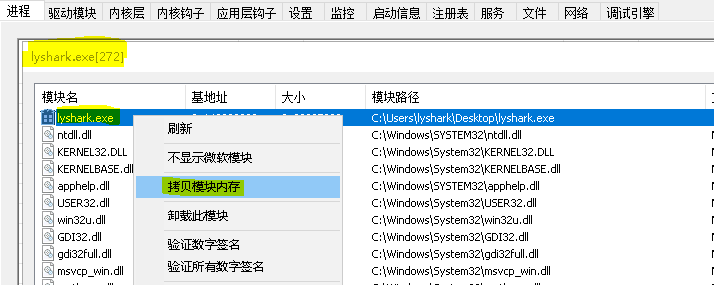
# 多数ARK反内核工具中都存在驱动级别的内存转存功能，该功能可以将应用层中运行进程的内存镜像转 存到特定目录下，内存转存功能在应对加壳程序的分析尤为重要，当进程在内存中解码后，我们可以很 容易的将内存镜像导出，从而更好的对样本进行分析，当然某些加密壳可能无效但绝大多数情况下是可 以被转存的。



在上一篇文章 《驱动开发：内核R3与R0内存映射拷贝》 介绍了一种方式 SafeCopyMemory\_R3\_to\_R0 可以将应用层进程的内存空间映射到内核中，要实现内存转储功能我们还是需要使用这个映射函数，只是需 要在此函数上增加一些功能而已。

在实现转存之前，需要得到两个东西，进程内 模块基地址 以及 模块长度 这两个参数是必不可少的，至于内核中如何得到指定进程的模块数据，在很早之前的文章 《驱动开发：内核中枚举进线程与模块》 中有详细的参考方法，这里就在此基础之上实现一个简单的进程模块遍历功能。

如下代码中使用的就是 枚举 进程 PEB 结构得到更多参数的具体实现，如果不懂得可以研读 《驱动开发：

内核通过PEB得到进程参数》 这篇文章此处不再赘述。



#include <ntddk.h> #include <windef.h>

// 声明结构体

typedef struct \_KAPC\_STATE

{

LIST\_ENTRY ApcListHead[2]; PKPROCESS Process;

UCHAR KernelApcInProgress; UCHAR KernelApcPending; UCHAR UserApcPending;

} KAPC\_STATE, PKAPC\_STATE;

typedef struct \_LDR\_DATA\_TABLE\_ENTRY

{

LIST\_ENTRY64 InLoadOrderLinks; LIST\_ENTRY64 InMemoryOrderLinks; LIST\_ENTRY64 InInitializationOrderLinks; PVOID DllBase;

PVOID EntryPoint;

ULONG SizeOfImage; UNICODE\_STRING FullDllName; UNICODE\_STRING BaseDllName;

ULONG Flags;



USHORT LoadCount;

USHORT TlsIndex;

PVOID SectionPointer;

ULONG CheckSum;

PVOID LoadedImports;

PVOID EntryPointActivationContext;

PVOID PatchInformation; LIST\_ENTRY64 ForwarderLinks; LIST\_ENTRY64 ServiceTagLinks; LIST\_ENTRY64 StaticLinks;

PVOID ContextInformation;

ULONG64 OriginalBase; LARGE\_INTEGER LoadTime;

} LDR\_DATA\_TABLE\_ENTRY, PLDR\_DATA\_TABLE\_ENTRY;

// 偏移地址

ULONG64 LdrInPebOffset = 0x018; //peb.ldr

ULONG64 ModListInPebOffset = 0x010; //peb.ldr.InLoadOrderModuleList

// 声明API

NTKERNELAPI UCHAR PsGetProcessImageFileName(IN PEPROCESS Process); NTKERNELAPI PPEB PsGetProcessPeb(PEPROCESS Process);

NTKERNELAPI HANDLE PsGetProcessInheritedFromUniqueProcessId(IN PEPROCESS Process);

// 根据进程ID返回进程EPROCESS，失败返回NULL PEPROCESS LookupProcess(HANDLE Pid)

{

PEPROCESS eprocess = NULL;

if (NT\_SUCCESS(PsLookupProcessByProcessId(Pid, &eprocess))) return eprocess;

else

return NULL;

}

// 枚举指定进程的模块

// By: LyShark.com

VOID EnumModule(PEPROCESS Process)

