Administrator: Command Prompt

Demystifying Modern Windows Rootkits

Bill Demirkapi

Independent Security Researcher

Administrator: Command Prompt

Who Am I?

- 19 years old
- Sophomore at the Rochester Institute of Technology
- ❤️ Windows Internals
- Mostly self-taught (with guidance)
- Strong “Game Hacking” background

What Is This Talk About?

In this talk, we'll go over...

- Loading a rootkit.
- Communicating with a rootkit.
- Abusing legitimate network communications.
- An example rootkit I wrote and the design choices behind it.
- Executing commands from kernel.
- Tricks to cover up the filesystem trace of your rootkit.

Administrator: Command Prompt

Introduction to Windows Rootkits

Administrator: Command Prompt

Windows Rootkits: An Overview

Why would you want to use a rootkit?

- Kernel drivers have significant access to the machine.
- Same privilege level as a typical kernel anti-virus.
- Less mitigations and security solutions targeting kernel malware.
- Anti-Virus often have less visibility into operations performed by kernel drivers.
- Kernel drivers are often ignored by anti-virus.

Administrator: Command Prompt

Example: Treatment by Anti-Virus

Anti-virus tends to treat kernel drivers with significant trust compared to user-mode applications.

```
 1 OB_PREOP_CALLBACK_STATUS __fastcall PreProcessThreadCallback(__int64 RegistrationContext, POB_PRE_OPERATION_INFORMATION
 2 {
54       currentProcess = IoGetCurrentProcess();
55       if ( PsGetProcessId(currentProcess) < 8 && OperationInformation->KernelHandle & 1 )
56         return 0;                        // if caller has a pid < 8 AND the handle is for the kernel, return 0
```

Excerpt from Malwarebytes' Process/Thread Handle callbacks

```
 1 OB_PREOP_CALLBACK_STATUS __fastcall PreProcessThreadCallback(__int64 RegistrationContext, POB_PRE_OPE
 2 {
 3   ULONG kernelHandleFlag; // eax
29   kernelHandleFlag = OperationInformation->KernelHandle;
30   callerProcessId = -1;
31   callerProcessCreateTime = 0i64;
32   if ( !(kernelHandleFlag & 1)                    // check if operation is on a kernel handle
33     && ExGetPreviousMode()                        // check if operation is from a UserMode thread
34     && !IsCsrss()
```

Excerpt from Carbon Black's Process/Thread Handle callbacks

Administrator: Command Prompt

Loading a Rootkit

Abuse Legitimate Drivers

There are *a lot* of “vulnerable” drivers. With some reversing knowledge, finding a “0-day” in a driver can be trivial.

Examples include…

- Capcom’s Anti-Cheat driver
- Intel’s NAL Driver
- Microsoft themselves!

Administrator: Command Prompt

Abuse Legitimate Drivers

Using legitimate drivers has quite a few benefits too:

- You only need a few primitives to escalate privilege.
- Finding a “vulnerable” driver is relatively trivial (OEM Drivers 😢).
- Difficult to detect due to compatibility reasons.

Administrator: Command Prompt

Abuse Legitimate Drivers

Abusing legitimate drivers comes with some strong drawbacks too…

- Major issue of compatibility across operating system versions depending on the primitives you have.
- Much more likely to run into stability issues.
- The last thing you want is your malware to BSOD a victim.

Just Buy a Certificate!

For some red teamers, buying a legitimate code signing certificate might be a good option.

- Useful for targeted attacks.
- No stability concerns.

But…

- Potentially reveals your identity.
- Can be blacklisted.

Administrator: Command Prompt

Abuse Leaked Certificates

Instead of buying a certificate yourself, why not just use one from someone else?

- There are quite a few public leaked certificates available to download.
- *Almost* has all the benefits of buying one without deanonymization.

But…

- The leaked certificate you use can be detected in the future.
- If the certificate was issued after July 29th, 2015, it won't work on secure boot machines running certain versions of Windows 10.

Abuse Leaked Certificates

In most cases, Windows doesn’t care if your driver has a certificate that has expired or was revoked.

```
C:\VM Certificate Testing>sc create RevokedDriver binPath= "C:\VM Certificate
 Testing\revoked.sys" type= kernel
[SC] CreateService SUCCESS

C:\VM Certificate Testing>sc start RevokedDriver

SERVICE_NAME: RevokedDriver
        TYPE               : 1  KERNEL_DRIVER
        STATE              : 4  RUNNING
                                (STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
        PID                : 0
        FLAGS              :
```

revoked.sys Properties

General | Digital Signatures | Security | Details | Previous Versions

Digital Signature Details

General | Advanced

Digital Signature Information

A certificate was explicitly revoked by its issuer.

