

Evil MSI. A story about vulnerabilities in MSI Files

 [cicada-8.medium.com/evil-msi-a-long-story-about-vulnerabilities-in-msi-files-1a2a1acaf01c](https://medium.com/@cicada-8/evil-msi-a-long-story-about-vulnerabilities-in-msi-files-1a2a1acaf01c)

CICADA8

August 30, 2024



CICADA8



Hello everybody, my name is [Michael Zhmailo](#) and I am a penetration testing expert in the MTS Innovation Center [CICADA8](#) team.

You have probably come across MSI files quite often. They are used by software manufacturers to provide their programs. This format is more convenient than the standard EXE format for the following reasons:

- Ability to restore, install certain components
- Data storage in well-structured tables that can be easily accessed via APIs
- Easy distribution via SCCM, WEB endpoints

There may be various vulnerabilities inside MSI files, most of which will lead to **privilege escalation**. These include both logical vulnerabilities: DLL/TypeLib/COM/Exe File/Script/etc hijacking, PATH Abusing, and vulnerabilities of the MSI file format itself: Custom Actions Abuse, abandoned credentials, privileged child processes.

You may have read about vulnerabilities in MSI files before:

These are great articles if you are just getting familiar with finding vulnerabilities inside MSI files. Our article can also serve as a great starting point for learning more about this topic. In addition, we have developed a tool called [MyMSIAnalyzer](#) that will make it easier to find vulnerabilities inside MSI files. You should read this article if you want to learn more about the insides of the MSI format and how the tool works.

MSI File Format

The MSI format itself is somewhat similar to SQL databases. Inside the MSI file there are tables with various data. There is a relationship between the tables. And this table is analyzed and used while installing MSI file.

msi; format



MSI File Format

I note that there are a lot of tables. We are interested in only a few of them. The full list of tables is described [here](#).

- — a special table inside which resources used by the application (images, shortcuts, icons, etc.) are located;
- Each resource is associated with a specific functionality. Therefore, there is a , which is linked to the Components Table through the ;
- — a table that specifies which files should be installed on the system;
- — table that contains information about the folder structure of the program to be installed;

- — table that contains actions to be performed during MSI file installation (create a shortcut, create a registry key, write a value);
- — actions that need to be performed during the installation process, however, they cannot be performed through Windows Installer API, so third-party programs, DLL files, cmd commands are used.

Collecting MSI files for analysis

Manually

The easiest way to locate all MSI files is to look in the `C:\Windows\Installer` folder. Here you will definitely find all MSI files of programs installed on your computer.

Имя	Дата изменения	Тип	Размер
(0CB63CB8-B9EC-42FE-813C-0ED7AB4BA67B)	17.08.2023 11:01	Папка с файлами	
(0E6EEAC9-4913-4C2F-B7D2-761B27C35D7C)	25.10.2022 18:39	Папка с файлами	
(0E992720-1330-4AB3-8155-255F79785535)	01.06.2023 20:49	Папка с файлами	
(1D35D9F5-2C5C-4570-BDCD-C8CE26BE9278)	02.08.2023 12:02	Папка с файлами	
(2DB3E4A5-19C0-417F-A871-852E39D4B4AA)	02.08.2023 12:02	Папка с файлами	
(2FDB79CE-5193-4A39-82BB-E00158CC1533)	06.03.2023 14:21	Папка с файлами	
(03BC58B0-243D-47CA-8BDB-9671CC98B08B)	08.04.2023 16:37	Папка с файлами	
(3E031701-D71E-4818-9419-2FE5B3D9D3F4)	04.08.2023 21:01	Папка с файлами	
(3EF1E212-048C-4527-88D8-0F9E1F7D9C02)	02.08.2023 12:01	Папка с файлами	
(4BC53C5C-0B22-4040-ABD5-7C036D2AD487)	30.10.2023 20:20	Папка с файлами	
(9B1D4AB1-1D0E-4F67-9542-742D78DCE071)	08.04.2023 16:37	Папка с файлами	
(13B6E68B1-7498-48AB-9D22-AD3A86532531)	11.09.2023 9:11	Папка с файлами	
(31EEE27D-9C67-46D6-BC89-0DCC5F538462)	01.05.2023 20:28	Папка с файлами	

