



## Micro Patching CVE-2021-41379 Variant

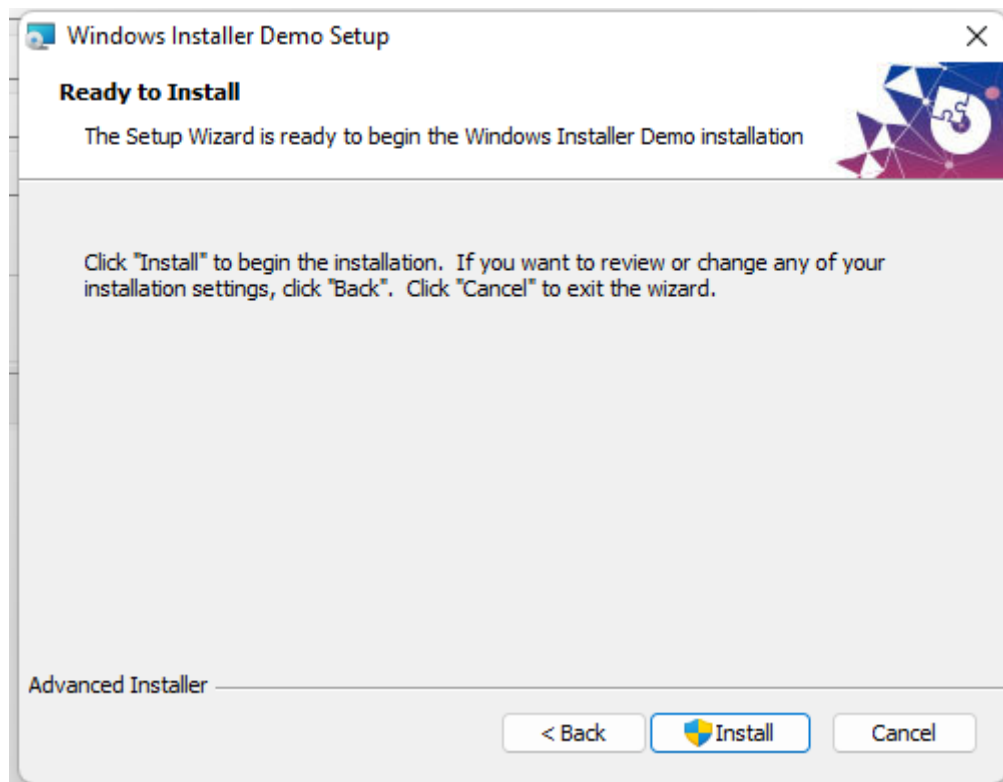
### ***Agenda:***

- Introduction To Windows Installer.
- Why Windows Installer Service Was Introduced ?
- The issue with Windows Installer in Windows Server Installations From Attacker Perspective.
- CVE-2021-41379 Variant Root Cause And Execution Flow
- The Current Work-Around Is Trash
- Micro-patching Windows Installer Service

*\*sorry this paper won't talk much about the installer because it was only written for the 0day bug*

*\*Tests are being done on a Hyper-V VM with Windows 10 21H2 installed*

# 1. Introduction To Windows Installer



Windows Installer was first introduced with Office 2000 as its installer and was released as a core component of Windows starting with Windows 2000.

The latest version of Windows Installer which is 4.5 exposes a variety of features to developers allowing them to Install, repair, uninstall, upgrade, patch, and advertise msi packages.