Signer information

Name:

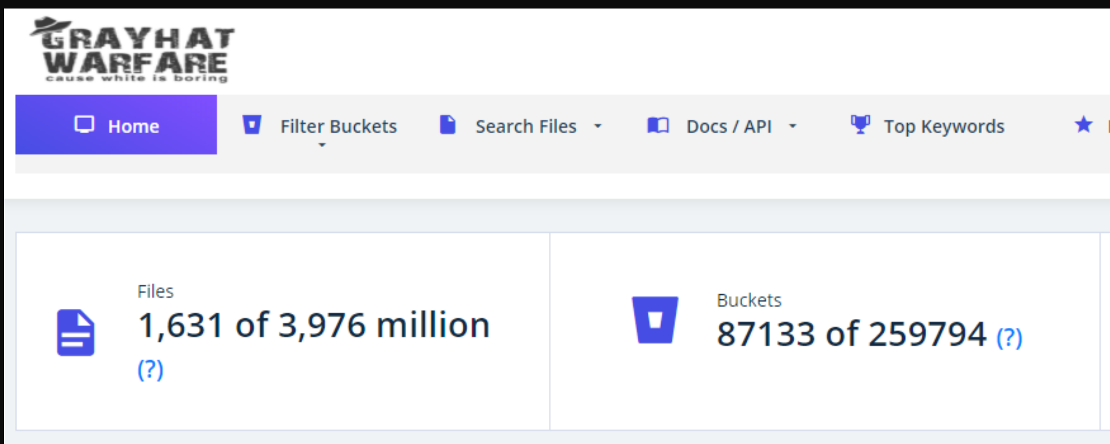
E-mail: Not available

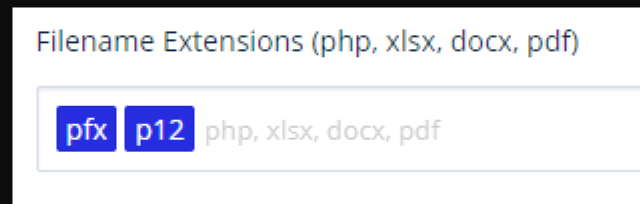
Signing time: Not available

Administrator: Command Prompt

Abuse Leaked Certificates

Several leaked certificates are already publicly posted, but it’s not impossible to find your own.



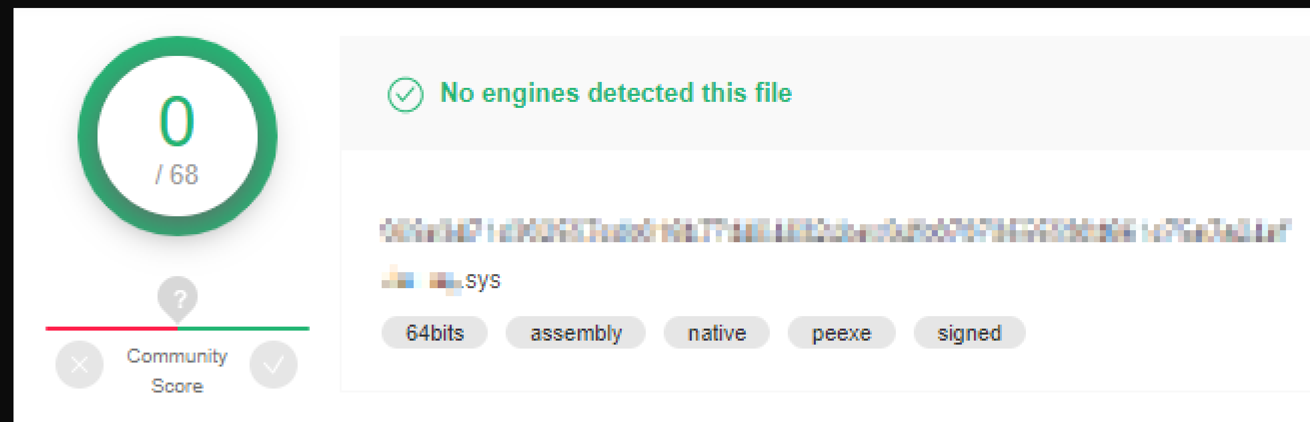


1 - 20 of 6005 results

Administrator: Command Prompt

Abuse Leaked Certificates

Oh and the best part…. most of them are undetected by the bulk of AV:



Administrator: Command Prompt

Communicating with a Rootkit

Beacon Out to a C2

A tried and true method that comes with some downsides is to "call home".

- Firewalls can block or flag outgoing requests to unknown/suspicious IP Addresses or ports.
- Advanced Network Inspection can catch some exfiltration techniques that try to "blend in with the noise".

Administrator: Command Prompt

Open a Port

Some malware takes the route that the C2 connects to the victim directly to control it.

- Relatively simple to setup.

But…

- Could be blocked off by a firewall.
- Difficult to “blend in with the noise”.

Application Specific Hooking

More advanced malware may opt to hook a specific application's communication as a channel of communication.

- Difficult to detect, especially if using legitimate protocol.

But…

- It's not very flexible.
- A machine might not have that service exposed.

Choosing a Communication Method

What I want…

1. Limited detection vectors.
2. Flexibility for various environments.

My assumptions…

1. Victims machines will have *some* services exposed.
2. Inbound and outbound access may be monitored.

Administrator: Command Prompt

Choosing a Communication Method

Application Specific Hooking was perfect for my needs, *except* for the flexibility.

Is there anyway we could change Application Specific Hooking to where it isn’t dependent on any single application?

Administrator: Command Prompt

Abusing Legitimate Communication

What if instead of hooking an application directly, we…

- Hook network communication, similar to tools like Wireshark.
- Place a special indicator in “malicious” packets, a “magic” constant.
- Send these “malicious” packets to legitimate ports on the victim machine.
- Search packets for this “magic” constant to pass on data to our malware.