Folder Contents #1

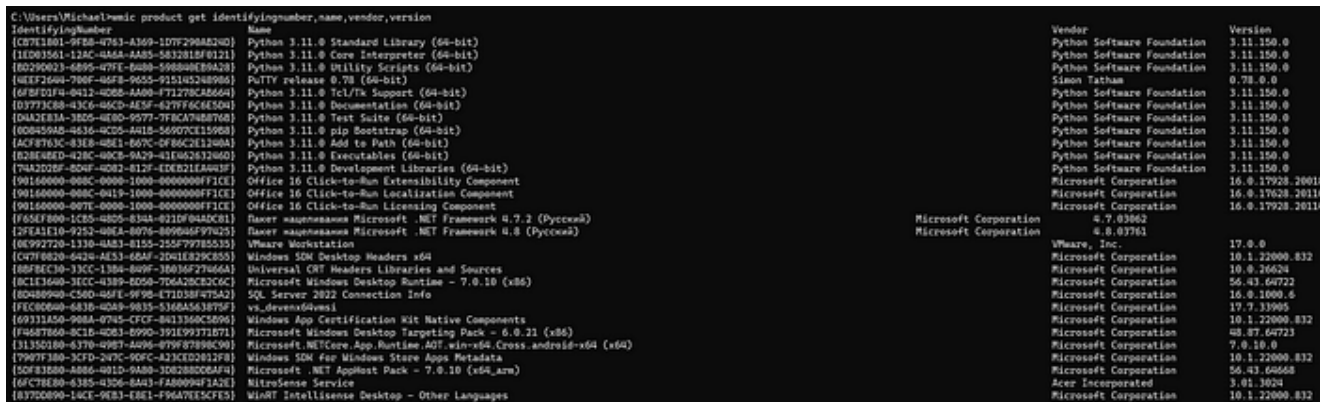
Имя	Дата изменения	Тип	Размер
MSIF2A5.tmp-	20.07.2023 15:54	Папка с файлами	
1c0cb64.msi	25.08.2024 23:36	Пакет установщика Wi...	304 КБ
1c0cb77.msi	25.08.2024 23:36	Пакет установщика Wi...	396 КБ
1c0cb82.msi	25.08.2024 23:36	Пакет установщика Wi...	276 КБ
1c0cb89.msi	25.08.2024 23:36	Пакет установщика Wi...	472 КБ
1c0cb90.msi	25.08.2024 23:36	Пакет установщика Wi...	148 КБ
1c48e0.msi	25.08.2024 23:36	Пакет установщика Wi...	174 КБ
1c48e1.msp	28.06.2011 22:21	Исправления установ...	4 529 КБ
1c54b5.msi	25.08.2024 23:36	Пакет установщика Wi...	148 КБ
1c54bc.msi	25.08.2024 23:36	Пакет установщика Wi...	148 КБ
1c54c3.msi	25.08.2024 23:36	Пакет установщика Wi...	140 КБ
1c54ca.msi	25.08.2024 23:36	Пакет установщика Wi...	140 КБ
1c919a9.msi	25.08.2024 23:36	Пакет установщика Wi...	184 КБ
1d6fbec.msi	25.08.2024 23:36	Пакет установщика Wi...	600 КБ

Folder Contents #2

Inside the folder you can find MSI files and other folders. Other folders often store various resources that the MSI file needs. Their name is GUID. This GUID can be seen in the *IdentifyingNumber* field of the installed software product.

You can examine the installed programs and make a mapping using these commands:

```
wmic product identifyingnumber,name,vendor,version
```



IdentifyingNumber	Name	Vendor	Version
{C0713801-9FEB-4761-A369-1D7F2968240D}	Python 3.11.0 Standard Library (64-bit)	Python Software Foundation	3.11.150.0
{11D83501-13AC-484A-A835-5832181F0121}	Python 3.11.0 Core Interpreter (64-bit)	Python Software Foundation	3.11.150.0
{82290923-6895-477C-6688-596668E80428}	Python 3.11.0 Utility Scripts (64-bit)	Python Software Foundation	3.11.150.0
{4E5F2664-706F-46F3-9653-915185248986}	PuTTY release 0.78 (64-bit)	Simon Tatham	0.78.0.0
{6F8FD1F4-8412-4058-AA09-F71279CA5664}	Python 3.11.0 Tcl/Tk Support (64-bit)	Python Software Foundation	3.11.150.0
{D3773C68-43C6-46CD-4E3F-627776C8E5D4}	Python 3.11.0 Documentation (64-bit)	Python Software Foundation	3.11.150.0
{0A82E111-3026-4036-9277-782C47483760}	Python 3.11.0 Test Suite (64-bit)	Python Software Foundation	3.11.150.0
{00045640-4616-4C23-4018-56907C115968}	Python 3.11.0 pip Bootstrapper (64-bit)	Python Software Foundation	3.11.150.0
{4CF8763C-83E8-48E1-667C-0F86C2E1260A}	Python 3.11.0 Add to Path (64-bit)	Python Software Foundation	3.11.150.0
{82848ED0-428C-480C-9A29-41E65263266D}	Python 3.11.0 Executables (64-bit)	Python Software Foundation	3.11.150.0
{79A2720F-6D4F-4C02-812F-1D6B21F8493F}	Python 3.11.0 Development Libraries (64-bit)	Python Software Foundation	3.11.150.0
{90166000-808C-4006-1000-9000000FF1CE}	Office 16 Click-to-Run Extensibility Component	Microsoft Corporation	16.0.17928.20011
{90166000-808C-4019-1000-9000000FF1CE}	Office 16 Click-to-Run Localization Component	Microsoft Corporation	16.0.17628.20111
{90166000-807E-8006-1000-9000000FF1CE}	Office 16 Click-to-Run Licensing Component	Microsoft Corporation	16.0.17928.20111
{F65E7880-1CE5-4805-8344-021D044DC81}	PowerShell 7.2.2 (Pyccexe)	Microsoft Corporation	4.7.03062
{27FA1110-9252-48E1-8076-809680F79629}	PowerShell 7.2.2 (Pyccexe)	Microsoft Corporation	4.7.03761
{86992720-1330-4A81-8155-265F79785351}	VMware Workstation	VMware, Inc.	17.0.0
{C970820-6424-4E51-60AF-2D01E829C855}	Windows SOM Desktop Headers x64	Microsoft Corporation	10.1.22000.832
{88F8EC10-13CC-1384-809F-38036F276664}	Universal CRT Headers Libraries and Sources	Microsoft Corporation	10.0.26624
{8C1E3680-10CC-4389-4D59-7D6A28C8C6C}	Microsoft Windows Desktop Runtime - 7.0.10 (x86)	Microsoft Corporation	56.43.66722
{80409980-5360-46F8-9978-171D33F47842}	SQL Server 2022 Connection Info	Microsoft Corporation	16.0.10800.6
{FEC80680-6810-4049-9835-5168A563875F}	vs.devenv@msi	Microsoft Corporation	17.7.13905
{69311A50-908A-9745-CFCF-841366C5896}	Windows App Certification Kit Native Components	Microsoft Corporation	10.1.22000.832
{F6687860-8C16-4081-0990-391E99771B71}	Microsoft Windows Desktop Targeting Pack - 6.0.21 (x86)	Microsoft Corporation	48.87.64723
{131352180-6370-4797-4096-979F07898C90}	Microsoft.NETCore.App.Runtime.AOT_win-x64_Cross.android-x64 (x64)	Microsoft Corporation	7.0.10.0
{7997F380-83FD-474C-90C5-523C02812F81}	Windows SOM for Windows Store Apps Metadata	Microsoft Corporation	10.1.22000.832
{50F3E800-4866-401D-9A00-3D628D206AF4}	Microsoft .NET Apollon Pack - 7.0.10 (x64_arm)	Microsoft Corporation	56.43.66668
{6FC78E80-6385-43D6-8A43-F480994F3A2E}	NitroSense Service	Acer Incorporated	3.01.3024
{8172D080-14C1-9E81-E8E1-F96A7E35CFE5}	WinRT Intelligence Desktop - Other Languages	Microsoft Corporation	10.1.22000.832

