Università di Genova
DIBRIS DIPARTIMENTO
DI INFORMATICA, BIOINGEGNERIA,
ROBOTICA E INGEGNERIA DEI SISTEMI

Security and Privacy in the Use of Copilot+: A Study of Digital Traces

Ignazio Liotta

Master's Thesis

Università di Genova, DIBRIS
Via Dodecaneso, 35 16146 Genova, Italy
https://www.dibris.unige.it/

Università
di Genova

Computer Science MSc
Computer Security and Engineering Curriculum

Security and Privacy in the Use of Copilot+: A Study of Digital Traces

Ignazio Liotta

Advisor: Giovanni Lagorio Examiner: Matteo dell'Amico

March, 2026

Abstract

In this thesis we explore Recall, an application introduced with Copilot+ PCs that captures snapshots (a combination of a screenshot and a metadata file) of the screen of the computer. Our goal is to find out how useful Recall can be in the field of digital forensics.

We start by presenting the history of Recall and explaining the differences between the older version of the application and the newest one. We then list all of the tests that we did on Recall and show the results paired with code snippets from the tools we wrote for this thesis.

Our results show that Recall captures potentially useful information in the snapshots, but its usefulness is limited. In fact, to be able to export the snapshots so that they can be analysed, the computer must be set to the European Economic Area. Applications can also easily filter themselves out of the snapshots without any input from the user.

In parallel to that, we demonstrate how Recall is lacking on the safety side of things, with snapshots containing sensitive information that should have been filtered out, and the general ease-of-access requiring only the PIN of the user after the initial setup.

Declaration on the use of AI

During the preparation of this thesis, the author used Copilot during the writing of the code presented here, in particular the Inline Suggestions in Visual Studio Code. The author has manually tested and adjusted the resulting code as needed and takes full responsibility for the content of this publication.

Table of Contents

Chapter 1

Introduction

Recall is a new tool developed by Microsoft, available on all Copilot+ computers. Copilot+ PCs are a new line of computers that are built with a Neural Processing Unit (NPU) specialized for AI operations, instead of the standard CPU (more in Section 2.1 about this topic). Recall periodically saves screen snapshots, enabling the users to search and retrieve prior on-screen content. As with every new tool that stores information about the activity of the users, we need to know what it saves and how secure is all that data against attackers. This concerns not only the adopters of this new technology, but possibly everyone that interacts with a computer with this feature enabled.

In Chapter 2 we provide a quick but detailed background on this new application. While a few years are long time, the development of Recall has been characterised by a few big iterations and long moments of silence from Microsoft: an initial release in 2024 that was heavily criticised, then a big update in early 2025 that focused on the security. Now the application is outside the Beta and no new big changes were made since the 2025 version.

In Chapter 3 we present two tools that were developed on the first version of Recall that we believe shaped the discourse on the application. We then discuss a great article on the 2025 security update that is still very much valid to this day.

In Chapter 4 we list how we approached this thesis and how we structured the tests. We provide a reference table between the scenarios that we built to examine Recall and their results in Chapter 5. We start with scenarios that are very detailed in their individual steps but are generic in their scope and as we move on the script flips, with steps that are more generic and the focus is narrower. This happens because as our mastery with the application increases, we understand which actions are going to have an impact on Recall and which can be omitted, and because we can narrow better what we are looking for.

In Chapter 5 we present the results of our tests. The results are not strictly in the chrono-

logical order, but are listed in a way to provide to the reader a gradual understanding on how Recall works, on the data we can retrieve from it and then the limits of this new application. Indeed, a big part of our tests was looking for ways that a program or the users might trick Recall, as that is a key concern on its security.

Finally, in Chapter 6 we look back at everything we have written in this thesis and draw our conclusions, not just on how well Recall does (or does not) work, but whether or not it might be useful to an investigator conducting a digital forensic analysis, which is the fundamental question that motivated us to research into this topic.

Chapter 2

Background

Recall is a tool developed by Microsoft that utilizes Windows Copilot Runtime to help you find anything you have seen on your PC [Lea24]. Users can search for keywords or scroll through the snapshots of their past activities. It also allows the users to jump back to the content seen in the snapshot if a *UserActivity* is provided by the application at the time of the snapshot.

A UserActivity refers to something specific the users were working within the application. For example, when the users are writing a document, a UserActivity could refer to the specific place in the document where the user left off writing. When listening to a music application, the UserActivity could be the playlist that the users last listened to. When drawing on a canvas, the UserActivity could be where the users last made a mark. In summary, a UserActivity represents a destination within a Windows application that users can return to so that they can resume what they were doing [Lea25b].

Recall takes advantage of the local Phi Silica model in Copilot+ PCs to connect actions to the content found by Recall using the AI-supported feature *Click to Do*, which runs on top of the snapshots made by Recall. Click to Do analyses what is in a snapshot and suggests actions to the users based on the content. For example, if the users select a piece of text, Click to Do might suggest summarising or rewriting it. If the users select an image, Click to Do might suggest blurring the background, erasing objects in the image, or searching the web for related [1] information [Lea25a].

[1] Figure 2.1 source is: https://support.microsoft.com/en-us/windows/click-to-do-in-recall-do-more-with-what-s-on-your-screen-967304a8-32d1-4812-a904-fad59b5e6abf

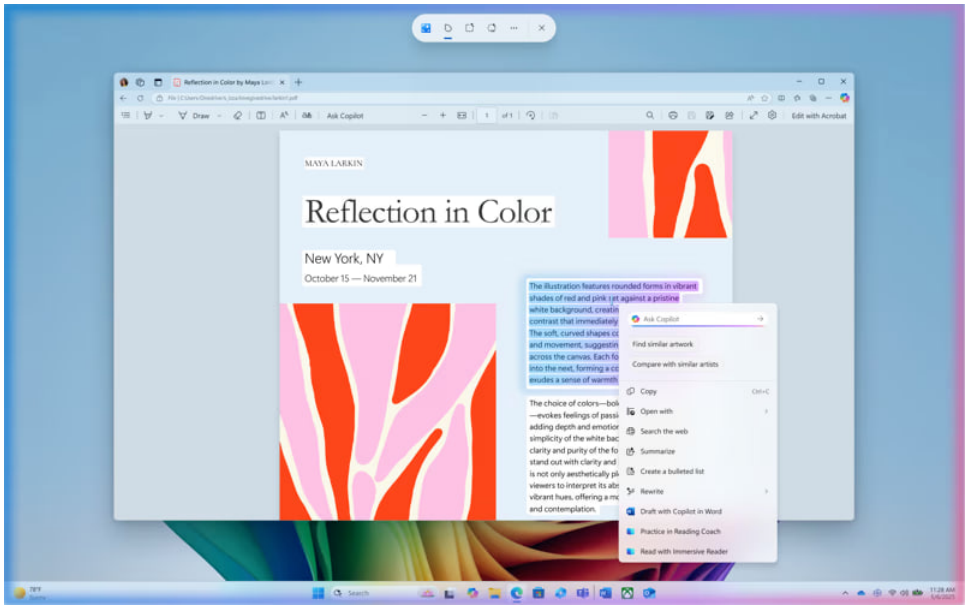

Figure 2.1: Example of Click to Do contextual menu

In this thesis we focus on the traces that Recall leaves on a computer and how their impact on the security and privacy of the users, not all the features related to Recall such as Click to Do or how to implement a UserActivity in an application.

2.1 Copilot+ PCs

Microsoft announced a new line of computers, called Copilot+ PCs, and Recall on May 20, 2024 [Kha24]. Copilot+ PCs initially were only ARM-based laptops, but now are now available on Qualcomm, Intel, and AMD platforms. These computers require a Neural Processing Unit (NPU), which offers 40 trillion of operations per second (TOPS). The NPU has also access to advanced AI models, which Recall requires to work [Sup24b].

2.2 Release and Reception

After the announcement of Recall, concerns about the security implications were raised. Less than a month later, Microsoft released three updates where they made Recall an optional feature that the user needed to opt-in, encrypted the data stored by Recall and made it necessary authenticating with Windows Hello the first time it is enabled. On April

10, 2025 Microsoft released a new Build to Insiders in the Preview Channel that would be gradually rolled out to the users. The new preview of Recall would be rolled out in the European Economic Area "later this year" [Tea25].