Administrator: Command Prompt

Hooking the User-Mode Network Stack

Administrator: Command Prompt

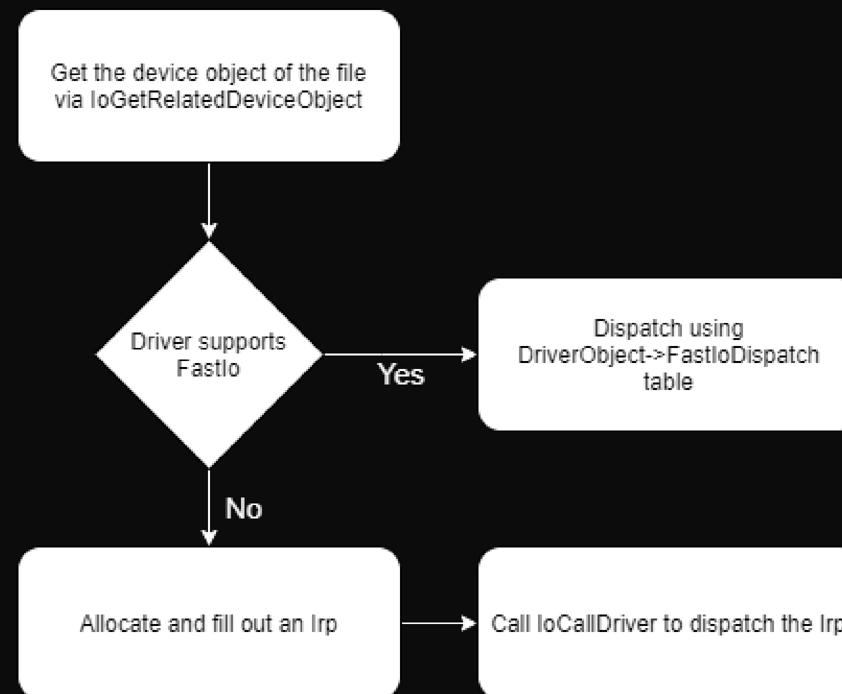
Hooking the Windows Winsock Driver

- A significant amount of services on Windows can be found in user-mode, how can we globally intercept this traffic?
- Networking relating to WinSock is handled by `Afd.sys`, otherwise known as the "Ancillary Function Driver for WinSock".
- Reversing a few functions in `mswsock.dll` revealed that a bulk of the communication was done through IOCTLs. If we could intercept these requests, we could snoop in on the data being received.

Administrator: Command Prompt

How Do Irps Know Where to Go?

When you call `NtDeviceIoControlFile` on a file handle to a device, how does the kernel determine what function to call?



Administrator: Command Prompt

Standard Methods of Intercepting Irps

There are a few ways we can intercept Irps, but let’s look at two common methods.

1. Replace the Major Function you’d like to hook in the driver’s object.
2. Perform a code hook directly on the dispatch handler.

Administrator: Command Prompt

Picking a method

To pick the best method of hooking, here are a few common questions you should ask.

- How many detection vectors are you potentially exposed to?
- How "usable" is the method?
- How expensive would it be to detect the method?

Hook a Driver Object

- How many detection vectors are you potentially exposed to?
  - Memory artifacts.
- How “usable” is the method?
  - For stability, by replacing a single function with an interlocked exchange, this method should be stable.
  - For compatibility, driver objects are well-documented and easy to find.
- How expensive would it be to detect the method?
  - Inexpensive, all anti-virus would need to do is enumerate loaded drivers and check that the major functions are within the bounds of the driver.

Administrator: Command Prompt

Hook a Driver’s Dispatch Function

- How many detection vectors are you potentially exposed to?
  - Memory artifacts.
- How “usable” is the method?
  - Unless the function is exported, you will need to find the function yourself.
  - Not all drivers are compatible with this method due to PatchGuard.
  - HVCI incompatible.
- How expensive would it be to detect the method?
  - Potentially inexpensive and several methods to detect hooking.

Hooking File Objects

I wanted a method that was…

- Undocumented.
- Stable.
- Relatively expensive to detect.

What if instead of hooking the original driver object, we hooked the file object instead?

Administrator: Command Prompt

How Do Irps Know Where to Go?

To retrieve the device associated with the Afd driver, the kernel calls `IoGetRelatedDeviceObject`.

```
typedef struct _FILE_OBJECT {
CSHORT Type;
CSHORT Size;
PDEVICE_OBJECT DeviceObject;
...
} FILE_OBJECT;
```

What's stopping us from overwriting this pointer?

Administrator: Command Prompt

Hooking File Objects

What we can do is…

1. Create our own device object and driver object.
2. Patch our copy of the driver object.
3. Replace the DeviceObject pointer of our file object with our own device.

Let’s talk about how we would go about doing this.

Administrator: Command Prompt

Hooking File Objects

Let's start by finding a file object to hook. We're after handles to \Device\Afd, but how can we find these objects?

ZwQuerySystemInformation function

Retrieves the specified system information.

Syntax

C++

```cpp
NTSTATUS WINAPI ZwQuerySystemInformation(
  _In_      SYSTEM_INFORMATION_CLASS SystemInformationClass,
  _Inout_   PVOID                    SystemInformation,
  _In_      ULONG                    SystemInformationLength,
  _Out_opt_ PULONG                   ReturnLength
);
```

```cpp
typedef enum _SYSTEM_INFORMATION_CLASS
{
...
SystemHandleInformation,
...
} SYSTEM_INFORMATION_CLASS,
*PSYSTEM_INFORMATION_CLASS;
```

Administrator: Command Prompt

Hooking File Objects

The `SystemHandleInformation` class allows us to query all handles on the system, including…

- The process ID the handle belongs to.
- The kernel pointer of the object associated with the handle.