List of the installed products

You can also use Powershell and add a filter by software.

```
Get-WmiObject - | ? { $_.Name -like } | select IdentifyingNumber,Name
```

Tools

Of course, it is more convenient to use automated tools to gather information and MSI files itself.

- This tool can be used to search for msi files, then download them and then analyze them directly on the attacker's machine to detect privilege escalation vectors (can be analyzed for privilege escalation vectors using, for example, our tool :) ;
- Great for extracting MSI files from SCCM. For example, from Distribution Points;

```
PS> Invoke-CMLootInventory -SCCMHost sccm01.domain.local -Outfile sccmfiles.txt
```

```
PS> Invoke- -InventoryFile .\sccmfiles.txt -Extension msi
```

— python version of CMLoot.

Web

You can find files for analysis on the internet as well. For example, you can use Google Dorks:

msi



GitHub
<https://github.com/download> · Перевести эту страницу

Download

Информация об этой странице недоступна.
Подробнее...



Far Manager
<https://www.farmanager.com/download> · Перевести эту страницу

Far Manager Official Site : download

Far Manager v3.0 build 6300 x86 (2024-04-07). Last change: 15c067185 · **download** · Far Manager v3.0 build 6300 x64 (2024-04-07). Last change: 15c067185 · **download**



Far Manager
<https://www.farmanager.com/download> · Перевести эту страницу

Far Manager Official Site : download

17 авг. 2024 г. — Stable builds · Far Manager v3.0 build 6300 x86 (2024-04-07). Last change: 15c067185 · **download** · Far Manager v3.0 build 6300 x64 (2024-04-07).



Teambition
<https://www.teambition.com/download> · Перевести эту страницу

Download

Информация об этой странице недоступна.
Подробнее...



windows.net
<https://hmsnetworks.blob.core.windows.net/nlw/docs...> · Перевести эту страницу

<https://hmsnetworks.blob.core.windows.net/nlw/docs...>

Информация об этой странице недоступна.
Подробнее...



downloadspin.com
<https://www.downloadspin.com/download-net-component-dll-to-process-pdf> · Перевести эту страницу

Download .NET Component DLL to Process PDF

Download Aspose.PDF to Process & Manipulate PDF Files. Open NuGet Package Manager within the Microsoft Visual Studio®, search for Aspose.PDF and install.



trimbleaccess.com
<https://tim.trimbleaccess.com/installation-manager> · Перевести эту страницу

Trimble - Installation Manager

Use Trimble Installation Manager to **download** and install the following software: Trimble SiteVision; Penmap Windows Field Data Collection; Penmap Project ...



equa.se
<https://es-so-portal.equa.se/esbo> · Перевести эту страницу

ESBO

Download ESBO Light - Free version* · **Download ESBO - Paid version***. *The programme is

Or use special resources with a list of MSI files:

Searching for vulnerabilities

Abandoned credentials

It's the simplest option. Inside MSI files, it is possible to find leftover passwords, API keys, endpoints and other data that may be of interest to us as attackers.

We have dedicated a [CredFinder](#) class in MyMSIAnalyzer for credential discovery. Searching for credentials works to the point of simplicity. It checks all properties of the MSI file and tries to find sensitive information by keywords.

```

static public string[] keywords = new string[]
{
    "USERNAME",
    "PASSWORD",
    "USER",
    "PASS",
    "PFX",
    "CERTIFICATE",
    "PRIVATE",
    "KEY",
    "API",
    "PATH",
    "FOLDER"
};

static public string[] blacklistedKeywords = new string[]
{
    "ALLUSERS",
    "nousername"
};

public static void FindCredentials(string msiPath)
{
    try
    {
        using (Database db = new Database(msiPath, DatabaseOpenMode.ReadOnly))
        {
            var sql = "SELECT `Property`, `Value` FROM `Property`";
            using (var view = db.OpenView(sql))
            {
                view.Execute();

                foreach (var record in view)
                {
                    var property = record.GetString("Property").ToLower();
                    var value = record.GetString("Value");
                    var containsBlackListedKeyword = false;

                    foreach (var blKw in blacklistedKeywords)
                    {
                        if (property.Contains(blKw.ToLower()))
                        {
                            containsBlackListedKeyword = true;
                            break;
                        }
                    }

                    if (containsBlackListedKeyword)
                        break;

                    foreach (var keyword in keywords)
                    {
                        if (property.Contains(keyword.ToLower()))
                        {
                            Console.WriteLine($"{\t[?]} Interesting property: {property}, Value: {value}");
                            break;
                        }
                    }
                }
            }
        }
    }
}

