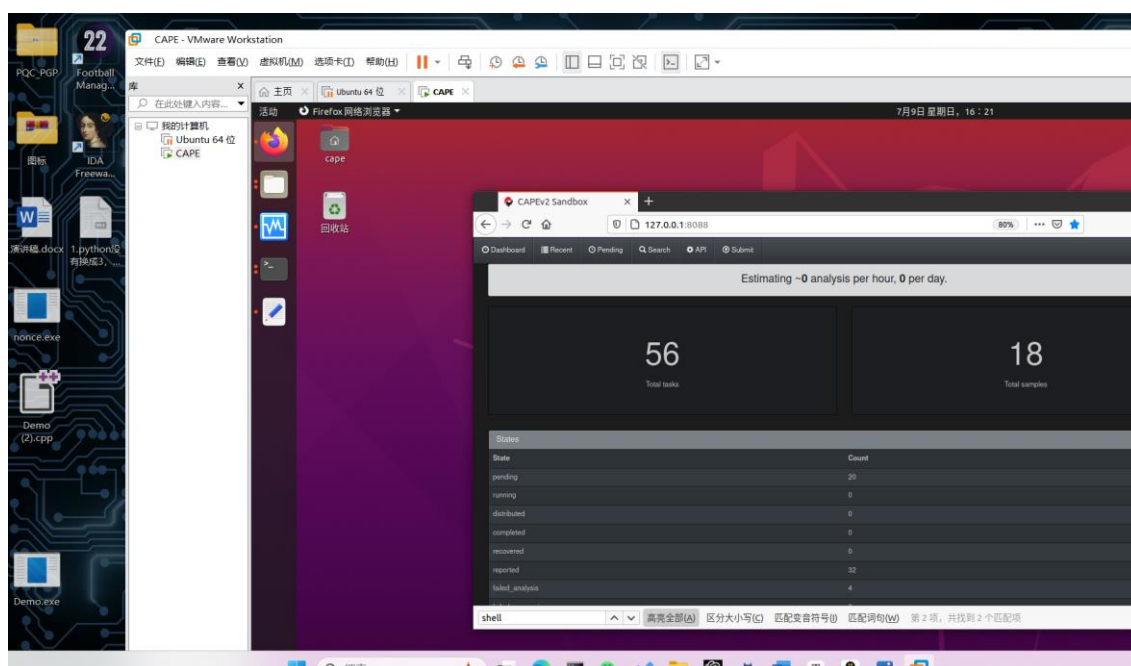


网上给出的解决方法是，重装 bootstrap5 模块，然后修改 cape 源代码，老师给出的建议是 cape 作为一个已发布的开源恶意软件检测框架，其结构功能较完善，不应出现需要修改源码解决错误的情况。遂继续思考发生该错误原因。

回想在运行.sh 文件的过程中，有许多 WARNING 警告，据分析，部分为依赖安装异常，虽然不影响后续的安装，但是在运行 cape 启动后端时，会出现大量依赖异常，虽然后续逐次安装，但通过阅读 Django 官方文档和 Github 中 issue 讨论，发现 Django 版本和 Python 版本具有对照关系，且各附加库对于版本也有要求，于是按照 python3.8 对应依次安装固定版本各 web 库，遂解决该问题，成功进入 Web 前端提交代码和查看分析包好的界面（截图是后来补充的，为了调试