If we open the Afd device ourselves, we can easily recognize file objects that are for the Afd device.

Administrator: Command Prompt

Hooking File Objects

Before we can overwrite the DeviceObject member, we need to create our fake objects first. Fortunately, the kernel exports the function it uses itself to create these objects.

All we need to do is call `ObCreateObject` passing the `IoDriverObjectType` or `IoDeviceObjectType` to create our fake objects.

We can copy the existing objects over to contain the same member values.

Hooking File Objects

With our fake objects created, we're almost ready to set the DeviceObject of the file object. First though, we need to hook our driver object.

We can use the standard "Hook a Driver Object" method, except instead of performing it on the original driver object, we'll use it on a fake driver object used *exclusively* for our hooks.

Administrator: Command Prompt

Hooking File Objects

To prevent race conditions while replacing the device object member, the original device object we use inside of our hooked dispatch must be set at the same time we the DeviceObject member of the file object.

To do this, simply perform an interlocked exchange of the original device object and the device object our hook uses.

```
//
// Atomically hook the device object of the file.
//
oldDeviceObject = RCAST<PDEVICE_OBJECT>(InterlockedExchange64(RCAST<PLONG64>(&FileObject->DeviceObject), RCAST<LONG64>(FileObjHook::FakeDeviceObject)));
```

Administrator: Command Prompt

Hooking File Objects

Now that we've hooked the file object, there is not much work left.

In our dispatch hook, we need to…

1. Check if we are hooking the MajorFunction being called.
    1. If we are, call the hook function passing the original device object and original dispatch function for that MajorFunction.
2. Make sure to restore the original DeviceObject when the MajorFunction is `IRP_MJ_CLEANUP`.

Hooking File Objects

- How many detection vectors are you potentially exposed to?
  - Memory artifacts.
- How “usable” is the method?
  - Most of the functions we use are at least semi-documented and unlikely to change significantly.
- How expensive would it be to detect the method?
  - Expensive, an anti-virus would have to replicate our hooking process and enumerate file objects to determine if the device/driver object was swapped.

Administrator: Command Prompt

How the Spectre Rootkit Abuses the User-Mode Network Stack

Administrator: Command Prompt

Abusing the Network

Using the File Object hook, we can now intercept Irps to the Afd driver.

This allows us to…

- Intercept all user-mode networking traffic.
- Send and receive our own data over any socket.

Administrator: Command Prompt

Abusing the Network

To review, our existing plan is to…

- Hook network communication, similar to tools like Wireshark.
- Place a special indicator in "malicious" packets, a "magic" constant.
- Send these "malicious" packets to legitimate ports on the victim machine.
- Search packets for this "magic" constant to pass on data to our malware.

How can we actually retrieve the content of packets that are received?

Abusing the Network

For receive operations, an IOCTL with the code IOCTL_AFD_RECV is sent to the Afd driver. Here is the structure sent in the input buffer.

```
typedef struct  _AFD_RECV_INFO {
PAFD_WSABUF BufferArray;
ULONG BufferCount;
ULONG AfdFlags;
ULONG TdiFlags;
} AFD_RECV_INFO, * PAFD_RECV_INFO;

typedef struct _AFD_WSABUF {
UINT len;
PCHAR buf;
} AFD_WSABUF, * PAFD_WSABUF;
```

Administrator: Command Prompt

Parsing Packets: Design

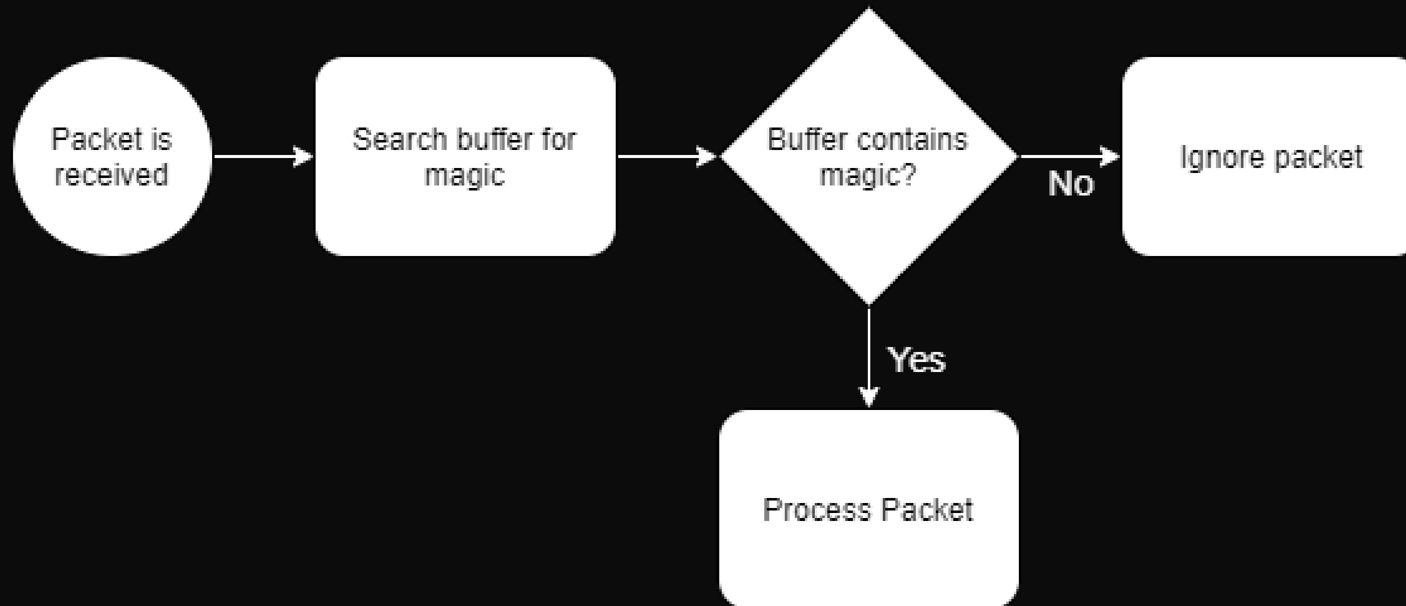
Let's talk about how the Spectre Rootkit was designed.