```


CredFinder.cs

Since MSI format is close to SQL format, you can get all properties with one query. However, if you need a portable option or don't know how to compile CSharp projects yet, you can use a Powershell script with same logic:

```
$installerPath = "C:\Windows\Installer"
$package = New-Object -ComObject WindowsInstaller.Installer

{
param (
    [string]$msiPath
)
try {
$database = $package.GetType().InvokeMember("OpenDatabase", "InvokeMethod", $null,
$package, @($msiPath, 0))
$view = $database.GetType().InvokeMember("OpenView", "InvokeMethod", $null,
$database, @("SELECT * FROM Property"))

$view.Execute()
while ($record = $view.Fetch()) {
$property = $record.StringData(1)
$value = $record.StringData(2)
if ($property -match"USERNAME|PASSWORD|USER|PASS") {
    Write-Host "File: , Property: , Value: "
}
} catch {
    Write-Host "Error processing file: " -ForegroundColor Red
}
}

Get-ChildItem -Path -Filter *.msi -
Recurse | ForEach-Object { AnalyzeMsiFile .FullName}
```

Behavioral analysis

MSI Repair Mode

Of course, inside MSI credentials can be found quite rarely. Most often only when analyzing MSI files that were stolen from SCCM. So if we are looking for a privilege escalation vector, we need to analyze the behavior of the MSI file.

And here we need to familiarize ourselves with an unusual functionality: the MSI file repair mechanism.

MSI's repair mechanism allows a Windows system to reinstall either the entire product or individual components of the product. In effect, fix the program if something went wrong during use or installation.


This functionality is most conveniently utilized using the CLI tool [msiexec](#).

Repair options


You can use this command to repair an installed package.

Syntax

```
msiexec.exe [/f{p|o|e|d|c|a|u|m|s|v}] <product_code>
```

 Copy

Parameters


 Expand table

Parameter	Description
/fp	Repairs the package if a file is missing.
/fo	Repairs the package if a file is missing, or if an older version is installed.
/fe	Repairs the package if file is missing, or if an equal or older version is installed.
/fd	Repairs the package if file is missing, or if a different version is installed.
/fc	Repairs the package if file is missing, or if checksum does not match the calculated value.
/fa	Forces all files to be reinstalled.
/fu	Repairs all the required user-specific registry entries.
/fm	Repairs all the required computer-specific registry entries.
/fs	Repairs all existing shortcuts.
/fv	Runs from source and re-caches the local package.

Examples

To force all files to be reinstalled based on the MSI product code to be repaired, {AAD3D77A-7476-469F-ADF4-04424124E91D}, type:

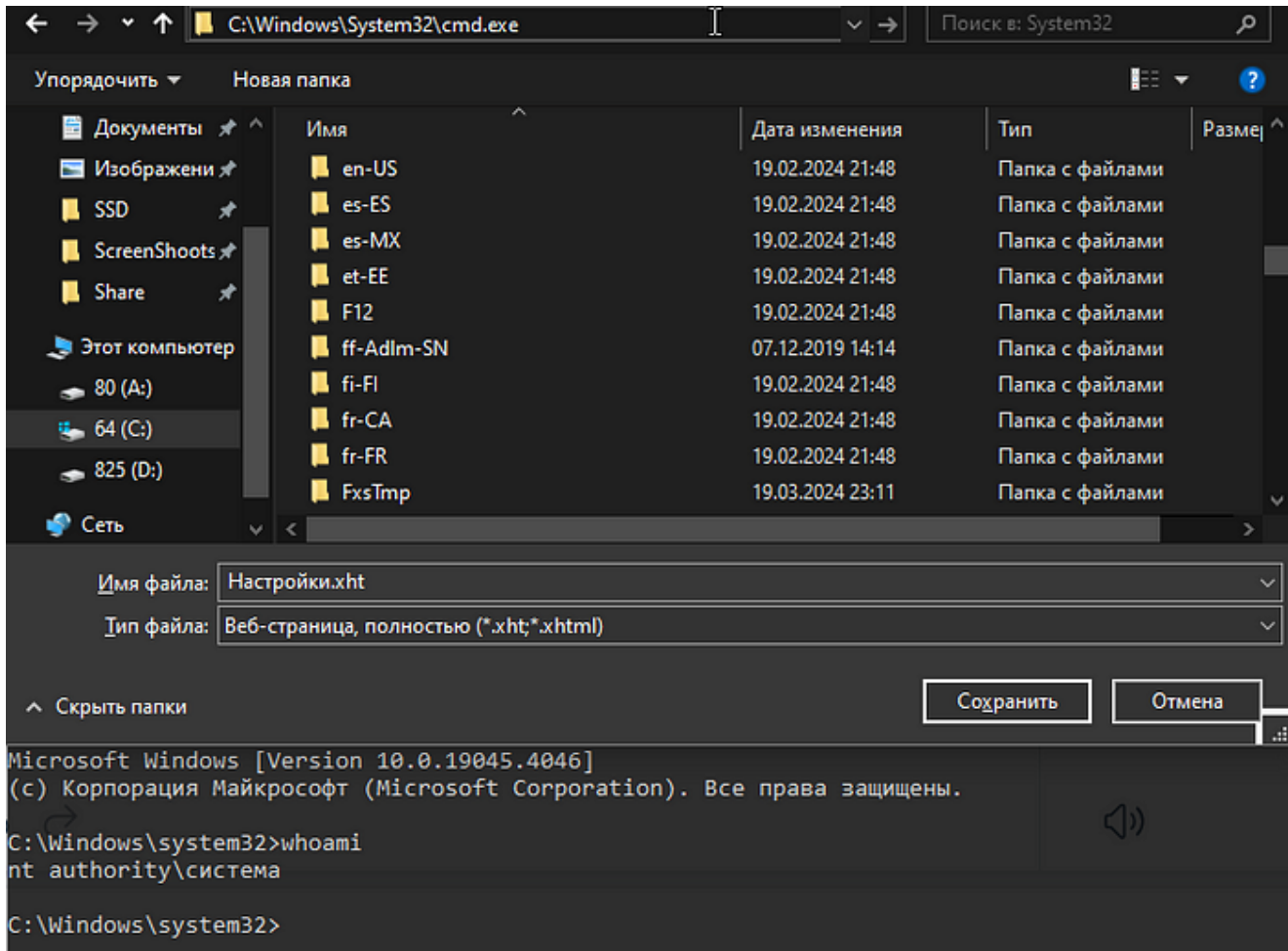
```
msiexec.exe /fa {AAD3D77A-7476-469F-ADF4-04424124E91D}
```