Since its first launch, many articles and blog posts have been written about Recall, many of them wary of the potential problems it could carry:

- May 20, 2024: Microsoft embraces the power of ChatGPT-4o and announces a new line of computers, called Copilot+ PCs. This computers will have a new feature called Recall that will allow the users to ask questions about what they a few days ago [Kha24].
- June 6, 2024: The snapshots taken by Recall can be easily grabbed with a new tool called *TotalRecall*. This is because the local database used by Recall is in plain text [Arn24].
- June 13, 2024: Microsoft delays the release of Recall after it is called a potential cybersecurity disaster by experts [War24a].
- October 31, 2024: Microsoft delays again Recall, Windows Insiders will not be able to try Recall until December [Ehr24].
- December 12, 2024: Recall is creepy, clever and compelling. It feels clunky and unfinished, but Recall does take snapshots of everything, from chat messages to drafts of unsent tweets. It can have its uses, but storing a digital trail on a computer can be concerning [War24b].
- April 10, 2025: Microsoft releases Windows 11 Build 26100.3902 to the Release Preview Channel. Recall is rolling out gradually, but countries inside the European Economic Area will receive it later [Tea25].
- April 21, 2025: Recall is rolling out to the end users devices. It is opt-in, the database is encrypted, but the biometric encryption is only needed only for the initial setup [Bea25].

After the April release, Recall has not changed much. Our tests began July 2025, when Recall was already available to countries inside the European Economic Area and the only noticeable update was when Recall left the Windows Insider Program and became available to all Copilot+ PC users.

Chapter 3

Related Work

In this chapter, we present the two important tools released on the early versions of Recall, and an article discussing vulnerabilities that were present on April 2025 and still are. These articles were instrumental in developing the tests presented in Chapter 5.

3.1 AmperageKit

AmperageKit was first uploaded to GitHub on May 5, 2024 [Alb24]. It allows users to enable Recall on `ARM64` machines that are not officially supported. However, it requires specific Windows builds to run that are now outdated.

This tool helped testing and reviewing Recall in its early stages, and we were thinking of using it before Microsoft patched many of the problems that were found. One such example of its use is in [Zac24], where the authors showed the wealth of information was readily available in the non-encrypted SQLite database. They provided some examples of the Snapshots and their metadata. They were doubtful that it could become widespread with the privacy problems of that version and in hindsight, they were not wrong as Recall faced quite a lot of backlash in the years to come.

3.2 TotalRecall

On June 4, 2024, *TotalRecall* was uploaded on GitHub [Hag24]. It is a tool that copies the database and snapshots taken by Recall, parsing them for potentially interesting artifacts. It allows the user to define dates to limit the extraction as well as searching for strings extracted via Recall OCR. The tool generates a summary of the extracted data, as well

as a text file listing all the captured data and search results. This was possible because Recall stored its data in a non-encrypted database, which now has been patched.

While we cannot say it with certainty, it is more than plausible that this tool and the articles it prompted pushed Microsoft in the right direction during the development of Recall.

3.3 Recent Tests on Security and Privacy

On April 21, 2025, Beaumont [Bea25] discussed the improvements of the latest Recall version, while highlighting the problems that are still present:

- Recall is now an opt-in feature, which is always a good thing with products that may have an impact on the privacy of the user.
- The database in which Recall stores its data is now encrypted.
- Sensitive data is now automatically filtered. However, the filters are not perfect and still require some tuning, as it sometimes fails to censor critical data such as a credit card's number or CVV (we demonstrate this in Section 5.3). This data is then easily retrievable with a quick search, as it is indexed with all the data from the snapshots. As the writer of the articles pointed out, it might be a problem of browser support, but it is still a problem nonetheless.
- In the initial setup Recall requires a biometric sign-in with Windows Hello. Unfortunately, it is required only for that step and later logins are done only with the user's PIN, which is not the strongest barrier against a potential attacker to say the least. Recall can also be disabled and reenabled without biometrics, meaning that an attacker could reenable Recall on a PC where the user had tried it and then disabled it. This, and with how easy is to search information with it, makes it undesirable by certain categories of vulnerable people.
- Recall is still unstable, sometimes stopping to work without a clear explanation.

While not being the "privacy nightmare" as it was described in the initial articles, Recall still has some problems that the end user should be aware of.

Chapter 4

Methodology

We started our analysis by testing the core functionalities of Recall. We began by writing down User Scenarios, with a list of all the actions that would be taken so that it could be compared to the data gathered by Recall. When writing down the scenarios, we aimed to recreate actions that a common user might perform, using tools already available on Windows 11 or easily accessible to anyone. All actions were done manually, as Recall takes periodical snapshots and the speed of an automated test would be counterproductive. From that we moved to more specific scenarios, to build tools to analyse the data we were able to gather and to find the limits of Recall. In Table 4.1 and Table 4.2 we summarise the tests we have done and refer to the sections in Chapter 5 where we discuss them.

Table 4.1: Scenarios

`Scenario`	`Default Applications`
Method	List of tasks using default Windows applications
Discussed In	Section 5.1
`Scenario`	`Browsers`
Method	Test supported and non-supported browsers
Discussed In	Section 5.2
`Scenario`	`Private Navigation Tabs`
Method	Repeat tests in Private mode
Discussed In	Section 5.2

Table 4.2: Scenarios (continue)

Scenario	Microsoft Store Applications
Method	Install and compare Discord from website and Microsoft Store
Discussed In	Section 5.13
Scenario	Data Permanence
Method	Reset Recall and then PC
Discussed In	Section 5.13
Scenario	Sensitive Data
Method	Simulate purchase on an online store
Discussed In	Section 5.3
Scenario	Chats
Method	Use different mainstream messaging applications
Discussed In	Section 5.3
Scenario	Registry Keys
Method	Capture changes when modifying Recall settings with Regshot
Discussed In	Section 5.8
Scenario	Regional Settings
Method	Set the PC location to the USA
Discussed In	Section 5.12
Scenario	Bypass Export Snapshots
Method	Retrieve files from *AsymStore* directory
Discussed In	Section 5.1
Scenario	Filter Applications and Executables
Method	Add Notepad++ to the filter list and write a small document
Discussed In	Section 5.10
Scenario	Hiding from Recall
Method	Create simple applications with *WDA_EXCLUDEFROMCAPTURE* on
Discussed In	Section 5.10

4.1 Default Application

We designed this scenario to simulate the activities an average user could do with just the default applications available on Windows 11. We were just starting and did not know

what to expect from the snapshots and what data we could gather from them, so we tried a bit of everything and so it is the most detailed scenario. We followed this list of actions:

1. Open *Notepad* and write the following text:

```
MEMO for the week:
Monday 11:30 - Doctor appointment for back pain
Wednesday 17:00 - Volleyball match
Thursday 20:00 - Pizza with friends at Koppa
Friday 09:30 - Grocery shopping
Sunday 09:00 - Grocery shopping for parents
```

2. Save the text file on the *Desktop*.
3. Close *Notepad*.
4. Open *Calculator*
5. Type $3 + 15 =$.
6. Click the *CE* button.
7. Type $4 + 12 =$.
8. Close *Calculator*.
9. Open *Microsoft Edge*.
10. Type "agv volley" in the search bar and press *Enter*.
11. Click the first result.
12. Click the *Facebook* redirect button.
13. Fill the login form with correct user data.
14. Click on the *Info* button.
15. Open a new tab and navigate to `https://www.giallozafferano.it`.
16. Click on *Ultime Ricette*.
17. Open *Snipping Tool*.

18. Capture a screenshot of the entire screen.
19. Save the screenshot in `Documents`.
20. Close *Edge*.
21. Open an *InPrivate* window in *Microsoft Edge*.
22. Type "cool cats" in the search bar and press *Enter*.
23. Capture a screenshot of the resulting web page with *Snipping Tools*.
24. Save the screenshot in `Documents`.
25. Close *Edge*.
26. Browse the `Documents` folder and delete the second screenshot.
27. Open and close *Media Player*.
28. Open *News* and click on the first article from the top left, then close it.
29. Open *Microsoft Store*.
30. Type "sea of thieves" in the search bar and press *Enter*.
31. Click on the first result.
32. Close *Microsoft Store*.
33. Open *Paint*.
34. Draw $4 + 2 = 6$.
35. Save the drawing in `Documents`.
36. Open *Clock*.
37. Click *Timer*.
38. Start a 1-minute timer and wait for its completion.
39. Close *Clock*.
40. Open *Command Prompt*
41. Change directory to Documents
42. Run `ipconfig`.