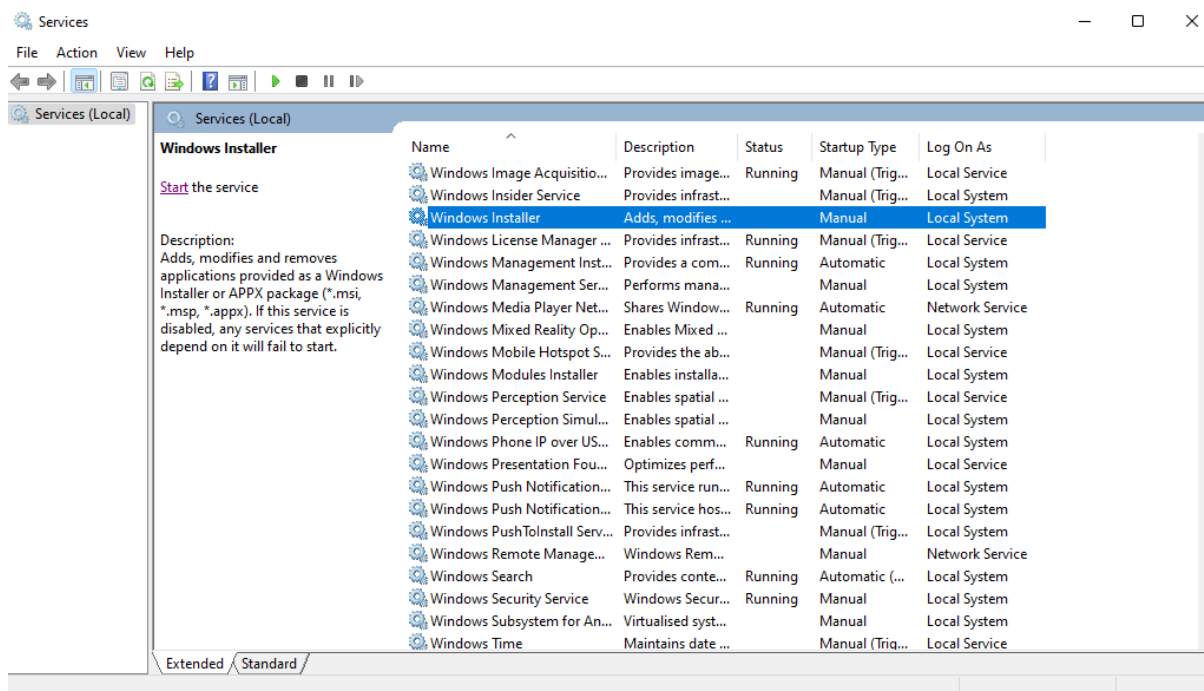
An Msi package isn't just a database or file packer, an Msi package more like an executable parsed by Windows Installer, ensuring the execution of every instruction and extracting/updating object with its appropriation location.

What makes windows installer unique, unlike any other solutions available is the fact that you can specify custom ACL to your newly created objects including but not limited to files/keys/services.

Another feature that makes windows installer another masterpiece of software, is the rollback mechanism. When an installation fail or a user click cancel during an update, Windows Installer rollback all the changes made to your computer and make your installation clean, just like never ever happened!

This of course is achieved through some overly complicated procedures which I won't detail about now.

## 2. Why Windows Installer was introduced ?



Windows Installer service was introduced around Windows XP, the service is named “msiserver” with display name “Windows Installer” with a default command line “C:\Windows\System32\msiexec.exe /V”

The service runs as Local System which is the highest privileges available on Windows installations to serve Windows Installations.

Windows Installer service also exposes a COM interface allowing all kinds of accounts to launch an MSI operation, including non-administrators.

When I started doing research on Windows Installer service, I always wondered why Windows Installer needed to run as Local System instead of just running as the caller. Windows actually has the concept of System Access Control List (SACL), this allows administrators to audit access to objects that support ACL in Windows. But the thing is, to access to SACL you need a special privilege known as SeSecurityPrivilege, this privilege is strictly assigned to Administrators and Local System account but not standard users accounts.

One of the reasons why Windows Installer service was introduced, is to ensure the honour of SACL during MSI operations. Let's take by example, if a user clicked cancel during an installation. The rollback mechanism will ensure to restore the SACL in this case. Which wouldn't be possible if the installer was running as the user.

But unfortunately, every feature comes with a cost. In this case, the cost is security vulnerabilities. If the installer service failed to parse an MSI package or allowed an elevated operation to be executed then attackers can abuse the installer for their personal gain.

While Microsoft puts great efforts to fix those bugs but unfortunately, it's still not enough to ensure the stability of such huge component.

### **3. The issue with Windows Installer in Windows Server Installations From Attacker Perspective**

While Windows Installer service remains accessible to standard users in desktop installations.

It's unfortunately, not the case in Windows Server installations. Thanks to group policy, windows installer is blocked from providing its services to non-administrators due to the default value of "Prohibit Users Install" which is set to 1 in windows server installation and 0 to desktop installations.