 Copy

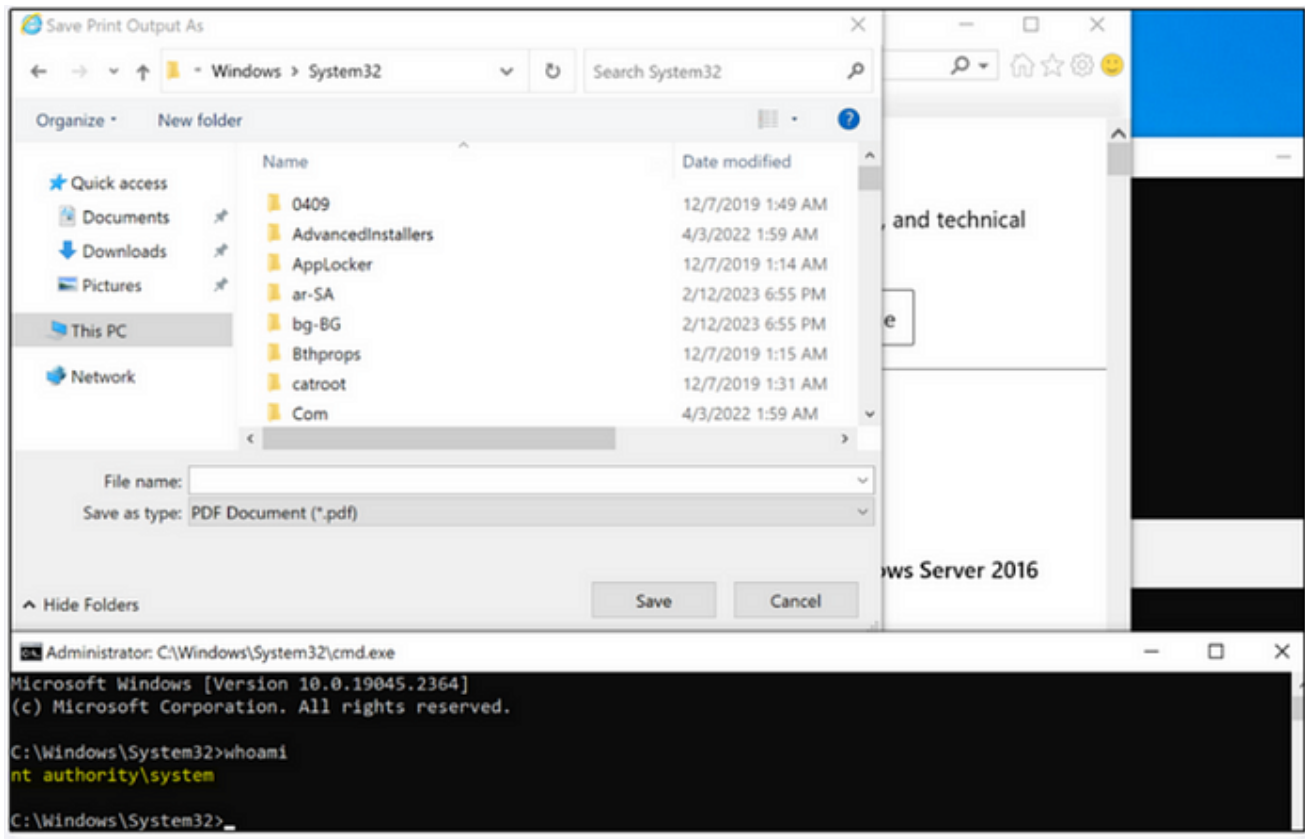
In addition, MSI Custom Actions, which were created by the developer of the MSI file, are invoked in recovery mode. Here too, there may be a vulnerability. If Custom Actions or the entire MSI file is misconfigured, the recovery process is performed on behalf of the NT AUTHORITY\SYSTEM user, which allows us to escalate privileges.

For example, if the developer has set Custom Actions to run cmd.exe, then during a normal installation cmd.exe will be run as the current user, but during recovery it will be run as the system user.

Also through Custom Actions can run some graphical applications on behalf of the system, from which you can make an analog of Kiosk Bypass, get out to explorer.exe and run cmd.exe from it. cmd.exe will be launched on behalf of the system.



GUI in custom actions abuse



Abuse example in Internet Explorer installer

How to detect it?