43. Run `ping 8.8.8.8`.
44. Close *Command Prompt*.
45. Open *Windows PowerShell*.
46. Create a new directory.
47. Write the erroneous command `notepad.txt`.
48. Create a text file inside the new directory.
49. Write "V stands for Viennetta" and save the document.
50. Close *Notepad*.
51. Delete the new directory and its file with *PowerShell* commands.
52. Close PowerShell.
53. Close Paint.

4.2 Browsers

We tested *Microsoft Edge*, *Firefox*[1], *Google Chrome*[2], *DuckDuckGo*[3], *Tor*[4] and *Min*[5] to see how Recall would behave. Later we tested also *Brave* when we read about its ability to hide from Recall. We believe an easy way to test each browser is:

1. Open a *Wikipedia* article and read it.
2. Open a *Youtube* video that is not a static picture and let it play.
3. Open *Facebook* and fill the login form.
4. Open an online shop, add a couple of items to the cart and then type the relevant payment information in the form.

With these three steps you can test pages with little activity going on, pages with high activity on screen and the ability of Recall to filter out sensitive data.

[1] https://www.firefox.com
[2] https://www.google.com/chrome/
[3] https://duckduckgo.com
[4] https://www.torproject.org
[5] https://minbrowser.org

4.3 Private Navigation Tabs

A quick way to test this functionality is to:

1. *Private* tab on a browser like *Edge* or *Chrome*.
2. Navigate to the *Wikipedia* homepage.
3. Minimize the browser.
4. Open *Notepad* and write a small document of at least a couple of lines.
5. Close the browser.

In this way we can see if Recall captures the activity of the browser or any activity at all while said browser is open in Private mode.

4.4 Microsoft Store Application

To test if there were any difference in the metadata if an application had been downloaded from the *Microsoft Store* or from somewhere else, we downloaded and installed *Discord* from their website and then from the Microsoft Store. In both instances we wrote a couple of messages in a chat and then uninstalled the application.

4.5 Data Persistence

To test if some of the data generated by Recall would remain after resetting the computer, we first did a reset with the option to keep all the files of the user, and then we did a clean installation from scratch.

4.6 Chats

After the test with Discord, we decided to see if any common messaging application would be filtered out by default by Recall. We tested *Telegram*, *WhatsApp*, *Teams* and *Outlook*.

4.7 Registry Keys

The Windows Registry is a central hierarchical databased used in Windows to store information that is necessary to configure the system for one or more users, applications and hardware devices. The Registry contains information that Windows continually references during operation, such as profiles for each user, the applications installed on the computer and the types of documents that each can create, property sheet settings for folders and application icons, what hardware exists on the system, and the ports that are being used. A Registry Hive is a group of keys, subkeys and values in the registry that has a set of supporting files that contain backups of its data [Lea26].

We wanted to see which Registry Hives and keys Recall might be associated to. For this experiment we used *Regshot*, an open-source registry compare utility [Sea19]. In simple terms, it allows the user to capture two "shots" of the Registry and then compare them, producing a text file with the changes in the registry that happened between the two. For our purposes we ran the following test six times:

1. Capture the 1st shot of the system with Regshot.
2. Open the Recall settings.
3. Change the maximum storage for snapshots from 75 GB to 25 GB.
4. Change the maximum storage duration for snapshots from 90 to 30 days.
5. Disable the "Filter sensitive information" option.
6. Capture the 2nd shot of the system with Regshot.

We then used the tool described in Section 5.8 to compare the results between all tests.

4.8 Regional Settings

To test if the block of the Export feature was avoidable in some way, we reset the computer and set the region to the United States during the initial setup, then we set its location back to Italy in the Settings menu.

4.9 Bypass Export Snapshots

We tested if the non-exported snapshots stored inside Recall could be decrypted in the same way as the exported ones. To do that, we copied the encrypted files stored inside

the *AsymStore* directory (see Section 5.1) and ran the decryption tool on them with the decryption key given by Recall during that setup. To track the resulting failure inside the decryption tool we added a *catch* to the an existing *try* and then made it print the error.

4.10 Filter Applications and Executables

We tested if the Filtered applications list could be tricked by adding *Notepad++* to the said list and changing the name of the executable to a different one. Then we:

1. Wrote a small text file with Notepad++ with a *Microsoft Edge* navigation tab visible under it.
2. Wrote a small document with Notepad while the Notepad++ was open on the side.
3. Browsed Wikipedia with the Notepad++ document in the background.

4.11 Hiding from Recall

To test if and how adding the *WDA_EXCLUDEFROMCAPTURE* flag worked we created an Empty C++ Project in Visual Studio and added the flag. Seeing how it would not work, we created a Windows Desktop Application and then added the flag. We then tried to have it captured by Recall and the *Snipping Tool*.

Chapter 5

Results

In this chapter we discuss the results of the tests that we have discussed in the previous chapter. We tried to get a better understanding on the functioning of Recall, the traces it leaves on a computer and what useful information an investigator might gather from it. The first tests were made several months ago, but no major update to the core functions of Recall was rolled out during this timeframe and so we believe that all the results that we present are still relevant at the moment of writing. The only real change was that Recall now can be enabled on an eligible computer without joining the Windows Insider Program.

We begin with an overall analysis on the data gathered by Recall, namely the screenshots (`.jpg` pictures) and their metadata (`.json` files), then we take a quick look at how Recall behaves with browsers and possibly sensible data and then we move to a deeper investigation on the metadata. After that we present some scripts that we believe can help in the investigation process. We then try, and fail, to track Recall on the Registry. Finally we discuss how a program can hide from Recall and finish with a detailed analysis on the decryption tool provided by Microsoft and how it can sometimes fail.

5.1 The Snapshots

The snapshots are stored inside `C:\Users\[user]\AppData\Local\CoreAIPlatform.00\UKP\{...}\AsymStore`[1], where `{...}` is short for a directory with an alphanumerical name that changes with every installation of Recall. To view the snapshots and their metadata outside of Recall the user needs to export them, an option available only if the computer is set to be inside the *European Economic Area*. To decrypt them there is a tool provided by Microsoft [Gud25], using the key provided by Recall during its setup. This key cannot be

[1] Administrative privileges are required to access UKP.

recovered and so the user must take care to store it safely. Inside `AsymStore` the snapshots and the metadata files are split and encrypted individually:

1. Before the export, inside `AsymStore` there are `x` files
2. After the export, inside the chosen output directory there are `x/2` files
3. After the decryption, inside the chosen output directory there are `x` files

We can safely assume that during the export, Recall merges each snapshot with their relative metadata file. We tried to decrypt files directly taken from the `AsymStore` directory without exporting them, but the process failed. Considering that the decryption fails during the stage of verification of the key, we believed that the files inside `AsymStore` are encrypted with a different key. This key is also different in each instance of Recall: if we reset Recall and put inside `AsymStore` the files that were previously stored there, Recall will not show them inside the application.

Inside `UKP\{...}` there is also the database `ukg.db`, which is encrypted via the Trusted Platform Module [Wes24]. The blog post confirmed our theory about `AsymStore` files, as they claim that those are also encrypted in the same fashion as the database.

The Trusted Platform Module technology is designed to provide hardware-based, security-related functions. A TPM chip is a secure crypto-processor that is designed to carry out cryptographic operations. The chip includes multiple physical security mechanisms to make it tamper-resistant, and malicious software is unable to tamper with the security functions of the TPM. The most common TPM functions are used for system integrity measurements and for key creation and use. During the boot process of a system, the boot code that is loaded (including firmware and the operating system components) can be measured and recorded in the TPM. The integrity measurements can be used as evidence for how a system started and to make sure that a TPM-based key was used only when the correct software was used to boot the system [Lea25d].

