

Forum:[Hack Forums](#)

Thread: [Privilege Escalation explanation by Abdelhamid Nasser](#)

AzureGod - December 20, 2022, 1:52 pm

Author Abdelhamid Nasser <https://twitter.com/thezdi>

If an arbitrary file was deleted under the NT AUTHORITY\SYSTEM account, this is a system breach and a denial of service incident.

This article discusses interesting tricks related to the deletion of arbitrary files and folders and other seemingly inefficient file system exploitation primitives.

Difficulties with deleting arbitrary files

There are two major hurdles when deleting arbitrary files in Windows:

1. Most critical Windows files are locked using DACLs. account SYSTEM . The OS files are mostly owned by the TrustedInstaller , and only that account can modify them. (Exercise: Find critical Windows files that you can still delete or overwrite as SYSTEM .)
2. Finding a file that can be deleted on behalf of SYSTEM is not enough for an attacker: a fail-open situation (deterioration of security) must occur during deletion.

There is a third potential complication: some important system files will always be inaccessible due to sharing violations.

In practice, it is very difficult to find a file whose deletion would make sense from the point of view of hacking the system and at the same time be allowed to it. If you search in the usual places, such as C:\Windows, C:\Program Files or C:\Program Data , you won't find anything suitable there. [Previously was shown](#) operation of anti-virus and other software However, the possibility of exploitation depends on the specific vulnerable behavior of programs.

There is a solution - in the Windows installer

In March 2021, a [vulnerability](#) was reported by a researcher named Abdelhamid Naceri, also known as [Halov](#) . The vulnerability consisted in the deletion of an arbitrary file through the User Profile service on behalf of the SYSTEM . The post also mentioned a trick that allows you to use deletion of a file for elevation of privilege (EoP) and run the command line as SYSTEM . The increase was achieved by deleting a file located in a very unexpected place.

The following is a simplified description of the Windows Installer service and the privilege escalation method mentioned.

The Windows Installer service is responsible for installing applications. The application author creates an .msi . This is a database that describes the changes needed to install an application: what folders to create, what files to copy, what registry keys to change, what special actions to perform, and so on.

At the same time, the Windows Installer service implements transaction management - to ensure system integrity in the event of an installation failure and to gracefully rollback the installation. When the Windows Installer makes a change to the system, it creates an entry for it each time. And when the installer overwrites a file on the system with a newer version from the package being installed, it keeps a copy of the old version. The entries allow the service to revert the system to its original state when the installation is rolled back. In the simplest scenario, the entries are stored in the C:\Config.Msi folder .

During installation, the Windows Installer service creates the C:\Config.Msi folder and writes the rollback information there. When changes are made to the system during installation, the service writes information about them each time to a file with the extension .rbs (rollback script) in the C:\Config.Msi folder . Also, when overwriting an older version of a file with a newer one, the installer saves a copy of the original file in the C:\Config.Msi folder . Copies are assigned the extension .rbf (rollback file). If you want to roll back an incomplete installation, the installer service accesses the .rbs and .rbf them to reset the system to its original state.

It is important to protect this mechanism from outside interference. If an attacker can modify the .rbs and .rbf files before they are read, the system could be arbitrarily modified during the rollback. Therefore, the Windows Installer enforces strict access restrictions on files in the C:\Config.Msi .

However, there is a loophole. An attacker could use an arbitrary folder deletion vulnerability to completely delete the C:\Config.Msi folder immediately after it was created by the Windows Installer. attacker can then recreate the C:\Config.Msi folder with reduced DACL permissions (normal users are allowed to create folders in the C:\). And when the Windows Installer creates the rollback files in C:\Config.Msi , the criminal can replace C:\Config.Msi folder with a fake version of it with modified .rbs and .rbf files . Then, during the rollback, the Windows Installer will make arbitrary changes to the system according to malicious scripts.

In this case, the only required exploitation primitive is the ability to delete an empty folder. The primitives of moving or renaming the folder will also work.