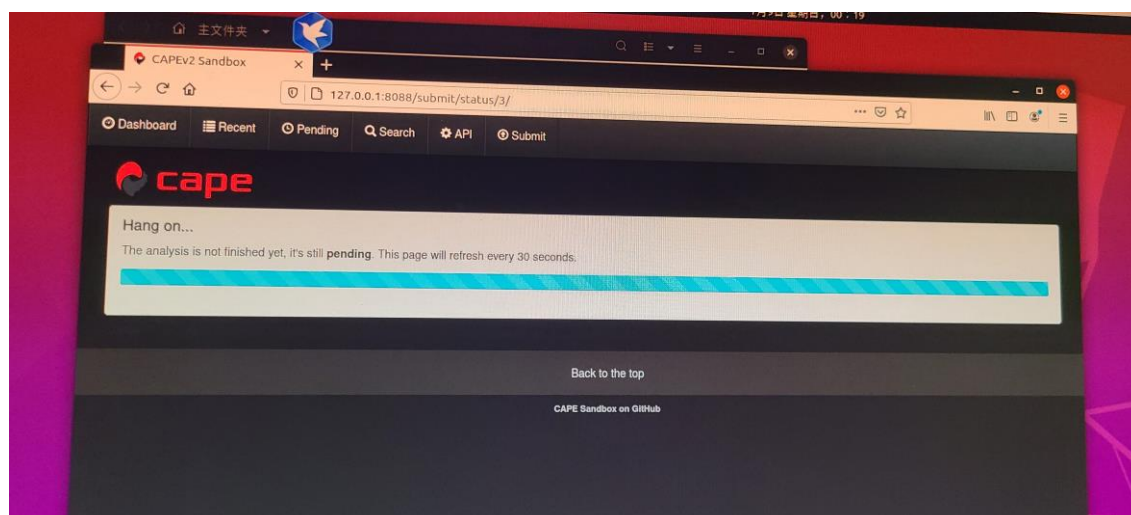


pending 问题上传了多个 exe，显示为 56 的任务，18 个生成成功）：



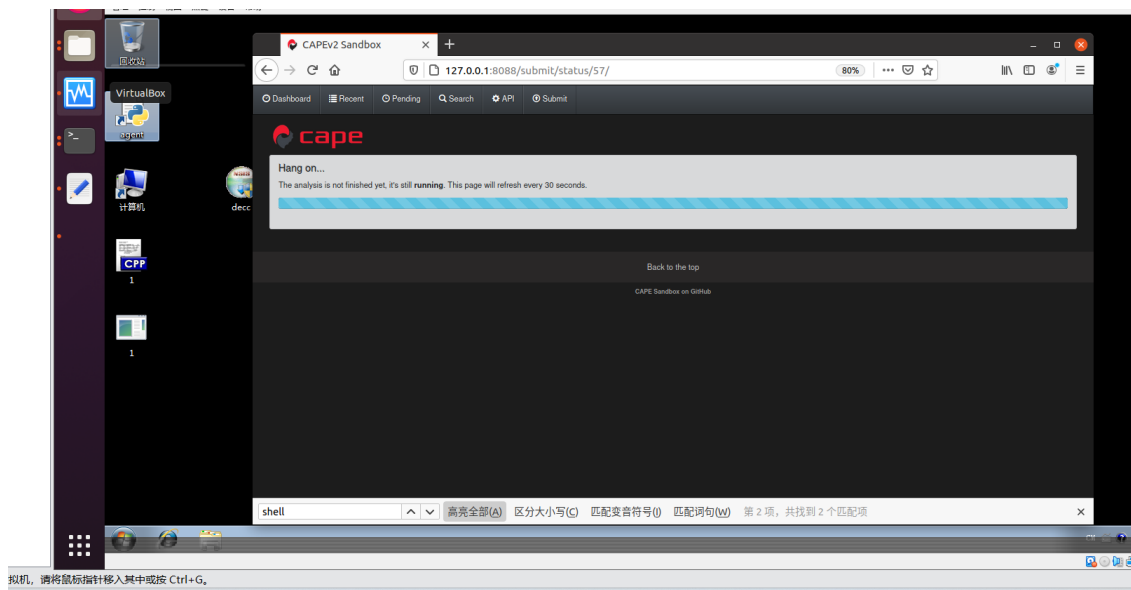
问题四，在网页提交的 exe 一直在等待队列 pending 问题：

当运行上传的 EXE 文件时，web 界面会一直停留在 pending 界面，即等待队列，却没有打开虚拟机运行，奇怪的是，上传.py 和.cpp 文件却可以分析：



尝试阅读 web 前端代码和 proecss.py 代码，没有发现 exe 一直处于 pending 状态的问题所在，既然无法在前端提交 exe，尝试利用后端直接调用虚拟机分析

exe，构造终端命令：【python3 ./utils/submit.py --timeout 60 --machine cuckoo1 /home/cape/sample/3.exe】，其中 machine cuckoo1 指定调用的沙盒 virtuaibox 虚拟机名称，timeout 60 指定超时时间（因为某些分析即使失败也不会退出，在调试发现会占用大量虚拟机时间），利用开发者编写的 submit.py 代码从终端提交，发现可以提交 exe，且成功打开虚拟机转入 running 状态，绕过前端提交 exe 不能 running 的问题：