Recall also provides a search function to skim through the snapshots, but its utility is limited by the the fact it operates like a dictionary search and so the query has to match one of the keywords extracted from a snapshot to have a match. In our experience, those keywords are not many and most likely part of the *Title* field (see Section 5.4 and Section 5.5 for more information about it).

5.2 Recall and Browsers

Microsoft lists Edge, Firefox, Opera, Google Chrome and Chromium based browsers as supported browsers [Sup24a]: all of them filter private browsing activity, while all but the

latter support filters to specified websites. We tested it ourselves:

- Edge, Firefox and Chrome do not appear in snapshots if a tab with private navigation is open.
- Tor appears in the snapshots as a regular browser.
- Min, a less known browser, also appears in the snapshots as normal.
- Brave and DuckDuckGo, both Chromium based, stop Recall from taking any snapshot.

In general, when a navigation tab is open Recall stops taking snapshots entirely. In Section 5.10 we will delve deeper into this topic.

5.3 Filtering Sensitive Data

Inside the settings, the user can set Recall to filter sensitive data like passwords or credit card data. Sometimes that does not work and snapshots of those are taken, such as in Figure 5.1. Recall does not consider chats such as *Telegram* and *WhatsApp* to contain sensitive information and will regularly take snapshots such as in Figure 5.2.

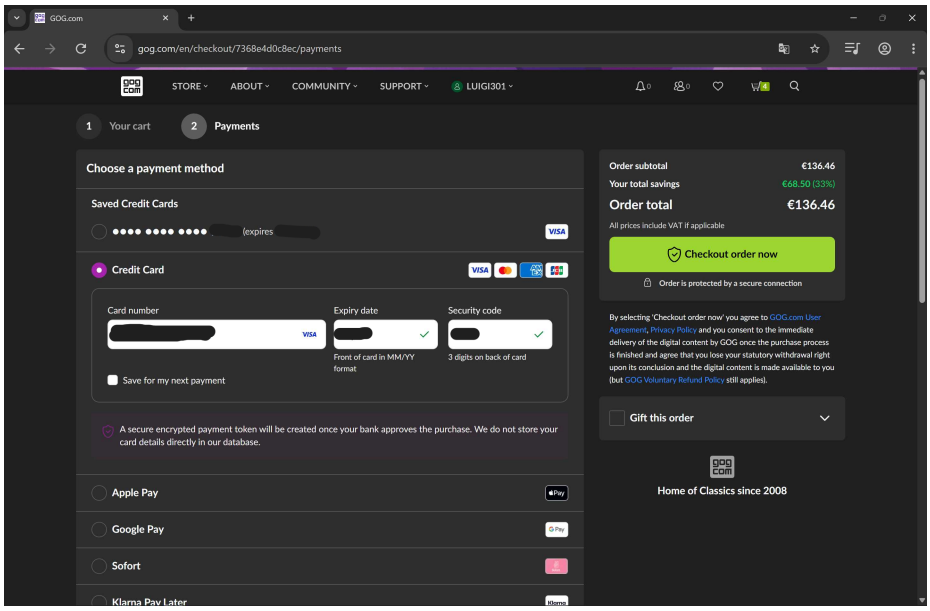

Figure 5.1: Manually censored snapshot of a Payment form

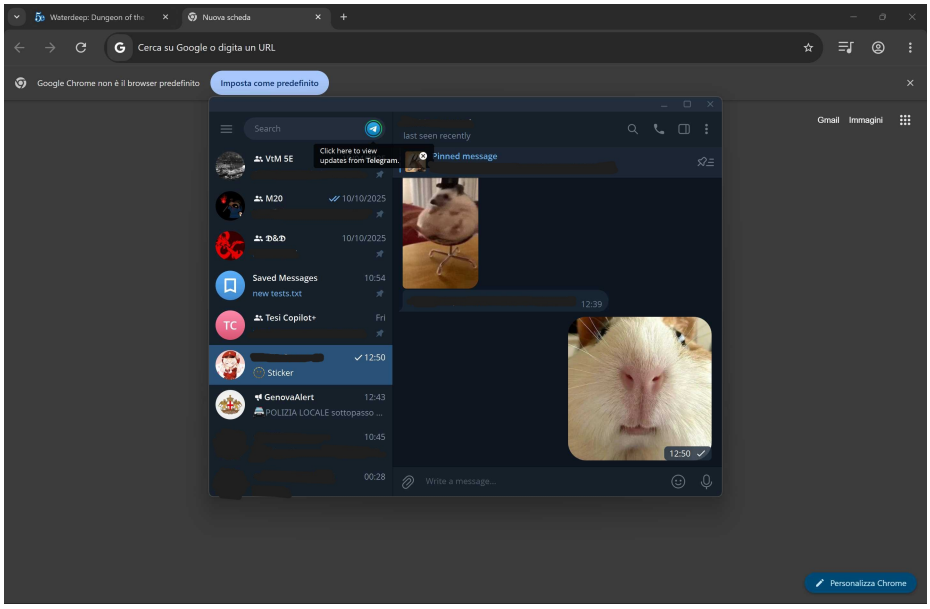

Figure 5.2: Manually censored snapshot of a Telegram chat

Chatting with someone that has Recall on their computer means that your messages will probably be stored in the database of the application. Even though Recall is strictly offline, in Chapter 3 we pointed out the safety problems it still has at the moment of writing. That could be less than desirable for some users, and we will see in Section 5.10 how Signal managed that.

5.4 The Metadata

Each metadata file contains one or more `DataItem`, structured like a `JSON` object. The key-value pairs inside the `DataItems` match roughly with what was visible in the database before it was encrypted. In Figure 5.3 we can see the tables that were present in the database when it was still not encrypted [Zac24]. Here we present a comparison between the old tables and the data found in the exported `JSON` files.

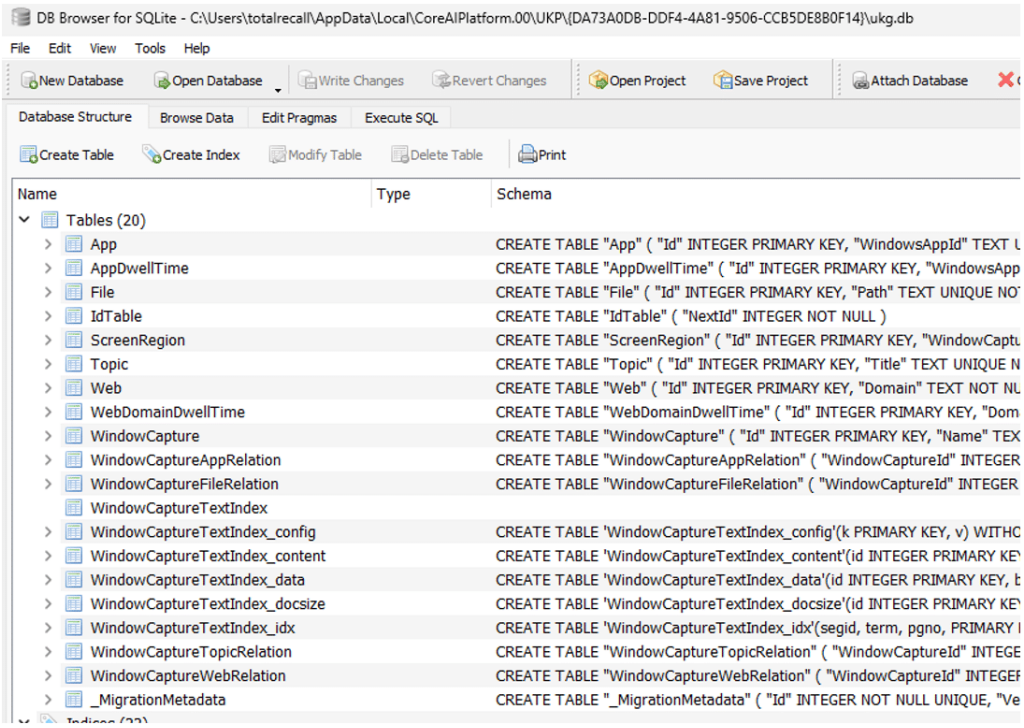

Figure 5.3: Database screenshot from CyberCX Blog

- App: WindowsAppId, Id used by Windows to identify the program, examples are
  - MSEdge → Microsoft Edge
  - Microsoft.WindowsTerminal_8wekyb3d8bbwe!App → Windows PowerShell[2]
  - Chrome → Google Chrome
  - Microsoft.Windows.Explorer → File Explorer
  - The full path of the executable in some cases, like with Registry Explorer
    - Registry Explorer → `C:\\Users\\luigi\\OneDrive\\Documents\\RegistryExplorer\\RegistryExplorer\\RegistryExplorer.exe`
    - Services → `C:\\Windows\\System32\\mmc.exe` [3]
- AppDwellTime: DwellTimeOfCapture

[2]Yes, weirdly enough it is called WindowsTerminal. Check Section 5.6 for an example of the ambiguity that it can cause.