Spectre Rootkit Packet Structure
Any prepended data
Magic Constant
Base Packet Structure
Optional Custom Structure
Any appended data

```
25  typedef struct _BASE_PACKET
26  {
27      ULONG PacketLength; // The length of the packet.
28      PACKET_TYPE Type;   // Indicates the type of packet.
29  } BASE_PACKET, *PBASE_PACKET;
```

Parsing Packets: Pre-Processing

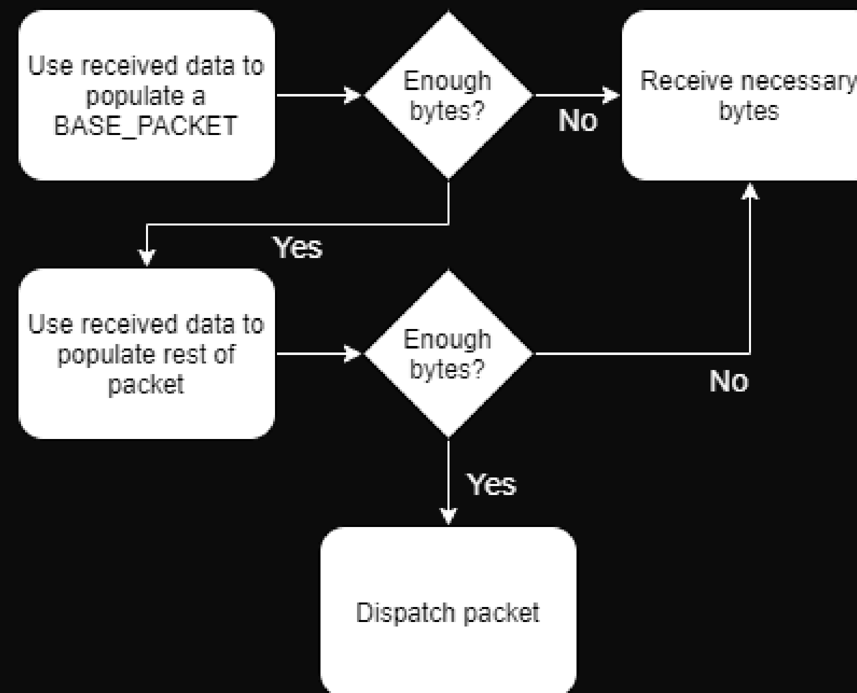
Here is the process used when the Spectre Rootkit receives a packet.



Administrator: Command Prompt

Parsing Packets: Processing

Before dispatching a packet, we need to create a complete packet.



Packet Handlers

Before we go any further, let’s talk about the concept of “Packet Handlers” in the Spectre Rootkit.

```
11  typedef class PacketHandler
12  {
13  protected:
14      //
15      // The packet dispatcher is used for sending and receiving network messages.
16      // It can also be used to dispatch a new packet.
17      //
18      PPACKET_DISPATCH PacketDispatch;
19  public:
20      PacketHandler (
21          _In_ PPACKET_DISPATCH Dispatcher
22          );
23
24      virtual NTSTATUS ProcessPacket (
25          _In_ PBASE_PACKET FullPacket
26          ) = 0;
27  } PACKET_HANDLER, *PPACKET_HANDLER;
```

Packet Handlers

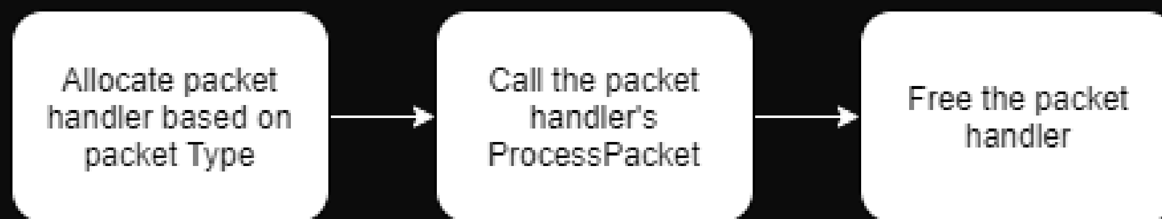
An example of a packet handler included with the Spectre Rootkit is the `PingPacketHandler`. This handler is used to determine if a machine/port is infected.

