Intercepting Password Changes With Function Hooking

By Posted on

Published: 2013-09-15 · Archived: 2026-04-06 01:21:14 UTC

Last week, Mubix published a malicious Windows password filter DLL (http://carnal0wnage.attackresearch.com/2013/09/stealing-passwords-every-time-they.html). The idea is simple, by installing this password filter, he can intercept the clear text credential whenever a user changes their password. There are two caveats with installing this password filter:

1. You must restart the computer for it to take affect
2. It will show up as an autorun and a loaded DLL in lsass.exe which may be noticed

I've been playing around with function hooks lately and thought this would be a great demonstration of their usefulness. The idea is that you hook the PasswordChangeNotify function in the default Windows password filter (rassfm.dll). Anytime PasswordChangeNotify is called, it will be rerouted to my malicious PasswordChangeNotify function, which will write the password to disk and return execution back to the original PasswordChangeNotify function.

To do this, I will write the function hooking code in a DLL and reflectively inject the DLL in to lsass.exe using Invoke-ReflectivePEInjection. The benefit to this approach is that no binaries are written to disk, no suspicious DLL's are loaded in lsass, no registry changes are made, and no reboot is required.

Since I know not everybody is familiar with writing a function hook, I'll explain the code (which can be found here: https://github.com/clymb3r/Misc-Windows-Hacking/tree/master/HookPasswordChange):

- Invoke-ReflectivePEInjection is used to inject the malicious hooking DLL in to lsass.

.\Invoke-ReflectivePEInjection –pepath .\HookPasswordChange.dll –procname lsass

- Invoke-ReflectivePEInjection calls the function VoidFunc in the reflectively loaded DLL, which installs the function hook. The function hook overwrites the first 12 bytes of PasswordChangeNotify with instructions to jump to another location.

The hook also allocates some RWX memory that holds byte code that returns execution flow to rassfm!PasswordChangeNotify (more on this later).

- Rassfm!PasswordChangeNotify is called; it immediately executes the following assembly which diverts execution to my malicious PasswordChangeNotifyHook function.

```
rassfm!PasswordChangeNotify:
000007f8`337e2614 48b8001a6005ba000000 mov rax,00b005e01a00h
000007f8`337e261e ffe0            jmp     rax
000007f8`337e2620 7424            je      rassfm!PasswordChangeNotify+0x32 (000007f8`337e
000007f8`337e2622 205741          and     byte ptr [rdi+41h],dl
000007f8`337e2625 56              push    rsi
000007f8`337e2626 4157            push    r15
000007f8`337e262B 4883ec30        sub     rsp,30h
000007f8`337e262c 4533ff          xor     r15d,r15d
000007f8`337e262f 498bf0          mov     rsi,r8
000007f8`337e2632 448bf2          mov     r14d,edx
000007f8`337e2635 4d85c0          test    r8,r8
000007f8`337e2638 7406            je      rassfm!PasswordChangeNotify+0x2c (000007f8`337e
000007f8`337e263a 410fb718        movzx   ebx,word ptr [r8]
000007f8`337e263e eb03            jmp     rassfm!PasswordChangeNotify+0x2f (000007f8`337e
```

Disassembly of PasswordChangeNotify after being hooked. The instructions after jmp rax disassemble incorrectly after hooking, but all the code after the hook is intact.

- PasswordChangeNotifyHook is a function written in c++, and it takes the same parameters as rassfm!PasswordChangeNotify function. This function takes the username and password and writes it to disk (or does something else that you program such as sending the output to a web site).
- Now it is time to return execution flow back to the real PasswordChangeNotify. PasswordChangeNotifyHook casts the memory address of the RWX memory allocated earlier to be a function pointer to a function with the same signature as rassfm!PasswordChangeNotify and calls this function pointer.
- The RWX memory contains the first 3 instructions (15 bytes worth of byte code) that we overwrote in rassfm!PasswordChangeNotify. Now that those instructions have been executed, load the memory address of PasswordChangeNotify+0xf to the EAX and jump to it. Execution flow has now been successfully returned to rassfm!PasswordChangeNotify.

```
000000fd`d2230000 48895c2408      mov     qword ptr [rsp+8],rbx
000000fd`d2230005 48896c2410      mov     qword ptr [rsp+10h],rbp
000000fd`d2230009 4889742420      mov     qword ptr [rsp+20h],rsi
000000fd`d223000f 48b82326be9cfc070000 mov rax,offset rassfm!PasswordChangeNotify+0xf (
000000fd`d2230019 ffe0            jmp     rax
```

Disassembly of the RWX memory I allocate which ends up returning execution to PasswordChangeNotify.

Originally I wanted to write inline asm in the PasswordChangeNotifyHook function to return control flow to rassfm!PasswordChangeNotify but unfortunately Visual Studio doesn't support inline asm for x64 or Itanium, only x86. This is why I had to allocate RWX memory to put the byte code in and trigger it by calling a function pointer.

Note that this is a proof of concept. I have tested it on Windows Server 2012 but it should work on 2008R2 as well. The code can be found at: https://github.com/clymb3r/Misc-Windows-Hacking/tree/master/HookPasswordChange.

The Invoke-ReflectivePEInjection script which is used to load the hook DLL can be found at: https://github.com/clymb3r/PowerShell/tree/master/Invoke-ReflectivePEInjection.

Source: https://clymb3r.wordpress.com/2013/09/15/intercepting-password-changes-with-function-hooking/