[3]Path to the *Microsoft Management Console*, which hosts administration tools such as Services, Event Viewer and Device Manager.

  - In cyber security Dwell Time is a critical risk indicator
  - It refers to the time between an attacker gaining unauthorized access to a system and the moment their presence is detected and removed
  - In this context, we are not sure what it might be referring to, as it is not present in all metadata files

- File: Path, the full path to the executable of the program
- ScreenRegion: ForegroundWindowBounds, the position on screen of the foreground application
  - Not present in all metadata files, we are not sure why
  - It does not seem to be related to the type of activity on screen, as in some browser snapshots it is present and in others it is not
- IdTable: there are multiple types of ids inside the metadata, hard to pinpoint which ones were stored there
- Topic: Title
  - Inside the metadata, Name is just the name of the program, like "Visual Studio" or "Microsoft Edge"
  - Title instead is more of a descriptor of the content, like "Eric Zimmerman's tools - Google Chrome" or "*I am making a test writing a test. - Notepad"
  - Inside the schema of the Topic table in Figure 5.3, it is even visible the attribute "Title"
- Web: Domain and possibly other web-related fields
  - Like with Title, the attribute "Domain" is visible inside the schema of the table Web

We cannot do a one-to-one comparison without access to the database. To do so we would need access to an older version of Recall, and so we are limited to guesswork and intuition. Still, we can see that in the years of releases things have not changed too much and the `JSON` files provide a wide breath of information. Of course, only if the computer is set to the European Economic Area. Outside of those countries, Recall provides only a timeline of snapshots to the investigator.

5.5 Extrapolating Information from the Metadata

The `JSON` files obtained by Recall are not formatted in a human-friendly way, and parsing each file one by one is suboptimal. To help us in this matter we wrote a script to extract information that we deemed important and save it in a summary report.

In Listing 1 we provide the directory with the exported files as input, a target output directory and the key to decrypt the files and then run the executable compiled from the code provided by Microsoft in [Gud25].

```
parser = argparse.ArgumentParser(description="Decrypt. Extract. Analyse.")
parser.add_argument('-input', type=str, required=True)
parser.add_argument("-decrypted", type=str, required=True)
parser.add_argument("-output", type=str, required=True)
parser.add_argument('-key', type=str, required=True)

args = parser.parse_args()
decrypt_args = ['RSE.exe', args.input, args.decrypted, args.key]
subprocess.run(decrypt_args)
```

Code Snippet 1: Decrypt exported files

The *Name* of the programs that were running, their *Path* and the *Timestamp* of the snapshot are all fundamental to reconstruct an overview of the timeline of the user. *Domain* and *Uri* appear only in snapshots that feature browsers, and it is obvious why they are important. The snippet in Listing 2 checks for nested objects, as each metadata file is made up of several `DataItems`, each one with their fields. It is inside these `DataItems` that the *Titles* are found while the *Timestamp* is outside, hence the difference in the loop.

```
for v in data.values():
  if isinstance(v, dict):
    update_info("name", v.get("Name"))
    update_info("path", v.get("Path"))
    update_info("domain", v.get("Domain"))
    update_info("uri", v.get("Uri"))
    update_info("timestamp", v.get("Timestamp"))
    for x in v.values():
      if isinstance(x, dict):
        update_info("name", x.get("Name"))
        update_info("path", x.get("Path"))
        update_info("domain", x.get("Domain"))
        update_info("uri", x.get("Uri"))
        update_info("title", x.get("Title"))

return info
```

Code Snippet 2: Iterate through all dict values (DataItems) and nested dicts

In Listing 3 we filter the *null* character that sometimes ends up in the metadata, ignore values in the *Name* field that are useless to the analysis and also create a human-readable Timestamp to go along with the one in milliseconds. With these values we create an easy to read summary that can then be used to get a quick understanding of what Recall captured in the snapshots.

```
def update_info(key, value):
  #filter \u0000 values (sometimes is present in the metadata, rare but possible)
  if value and "\u0000" in value:
    value = value.replace("\u0000", "")
  #ignore WindowCaptureEvent values (there are multiple name fields, but
  #besides the one we want, others are only WindowCaptureEvent)
  if value == "WindowCaptureEvent":
    return
  #if the current info value is N/A and the new value is not empty, update it
  if info[key] == "N/A" and value:
    info[key] = value
  elif info[key] != "N/A" and value:
    #append additional names found
    info[key] += f"; {value}"
  #convert timestamp to human-readable format
  if key == "timestamp" and value:
    try:
      #need to divide by 1000 to convert from milliseconds to seconds
      info["timestampMilliseconds"] = value
      info[key] = (datetime.fromtimestamp(float(value)/1000)
                   .strftime('%Y-%m-%d %H:%M:%S'))
    except ValueError:
      pass
```

Code Snippet 3: Helper function to correctly set the value

Finally, in Listing 4 we used the library `pandas` to write the summary report in a `.csv` file.

```
df = pd.DataFrame(extracted_data)
df = df.sort_values(by='timestamp')
df.to_csv(output_csv, index=False)
print(f"Extracted information saved to {output_csv}")
```

Code Snippet 4: Write csv summary

In Table 5.1 and Table 5.2 we have respectively the metadata of a snapshot of a File Explorer window and the metadata of a snapshot of a Google Chrome navigation tab.

Table 5.1: Desktop Application Snapshot

Field	Value
Filename	file_explorer.json
Name	File Explorer
Path	`C:\Windows\explorer.exe`
Domain	N/A
Uri	N/A
Title	Downloads - File Explorer
Timestamp	2026-01-21 01:54:22
Timestamp (Milliseconds)	1768956862288

Table 5.2: Browser Snapshot

Field	Value
Filename	eric_zimmerman.json
Name	Google Chrome
Path	`C:\ProgramFiles\Google\Chrome\Application\chrome.exe`
Domain	ericzimmerman.github.io
Uri	https://ericzimmerman.github.io/
Title	Eric Zimmerman's tools - Google Chrome
Timestamp	2025-10-24 10:49:51
Timestamp (Milliseconds)	1761295791263

5.6 Reconstructing a Timeline

Using the script we just described, we analysed the Scenario presented in Section 4.1 to see how much information Recall captured. With the tool described in Section 5.5 we can extract and summarise the results of that Scenario and examine what we can learn from them.