Of course, this decision wasn't taken randomly. A desktop user is more likely to install a per-user msi package than a server user.

While this mitigates a great attack surface on windows server but windows installer support a feature that doesn't honour group policy, which is the Administrative Install.

The name and the feature by itself aren't too common, the fact that non-administrators are allowed to launch administrative installs even in a windows server make it strangely weird, maybe this behaviour wasn't intended at all ?

I wasted several hours around this feature, but as far as I can conclude. This feature never executes the instruction of the package but only extract its content. And for some reasons, the extraction only focuses on files but not any other objects such as registry keys, services, COM+ interfaces...

Although this block a great attack surface but bugs such as CVE-2020-0841, CVE-2021-41379 and its variants are still exploitable.

## 4. CVE-2021-41379 Variant Root Cause

Source : <https://www.secpod.com/blog/new-windows-installer-zero-day-flaw-exploited-in-the-wild/>

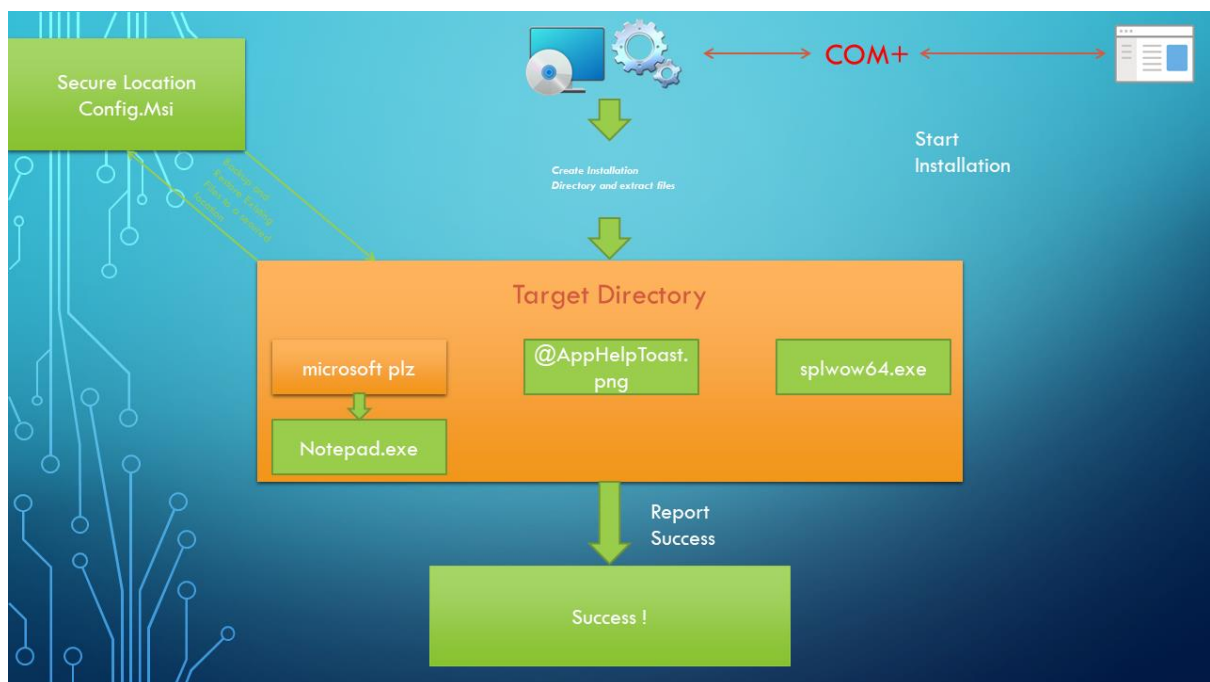


The variant was discovered during the analysis of CVE-2021-41379, this bug is not a bypass of CVE-2021-41379 but more like a variant. But somehow the media still claimed that this bug is a bypass.

I publicly disclosed the issue in November 22 because Microsoft were idiots about their bounty program recently, but that's not our topic.