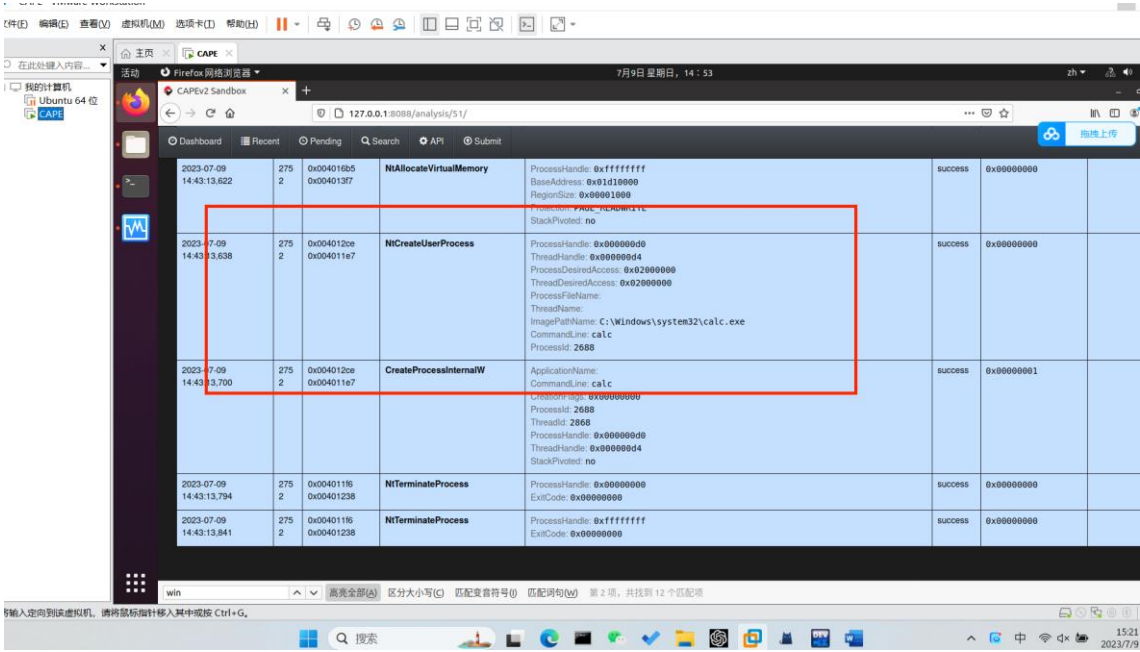
同时，在 WEB 段成功生成了检测报告，说明 capev2 sandbox 基本搭建完成。

**问题五，搭建好的检测系统检测不到 WinExecAPI 的问题：**

该问题考虑为默认 hook API 端口的问题，在 3.3 部分中讨论。

3.3 使用本地的 cape sandbox 重复 3.1 的实验，本地的 cape sandbox 能否监控到 Demo.exe 的行为？

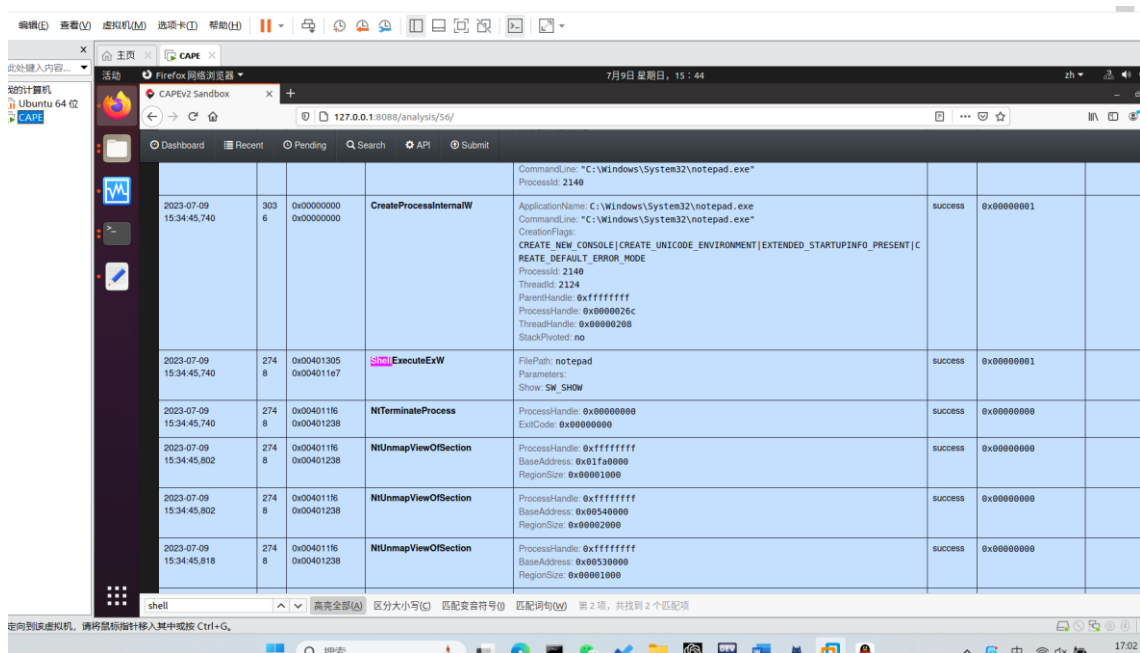
使用本地 cape sandbox 重复 3.1 的实验，可以看到报告中并没有检测到 WinExec 行为，但是检测到了 WinExec 之后创建进程打开 calc 的过程：



