操作系统安全实验

从内存搞到某个应用进程使用的密钥

姓名：孔令伟	学号：201500301267	班级：大三三班
实验日期：2018/04/07	实验学时：2	实验题目：从内存获得进程信息

一. *背景*

内存是计算机中重要的部件之一，它是与 CPU 进行沟通的桥梁。计算机中所有程序的运行都是在内存中进行的，因此内存的性能对计算机的影响非常大。内存(Memory)也被称为内存储器，其作用是用于暂时存放 CPU 中的运算数据，以及与硬盘等外部存储器交换的数据。只要计算机在运行中，CPU 就会把需要运算的数据调到内存中进行运算，当运算完成后 CPU 再将结果传送出来，内存的运行也决定了计算机的稳定运行。

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位.线程是进程的一个实体,是 CPU 调度和分派的基本单位,它是比进程更小的能独立运行的基本单位.线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源.

二. *技术或漏洞实现*

首先我们需要知道某个程序进程的 ID，从而对该进程进行信息的查看，具体的方法我们可以通过参考百度百科 https://jingyan.baidu.com/article/647f0115cd62977f2148a88d.html 来实现。

1）尝试使用编程实现

1. 切换目标进程的 CR3

通常,跨进程读写内存,用到 ReadProcessMemory，WriteProcessMemory，但需要进程句柄,如果目标进程受到保护,可能获得进程句柄会失败.

ReadProcessMemory 最后会调用到 KeStackAttachProcess 附加到目标进程上切换进程环境进行拷贝的，所以想到拿到目标进程的虚拟内存内容,可以将目标进程的页目录基地址放入 CR3 中即可.

首先要获得目标进程的 cr3 寄存器,即页目录基地址(开启 PAE，页目录指针表),

每个进程在内核里都有一个 EPROCESS 结构.

```
nt!_EPROCESS
+0x000 Pcb : _KPROCESS
+0x06c ProcessLock : _EX_PUSH_LOCK
+0x070 CreateTime : _LARGE_INTEGER
+0x078 ExitTime : _LARGE_INTEGER
+0x080 RundownProtect : _EX_RUNDOWN_REF
```

Pcb 中就有我们想要得到的 CR3

```
nt!_KPROCESS
+0x000 Header : _DISPATCHER_HEADER
+0x010 ProfileListHead : _LIST_ENTRY
+0x018 DirectoryTableBase : [2] Uint4B
+0x020 LdtDescriptor : _KGDTENTRY
+0x028 Int21Descriptor : _KIDTENTRY
```

那只需要获得目标进程 EPROCESS 就可以得到 CR3 了

遍历 EPROCESS 里的 ActiveProcessLinks 的链表获取指定进程的 EPROCESS

```
// 获得当前进程 EPROCESS 信息
```

```
ULONG uEprocess = 0;
__asm
{
	mov eax, fs:[0x124] // _ethread
	mov eax, [eax+0x44] // _kprocess
	mov uEprocess, eax
}

KdPrint(("EPROCESS: 0x%08x\n", uEprocess));
LIST_ENTRY ListHead;
InitializeListHead(&ListHead);

ULONG uFirstEprocess = uEprocess;
ULONG uCount = 0;
PLIST_ENTRY pActiveProcessLinks;
ProcessInfoList *pProcssList = NULL;

ULONG uNameOffset = GetPlantformDependentInfo(FILE_NAME_OFFSET);
ULONG uPidOffset = GetPlantformDependentInfo(PROCESS_ID_OFFSET);
ULONG uLinkOffset = GetPlantformDependentInfo(PROCESS_LINK_OFFSET);
ULONG uExitTime = GetPlantformDependentInfo(EXIT_TIME_OFFSET);
// 遍历链表获得进程信息
do {
	pProcssList= (ProcessInfoList *)ExAllocatePool(PagedPool, sizeof(ProcessInfoList));
	if (pProcssList == NULL){
		status = STATUS_INSUFFICIENT_RESOURCES;
		break;
	}
	PLARGE_INTEGER ExitTime;
	ExitTime = (PLARGE_INTEGER)(uEprocess + uExitTime);
	if (ExitTime->QuadPart == 0){
		if (*(int *)(uEprocess + uPidOffset) <= 0){
			pProcssList->ProcInfo.uProcessId = 0;
			pProcssList->ProcInfo.uEprocess = uEprocess;
			pProcssList->ProcInfo.uCR3 = *(PULONG)(uEprocess + 0x18);
			RtlCopyMemory(pProcssList->ProcInfo.pszImageFileName, "Idle", 16);
			InsertHeadList(&ListHead, &pProcssList->ListEntry);
			KdPrint(("PID: %d, EPROCESS: 0x%08x, FileName: %s, CR3: 0x%08x\n",
			pProcssList->ProcInfo.uProcessId,
			pProcssList->ProcInfo.uEprocess,
			pProcssList->ProcInfo.pszImageFileName,
			pProcssList->ProcInfo.uCR3));
		}else{
			pProcssList->ProcInfo.uEprocess = uEprocess;
```

```
                pProcssList->ProcInfo.uCR3 = *(PULONG)(uEprocess + 0x18);
                pProcssList->ProcInfo.uProcessId = *(PULONG)(uEprocess + uPidOffset);
                RtlCopyMemory(pProcssList->ProcInfo.pszImageFileName,
                        (PVOID)(uEprocess+uNameOffset), 16);
                InsertHeadList(&ListHead, &pProcssList->ListEntry);
                KdPrint(("PID: %d, EPROCESS: 0x%08x, FileName: %s, CR3: 0x%08x\n",
                pProcssList->ProcInfo.uProcessId,
                pProcssList->ProcInfo.uEprocess,
                pProcssList->ProcInfo.pszImageFileName,
                pProcssList->ProcInfo.uCR3));
            }
            uCount++;
        }
        pActiveProcessLinks = (PLIST_ENTRY)(uEprocess + uLinkOffset);
        uEprocess = (ULONG)pActiveProcessLinks->Blink - uLinkOffset;
        if (uEprocess == uFirstEprocess){
            break;
        }
} while (uEprocess != 0);
```