Let's start by checking the entire MSI file. There are only two things we need to monitor:

- Presence of a GUI interface, if we want to promote via explorer.exe escape
- The name of the user on whose behalf the MSI file is run in recovery mode, if we want to examine the file for other vulnerabilities, such as DLL Hijacking

The easiest way to detect such MSI files is to use the [GuiFinder](#) tool:

```
.\GuiFinder C:\Temp
```

```
PS A:\ssd\Projects\VS\CApp\Release> .\CApp.exe --folder C:\Temp
Recursive Scanning Directory: C:\Temp
File: C:\Temp\3yedfvqw.m3w\TestMsi\Microsoft.VisualStudio.Setup.TestMsi.msi [+] HAS GUI [?] Running from: WINPC\Michael
```

If you find yourself running from the NT AUTHORITY\System and the presence of a graphical interface, you can attempt to perform an escape from the environment as described above.

What about custom actions?

Custom Actions can also be executed on behalf of the NT AUTHORITY\SYSTEM. To do so, they must be configured with the *Impersonate="no"* option. For example, as here

```
<?xml version= encoding=?>
< =>
< = = = = = =>
< = = = = = = = = />
```

```
< = = = = />
< = = = = = = />
< = =>
< =>
< = = =>
< = = =>
< = />
</>
</>
</>
</>
< = = />
< = = =>
< = />
</>
< = = />
```

1 1

This will cause Custom Actions to be run on behalf of the NT AUTHORITY\SYSTEM. We created a ActionAnalyzer class to analyze Custom Actions.

First, we also highlighted keywords whose presence within Custom Actions will result in privilege escalation. After all, it will be easy to abuse this functionality.

```

static public string[] keywords = new string[]
{
    "AdminToolsFolder",
    "AppDataFolder",
    "DesktopFolder",
    "FavoritesFolder",
    "LocalAppDataFolder",
    "MyPicturesFolder",
    "NetHoodFolder",
    "PersonalFolder",
    "PrintHoodFolder",
    "ProgramMenuFolder",
    "RecentFolder",
    "SendToFolder",
    "StartMenuFolder",
    "StartupFolder",
    "TempFolder",
    "APPDATA",
    "HomePath",
    "LOCALAPPDATA",
    "TMP",
    "USERPROFILE",
    "conhost.exe",
    "cmd.exe",
    "powershell.exe",
    "-noprofile",
    "HKCU",
    "HKEY_CURRENT_USER",
    "C:\\Windows\\Tasks",
    "C:\\windows\\tracing",
    "C:\\Windows\\Temp",
    "C:\\Windows\\tracing",
    "C:\\Windows\\Registration\\CRMLog",
    "C:\\Windows\\System32\\FxsTmp",
    "C:\\Windows\\System32\\Tasks",
    "C:\\Windows\\System32\\AppLocker\\AppCache.dat",
    "C:\\Windows\\System32\\AppLocker\\AppCache.dat.LOG1",
    "C:\\Windows\\System32\\AppLocker\\AppCache.dat.LOG2",
    "C:\\Windows\\System32\\Com\\dmp",
    "C:\\Windows\\System32\\Microsoft\\Crypto\\RSA\\MachineKeys",
    "C:\\Windows\\System32\\spool\\PRINTERS",
    "C:\\Windows\\System32\\spool\\SERVERS",
    "C:\\Windows\\System32\\spool\\drivers\\color",
    "C:\\Windows\\System32\\Tasks\\OneDrive",
    ""
};

```

Interesting Keywords

The tool then checks that Custom Actions will in principle be called. To do this, they must be in the call sequence between InstallExecuteSequence and InstallFinalize.

```
var installExecuteSequenceOrder = installExecuteSequenceList.OrderBy(row => row.Sequence).ToList();
var installExecuteIndex = installExecuteSequenceOrder.FindIndex(row => row.Action == "InstallValidate");
if (installExecuteIndex != -1)
{
    installExecuteIndex = installExecuteSequenceOrder.FindIndex(row => row.Action == "InstallInitialize");
}
var installFinalizeIndex = installExecuteSequenceOrder.FindIndex(row => row.Action == "InstallFinalize");
```

Getting sequence indexes

The validation of these indices is done a little later. After getting the indexes, we extract all Custom Actions and check for the most important parameter Impersonate: NO


```

var customActionsList = new List<CustomAction_>();
try
{
    customActionsList = database.CustomActions.ToList();
} catch (Exception ex)
{
    return;
}

if (!customActionsList.Any())
{
    return;
}

foreach (var customAction in customActionsList)
{
    if ((customAction.Type & WixToolset.Dtf.WindowsInstaller.CustomActionTypes.NoImpersonate) != 0 &&
        ((customAction.Type & WixToolset.Dtf.WindowsInstaller.CustomActionTypes.Commit) != 0 ||
         (customAction.Type & WixToolset.Dtf.WindowsInstaller.CustomActionTypes.Rollback) != 0))
    {

        var actionInSequence = installExecuteSequenceList.FirstOrDefault(seq => seq.Action == customAction.Action);

        if (actionInSequence != null)
        {
            var actionIndex = installExecuteSequenceOrder.IndexOf(actionInSequence);
            if (actionIndex > installExecuteIndex && actionIndex < installFinalizeIndex)
            {
                Console.WriteLine($"{t[+]} Interesting Custom Action found");
                Console.WriteLine($"{tIndex: {actionIndex}");
                Console.WriteLine($"{tAction: {customAction.Action}");
                Console.WriteLine($"{tType: {customAction.Type}");
                Console.WriteLine($"{tSource: {customAction.Source}");

                var target = customAction.Target;
                Console.Write($"{tTarget: ");
            }
        }
    }
}