考虑是否是环境安装有问题，不能检测到 proeexec 中的 API，于是听从老师的建议，修改模拟恶意软件的源码，增加另一条调用记事本命令，ShellExecute，观察报告中是否有新的 API 接口出现：

```
#include<stdio>
#include<windows>
#include<shellapi.h>
int main()
{
    WinExec("calc",SW_SHOWNORMAL);
    ShellExecute(NULL,"open","notepad",NULL,NULL,SW_SHOW);
    return 0;
}
```

将新产生的 exe 提交到 capev2 sandbox 中，成功得到问题报告，可以看到在虚拟机中成功打开了计算器和记事本程序，同时在报告中出现了 ShellExecute 得 API 接口：



故猜测在默认的初始配置中，注释掉了 hook WinE 的代码，致使检测不到特定 API，将在 3.5 中予以验证

3.4 阅读 cape sandbox 源码，cape sandbox 的进程（process）监控模块<sup>5</sup>，共监控了哪些 API？

阅读 capemon 源码中的文件 hook.c，在 full\_hooks 中可看到其监控的模块：

```
// Process Hooks
```

1. HOOK(ntdll, NtAllocateVirtualMemory),
2. HOOK(ntdll, NtAllocateVirtualMemoryEx),
3. HOOK(ntdll, NtReadVirtualMemory),
4. HOOK(kernel32, ReadProcessMemory),
5. HOOK(ntdll, NtWriteVirtualMemory),
6. HOOK(kernel32, WriteProcessMemory),
7. HOOK(ntdll, NtWow64WriteVirtualMemory64),
8. HOOK(ntdll, NtWow64ReadVirtualMemory64),
9. HOOK(ntdll, NtProtectVirtualMemory),
10. HOOK(kernel32, VirtualProtectEx),
11. HOOK(ntdll, NtFreeVirtualMemory),
12. HOOK(ntdll, NtCreateProcess),
13. HOOK(ntdll, NtCreateProcessEx),
14. HOOK(ntdll, RtlCreateUserProcess),

---

5 进程（process）监控模块：hook\_process.c

```

15. HOOK(ntdll, NtOpenProcess),
16. HOOK(ntdll, NtTerminateProcess),
17. HOOK(ntdll, RtlReportSilentProcessExit),
18. HOOK(ntdll, NtResumeProcess),
19. HOOK(ntdll, NtCreateSection),
20. HOOK(ntdll, NtDuplicateObject),
21. HOOK(ntdll, NtMakeTemporaryObject),
22. HOOK(ntdll, NtMakePermanentObject),
23. HOOK(ntdll, NtOpenSection),
24. HOOK(ntdll, NtMapViewOfSection),
25. HOOK(ntdll, NtMapViewOfSectionEx),
26. HOOK(ntdll, NtUnmapViewOfSection),
27. HOOK(ntdll, NtUnmapViewOfSectionEx),
28. HOOK(kernel32, WaitForDebugEvent),
29. HOOK(ntdll, DbgUiWaitStateChange),
30. HOOK(advapi32, CreateProcessWithLogonW),
31. HOOK(advapi32, CreateProcessWithTokenW),
32. HOOK(kernel32, CreateToolhelp32Snapshot),
33. HOOK(kernel32, Process32FirstW),
34. HOOK(kernel32, Process32NextW),
35. HOOK(kernel32, Module32FirstW),
36. HOOK(kernel32, Module32NextW),
37. HOOK(kernel32, CreateProcessA),
38. HOOK(kernel32, CreateProcessW),
39. HOOK(kernel32, WinExec),
40. //HOOK(kernel32, VirtualFreeEx),
41. // all variants of ShellExecute end up in ShellExecuteExW
42. HOOK(shell32, ShellExecuteExW),
    43.     HOOK(msvcrt, system),