From deleting, moving or renaming an arbitrary folder to privilege escalation to the SYSTEM level

In addition to the article , [here](#) is the source code for the privilege escalation exploit described by Abdelhamid Nasser. The exploit can be widely used when there is a primitive to delete, move or rename an arbitrary empty folder in the context of the SYSTEM account or an administrator account. When building an exploit, the Release configuration is selected based on the architecture of the target system (x64 or x86). Once launched, the exploit will prompt you to initiate deletion of the C:\Config.Msi folder . For deletion, you can activate the arbitrary folder deletion vulnerability. Or, for testing purposes, you can simply run rmdir C:\Config.Msi from an elevated command prompt. After a successful launch, the exploit will create a file at C:\Program Files\Common Files\microsoft shared\ink\HID.DLL . Then you can launch the on-screen keyboard (osk.exe), switch to the secure desktop (for example, using the Ctrl-Alt-Delete keys) and thus get a command line with SYSTEM .

The exploit contains an .msi . file .msi , one causes a short delay and the second generates an error. When the Windows Installer service tries to install the .msi , the installation will stop in half and then roll back. By the time the rollback begins, the exploit will have already replaced the contents of the C:\Config.Msi folder with malicious .rbs and .rbf files . The .rbf will contain fragments of the malicious HID.DLL file. And the .rbs file will tell the Windows Installer to "restore" the malicious file to the correct path (C:\Program Files\Common Files\microsoft shared\ink\).

The full principle of the exploit for escalation of privileges looks like this:

The exploit creates a dummy directory C:\Config.Msi and sets an oplock on it.

An attacker activates the vulnerability and deletes the C:\Config.Msi folder (or moves C:\Config.Msi to another location) under the context of the SYSTEM account (or administrator account). account is SYSTEM forced into a waiting state.

The exploit runs a callback to create a soft lock. The following actions are performed in the callback function.

The exploit moves the dummy folder C:\Config.Msi to another location. Thus, the soft lock is maintained and the vulnerable process continues its forced wait. At this time, the C:\Config.Msi becomes available for other purposes (see below).

The exploit creates a new thread that calls the Windows Installer service to install an .msi with a disabled user interface.

The callback thread then resumes and starts checking for the existence of the C:\Config.Msi folder . For some unknown reason, Windows Installer creates the C:\Config.Msi folder , uses it briefly to store a temporary file, deletes it, and then recreates it to store the rollback scripts in the folder. The callback thread checks for the existence of the C:\Config.Msi folder in anticipation of each of the listed actions.

When the exploit detects that the Windows Installer has created the C:\Config.Msi folder a second time, the callback thread ends and releases the soft lock. This allows the affected process to proceed and delete, move, or rename C:\Config.Msi folder created by the Windows Installer.

The main exploit thread resumes. It regularly tries to create C:\Config.Msi folder with a reduced DACL permission. Once the affected process deletes, moves, or renames the C:\Config.Msi folder , the exploit successfully creates a folder with the same name.

The exploit watches the contents of C:\Config.Msi and waits for the Windows Installer to create a .rbs .

The exploit repeatedly tries to move the C:\Config.Msi folder to another location. As soon as the Windows Installer closes the .rbs , the move is executed and the exploit continues.

The exploit creates C:\Config.Msi one last time. There it places a malicious .rbs with the same name as the original .rbs . along with the .rbf file is placed in the folder .rbs .

After implementing the delay and generating the error specified in the .msi , the Windows Installer rolls back the changes. It takes the planted malicious .rbs and .rbf files , and a malicious DLL is injected into the system.

Note that there is a race condition in step 7 that sometimes leads to problems. If the affected process does not immediately activate and delete the C:\Config.Msi folder in time, the opportunity for exploitation of the vulnerability may be missed: the Windows Installer will soon open a handle C:\Config.Msi folder and start writing the .rbs . At this time deleting C:\Config.Msi won't work because the folder won't be empty. This can be avoided by running the exploit on a system with at least 4 processor cores. And the ideal option for an attacker would probably be a "silent" system, not too busy with other activity. In case of failure, you will need to rerun the exploit and activate the vulnerability again.