然后是读写内存：

```
_try
{
        WriteMemoryInfo *pInfo =
            (WriteMemoryInfo *)ExAllocatePool(PagedPool, sizeof(WriteMemoryInfo));
        RtlCopyMemory(pInfo, pIoBuffer, sizeof(WriteMemoryInfo));
        PVOID pWrite = ExAllocatePool(PagedPool, pInfo->nWriteSize);
        RtlCopyMemory(pWrite, pInfo->pData, pInfo->nWriteSize);
        //pInfo->pData = (PBYTE)ExAllocatePool(PagedPool, pInfo->nWriteSize);
        ULONG uOldCr3 = 0;
        ULONG uCurrentCr3 = *(PULONG)(pInfo->nEprocess + 0x18);
        if (pInfo->nMemoryAddr == 0){
            status = STATUS_UNSUCCESSFUL;
            break;
        }
        __asm{
            mov eax, cr3
            mov uOldCr3, eax
            mov eax, uCurrentCr3
            mov cr3, eax
        }
        KIRQL oldIrql;
        oldIrql = KeRaiseIrqlToDpcLevel();
        __asm{
```

```
            cli
            push eax
            mov eax, cr0
            and eax, not 10000h
            mov cr0, eax
        }
        RtlCopyMemory((PVOID)pInfo->nMemoryAddr,
        pWrite, pInfo->nWriteSize);
        __asm{
            mov eax, CR0
            or eax, 10000h
            mov cr0,eax
            pop eax
            sti
        }
        KeLowerIrql(oldIrql);
        __asm{
            mov eax, uOldCr3
            mov cr3, eax
        }
        uOutSize = pInfo->nWriteSize;
        if (pInfo != NULL){
            ExFreePool(pInfo);
            pInfo = NULL;
        }
        status = STATUS_SUCCESS;
}
```

2. 根据分页机制,进行手工转换,得到虚拟地址的映射的物理地址,读其物理地址得到目标进程虚拟地址的内容.

获得了 CR3,,接下来,就是根据分页机制,把虚拟地址转换为物理地址勒.

在转换之前要判断是否开启 PAE,非 PAE 和开启 PAE 的转换有所不同

控制寄存器 CR4 的第 5 位标记是否开启 PAE

3.经过一系列操作可以获得页表，根据页目录项的第 7 位判断页大小,2MB,还是 4KB，大小不同，其页表基地址和偏移也各有所不同，通过分析后在代码中提取，即可获得对应内存的信息

```
DWORD dwPageOffsetMB = dwVirtualAddr & 0x000fffff;
DWORD dwDirBaseAddr = (DWORD)(nPageDirEntry & 0x00000007fff00000);
PVOID pReadBuf = new BYTE[dwReadSize];
BOOL bRet = ReadPageMemoryPAE(pReadBuf,
dwPageOffsetMB,
dwReadSize,
dwDirBaseAddr);
if (bRet == FALSE){
    return;
}
```

这样可以得到目标进程虚拟地址的内存数据

2）除此之外，还可以使用 cheat engine 等工具获取进程数据，同时对特定项目进行修改。

3）同时从网络了解到，可以使用 TOOLHELP32 工具、PSAPI 等工具实现内训信息的获取。相比较之下，PSAPI 与 TOOLHELP32 大致相同，也比较简洁方便。但是在获取驱动、内存等方面，由于 PSAPI 有很多方法支持，相比而言，会更加简单实用。PASPI 主要通过 EnumProcess 函数来获取所有运行进程的 ID，然后再分别通过 EnumProcessModules、EnumDeviceDrives 等函数，获得相应的计数器或者指针信息。该工具在 MSDN 也有示例代码，可以用来学习

三．原理分析

控制寄存器（CR0～CR3）用于控制和确定处理器的操作模式以及当前执行任务的特性。CR0 中含有控制处理器操作模式和状态的系统控制标志；CR1 保留不用；CR2 含有导致页错误的线性地址；CR3 中含有页目录表物理内存基地址，因此该寄存器也被称为页目录基地址寄存器 PDBR。以上代码通过对寄存器进行读写，实现了目标基址的获取和内存的读取。由于篇幅限制，以上代码并非全部。

四．总结与预防

本次实验并不能算作完成，因为通过编程仅仅实现了进程信息内容的获取，然而并不能获取某个特定数据例如密码等内容。编程之余我也试用了 cheat engine 读取进程内容，发现还是比较好用的，同时我在其官网发现，ce 是开放源码的，这对于学习有很大的意义，目前仍在学习中，项目过大因此并不指望能够在本学习将其学透，只能作粗浅的了解。

Cheat engine 官网：https://www.cheatengine.org/

GitHub 源码地址：https://github.com/cheat-engine/cheat-engine/