With luck and using some code coverage techniques, I was successfully able to produce an msi that triggers a unique behaviour in windows installer service during administrative install. Surprisingly, the package which bypasses CVE-2021-41379 patch completely broke windows installer but this one does not, that's why I considered dropping this bug instead of CVE-2021-41379 bypass.

This is the normal execution flow of test pkg.msi



While this package is perfectly safe, ESET decided to be smart and block it

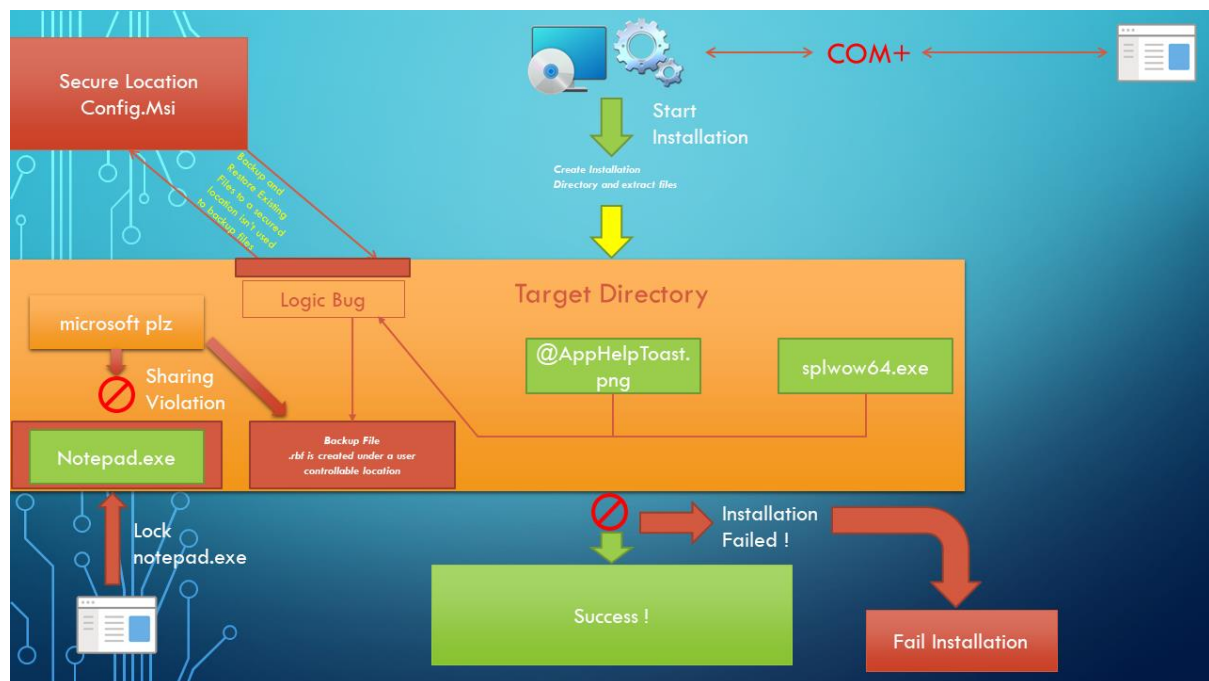
The screenshot shows the ESET security interface. On the left, there is a circular progress indicator with the number "1" and "/ 56". Below it, there is a "Community Score" section with a question mark icon and a "Community Score" label. On the right, there is a red warning icon and the text "1 security vendor flagged this file as malicious". Below this, the file hash "37ff2ed054e6ef0d9f792ee8d190bdb6951ce0aace044611d91d8b595c5737d7" is displayed, followed by the filename "test pkg.msi". Below the filename, there are several tags: "checks-usb-bus", "cve-2021-41379", "direct-cpu-clock-access", "exploit", "msi", and "runtime". At the bottom, there is a navigation bar with tabs: "DETECTION", "DETAILS", "RELATIONS", "BEHAVIOR", and "COMMUNITY" (which is selected and has a "2" badge). Below the "DETECTION" tab, the text "ESET-NOD32" is displayed, followed by a red warning icon and the text "Win32/Exploit.CVE-2021-41379.B".

I'm pretty sure if they don't use hash-based detection, they'll end up having false positives.

While reverse engineering, I noticed some strange behaviour in windows installer. The installer sometimes creates the rollback file (.rbf) outside the secured directory Config.Msi without impersonation.

