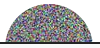
mavinject.exe Functionality Deconstructed



Among the suspicious APIs I scanned for, mavinject makes use of the following injection-related functions used commonly by malware:

- VirtualProtectEx — used to change memory page permissions in another process
- CreateRemoteThread — used to launch a thread in another process
- VirtualAllocEx — used to allocate memory in another process using the page permission of the developer's choosing

1. Calls OpenProcess to get a handle to the target process. It requests the following access: 0x10043A (SYNCHRONIZE | PROCESS_QUERY_INFORMATION | PROCESS_VM_WRITE | PROCESS_VM_READ | PROCESS_VM_OPERATION | PROCESS_CREATE_THREAD). From a detection perspective, for those familiar with Sysmon, it would be a reasonable assumption to build a ProcessAccess rule, but be mindful that requested access will not always match the GrantedAccess field. Additionally, I am not aware of the uniqueness of this process access but that would be easy enough to find out by letting Sysmon capture ProcessAccess events for a while

UpdateImports32/64 function. The `/HMODULE=0x` parameter would be used as follows:

```
mavinject.exe 4964 /HMODULE=0x013C0000 foo.dll 4
```

In this example, the 32-bit mavinject injects an import table entry consisting of “foo.dll” that “exports” a function with an ordinal of 4 into a 32-bit process (PID 4964) into the module at base address 0x013C0000 (the powershell.exe

I found this functionality to be very interesting because ideally, I thought I might be able to somehow redirect import address table (IAT) entries. In practice, I was unable to achieve that goal and I'm not convinced that using mavinject *on its own* can be used to achieve arbitrary import table hooking. There *might* be some abuse potential mixing import table injection with other AppV-related DLLs that are intended to be injected like AppVEntSubsystems[32|64].dll, but I didn't investigate that in too much depth.

I did find an interesting bug, however, in how mavinject applies

```
0:009> !dh -i 013C0000
  _IMAGE_IMPORT_DESCRIPTOR 04870000
    foobar.dll
              048700F8 Import Address Table
              048700F0 Import Name Table
                     0 time date stamp
                     0 Index of first forwarder reference

0000004E 6854 is program cannot be run in DOS mode.
```

I have yet to find a compelling way to abuse this in any useful fashion but I thought it was interesting. Also, a notable side effect of the

the contents of the PE header in the main module in a `.detours` section of memory. This article explains what the `.detours` section is used for. There appears to be no way to introduce custom hook functions into the `.detours` section using mavinject alone, though.

So again, ultimately, I was unable to abuse import table injection as a useful primitive for anything interesting (aside from arbitrary data injection) even with the wraparound bug. I am documenting this functionality, though, in the hopes that it might inspire someone else to invest the time into abusing Detours functionality in signed code. For those interested in identifying

however that an attacker can modify these fields but in doing so, the signature will be invalidated.

- Here is a sample set of Sysmon rules to detect mavinject usage:

```
<ProcessCreate onmatch="include">
    <!-- Catch mavinject DLL injection -->
    <CommandLine condition="contains">INJECTRUNNING</CommandLine>
    <!-- Catch mavinject regardless of its filename and command
line usage -->
    <!-- Note: an attacker can modify this field in the binary. It
```

about but which I am *very* confident will have a lot of abuse/tradecraft potential.

- I encouraged you to consider blogging about your research findings and methodology even if you weren't ultimately successful. After all, failure breeds inspiration which ultimately, leads to eventual success.

Mavinject | Reverse Engineering | Dll Injection