The incoming packet has no actual data, other than indicating its `Type` is a `Ping`. The handler responds to the client with an empty base packet with the `Type` set to `Ping`.

Administrator: Command Prompt

Parsing Packets: Dispatching

Once a packet is completely populated, the “packet dispatcher” will…



Here’s why the “packet dispatcher” is awesome: by passing a pointer to itself to the relevant packet handler, that packet handler can recursively dispatch a new packet!

Administrator: Command Prompt

Packet Handlers: XorPacketHandler

The best way to explain the recursive nature of the "packet dispatcher" is through an example, such as the `XorPacketHandler`.

The `XorPacketHandler` takes a `XOR_PACKET` structure:

```
40  typedef struct _XOR_PACKET
41  {
42      BASE_PACKET Base;   // Contains standard information about the packet.
43      BYTE XorKey;        // The XOR key used to obfuscate the packet.
44      BYTE XorContent[1]; // The XOR'd packet to dispatch.
45  } XOR_PACKET, *PXOR_PACKET;
```

This XOR_PACKET does not actually perform a malicious operation. Instead, it acts as an encapsulating packet.

Packet Handlers: XorPacketHandler

When the `XorPacketHandler` receives a packet, it will…

1. Use the XorKey to deobfuscate the XorContent.
2. Recursively dispatch the XorContent as a new packet.

The model that the Spectre Rootkit uses allows you to create infinite layers of encapsulation.

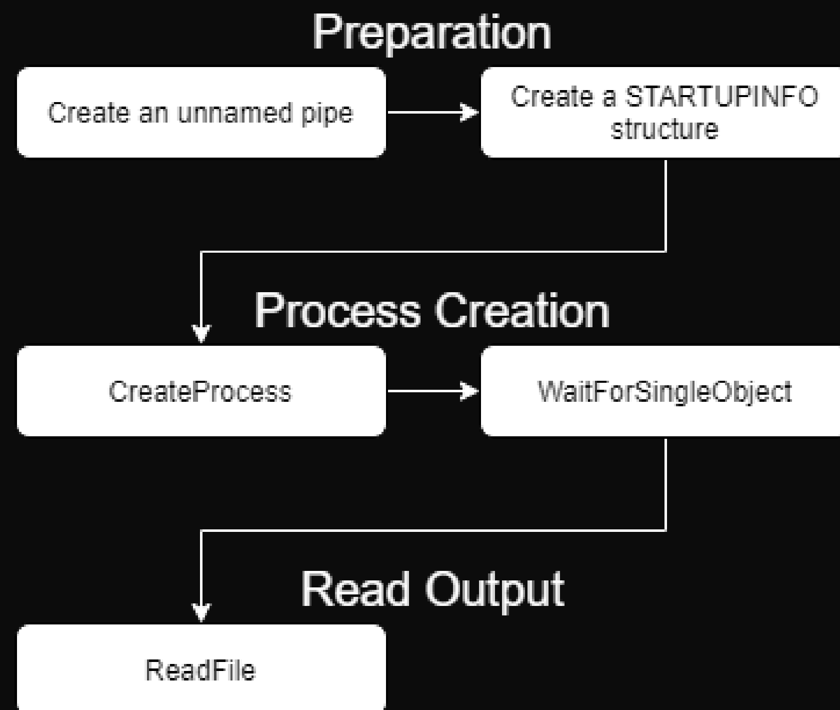
Executing Commands

Let’s take a look at how we can execute commands from our rootkit, a common feature seen in a variety of Windows malware.

Before we get into starting a process from a kernel driver, it’s important to understand how we would execute commands from a user-mode context.

Administrator: Command Prompt

Executing Commands: User-mode

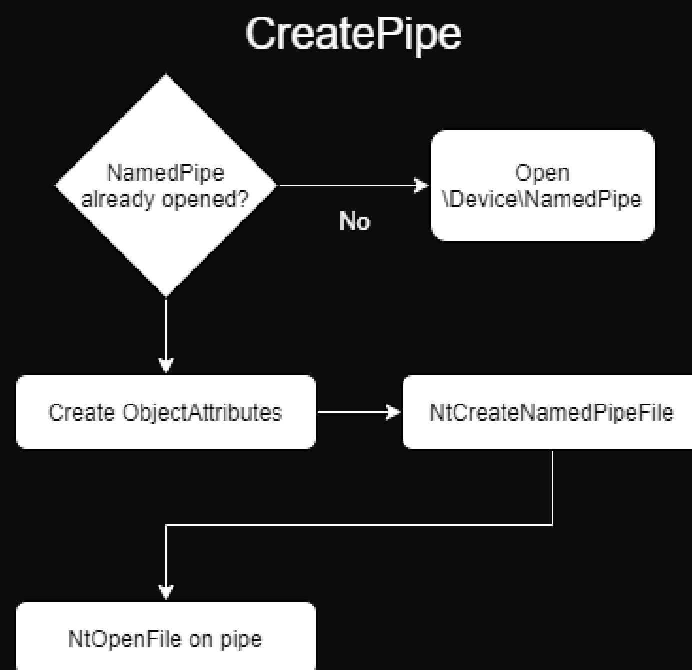


Administrator: Command Prompt

Executing Commands: Kernel-mode

Let's start by creating the pipes we need to obtain output.

Here is what `CreatePipe` does in the background…



Administrator: Command Prompt

Executing Commands: Kernel-mode

Now that we have pipes, we need to create the actual process. We'll use `ZwCreateUserProcess` because that's what `kernelbase.dll` uses itself to create processes.