Table 5.3: Default Applications Timeline 1-9

`Timestamp`	`Description`
11:00:04	User creates empty *Notepad* text file
11:00:29	User is writing the Monday memo
11:00:59	User is writing about the Monday appointment
11:01:04	User is still writing about the Monday appointment
11:03:35	User finishes to write the full memo
11:03:40	User is still on the memo text file
11:04:00	User is saving the text file inside `Documents`
11:05:23	User has typed "4+" in *Calculator*
11:06:40	User is on the website of *AGV Campomorone*

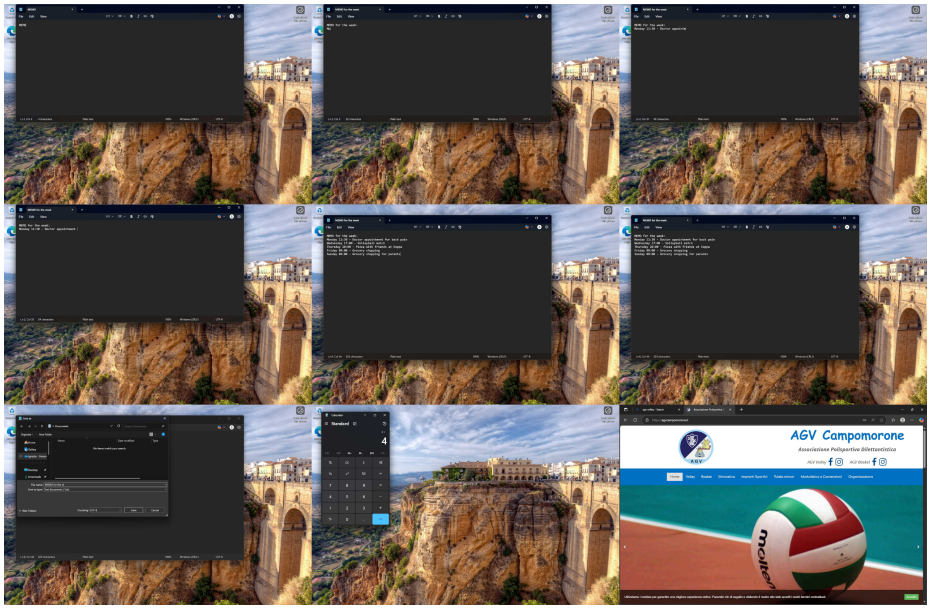

Figure 5.4: Notepad, Calculator and Edge

Table 5.4: Default Applications Timeline 10-18

Timestamp	Description
11:06:50	User is reading the cookie options for *Facebook*
11:09:38	User is looking at the latest recipes on *Giallo Zafferano*
11:09:59	User is taking a screenshot of the latest recipes
11:10:15	*Snipping Tool* has automatically saved the screenshot
11:10:45	User is manually saving the screenshot inside `Documents`
11:13:37	User took a screenshot of *Cool Cats NFTs*
11:14:39	User is looking at the memo and screenshots inside `Documents`
11:15:20	User deletes the *Cool Cats NFTs* screenshot from `Documents`
11:16:36	User is on the *News* application Home tab

[H]

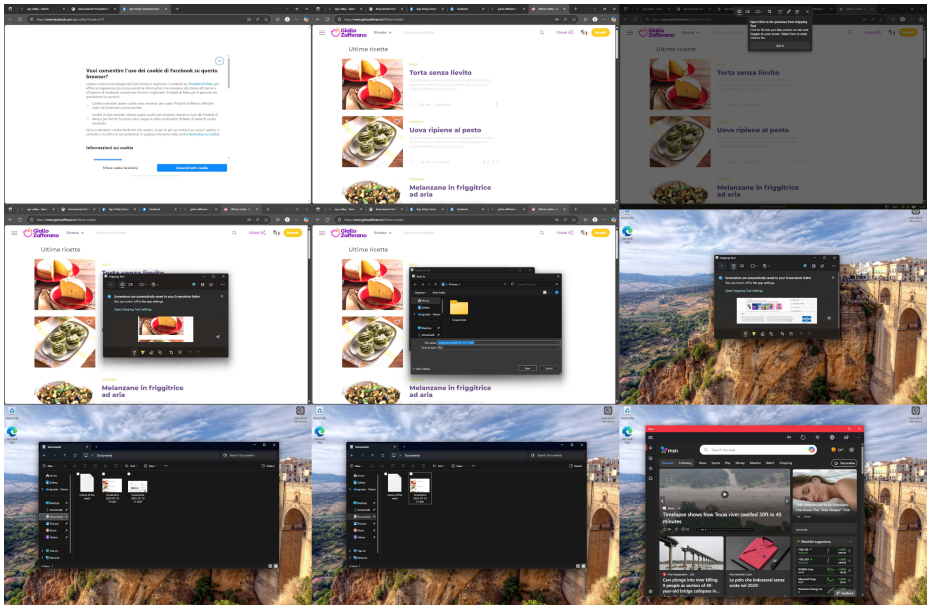

Figure 5.5: Edge, Snipping Tools, File Explorer and News

Table 5.5: Default Applications Timeline 19-27

Timestamp	Description
11:17:06	The *News* page is loading
11:17:36	User is reading an article about the Djokovic-Cobolli match
11:18:17	User is on the *Microsoft Store* Home tab
11:18:57	The *Thieves Helper* page on the Store is loading
11:19:12	The *Thieves Helper* page on the Store has loaded
11:19:42	Empty *Paint* canvas
11:20:13	User is saving a *Paint* drawing
11:20:18	User is still saving a *Paint* drawing
11:21:09	Four *Clock* timers, *Paint* drawing in the background

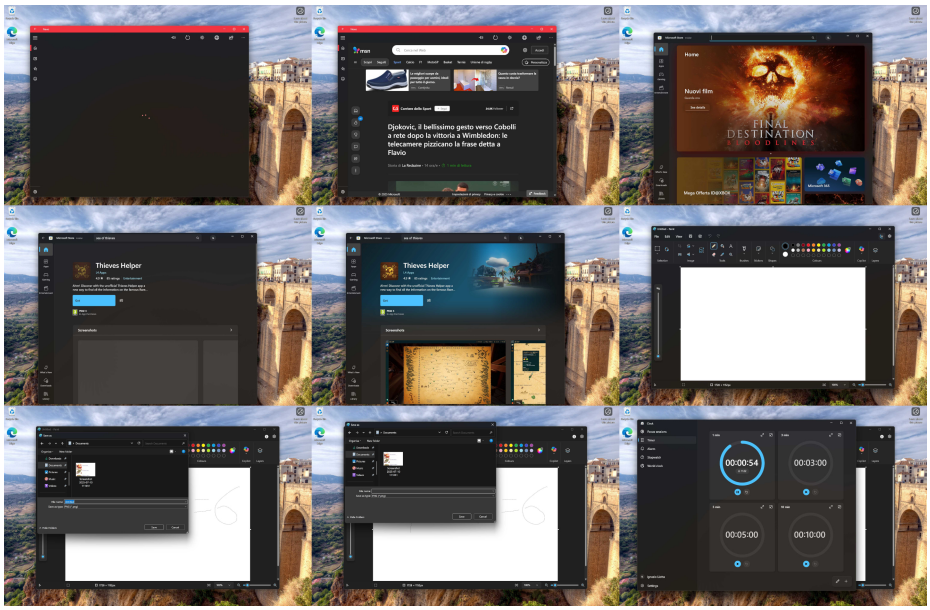

Figure 5.6: News, Microsoft Store, Paint and Clock

Table 5.6: Default Applications Timeline 28-36

Timestamp	Description
11:21:54	Four *Clock* timers*
11:21:59	Four *Clock* timers*
11:22:57	User is typing *cd Document* on a *Command Prompt* terminal*
11:23:11	User has run *ipconfig* on the terminal*
11:23:57	User opens a *Windows PowerShell* terminal*
11:24:22	User has run the *mkdir test* and *cd test* commands*
11:24:39	User opens the memo text file**
11:24:49	User closes the text file, revealing that it had opened it via the terminal*
11:25:09	User types *notepad.txt* in the terminal and gets an error*

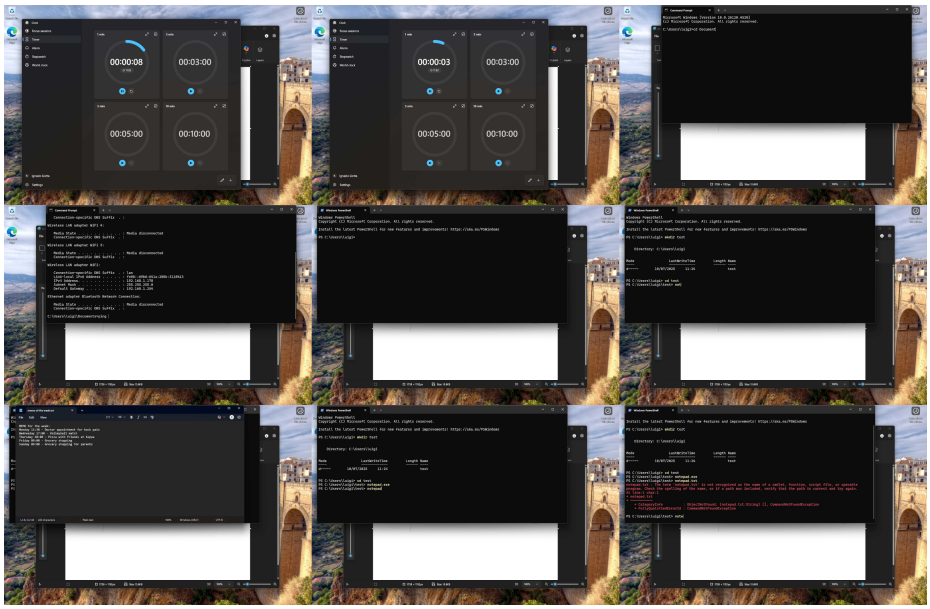

Figure 5.7: Clock, Terminal and Paint

Table 5.7: Default Applications Timeline 37-39

Timestamp	Description
11:26:02	User creates an empty text file*
11:26:12	User is writing inside the document*
11:27:02	User is looking at the *Paint* drawing

