# What are the methods to find hooked functions and APIs?

Published: 2012-07-31 · Archived: 2026-04-05 23:46:43 UTC

Hooks are implemented in a whole bunch of ways:

- Modifying legitimate jump instructions to point at hooks instead of the normal code.
- User call table (IAT) hooking - modifying the addresses of user-mode APIs in a process.
- Kernel call table hooking (e.g. SSDT / GDT ) - replacing a call table pointer with the address of your hook.
- WndProc hooks (e.g. `PeekMessage`) - hooking onto window notification messages.
- Legitimate callbacks like `PsSetCreateProcessNotifyRoutine`.

I'm guessing you're most interested in the first two types.

Jump hooks can be created in a near-infinite number of ways. This makes it almost impossible to write a tool to identify the hooks. However, you can use integrity checking tricks, e.g. comparing code in the binary file (e.g. `exe` or `dll`) to the code in memory. You could also hook `WriteProcessMemory` and other such APIs to detect modification of process memory, though this only works against user-mode attacks.

IAT hooks are a little easier to check for. Take a snapshot of the IAT of a process when it starts (e.g. from the static binary) and compare the in-memory IAT to the real addresses of the functions that should be in there. For example, if you know IAT entry 4 points to `user32.MessageBoxA`, you can use `GetProcAddress` to find the real address of that function and compare the address in the IAT to that. If they don't match, you know it's been hooked.

For further reading, there's a great paper on the subject, and I highly recommend reading "The Rootkit Arsenal" by Bill Blunden.

---

Source: https://security.stackexchange.com/questions/17904/what-are-the-methods-to-find-hooked-functions-and-apis