Let's start with the attribute list for the process.

- The most important attribute we have to set is the PsAttributeImageName attribute. This will specify the image file name for the new process.

Executing Commands: Kernel-mode

Next, we have to fill out an RTL_USER_PROCESS_PARAMETERS structure for the process.

In this structure, we need to set…

1. The window flags and the output handles to our pipes.
2. The current directory, the command line arguments, the process image path, and the default desktop name.

Administrator: Command Prompt

Executing Commands: Kernel-mode

From there, all it takes is a call to `ZwCreateUserProcess` to start the process.

Once the process has exited, similar to what we do in user-mode, we can call `ZwReadFile` to read the output from the unnamed pipe.

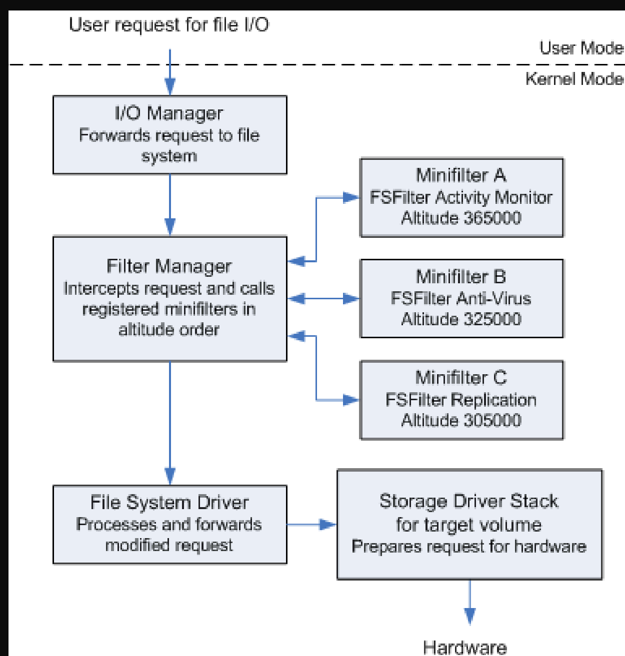
Administrator: Command Prompt

Hiding a Rootkit

Administrator: Command Prompt

Introduction to Mini-Filters

Mini-filter drivers allow you to attach to volumes and intercept certain file I/O. This is performed by registering with the Filter Manager driver.



Source: Microsoft Docs

Introduction to Mini-Filters

Mini-filters can be useful to mask the presence of our rootkit on the filesystem.

For example, a mini-filter can direct all file access for a certain file to another file. We can use this functionality to redirect access to our driver file to another legitimate driver.

Administrator: Command Prompt

Picking a method

To pick the best method of hooking, here are a few common questions you should ask.

- How many detection vectors are you potentially exposed to?
- How "usable" is the method?
- How expensive would it be to detect the method?

Administrator: Command Prompt

Become a Mini-Filter

The easiest way to abuse the functionality of a mini-filter is to become one yourself. Here are the minimum requirements for `FltRegisterFilter`:

1. Create [ServiceKey]\Instances
2. Create [ServiceKey]\Instances\[An instance name]
3. In [ServiceKey]\Instances add a "DefaultInstance" and set it to your instance name used in step 2.
4. In [ServiceKey]\Instances\[An instance name], add the "Altitude" and "Flags" values.

Become a Mini-Filter

- How many detection vectors are you potentially exposed to?
  - Registry and memory artifacts.
- How “usable” is the method?
  - No concerns from stability or usability, this is how other legitimate drivers register as mini-filters.
- How expensive would it be to detect the method?
  - Inexpensive. Besides the registry artifacts, drivers that are registered as mini-filters can easily be enumerated through API such as `FltEnumerateFilters`.

Hook a Mini-Filter

Another method is to simply hook an existing mini-filter. There are a couple of routes you could take.

- Code hook the callback for an existing filter.
- Overwrite the `FLT_REGISTRATION` structure before the victim driver uses it to have your own callback.
- DKOM an existing filter instance and replace the original callback with yours.

Administrator: Command Prompt

Hook a Mini-Filter: Code Hook

One of the easiest way to intercept callbacks to an existing mini-filter is to simply perform a code hook.

This can be as simple as a jmp hook, but it comes with quite a few drawbacks, similar to those we saw in an earlier section where we discussed intercepting Irps.

Administrator: Command Prompt

Hook a Mini-Filter: Code Hook

- How many detection vectors are you potentially exposed to?
  - Memory artifacts.
- How “usable” is the method?
  - Unless the function is exported, you will need to find the function yourself.
  - Not all drivers are compatible with this method due to PatchGuard.
  - HVCI incompatible.
- How expensive would it be to detect the method?
  - Potentially inexpensive and several methods to detect hooking.

Hook a Mini-Filter: DKOM

A semi-documented method of hooking an existing mini-filter is through DKOM.

You can enumerate filters and instances through the documented APIs `FltEnumerateFilters` and `FltEnumerateInstances`.

The function that gets called for a certain operation is specified in the `CallbackNodes` array in the `FLT_INSTANCE` structure.