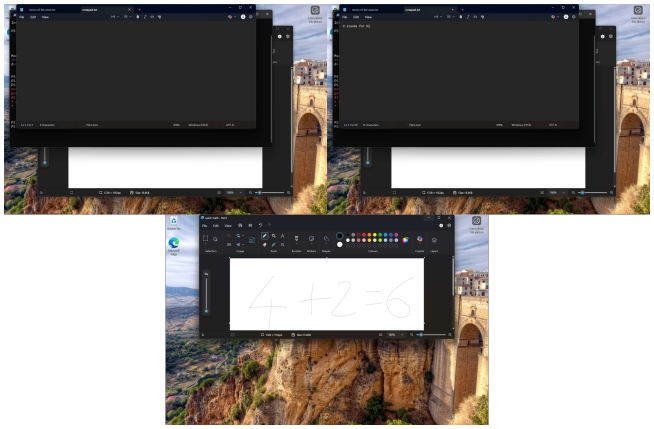

Figure 5.8: Notepad and Paint

**Paint* drawing in the background
***Paint* drawing and *Windows PowerShell* terminal in the background

Table 5.8: Websites captured by Recall

https://agvcampomorone.it/
https://www.facebook.com/agv.volley/?locale=it_IT
https://www.giallozafferano.it/Ultime-ricette/

Table 5.9: Programs captured by Recall

Program Name	Path
Notepad	C:\ProgramFiles\WindowsApps\[...]\Notepad\Notepad.exe
Calculator	C:\ProgramFiles\WindowsApps\[...]\CalculatorApp.exe
Edge	C:\ProgramFiles(x86)\Microsoft\Edge\Application\msedge.exe
Snipping Tool	C:\ProgramFiles\WindowsApps\[...]\SnippingTool\SnippingTool.exe
PickerHost	C:\Windows\System32\PickerHost.exe
File Explorer	C:\Windows\explorer.exe
News	C:\ProgramFiles\WindowsApps\[...]\Microsoft.Msn.News.exe
Microsoft Store	C:\ProgramFiles\WindowsApps\[...]\WinStore.App.exe
Paint	C:\ProgramFiles\WindowsApps\[...]\PaintApp\mspaint.exe
Clock	C:\ProgramFiles\WindowsApps\[...]\\PaintApp\mspaint.exe
Terminal	C:\ProgramFiles\WindowsApps\[...]\WindowsTerminal.exe

All of the data presented in Table 5.3-Table 5.4-Table 5.5-Table 5.6-Table 5.7,Table 5.8 and Table 5.9 was extracted by the metadata, excluding the description that was manually

done by us. Almost all the paths had to be shortened due to length limits, but in the `.csv` file they are complete. We can see that Recall captured a good amount of the activity of the User in the Scenario and the resulting timeline is close to the original one presented in Section 4.1, with only a few things missing:

1. Only the second equation done in *Calculator* was captured, and only partially.
2. The *Facebook* login form is missing, but that can be expected whenever the filter for sensitive data actually works.
3. The *Info* tab of the *AGV Volley* Facebook page was not captured, but we can still understand that the user was heading to that page from the *url* in the search tab.
4. There is no trace of the *InPrivate* navigation, which is to be expected. Recall also does not censor/skip the screenshot of the private navigation.
5. The gap in the timeline between *11:10:45* and *11:13:37* is not exceptionally long so that it might draw immediately the attention, but in a case where the User has done private navigation (or just simply paused Recall) for a longer period of time that would be more obvious.
6. Media Explorer was not caught, but being a single quick step with little impact on the screen that is not too surprising to us.
7. Recall does not distinguish between *Command Prompt* and *Windows PowerShell*.
8. The deletion of the files through the command line was not caught by Recall. This is important information that shows that the timeline obtained by Recall is not nearly enough by itself and must be paired with other tools and techniques of forensic analysis.

5.7 OCR to Scan Snapshots

As we said in Section 5.1, the search function leaves to be desired. That inspired us to make a script to perform an OCR reading of the exported snapshots and from the result do a search based on a list of keywords.

```python
def search_and_copy(input_folder, output_folder, search_string, languages):
    #initialize easyocr reader and suppress warnings
    reader = easyocr.Reader(languages, gpu=False, verbose=False)

    #ensure output folder exists
    os.makedirs(output_folder, exist_ok=True)

    #iterate through files in input folder
    for filename in os.listdir(input_folder):
        if filename.lower().endswith(('.jpg')):
            print(f"Processing image: {filename}")
            image_path = os.path.join(input_folder, filename)
            result = reader.readtext(image_path, detail=0)
            extracted_text = ' '.join(result)

            #check if search string is in extracted text
            if search_string.lower() in extracted_text.lower():
                #copy image file
                dest_image_path = os.path.join(output_folder, filename)
                shutil.copy2(image_path, dest_image_path)
                print(f"Copied image: {filename}")

                #copy corresponding json file if it exists
                json_filename = filename.rsplit('.', 1)[0] + '.json'
                json_path = os.path.join(input_folder, json_filename)
                if os.path.exists(json_path):
                    dest_json_path = os.path.join(output_folder, json_filename)
                    shutil.copy2(json_path, dest_json_path)
                    print(f"Copied JSON: {json_filename}")
                else:
                    print(f"JSON file not found for image: {filename}")
            else:
                print(f"Search string not found in image: {filename}")
```

Code Snippet 5: OCR reading of snapshots and filtering through a search string

A possible example is looking for the keyword "Telegram" inside a batch of snapshots. With this tool it will find instances of Telegram both as a desktop application and a website, and possible mentions in a text file and so on. All the relevant snapshots are then put in a folder specified by the user.

This tool is not particularly quick and could take a couple of minutes to analyse a single snapshot, but we believe that on a large batch of files it could save time to the investigator as they could let it run in the background while they work on something else and come back when it is done so that they can work on a subset of snapshots with the relevant content that they were looking for.

5.8 Analysing the Register Keys

We asked ourselves which registry keys are tied to Recall, and so we did a test to try to find some of them. We used *Regshot* to capture the changes to the registry keys when we set the following values for Recall:

- Maximum storage for snapshots from 75 to 25 GB
- Maximum storage duration for snapshots from 90 to 30
- Disable Filter sensitive information

```
Regshot 1.9.0 x86 ANSI
Comments:
Datetime: 2025/12/4 12:20:31 , 2025/12/4 12:21:38
Computer: LUIGI-DF , LUIGI-DF
Username: super , super

---------------------------------------------------

Keys deleted: 1

---------------------------------------------------

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Group Policy\{...}

---------------------------------------------------

Keys added: 7

---------------------------------------------------

HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Group Policy\{...}
HKLM\SOFTWARE\Classes\Local Settings\Software\Microsoft\{...}
HKLM\SOFTWARE\Classes\Local Settings\Software\Microsoft\{...}
HKLM\SOFTWARE\Classes\Local Settings\Software\Microsoft\{...}
HKLM\SOFTWARE\Classes\Local Settings\Software\Microsoft\{...}
HKLM\SOFTWARE\Classes\Local Settings\Software\Microsoft\{...}
HKU\S-1-5-21-1220528845-3739270546-3924994166-1001\{...}
```

Code Snippet 6: Example of Regshot output

We did this test six times and compared the results to see if there were common changes in the system across the tests, comparing the files with Listing 7. This script is slow as it compares each line in `file_a` to all lines in `file_b`.

```python
def compare_files(file_a_path, file_b_path):
    with open(file_a_path, 'r') as file_a, open(file_b_path, 'r') as file_b:
        lines_a = file_a.readlines()
        lines_b = file_b.readlines()

    line_map = {}
    for index_b, line_b in enumerate(lines_b):
        line_map[line_b.rstrip()] = index_b + 1  # Store line number (1-based)

    matches = []
    for index_a, line_a in enumerate(lines_a):
        line_a_stripped = line_a.rstrip()
        if line_a_stripped in line_map:
            if line_a_stripped != '':
                matches.append((index_a + 1, line_map[line_a_stripped]))

    return matches
```

Code Snippet 7: Compare two text files and return matching lines

The results were surprising:

- The files obtained by Regshot can range from 33 KB to 83 MB.
- There is no key that appears in the same way in all comparisons. Some appear in a couple, but the final result of comparing all of the tests is an empty file.