From Deleting an Arbitrary File to Elevating Privilege to SYSTEM

The technique described above assumes the presence of a primitive that allows you to delete an arbitrary empty folder. However, it is not uncommon to use the primitive for deleting files rather than folders. This was the case when exploiting a bug discovered by Abdelhamid Nasser in the user profile service. Back then, the exploit used one additional trick to elevate privileges to the SYSTEM level, described below.

In NTFS, the metadata (index data) associated with a folder is stored in an alternate data stream for that folder. If the folder is named C:\MyFolder , then the index data is in a stream named

C:\MyFolder::\$INDEX_ALLOCATION . Some details about this mechanism are here. However, for current purposes, it is enough to know the following. Deleting a folder stream ::\$INDEX_ALLOCATION successfully removes the folder from the file system, and a stream name such as C:\MyFolder::\$INDEX_ALLOCATION can be passed to API functions that take a filename as an argument. Among them, for example, is the DeleteFileW .

If you force a process running under the SYSTEM or administrator account to pass an arbitrary string to the DeleteFileW , then this process can be used not only as a file deletion primitive, but also as a folder deletion primitive. account level SYSTEM using the above trick. In this case, the passed string will look like C:\Config.Msi::\$INDEX_ALLOCATION .

Important: success depends on the specific code present in the affected process. If the affected process simply calls the DeleteFileA/DeleteFileW function , everything will work out. However, a process with the necessary privileges can perform other related actions, such as checking the attributes of a specified file. That is why the described scenario cannot be checked from the command line using the del C:\Config.Msi::\$INDEX_ALLOCATION command .

From deleting the contents of a folder to escalating privileges to the SYSTEM level

Let's take it to the next level and assume that a vulnerable process with SYSTEM does not allow you to specify an arbitrary folder or file to delete. However, an attacker can force a process to delete the contents of an arbitrary folder or recursively delete files from a writable folder. This is also how privilege escalation can be achieved: Abdelhamid Nasser demonstrated this in July 2021. He detailed the vulnerability in the SilentCleanup running as SYSTEM . The SilentCleanup through the contents of the temporary folder and removes every file it finds. The researcher used the following technique:

- Create subfolder temp\folder1 .

- Create file temp\folder1\file1.txt .

- Set a soft lock on the file temp\folder1\file1.txt .

- Wait for the affected process to examine the contents of the temp\folder1 folder and attempt to delete the file file1.txt . This will activate the soft lock.

- After activating the soft lock, do the following in the callback function:

- Move file1.txt to another location so that temp\folder1 is empty and can be deleted. The file file1.txt needs to be moved, not just deleted: deletion would require the soft lock to be released first. As a result, the soft lock is maintained and the vulnerable process continues to wait. At this time, the next step

- Re-create temp\folder1 folder as a junction to the \RPC Control in the object namespace.

- Create a symbolic link \RPC Control\file1.txt pointing to

- C:\Config.Msi::\$INDEX_ALLOCATION .

- When the callback ends, the soft lock is released and the affected process continues to run. Deleting the file1.txt deletes the C:\Config.Msi folder .

may recall the technique of exploiting symbolic links and \RPC Control from [symbolic link testing tools](#) James Forshaw However, it is important to note that it is not enough to establish a connection from the temp\folder1 folder to \RPC Control for the arbitrary file deletion vulnerability to work. can \RPC Control not be list-listed, so the affected process cannot find the \RPC Control\file1.txt object by iterating through the file list. You must first create a real file temp\folder1\file1.txt , which will be found by the vulnerable process. Only when the affected process tries to open the file for deletion should the temp\folder1 into a connection pointing to the

object namespace.

The working code of the exploit can be viewed in the FolderContentsDeleteToFolderDelete . The built-in malware detection feature in Windows will mark this process as malicious and terminate it. Therefore, you can add a Process exception for the FolderContentsDeleteToFolderDelete.exe . Two exploits can be chained together. First, the attacker will run the FolderOrFileDeleteToSystem and wait until the file prompts to initiate privileged deletion of the Config.Msi folder. The offender will then execute the command FolderContentsDeleteToFolderDelete /target C:\Config.Msi . You will be prompted to initiate a privileged delete of the contents of the C:\test1 . You can override the location, if needed, with the /initial . For testing purposes, you can run del /q C:\test1* at an elevated command prompt to mimic the privileged folder deletion primitive. As a result FolderContentsDeleteToFolderDelete will delete C:\Config.Msi folder and this will allow the FolderOrFileDeleteToSystem create the HID.DLL file. It remains to call the on-screen keyboard, press the Ctrl-Alt-Delete keys and get a shell with SYSTEM .