Hook a Mini-Filter: DKOM

- The `CallbackNodes` array index is associated with the major function you're hooking.

```
0: kd> dt fltmgr!_FLT_INSTANCE
   +0x000 Base                   : _FLT_OBJECT
   +0x030 OperationRundownRef    : Ptr64 _EX_RUNDOWN_REF_CACHE_AWARE
   +0x038 Volume                 : Ptr64 _FLT_VOLUME
   +0x040 Filter                 : Ptr64 _FLT_FILTER
   +0x048 Flags                  : _FLT_INSTANCE_FLAGS
   +0x050 Altitude               : _UNICODE_STRING
   +0x060 Name                   : _UNICODE_STRING
   +0x070 FilterLink             : _LIST_ENTRY
   +0x080 ContextLock            : _EX_PUSH_LOCK
   +0x088 Context                : Ptr64 _CONTEXT_NODE
   +0x090 TransactionContexts    : _CONTEXT_LIST_CTRL
   +0x098 TrackCompletionNodes   : Ptr64 _TRACK_COMPLETION_NODES
   +0x0a0 CallbackNodes          : [50] Ptr64 _CALLBACK_NODE
```

```
0: kd> dt fltmgr!_CALLBACK_NODE
   +0x000 CallbackLinks          : _LIST_ENTRY
   +0x010 Instance               : Ptr64 _FLT_INSTANCE
   +0x018 PreOperation           : Ptr64     _FLT_PREOP_CALLBACK_STATUS
   +0x020 PostOperation          : Ptr64     _FLT_POSTOP_CALLBACK_STATUS
   +0x018 GenerateFileName       : Ptr64     long
   +0x018 NormalizeNameComponent : Ptr64     long
   +0x018 NormalizeNameComponentEx : Ptr64     long
   +0x020 NormalizeContextCleanup : Ptr64     void
   +0x028 Flags                  : _CALLBACK_NODE_FLAGS
```

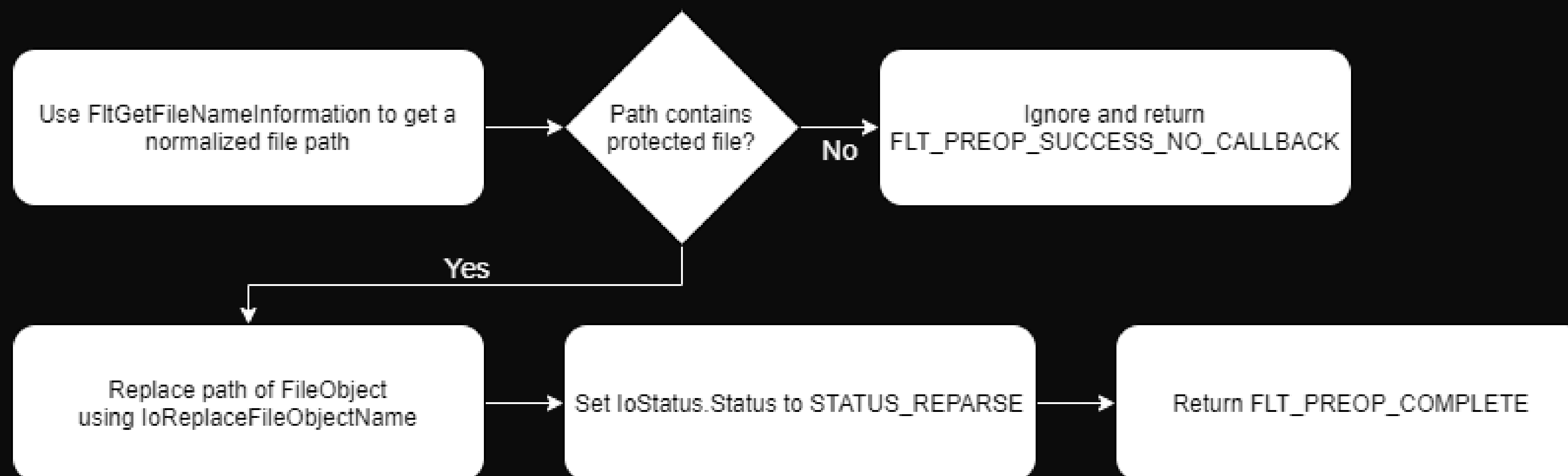
Hook a Mini-Filter: DKOM

- How many detection vectors are you potentially exposed to?
  - Memory artifacts.
- How “usable” is the method?
  - For stability, although obtaining a `FLT_INSTANCE` structure is documented, the `FLT_INSTANCE` structure itself is undocumented.
- How expensive would it be to detect the method?
  - Inexpensive, an anti-virus would need to occasionally enumerate registered filters and their instances for hooks in the `CallbackNodes` array.

Administrator: Command Prompt

Example: Abusing a Mini-Filter

Let's say you want to protect a certain file, what's an example of redirecting access to it?



Administrator: Command Prompt

Wrap Up

Administrator: Command Prompt

Thanks to...

Alex Ionescu (@aionescu)

- Long-time mentor very experienced with Windows Internals.

ReactOS

- A fantastic reference for undocumented functions and structures.

Nemanja Mulasmajic (@0xNemi) and Vlad Ionescu (@ucsenoi)

- Helped review this presentation.

Demystifying Modern Windows Rootkits – DEF CON 28

Administrator: Command Prompt

Contact / Questions

Thanks for sticking around! Now is the time for any questions.

Twitter

@BillDemirkapi

Blog

https://billdemirkapi.me

Spectre Rootkit

https://github.com/D4stiny/spectre