```

CustomAction flags analyzing

After making sure that the Custom Action is executed on behalf of the system and is within the correct action sequence, a keyword check is performed

```

if (actionInSequence != null)
{
    var actionIndex = installExecuteSequenceOrder.IndexOf(actionInSequence);
    if (actionIndex > installExecuteIndex && actionIndex < installFinalizeIndex)
    {
        Console.WriteLine($"\\t[+] Interesting Custom Action found");
        Console.WriteLine($"\\tIndex: {actionIndex}");
        Console.WriteLine($"\\tAction: {customAction.Action}");
        Console.WriteLine($"\\tType: {customAction.Type}");
        Console.WriteLine($"\\tSource: {customAction.Source}");

        var target = customAction.Target;
        Console.Write("\\tTarget: ");

        bool containsSensitiveKeywords = false;
        foreach (var keyword in keywords)
        {
            if (target.ToLower().Contains(keyword.ToLower()))
            {
                Console.ForegroundColor = ConsoleColor.Red;
                Console.Write(keyword + " ");
                Console.ResetColor();
                containsSensitiveKeywords = true;
            }
        }

        if (!containsSensitiveKeywords)
        {
            Console.Write(target);
        }

        Console.WriteLine();
        Console.WriteLine();
    }
}
}

```

Keyword checking

Lets run the tool.

```
Source: VBoxInstallHelper
Target: UninstallNetLwf

[?] Analyzing file: C:\Windows\Installer\510b3.msi
[+] Interesting Custom Action found
Index: 36
Action: ca_RollbackUninstallNetLwf
Type: Dll, Rollback, InScript, NoImpersonate
Source: VBoxInstallHelper
Target: InstallNetLwf

[?] Analyzing file: C:\Windows\Installer\524f4c4.msi
[+] Interesting Custom Action found
Index: 80
Action: PublishActiveSetup
Type: JScript, OncePerProcess, InScript, NoImpersonate, SixtyFourBitScript
Source: ActiveSetupCA.js
Target: PublishActiveSetup

[?] Analyzing file: C:\Windows\Installer\524f4c4.msi
[+] Interesting Custom Action found
Index: 56
Action: UninstallTUNTAPAdaptersCommit
Type: Dll, OncePerProcess, InScript, NoImpersonate
Source: libopenvpnmsica.dll
Target: ProcessDeferredAction

[?] Analyzing file: C:\Windows\Installer\524f4c4.msi
[+] Interesting Custom Action found
Index: 54
```

Example output

The tool will then bring up some interesting CustomActions. And you can start finding ways to abuse them. For example, perform DLL Sideload, launch a file from User Writable Paths, find another vulnerability. I encourage you to study [this article](#) and [this](#). It covers the most common ways to abuse CustomActions.

You can learn how to use our tool and hone your skills on the following vulnerabilities:

- CVE-2023-26077 (MSI Installer DLL Hijacking)
- CVE-2023-21800 (Symlink Abuse)
- CVE-2023-26078 (Escape to cmd.exe)

Custom Actions Overwriting

There is an even more interesting vector. We can find a CustomAction that runs on behalf of the system. However, we may not be able to abuse it. In that case, we can try to overwrite it! If the permissions on the MSI file allow, of course. In some cases, administrators override the default DACL, which will result in elevated privileges.

To find this vector, we created a [Writer](#) class.

```
l/ Can write custom actions: FALSE
[+] File C:\Windows\Installer\1d83d0e.msi
    [?] Signature: valid
    [?] Interesting property: registrykey, Value: Software\Python\PythonCore\3.11
    [?] Interesting property: rootregistrykey, Value: Software\Python\PythonCore
    [?] Can write custom actions: FALSE
[+] File C:\Windows\Installer\1d83d15.msi
    [?] Signature: valid
    [?] Can write custom actions: FALSE
[+] File C:\Windows\Installer\1d83d1c.msi
    [?] Signature: valid
    [?] Interesting property: registrykey, Value: Software\Python\PythonCore\3.11
    [?] Can write custom actions: FALSE
[+] File C:\Windows\Installer\1d83d23.msi
    [?] Signature: valid
    [?] Interesting property: registrykey, Value: Software\Python\PythonCore\3.11
    [?] Can write custom actions: FALSE
[+] File C:\Windows\Installer\1d83d2a.msi
    [?] Signature: valid
    [?] Interesting property: registrykey, Value: Software\Python\PythonCore\3.11
    [?] Can write custom actions: FALSE
[+] File C:\Windows\Installer\1d83d31.msi
    [?] Signature: valid
    [?] Can write custom actions: FALSE
[+] File C:\Windows\Installer\1d83d38.msi
    [?] Signature: valid
    [?] Interesting property: registrykey, Value: Software\Python\PythonCore\3.11
    [?] Can write custom actions: FALSE
[+] File C:\Windows\Installer\1d83d3f.msi
    [?] Signature: valid
    [?] Can write custom actions: TRUE
[+] File C:\Windows\Installer\1d83d46.msi
    [?] Signature: valid
    [?] Interesting property: registrykey, Value: Software\Python\PythonCore\3.11
    [?] Can write custom actions: TRUE
```

Output example

Diff

MSI files both contain vulnerabilities and fixes them! So we needed a convenient way to implement diff of two files to analyze patches.

