WinAPI User-Mode Thread Token Impersonation Privilege Leak

Author: Borja Gómez

Country: Spain

E-Mail: kub0x@elhacker.net

Date of discovery: 22/09/2014

Estimated Microsoft employee,

I leave here the technical details as the requirements, exploitation and contra measures of the discussed vulnerability. Hope you fix it the fast as you can.

General Information:

Exploit Type: Local privilege escalation in user-mode.

Tested on:

- Windows 7 Professional 32 bits & 64 bits

- Windows 8.1 Professional 64 bits

***** May affect all Windows 7 OS families, same for Windows 8.0/8.1 & Vista.

Requirements:

- User belongs to the administrators group.

- UAC is enabled in any option, except the highest one " ".

Additional Software:

My exploit has been coded on Visual Studio 2013 so you need to install the latest C++ Redistributable in order to execute the POC, but this could be avoided including essential libraries in the executable.

- **Microsoft Visual C++ 2012 Redistributable (x86) 11.0.61030**

- **Microsoft Visual C++ 2012 Redistributable (x86) 11.0.61030**
- **Microsoft Visual C++ 2013 Redistributable (x86) 12.0.21005**
- **Microsoft Visual C++ 2013 Redistributable (x64) 12.0.21005**

Vulnerability details:

Security flaw appears when combining an auto-elevate process execution with the use of its token to impersonate in a user-mode program thread.

What does this means?

When impersonating a token, the actions that we make are executed in the same security context as the process that the token belongs to. This means that if we are able to impersonate an administrative token we will be capable of executing actions like we were an administrator.

How can you impersonate in user-mode a token that belongs to a process running as an Administrator?

By default, in user-mode, we cannot open a process handle out of our scope. Basically we cannot neither retrieve a process handle running as an Administrator nor a process handle running as SYSTEM.

The vulnerability resides in the fact that in user-mode, we can execute an auto-elevated process, grab its process handle without opening it, and do the impersonation in our process current thread, so we are able to execute code in Admin security context.

Remark: UAC won't prompt when executing an auto-elevated process under a user account that belongs to the administrator groups.

Technical details:

Here I leave some code that demonstrates the security flaw:

SHELLEXECUTEINFOW structure contains a member that contains the process handle of the executed process. That member is named "**hProcess**". When a process is executed by calling **ShellExecuteExW**, hProcess will point to the opened process handle.

In my POC I use DeviceEject.exe. It is located at C:\Windows\System32 and besides belongs to the auto-elevated processes list. It's a good example since it terminates too fast.

First, we are going to execute DeviceEject and grab its handle to retrieve its process token. This function returns the process handle of the specified process (DeviceEject)

```
HANDLE GetPrivilegedToken(const wchar_t *pFile){

	wchar_t windir[MAX_PATH];
	size_t nSize = _countof(windir);
	_wgetenv_s(&nSize, windir, L"WINDIR");

	wstring dir = wstring(windir).append(L"\\System32\\");
	wstring file = wstring(dir).append(pFile);

	SHELLEXECUTEINFOW shinf = { 0 };

	shinf.cbSize = sizeof(shinf);
	shinf.fMask = SEE_MASK_NOCLOSEPROCESS;
	shinf.lpFile = file.c_str();
	shinf.lpDirectory = dir.c_str();
	shinf.nShow = SW_SHOW;

	ShellExecuteExW(&shinf);

	return shinf.hProcess;
}
```

Then we duplicate the token specifying the TokenImpersonation value (later explained) and we also set the Medium mandatory level (SID) in the duplicated token.

```
if (OpenProcessToken(hProcess, MAXIMUM_ALLOWED, &hToken))

	if (DuplicateTokenEx(hToken, MAXIMUM_ALLOWED, NULL,
SecurityImpersonation, TokenImpersonation, &hNewToken))

	if (ConvertStringSidToSid(wszIntegritySid, &pIntegritySid)){



		TIL.Label.Attributes = SE_GROUP_INTEGRITY;

		TIL.Label.Sid = pIntegritySid;

[..]

	}
```

We swap the current thread's token with the impersonation token in the caller thread. Note that we can restore this procedure by calling RevertToSelf(), if needed.

```
if (SetTokenInformation(hNewToken, TokenIntegrityLevel, &TIL,
sizeof(TOKEN_MANDATORY_LABEL) + GetLengthSid(pIntegritySid)))

	if (SetThreadToken(NULL, hNewToken)){

		CopyFileW(L"PocMsg.txt",
		wstring(windir).append(L"\\System32\\PocMsg.txt").c_str(),
		FALSE);
		RegisterPrivilegedTask();
		StartPrivilegedTask();
		DisableUAC();

	}

[..]
```

Finally we are able to make certain actions for example: registering tasks under SYSTEM/Admin accounts, copying to protected directories to hijack DLLs and write to sensitive and relevant registry keys. All this in user-mode. **NOTE:** a user-mode exploiting this vulnerability could disable UAC without even prompting or asking for elevation.

The **code is located at "TokenPoC" directory**, inside the .rar that I provided. Please read the **README** as it specifies the documentation and procedures that have been taken.

There are two executable files, one for each processor architecture (32/64 bits).

Execute the POC code fulfilling the requirements and **without** running as an administrator.

In this **PoC (TokenPoC)** I will execute three actions that require privileges:

1) Task registration under SYSTEM account. The task's command will be executed suddenly or when system boots by default. **NOTE:** This works due to the impersonation technique exposed above. In a normal user-mode process this won't register the task because of the lack of privilege.

2) File copying on protected directories. **NOTE:** this could conduce to DLL Hijacking attacks. A POC of DLL Hijacking is explained in my other White Paper using another exploit. DLL Hijacking technique COULD BE APPLIED in this POC, so we have two ways of executing unwanted DLL files with privileges.

3) Disables UAC via registry API. **NOTE:** As said before, this only works when exploiting the discussed vulnerability.

As you can see, you can register privileged tasks in user-mode with this technique. Privileged Tasks are registered using COM objects that need elevation in certain point, so I imagine that COM objects that need elevation will work here without asking for it.

Vector Attacks:

- A good vector attack will be the execution of a DLL that exploits the vulnerability via rundll32.

- Also an ActiveX object will be a good choice for executing the exploit.

- Driven-by attack to enterprises o whatever target that can offer us some information and profit.

Contra measures:

There are known contra measures to apply in order to fix this issue:

- Run under a **standard user account**. This means that the users does not belong to the administrators group, so UAC prompts and ask for the administrator credentials.

- Mark "**Always Notify**" option in the User Access Control Settings Panel. Is the highest option, so will prompt UAC window for each task that involves elevation, including the execution of auto-elevated processes.

- **DLL Hijacking** contra measures are explained in the **second white paper**. Also I wonder why Windows has not thought about this yet. Please read them too.

Thanks for reading. Tt took me a long time, effort and dedication to have discovered the flaw, same for the writings/whitepapers. I hope you answer soon!

Best regards,

Borja.