{

SIZE\_T Peb = 0; SIZE\_T Ldr = 0;

PLIST\_ENTRY ModListHead = 0; PLIST\_ENTRY Module = 0; ANSI\_STRING AnsiString; KAPC\_STATE ks;

// EPROCESS地址无效则退出

if (!MmIsAddressValid(Process)) return;

// 获取PEB地址

Peb = (SIZE\_T)PsGetProcessPeb(Process);

// PEB地址无效则退出

if (!Peb)



return;

// 依附进程

KeStackAttachProcess(Process, &ks);

try

{

// 获得LDR地址

Ldr = Peb + (SIZE\_T)LdrInPebOffset;

// 测试是否可读，不可读则抛出异常退出

ProbeForRead((CONST PVOID)Ldr, 8, 8);

// 获得链表头

ModListHead = (PLIST\_ENTRY)( (PULONG64)Ldr + ModListInPebOffset);

// 再次测试可读性

ProbeForRead((CONST PVOID)ModListHead, 8, 8);

// 获得第一个模块的信息

Module = ModListHead->Flink;

while (ModListHead != Module)

{

//打印信息：基址、大小、DLL路径

DbgPrint("模块基址 = %p | 大小 = %ld | 模块名 = %wZ | 完整路径= %wZ \n", (PVOID)(((PLDR\_DATA\_TABLE\_ENTRY)Module)->DllBase),

(ULONG)(((PLDR\_DATA\_TABLE\_ENTRY)Module)->SizeOfImage),

&(((PLDR\_DATA\_TABLE\_ENTRY)Module)->BaseDllName), &(((PLDR\_DATA\_TABLE\_ENTRY)Module)->FullDllName)

);

Module = Module->Flink;

// 测试下一个模块信息的可读性

ProbeForRead((CONST PVOID)Module, 80, 8);

}

}

except (EXCEPTION\_EXECUTE\_HANDLER){ ; }

// 取消依附进程

KeUnstackDetachProcess(&ks);

}

VOID DriverUnload(IN PDRIVER\_OBJECT DriverObject)

{

}

NTSTATUS DriverEntry(IN PDRIVER\_OBJECT DriverObject, IN PUNICODE\_STRING RegistryPath)

{

DbgPrint("hello lyshark.com \n");

ULONG i = 0;

PEPROCESS eproc = NULL;

for (i = 4; i<100000000; i = i + 4)

{

eproc = LookupProcess((HANDLE)i); if (eproc != NULL)

{

ObDereferenceObject(eproc);

if (strstr(PsGetProcessImageFileName(eproc), "lyshark.exe") != NULL)

{

EnumModule(eproc);

}

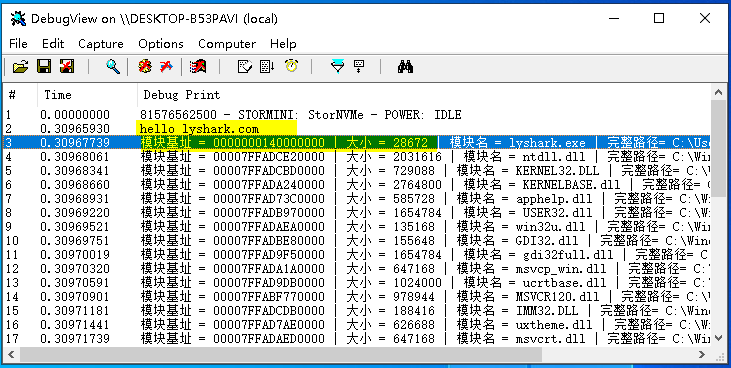
}

}

DriverObject->DriverUnload = DriverUnload; return STATUS\_SUCCESS;

}

如上我们指定获取应用层 lyshark.exe 进程的模块信息，并可得到以下输出效果:



# 上篇文章中的代码就不再啰嗦了，这里只给出内存转存的核心代码，如下代码：

RtlInitUnicodeString 用于初始化转存后的名字字符串

ZwCreateFile 内核中创建文件到应用层

ZwWriteFile 将文件写出到文件

ZwClose 最后是关闭文件并释放堆空间

很简单只是利用了 SafeCopyMemory\_R3\_to\_R0 将进程内存读取到缓冲区内，并将缓冲区写出到C盘目录下。

// 进程内存拷贝函数

// By: LyShark.com

NTSTATUS ProcessDumps(PEPROCESS pEprocess, ULONG\_PTR nBase, ULONG nSize)