Triggering this behaviour wasn't easy at all and somehow, I was lucky enough to produce this test pkg.msi that trigger this behaviour under certain circumstances.

This the execution flow of the msi package while notepad.exe is locked by file sharing and locking process is inaccessible by the calling user



As you might notice the rollback file is created under a user controllable location without impersonation. This happen due to a design flaw, CMsiOpExecute::ixfFileCopy actually end up calling MsiCreateFileWithUserAccessCheck without impersonation in a user write-able location.

Although there's a path verification in CMsiFileCopy::OpenDestination

```
v25 = (*(__int64 (__fastcall **)(_QWORD)))(**((_QWORD *)this + 8) + 80i64))(*((_QWORD *)this + 8));
LOBYTE(v26) = v4;
LODWORD(dwCreationDisposition) = v24;
v27 = MsiCreateFileWithUserAccessCheck(v25, &v58, v15, v26);
*((_QWORD *)this + 6) = v27;
if ( v27 != -1 && !CMsiFileCopy::VerifyDestination(this, v4) )
```

But it happens a bit too late because MsiCreateFileWithUserAccessCheck already set a write-able security descriptor for the file.

The bug can be simply abused by redirecting the rollback file creation into an arbitrary location, it's not as easy as it sounds but it's already implemented in the PoC. So, I won't talk about it too much, go read some code.



## 5. The Current Work-Around Is Trash

I managed to get my hands on several papers that attempted to mitigate this issue, all of them were completely inefficient to mitigate the full potential of this vulnerability.

While some actually attempted to block any package called “test pkg.msi” from being installed, seriously, is this your best ?

And for the others, they blocked Microsoft Edge elevation service binary hijacking. I found it too insulting to talk about it. I mean you can literally hijack any \*other\* binary and still have system.

The only ones who actually DO something about a bug are 0patch, but I guess they still got a lot of work to do.

Also, I’m pretty sure that Microsoft won’t fix that till January 2022. It’s really a mess there in windows installer.

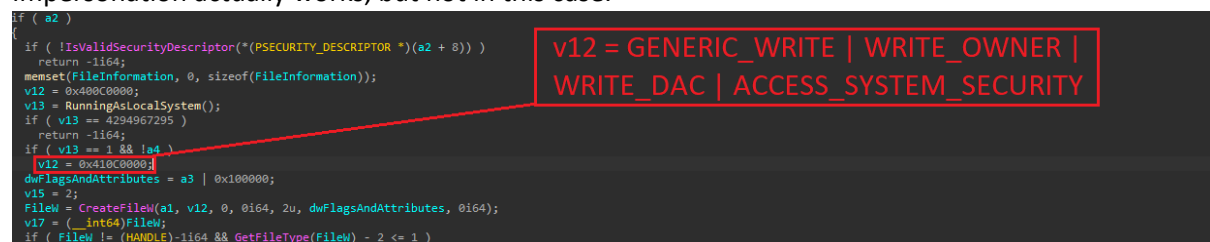
## 6. Micro-patching Windows Installer Service

While I claimed in the repository that the bug is incredibly is not possible to fix, I was kind of wrong and right. I'll explain why.

Micro Patching this kind of issue can be usually done by either verifying the path integrity using `GetFinalPathNameByHandle` or impersonating the caller in case of attacker having control of the target path.

### Why Impersonation didn't work?

Impersonation actually works, but not in this case.



```
if ( a2 )
{
    if ( !IsValidSecurityDescriptor(*(PSECURITY_DESCRIPTOR *)(a2 + 8)) )
        return -1i64;
    memset(FileInformation, 0, sizeof(FileInformation));
    v12 = 0x40000000;
    v13 = RunningAsLocalSystem();
    if ( v13 == 4294967295 )
        return -1i64;
    if ( v13 == 1 && !a4 )
        v12 = 0x410C0000;
    dwFlagsAndAttributes = a3 | 0x100000;
    v15 = 2;
    FileW = CreateFileW(a1, v12, 0, 0i64, 2u, dwFlagsAndAttributes, 0i64);
    v17 = (__int64)FileW;
    if ( FileW != (HANDLE)-1i64 && GetFileType(FileW) - 2 <= 1 )
```

**v12 = GENERIC\_WRITE | WRITE\_OWNER | WRITE\_DAC | ACCESS\_SYSTEM\_SECURITY**

The installer attempts to access to the SACL of the rollback file, if we simply impersonated the user. Accessing to the file will fail with `ERROR_PRIVILEGE_NOT_HELD`

### Why Path Verification didn't work?

Actually, it worked but not for long. I actually wrote an entire module, which hook `CreateFileW` calls using detours library from Microsoft and look for specific `CreateFileW` calls which try to access to .rbf files with the access mask shown above.