```

其调用的系统 API 包括 WinExec, ShellExecute, system。

同时，各监控模块函数具体实现可以在 hook\_process.c 中查看到，针对该部分代码，将在思考讨论中进行分析。

### 3.5 修改 cape sandbox 源码，使得它的进程（process）监控功能只监控一个 API “WinExec”，不再监控其它 API。

提示：Step1. 下载 cape sandbox 的组件 capemon<sup>6</sup>的代码到本地。

Step2 参考官方文档 “How to add hooks to capemon<sup>7</sup>” 修改 capemon 源码；编译生成

---

<sup>6</sup> Capemon: <https://github.com/kevoreilly/capemon>

<sup>7</sup> “How to add hooks to capemon”: <https://github.com/kevoreilly/capemon>

新的 capemon.dll。

Step3: 参考文档 “How to compile capemon<sup>8</sup>” 用新的 capemon.dll 替换掉原 capemon.dll。

安装官方文档的说明，下载 capemon 组件的源码，修改 hook.c 文件中的 full\_hook 对 process 的调用，需要修改的代码部分与 3.4 中提到的代码相同（另外注释掉 special API 部分），将其全部注释掉：

```
//HOOK(kernel32, WaitForDebugEvent),
//HOOK(ntdll, DbgUiWaitStateChange),
//HOOK(advapi32, CreateProcessWithLogonW),
//HOOK(advapi32, CreateProcessWithTokenW),
//HOOK(kernel32, CreateToolhelp32Snapshot),
//HOOK(kernel32, Process32FirstW),
//HOOK(kernel32, Process32NextW),
//HOOK(kernel32, Module32FirstW),
//HOOK(kernel32, Module32NextW),
//HOOK(kernel32, CreateProcessA),
//HOOK(kernel32, CreateProcessW),
HOOK(kernel32, WinExec), //只保留这个函数的声明
//使调用时不会显示其他HOOK函数接口
//HOOK(kernel32, VirtualFreeEx),
// all variants of ShellExecute end up in ShellExecuteExW
//HOOK(shell32, ShellExecuteExW),
//HOOK(msvcrt, system),
```

通过安装 VS2017 工具集，成功编译该代码，得到 DLL 文件：

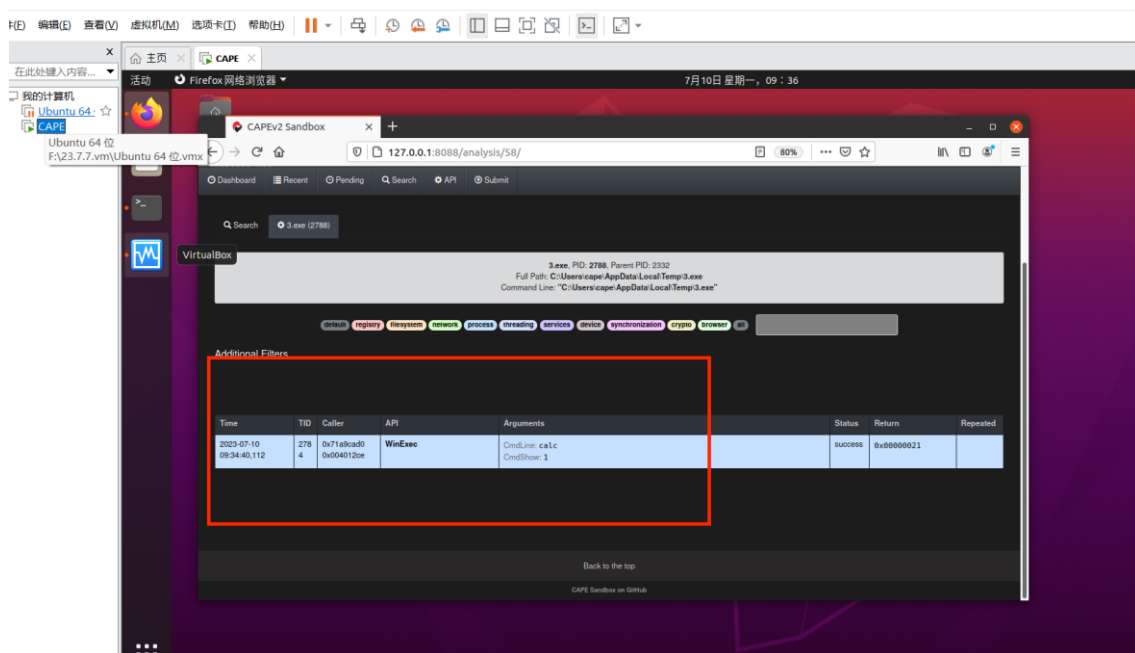
名称	修改日期	类型	大小
今天			
capemon.dll	2023/7/10 9:18	应用程序扩展	2,384 KB
capemon_x64.dll	2023/7/10 9:18	应用程序扩展	3,040 KB
昨天			

