Microsoft Windows 10 - Child Process Restriction Mitigation Bypass

EDB-ID: 44888	**Author**: Google Security Research	**Published**: 2018-06-13
CVE: CVE-2018-0982	**Type**: Local	**Platform**: Windows
Aliases: N/A	**Advisory/Source**: Link	**Tags**: Local
E-DB Verified: ✔	**Exploit**: Download / View Raw	**Vulnerable App**: N/A

« Previous Exploit

Next Exploit »

```
Windows: Child Process Restriction Mitigation Bypass
Platform: Windows 10 1709 (not tested other versions)
Class: Security Feature Bypass

Summary:

It’s possible to bypass the child process restriction mitigation policy by impersonating the anonymous
token leading to a security feature bypass.

Description:

Windows 10 has a mitigation policy to restrict a process creating new child processes. I believe the main
rationale is to prevent escaping some of the other mitigations which are not inherited across to new child
processes as well as bugs which can only be exploiting from a fresh process. The policy is enforced as a
flag in the token rather than on the process which allows the restriction to be passed across process
boundaries during impersonation, which would also kill abusing WMI Win32_Process and similar.

During process creation the token flag is checked in SeSubProcessToken which creates the new primary token
for the new process. It’s possible to also specify a flag for overriding the behavior, the code looks
```

something like the following:

```
if (ChildProcessOptions & PROCESS_CREATION_CHILD_PROCESS_OVERRIDE)
{
    PTOKEN CurrentToken = PsReferenceEffectiveToken(
            KeGetCurrentThread(),
            &Type,
            &CopyOnOpen,
            &ImpersonationLevel);
    if ( Type == TokenImpersonation && ImpersonationLevel < SecurityImpersonation
      || (SeTokenIsNoChildProcessRestrictionEnforced(CurrentToken) != 0 && Type != TokenPrimary))
    {
      return STATUS_CHILD_PROCESS_BLOCKED;
    }
}
```

This checks if the PROCESS_CREATION_CHILD_PROCESS_OVERRIDE is set then either the primary or impersonation token do not have the restrict child process flag set. If the token does have the flag then STATUS_CHILD_PROCESS_BLOCKED is returned and process creation fails. The problem with this code is it entirely relies on a process not being able to get an impersonation token without the flag. For a normal user process this would be trivial (of course it's trivial to bypass this restriction from a normal process anyway) but from an AppContainer it should be much more difficult.

There is an easy token we can impersonate which doesn't have the flag set, the Anonymous token. The problem with this is if we impersonate over the entire process creation then it will fail because the kernel will not be able to open the target executable. Fortunately the check for child process creation is after opening the file so we can abuse oplocks and from a separate thread assign the impersonation token while the thread is still running kernel code. So the following steps can be used to create an arbitrary child process:

1. Place an oplock on the image file for the process we want to create and wait for completion.
2. In a separate thread create a new process with the desired image file.
3. Once oplock has completed impersonate the anonymous token on the thread calling create process. Release oplock.
4. Original thread should continue process creation and check the anonymous token for the restricted flag bypassing the mitigation.

Note that you could probably also abuse the conhost creation inside ConDrv as that runs with kernel permissions so won't actually care about the anonymous token but it would be nicer to demonstrate this bypass with an arbitrary process.

From a fixing perspective I'm not entirely clear what the purpose of checking the impersonation token is. I'm guessing it's supposed to allow a secondary process without restriction to use a process which has the restriction as a parent process using a process attribute. In that case perhaps you need a check that the

```
parent process attribute is set and we're not being called from the same process or something similar so
that only that use case can pass the override flag.

Proof of Concept:

I've provided a PoC as a C# project. It will first respawn itself into an AppContainer with the child
process restriction mitigation enabled. The use of a AppContainer shows that this would be normally much
more difficult to circumvent as you can't just open other processes. It will then use the outlined attack
to bypass the restriction and respawn itself a second time. If successful there should be three copies of
the poc running, two with child process creation restrictions inside an AppContainer.

1) Compile the C# project. It will need to grab the NtApiDotNet from NuGet to work.
2) Apply the ALL_APPLICATIONS_PACKAGES Read/Execute ACE to the POC's directory otherwise respawning as an
AC will not work.
2) Execute the PoC

Expected Result:
The second process should fail to create a new process.

Observed Result:
The second process creates a new process and the third process in the chain shows a Hello message box.


Proof of Concept:
https://github.com/offensive-security/exploit-database-bin-sploits/raw/master/bin-sploits/44888.zip
```