The hook was efficient enough against the bug. But unfortunately, I bypassed my own patch.

I managed to drop an arbitrary .rbf file in System32 with a write-able ACL, the file name cannot be controlled by the user. But we can't accept the risk of allowing users to create write-able files under secured locations.

## How did the path verification work?

```
// we are specifically looking to hook CreateFileW in MsiCreateFileWithUserAccessCheck, so if a CreateFileW arguments doesn't match those, we will simply ignore them
if (dwDesiredAccess != 0x40C0000 || dwShareMode != NULL || dwCreationDisposition != CREATE_ALWAYS)
    goto ContinueUnhooked;
extension = PathFindExtensionW(lpFileName);
if (wcsncmp(extension, L".rbf") != 0)
    goto ContinueUnhooked;
dwDesiredAccess |= DELETE;
return_value = UnhookedCreateFileW(lpFileName, dwDesiredAccess, dwShareMode, lpSecurityAttributes, OPEN_ALWAYS, dwFlagsAndAttributes, hTemplateFile);
if (return_value == INVALID_HANDLE_VALUE)
    return return_value;
//handle existing file
if (GetLastError() == ERROR_ALREADY_EXISTS) {
    DWORD lasterr = 0;
    IsTrusted = VerifyPathIntegrity(return_value, (WCHAR*)lpFileName);
    if (!IsTrusted) {
        SetLastError(ERROR_ACCESS_DENIED);
        CloseHandle(return_value);
        return INVALID_HANDLE_VALUE;
    }
    //overwrite the file
    FILE_END_OF_FILE_INFO end_of_file_info = { 0 };
    end_of_file_info.EndOfFile.QuadPart = 0;
    SetFileInformationByHandle(return_value, FileEndOfFileInfo, &end_of_file_info, sizeof(end_of_file_info));
    SetLastError(lasterr);
    return return_value;
}
//handle a newly created file
IsTrusted = VerifyPathIntegrity(return_value, (WCHAR*)lpFileName);
if (!IsTrusted) {
    _FILE_DISPOSITION_INFO fdi = { TRUE };
    SetFileInformationByHandle(return_value, FileDispositionInfo, &fdi, sizeof(fdi));
    SetLastError(ERROR_ACCESS_DENIED);
    CloseHandle(return_value);
    return INVALID_HANDLE_VALUE;
}
return return_value;
ContinueUnhooked:
return UnhookedCreateFileW(lpFileName, dwDesiredAccess, dwShareMode, lpSecurityAttributes, dwCreationDisposition, dwFlagsAndAttributes, hTemplateFile);
```

The path verification checks if the file being accessed is the .rbf file, the PoC then open the file with OPEN\_ALWAYS instead of CREATE\_ALWAYS to not overwrite the file and causes arbitrary file overwrite.

If opened existing file, the module will verify the path integrity and if it matches the PoC Overwrite the file and continue. Otherwise, it fails with ACCESS\_DENIED.

If created new file, the module verifies path integrity and if it matches it returns a success. Otherwise, the newly created file is removed to avoid any issues and ACCESS\_DENIED is returned.