将 DLL 放入指定路径，可以看到界面只显示了指定 API-WinExec，同时也证明了 3.4 猜想,即安装的 cape sandbox 初始默认不检测 WinExec API:

---

<sup>8</sup> “How to add hooks to capemon”: <https://github.com/kevoreilly/capemon>





### 三. 实验总结

通过本次实验对恶意代码的静态和动态分析，我初步学会了如何对一个可疑文件进行分析和检测，同时，通过手动搭建本地沙盒检测环境以及修改其源代码，基本了解了 cape sandbox 的运行机制和原理。在手动搭建本地环境的过程中，通过实操熟悉了 linux 系统的基本用法和常见权限/下载错误，了解了 linux 与 windows 在操作等方面的区别。

在操作过程中，我思考了 IDA 静态分析的原理，了解到可执行二进制文件（.exe）中的二进制数据为计算机可以直接执行的机器语言指令，反编译软件如 IDA 通过分析.exe 的二进制代码，推测其汇编语言和结构。但是通过前面的实验可以发现，IDA 的反编译功能并非完全精确，它只能还原出代码的静态部分，对于加壳后的 exe，则无法有效剔除干扰。

那么很自然的可以想到，加壳是怎样干扰静态编译的呢？通过 UPX 的官方 Github 文档和网络资料，了解到基础方法为通过对指令进行编码或者引入无效的指令来干扰反汇编器的准确性，改变函数和变量的名称、重新排列指令的顺序、添加无关代码，使反编译器无法准确找到真实指令。同时我了解到，部分加壳程序还可以使用加解密的方法避免反编译，即将二进制代码加密，只要运行时才可以解密运行；或者采取防调试与反反汇编，即检测到反编译时通过修改代码执行路径等方式避免被分析。分析之前 IDA 的报错信息，可以发现应该是被反反编译

功能提前终止了反编译进程，所以得到 analysis failed。

之后，尝试分析 sanbox 是如何捕获 API 调用的，尝试阅读 hook\_prosess.c 的源代码，以本次实验的 API 对象 WinExec 为例，分析 HOOK\_WinExec，其代码如下：

```
HOOKDEF(UINT, WINAPI, WinExec,  
    __in LPCSTR lpCmdLine,  
    __in UINT    uCmdShow  
) {  
    UINT ret = Old_WinExec(lpCmdLine, uCmdShow);  
    LOQ_nonzero("process", "si", "CmdLine", lpCmdLine, "CmdShow", uCmd  
Show);  
    return ret;  
}
```

首先了解到，Hook 是一种修改函数行为的技术，可以在函数执行前后注入自定义逻辑，该代码使用了 HOOKDEF 宏来定义 WinExec 的 Hook，在函数体内部，首先调用了 Old\_WinExec 函数，该函数是保存了原始 WinExec 函数的指针，用于调用原始函数。

然后，在调用原始函数之后，通过 LOQ\_nonzero 函数将执行命令行和窗口显示方式作为参数输出日志。最后，将原始函数的返回值作为本函数的返回值返回父函数。当源程序调用 HOOK 对应的 API 时，HOOK 函数可以捕获这个 API 调用的日志等信息，并发送给 log 存储，实现捕获事件。

最后，是一些对于学习计算机和信息安全过程的收获，通过本课程的学习，除了学习到了许多 web 渗透和恶意软件分析的实用技能，同时也在培养发现问题分析问题解决问题的能力，在本次实验中，出现了一系列的问题，但是思考分析解决问题的过程比搭建环境本身更让我理解了 cape sandbox 的逻辑/架构/依赖库等知识，在解决问题的过程中，我阅读了 cape sandbox 的代码，了解了其运行的基本原理，明白了老师为什么让我们从头编译一遍 capev2 sandbox 环境，这个过程虽然很困难，但是只要从头开始，一步一步的推进，才能做到对一个项目完全理解并掌握。