The simplest way is to use msidiff, its syntax is self-explanatory

```

root@wifi:~# msidiff /mnt/hgfs/Share/1d83d23.msi /mnt/hgfs/Share/1d83d31.msi
diff -r -Nup old/Component.idt new/Component.idt
--- old/Component.idt 2024-08-30 09:47:19.812013927 -0400
+++ new/Component.idt 2024-08-30 09:47:20.568013966 -0400
@@ -1,1325 +1,98 @@
Component      ComponentId      Directory_      Attributes      Condition      KeyPath
s72      s38      s72      12      5255      572
Component      Component
-Lib_test_allsans.pem {DFE13381-AE7A-5286-8617-265C2ECE24EA} Lib_test 256 Lib_test_allsans.pem
-Lib_test_ann_module.py {0B606000-F0EF-5D89-BF85-EC60E0576FA3} Lib_test 256 Lib_test_ann_module.py
-Lib_test_ann_module2.py {5E3FE8AC-28EA-5695-8AFA-E0F92DD6685C} Lib_test 256 Lib_test_ann_module2.py
-Lib_test_ann_module3.py {ACAD80EA-AC09-540C-8498-91E1608F73D5} Lib_test 256 Lib_test_ann_module3.py
-Lib_test_ann_module4.py {A3683F45-86E8-5175-908A-445555FC7E8B} Lib_test 256 Lib_test_ann_module4.py
-Lib_test_ann_module5.py {B7AD0860-DA82-58FD-A420-2F42E9DF5E1E} Lib_test 256 Lib_test_ann_module5.py
-Lib_test_ann_module6.py {F2BC0E73-7CE8-5C5D-83D8-342C621773C0} Lib_test 256 Lib_test_ann_module6.py
-Lib_test_ann_module7.py {29944982-96E0-57D8-9597-A4FE93CD9F9F} Lib_test 256 Lib_test_ann_module7.py
-Lib_test_ann_module8.py {F17FBF5D-0843-51AD-AD16-313F64FFB8BA} Lib_test 256 Lib_test_ann_module8.py
-Lib_test_audiodata_pluck_alaw.aifc {F9565785-F360-56C3-B588-0003ED314991} Lib_test_audiodata 256 Lib_test_audiodata_pluck_alaw.aifc
-Lib_test_audiodata_pluck_pcm16.aiff {C2AD4606-7100-5A5F-8981-64D37C57F446} Lib_test_audiodata 256 Lib_test_audiodata_pluck_pcm16.aiff
-Lib_test_audiodata_pluck_pcm16.au {88D10CAC-2AF8-5FD8-9433-D5F401D1A0C3} Lib_test_audiodata 256 Lib_test_audiodata_pluck_pcm16.au
-Lib_test_audiodata_pluck_pcm16.wav {70D0AC05-3A19-5F07-8619-F614435A4EFD} Lib_test_audiodata 256 Lib_test_audiodata_pluck_pcm16.wav
-Lib_test_audiodata_pluck_pcm24.aiff {4DBF083C-8E3E-5795-B38F-17EC0B000803} Lib_test_audiodata 256 Lib_test_audiodata_pluck_pcm24.aiff
-Lib_test_audiodata_pluck_pcm24.au {D872CA0D-4932-5EF9-A68B-CDEB0926898D} Lib_test_audiodata 256 Lib_test_audiodata_pluck_pcm24.au
-Lib_test_audiodata_pluck_pcm24.wav {6C40B3CC-EE64-5859-AA5F-A45E61664EF5} Lib_test_audiodata 256 Lib_test_audiodata_pluck_pcm24.wav
-Lib_test_audiodata_pluck_pcm32.aiff {A21F8FDA-64E4-59AF-B5CF-87A41184F9AA} Lib_test_audiodata 256 Lib_test_audiodata_pluck_pcm32.aiff
-Lib_test_audiodata_pluck_pcm32.au {DEA8A176-28F2-582E-B893-9F9F4F4F0554} Lib_test_audiodata 256 Lib_test_audiodata_pluck_pcm32.au
-Lib_test_audiodata_pluck_pcm32.wav {D797ADAD-4DBF-573B-9E91-78ADE1882A1E} Lib_test_audiodata 256 Lib_test_audiodata_pluck_pcm32.wav
-Lib_test_audiodata_pluck_pcm8.aiff {989FED37-DF5E-56C5-AEF8-56240AF62034} Lib_test_audiodata 256 Lib_test_audiodata_pluck_pcm8.aiff
-Lib_test_audiodata_pluck_pcm8.au {82744983-24F7-58D6-8227-69A4AE4D45C6} Lib_test_audiodata 256 Lib_test_audiodata_pluck_pcm8.au
-Lib_test_audiodata_pluck_pcm8.wav {4CACE03F-8160-5F38-A1E2-3C17525FBFFC} Lib_test_audiodata 256 Lib_test_audiodata_pluck_pcm8.wav
-Lib_test_audiodata_pluck_ulaw.aifc {7FC71C86-5997-580B-9F39-81045CFDB687} Lib_test_audiodata 256 Lib_test_audiodata_pluck_ulaw.aifc
-Lib_test_audiodata_pluck_ulaw.au {3E889860-8E35-5CC1-9980-8EC3F486A367} Lib_test_audiodata 256 Lib_test_audiodata_pluck_ulaw.au
-Lib_test_audiotests.py {6E4523D0-8050-5252-AA65-5FF3E2E53A33} Lib_test 256 Lib_test_audiotests.py
-Lib_test_audiotests.py {50880828-97E7-548F-8798-A1810700814F} Lib_test 256 Lib_test_audiotests.py
-Lib_test_audit_tests.py {D4E96F55-4718-5E36-8841-3209A8A523D6} Lib_test 256 Lib_test_audit_tests.py
-Lib_test_autotest.py {189F7004-2E02-5651-A3E7-823F6A99CFD2} Lib_test 256 Lib_test_autotest.py

```

Diff example

Conclusion

MSI files are used quite often in the Windows infrastructure. Often an improper approach to developing or deploying such files will lead to the possibility of privilege escalation on the host.