## Before the Patch:

The screenshot displays the Windows Event Viewer interface. The main pane shows a list of events with columns for Time, Process Name, PID, Operation, Path, Result, and Detail. The events are filtered by 'Process Name' and show a sequence of file operations. A specific event is highlighted in blue, and its details are shown in the right pane. The event details include:

- Date:** 29/11/2021 01:54:25.3924160
- Thread:** 3664
- Class:** File System
- Operation:** CreateFile
- Result:** SUCCESS
- Path:** C:\test\microsoft.ppt\11877.rbf
- Duration:** 0.0001798
- Desired Access:** Generic Read, Read Attributes, Write DAC, Write Owner, Access System Security
- Disposition:** OverwriteIf
- Options:** Synchronous IO Non-Alert, Non-Directory File, Open No Recall
- Attributes:** n/a
- ShareMode:** None
- AllocationSize:** 0
- OpenResult:** Created

The event details also show a 'Desired Access' section with a list of permissions and a 'Result' section with a list of permissions. The event is categorized as 'Information' and is part of the 'System' log.

## After the Patch:

The screenshot displays the Windows Event Viewer interface. On the left, a list of events is shown, with the 'CreateFile' event selected. The right pane provides a detailed view of this event. The 'Event Properties' window is open, showing the 'Process' tab. The 'Process' tab indicates that the event was triggered by the 'C:\inetmimicsoft\pic\15349b7\df' process. The 'Event Properties' window also shows the 'Disposition' as 'Created' and the 'Path' as 'C:\inetmimicsoft\pic\15349b7\df'.

Why didn't this work?

As I said before, the attacker has a dangerous control over the target directory.

In this case the attacker completely controls the target installation directory, so if he supplies C:\Windows\System32 as an installation directory, the checks will return a success and the patch will be bypass. Although the attacker won't control the file name but let's say if someone discovered a file creation weaponization technique that's similar to Microsoft Diagnostic Hub dll load technique, he may still end up having LPE which is something we won't accept.

**HELP ????!?!??? Impersonation and path checking didn't work !!!**

The bug can't be patched, we are under Microsoft mercy which isn't going to be any good till the next patch, AAAAAAGGGGHHH.

## Attempt #2 to patch it

Instead of lurking around with CreateFileW, I decided to actually fix the root cause.

First Windows Installer never create .rbf files outside of Config.Msi directories except for the case of file shares and any other local drive rather than Systemroot.

I decided then to block any rollback file creation outside the Config.Msi directories and inside Systemroot driver, this worked perfectly fine. And the issue was mitigated.

Before the Patch:

The screenshot displays the Windows Event Viewer interface. On the left, a list of events is shown, with one event selected and highlighted in blue. The main pane on the right shows the details of this event, including its date, time, and a description. The event is a 'Generic: Write, Read Attributes, Write DAC, Write Owner, Access System Security' operation. The description indicates that the operation was successful and that the file was created. The event is categorized as 'Information' and is associated with the 'Microsoft Windows' source.

After the patch:

The screenshot displays the Windows Event Viewer interface after the patch. The list of events on the left is shorter than in the previous screenshot. The main pane on the right shows the details of a selected event, which is a 'Generic: Write, Read Attributes, Write DAC, Write Owner, Access System Security' operation. The description indicates that the operation was successful and that the file was created. The event is categorized as 'Information' and is associated with the 'Microsoft Windows' source.

Now after the patch, the installer never creates .rbf files inside Config.Msi which mitigate the issue.

Tests were done with several .msi package while the hooking module was loaded, none of them shown any unattended behaviour. Which mean, Great Success!

## How To Use?

Make sure you're on x64 installation, and inject msi\_hook.dll (x64 release build) inside msixec.exe process that runs as SYSTEM.

I used this guy tool, <https://github.com/master131/ExtremInjector/releases/tag/v3.7.3> with manual mapping to inject dll into windows installer.

Most stable version of PoC -

<https://github.com/klinix5/InstallerFileTakeOver/tree/e7751f3e6ed992d0b2262670941c4b6fc1173a9d>

Msi\_Hook Repository - [https://github.com/klinix5/InstallerFileTakeOverPatch/tree/main/msi\\_hook](https://github.com/klinix5/InstallerFileTakeOverPatch/tree/main/msi_hook)

Oh wait, you're Microsoft?

Tired of seeing your 0day bugs being publicly disclosed ?

You can always hire me; you know where to find me !

Hold on, you're not Microsoft?

You got a position for me in EU, UK or Canada?

You can still hire me, get in touch with me in LinkedIn!

*\*And no, I'm not interested in remote positions. Sorry!*

*Skills: I do now this stuff + I can develop win32 apps and kernels.*