{

BOOLEAN bAttach = FALSE; KAPC\_STATE ks = { 0 };

PVOID pBuffer = NULL;

NTSTATUS status = STATUS\_UNSUCCESSFUL;

if (nSize == 0 || pEprocess == NULL)

{

return status;

}

pBuffer = ExAllocatePoolWithTag(PagedPool, nSize, 'lysh'); if (!pBuffer)



{

return status;

}

memset(pBuffer, 0, nSize);

if (pEprocess != IoGetCurrentProcess())

{

KeStackAttachProcess(pEprocess, &ks); bAttach = TRUE;

}

status = SafeCopyMemory\_R3\_to\_R0(nBase, (ULONG\_PTR)pBuffer, nSize);

if (bAttach)

{

KeUnstackDetachProcess(&ks); bAttach = FALSE;

}

OBJECT\_ATTRIBUTES object; IO\_STATUS\_BLOCK io;

HANDLE hFile; UNICODE\_STRING log;

// 导出文件名称

RtlInitUnicodeString(&log, L"\\??\\C:\\lyshark\_dumps.exe"); InitializeObjectAttributes(&object, &log, OBJ\_CASE\_INSENSITIVE, NULL, NULL);

status = ZwCreateFile(&hFile, GENERIC\_WRITE,

&object, &io, NULL,

FILE\_ATTRIBUTE\_NORMAL, FILE\_SHARE\_WRITE, FILE\_OPEN\_IF, FILE\_SYNCHRONOUS\_IO\_NONALERT, NULL,

0);

if (!NT\_SUCCESS(status))

{

DbgPrint("打开文件错误 \n"); return STATUS\_SUCCESS;

}

ZwWriteFile(hFile, NULL, NULL, NULL, &io, pBuffer, nSize, NULL, NULL);

DbgPrint("写出字节数: %d \n", io.Information); DbgPrint("[ ] LyShark.exe 已 转 存 "); ZwClose(hFile);

if (pBuffer)

{

ExFreePoolWithTag(pBuffer, 'lysh'); pBuffer = NULL;

}

return status;

}

VOID UnDriver(PDRIVER\_OBJECT driver)

{

DbgPrint(("Uninstall Driver Is OK \n"));

}

// lyshark.com

NTSTATUS DriverEntry(IN PDRIVER\_OBJECT Driver, PUNICODE\_STRING RegistryPath)

{

DbgPrint("hello lyshark.com \n");

NTSTATUS ntStatus;

PEPROCESS pCurProcess = NULL;

try

{

ntStatus = PsLookupProcessByProcessId((HANDLE)272, &pCurProcess); if (NT\_SUCCESS(ntStatus))

{

// 设置基地址以及长度

ntStatus = ProcessDumps(pCurProcess, 0x140000000, 1024); ObDereferenceObject(pCurProcess);

}

}

except (1)

{

ntStatus = GetExceptionCode();

}

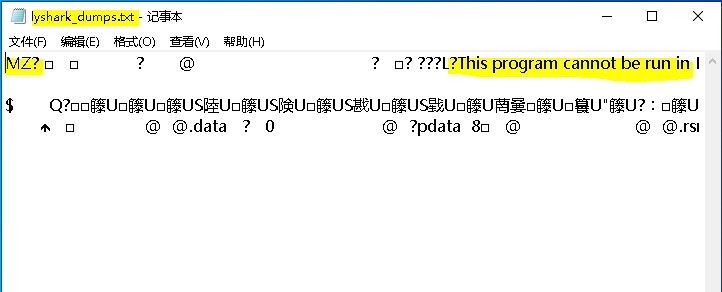
Driver->DriverUnload = UnDriver; return STATUS\_SUCCESS;

}

# 转存后效果如下所示：



至于导出的进程无法运行只是没有修复而已(后期会讲)，可以打开看看是没错的。



本书作者： 王瑞 (LyShark)

作者邮箱： m [e@lyshark.com](mailto:e@lyshark.com)

作者博客： h ttps://lyshark.cnblogs.com

团队首页： w ww.lyshark.com