« Previous Exploit

Next Exploit »

Related Exploits

Trying to match CVEs (1): CVE-2018-0982

Other Possible E-DB Search Terms: *Microsoft Windows 10*, **Microsoft Windows**

Date	D	V	Title	Author
1999-01-07		✔	Microsoft Windows - 'April Fools 2001' Set Incorrect Date	Richard M. ...

Date	D	V	Title	Author
2010-07-08		✓	Microsoft Windows - 'cmd.exe' Unicode Buffer Overflow (SEH)	bitform
2005-09-06		✓	Microsoft Windows - 'keybd_event' Local Privilege Escalation	AndrÃ©s Acu...
2005-08-01		✓	Microsoft Windows - 'LegitCheckControl.dll' Genuine Advantage Validation Patch	HaCkZaTaN
2017-06-22		✓	Microsoft Windows - 'win32k!ClientPrinterThunk' Kernel Stack Memory Disclosure	Google Secu...
2014-11-22		◷	Microsoft Windows - 'win32k.sys' Denial of Service	Kedamsky
2006-02-12		✓	Microsoft Windows - ACLs Privilege Escalation (2)	Andres Tarasco
2012-11-29		✓	Microsoft Windows - AlwaysInstallElevated MSI (Metasploit)	Metasploit
2007-03-10		✓	Microsoft Windows - DCE-RPC svcctl ChangeServiceConfig2A() Memory Corruption	h07
2012-10-16		✓	Microsoft Windows - Escalate Service Permissions Privilege Escalation (Metasploit)	Metasploit
2012-10-10		✓	Microsoft Windows - Escalate UAC Execute RunAs (Metasploit)	Metasploit
2012-10-10		✓	Microsoft Windows - Escalate UAC Protection Bypass (Metasploit)	Metasploit
2017-08-22		✓	Microsoft Windows - Escalate UAC Protection Bypass (Via COM Handler Hijack) (Metasploit)	Metasploit
2016-08-19		◷	Microsoft Windows - Fileless UAC Protection Bypass Privilege Escalation (Metasploit)	Pablo Gonzá...
2010-08-21		◷	Microsoft Windows - IcmpSendEcho2Ex Interrupting Denial of Service	l3D
2010-09-07		✓	Microsoft Windows - Local Procedure Call (LPC) Privilege Escalation	yuange
2018-01-08		✓	Microsoft Windows - Local XPS Print Spooler Sandbox Escape	Google Secu...
2013-01-25		✓	Microsoft Windows - Manage Memory Payload Injection (Metasploit)	Metasploit
2006-01-15		✓	Microsoft Windows - Metafile '.WMF' Arbitrary File Download (Generator)	darkeagle
2012-08-15		✓	Microsoft Windows - Service Trusted Path Privilege Escalation (Metasploit)	Metasploit

Date	D	V	Title	Author
2014-05-22		✓	Microsoft Windows - Touch Injection API Local Denial of Service	Tavis Ormandy
2018-06-04		✓	Microsoft Windows - UAC Protection Bypass (Via Slui File Handler Hijack) (Metasploit)	Metasploit
2010-08-17		✓	Microsoft Windows - Win32k!xxxRealDrawMenuItem() Missing HBITMAP Bounds Checks	Tavis Ormandy
2005-05-02		✓	Microsoft Windows - WINS Vulnerability + OS/SP Scanner	class101
2015-10-26		◷	Microsoft Windows 10 - 'pcap' Driver Privilege Escalation	Rootkitsmm