From random folder creation

There is one more trick left. It was also reported by Abdelhamid Nasseri. account is required SYSTEM or administrator At first glance, this may not affect security at all (unless the folder is created with reduced DACL permissions). However, it is actually possible to implement malicious actions. The task is to create a folder like this: C:\Windows\System32\cng.sys .

Typically, a file or folder with that name does not exist. But if an attacker places an extraneous file or even an empty folder in the specified location of the file system, the Windows boot process will be disrupted.

The principle of operation of this method is not completely clear. It looks like Windows is trying to load the cng.sys from the wrong location and is failing. At the same time, the retry logic is not implemented, which would allow the system to continue searching and find the required driver. The result is a complete inability to boot the system. Other drivers can be used to achieve the same effect.

Depending on the vulnerability, the described exploit can even be used remotely. It does not require anything other than the ability to embed a folder or file in the right place.

Conclusion

The considered techniques prove that some rather weak exploitation primitives can be effectively used. The following was shown:

- account SYSTEM can be used to elevate privileges to the SYSTEM .

- account SYSTEM can usually be used to elevate privileges to the SYSTEM .

- account SYSTEM can be used to elevate privileges to the SYSTEM .

- Recursively deleting the contents of a fixed folder can be used to elevate privileges to the SYSTEM if the folder is writable by an attacker as the SYSTEM or administrator account.

- account SYSTEM (even without content control) can be used to implement malicious actions.

Author Abdelhamid Nasseri <https://twitter.com/thezdi>

InspectorGadget - December 20, 2022, 1:56 pm

So you just ripped somebody's tutorial and gave a one line of credit at the end..... and then spelt Explanation wrong in the title. Jesus, doesn't even deserve a GG.

AzureGod - December 20, 2022, 1:58 pm

(Dec 20, 2022 02:56 PM)GismoGuy Wrote: [\[CITE\]](#) So you just ripped somebody's tutorial and gave a one line of credit at the end..... and then spelt Explanation wrong in the title. Jesus, doesn't even deserve a GG.
Better now?

InspectorGadget - December 20, 2022, 2:00 pm

(Dec 20, 2022 02:58 PM)AzureGod Wrote: [\[CITE\]](#) Better now?
Yeah, thanks! Also having re-read my post it seems abit too aggressive - sorry as it seemed I took my bad day out on you!

AzureGod - December 20, 2022, 2:01 pm

(Dec 20, 2022 03:00 PM)GismoGuy Wrote: [\[CITE\]](#) Yeah, thanks! Also having re-read my post it seems abit too aggressive - sorry as it seemed I took my bad day out on you!
Hey, i am happy you are alert, thank you for the help bud!

memory - December 21, 2022, 7:11 pm

This is a fairly cool article however it would've been even cooler for a PoC or direct link to the tweet.

Thanks for sharing!

Hackers - December 22, 2022, 5:28 am

I did not read the entire article, but most of the first half. This seems logical & I can understand how hackers were able to manipulate the system perms. It's a little above my head in the sense that the chances of me ever discovering a vuln like this on my own are slim to none - as of today - and my chances of using something like this in the future are pretty slim as well.

Good post & I like how it touches on so many aspects of security with owner permissions. Symlinks were the bane of my existence at one point. We had a server get hacked and for over a year it was constantly targeted. Turns out the guy had made a symlink and that's how he was able to keep accessing the server because the file he sync'd with was a config file, and the new file was in a directory they could access publicly, so every time we locked down server he always had access to MySQL. Then he manipulated MySQL by putting in paths with ../../ (file path traversal) in it. This affected that software because it would use the values in the database (that were pre-sanitized / validated on INSERT, but not on READ ops) to generate new files, so the hacker was able to use this to create php files.