Having such a wide breath of results from Regshot and still no matches between all of them was the last thing that we expected. Maybe none of those settings touch registry keys, but in that case we are left with no easy ways to track the interactions between Recall and the Registry.

5.9 Frequency of snapshots

To know the frequency at which Recall takes the snapshots can be useful when reconstructing the actions taken by a user, and so we decided to test four different scenarios each one over a period of one hour: a black rectangle, a short GIF of a Pong game[4], a

[4] https://hadicya.dev/building-pong-with-monogame-a-step-by-step-guide

static Wikipedia page[5] and a YouTube video[6]. Here are the results:

1. Black rectangle: to test how Recall behaves in a situation akin to the user being away from their PC with something on screen.

   Number of snapshots: 1

   Frequency: 1 when the file was opened

2. Pong GIF: to test how Recall behaves with when the content on screen is repeated in a loop.

   Number of snapshots: 8

   Frequency: 1 every 30 seconds after the file is opened, after 4 minutes it stops.

3. Static Wikipedia page: there is very little happening on screen, but Recall might have a more ad-hoc behaviour when it comes to browsers as they seem to be a point of focus for the development of this application.

   Number of snapshots: 3

   Frequency: web page was opened at 10:39, first snapshot at 10:39, second one at 10:40, third and last one at 10:42.

4. Youtube video: a video with a person speaking and moving, slow moving background and few sudden cuts and jumps.

   Number of snapshots: 121

   Frequency: 1 every 30 seconds.

List 8: Snapshot Frequency Results

From these results, we can make some educated guesses at how Recall behaves:

- Recall can recognize when there is no activity on screen and so avoids wasting memory if nothing happens on screen.
- Recall can also recognize when on screen there is a short but frequent pattern and after a few snapshots it will stop until something different happens.
- Recall seems to have a bias towards web pages, even ones where there is no pop up, banner or anything remotely animated. Still, the snapshots are taken at a lower

[5] https://it.wikipedia.org/wiki/Cybersecurity_and_Infrastructure_Security_Agency
[6] https://www.youtube.com/watch?v=oDIm88df2Xo

frequency compared to the *gif* and *video* scenarios and it quickly stops after a couple of minutes.

- When there is high activity on screen but no inputs from the user, the frequency of snapshots averages to 1 per 30 seconds.

User input can change the frequency quite dramatically: while disabling the option for the screen to turn off after a period of inactivity in the Settings in preparation for the tests, Recall captured three snapshots in less than a minute, two of them just 10 seconds apart.

5.10 Hiding from Recall

As stated before, Recall allows the user to add applications and websites to a filter list so that Recall does not capture them in the snapshots when said applications or websites are in the foreground [Sup24a]. We decided to investigate further into the matter, starting with the applications in the filter list. We wanted to see if changing the name of the executable of an application already in the filter list would stop said filter to work and we tested it with *Notepad++*. We confirmed that Recall correctly filtered out any presence of Notepad++ on the snapshots, but also discovered that if the application is "in focus" (in this case when we were typing in the Notepad++ window) Recall stops entirely to take any snapshot, while if any other window is in focus Recall will simply hide the presence of the filtered application and take snapshots regularly. When we moved to testing with the browsers we saw that Brave stopped Recall from taking any snapshot while it was open and we got curious on how that was achieved. Sahib [Sah25] explains in an article that engineers at Brave leveraged the fact private browsing is not saved as snapshots, and so "extended that logic to all Brave browser windows", making every tab private. The user can disable this feature in the settings of the browser, but it is on by default as a safety measure: Sahib acknowledges that improvements have been made on the safety on Recall, but they claim to want to focus privacy first with their product. In their article they mention the work done by Signal [Lun25] as a partial inspiration for their solution, a different and more "aggressive" one that stops the application from appearing in any screenshot, not just the ones made by Recall. Signal achieves this by is setting the value of the Digital Rights Management flag [Lea25c]. We tried it by creating a simple Windows Desktop Application and adding the following line:

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
  //...
  SetWindowDisplayAffinity(hWnd, WDA_EXCLUDEFROMCAPTURE);
  //...
}
```

Code Snippet 9: Setting the DRM flag for a window

The result is that that window does not appear on any screenshot, showing the rest of the screen.

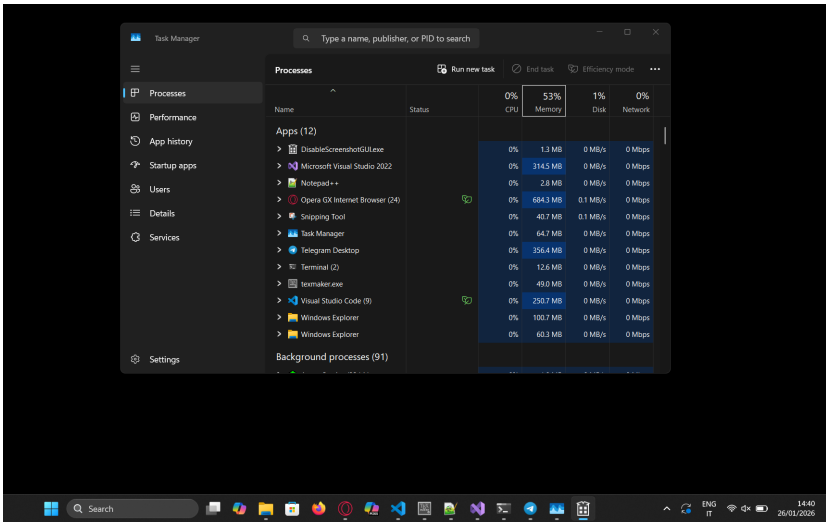

Figure 5.9: Example of program with the DRM flag set to hide from screenshots

In Figure 5.9 the icon on the taskbar of the program is still visible, but any content displayed on the foreground window is not. In regards to Recall, the program does not appear in the snapshot nor in the metadata. The contents of the program are successfully hidden from Recall and parsing only through the `JSON` files would not reveal its presence.

Going back to the browsers, we did not find any statement about DuckDuckGo blocking Recall, but one of their advertised features is default private browsing [Duc08], so that is most likely blocking Recall from taking any snapshot like with Brave.

5.11 Decryption Failing

While not directly associated with how Recall works and the data it generates, this problem does affect the ability to extract anything from the exported snapshots. In fact during all the other tests it was clear that sometimes the program used to decrypt said snapshots would fail to decrypt one or more `JSON` files, creating empty files instead. This only happens with the `JSON` files and not the related pictures. To test this bug we ran a 1000 times the decryption tool on batches of different sizes to see if the amount of files to decrypt in a single run mattered in some way. The smaller snapshot batches are all subsets of the bigger ones.

Table 5.10: Decryption Failures

Sample Size	Total Failures	Average Failure Ratio
25	291	0.0116
50	646	0.0129
100	1403	0.0128
200	2213	0.0111

Given the data above, we believe that the size of the sample does not impact the percentage of failures, and the percentage averages to 1.2%. We ave also seen that there are no corrupted files that might be causing this, as any file from the batches could fail in one and succeed in another one. If we then treat every decryption as an independent event, the probability of at least one `JSON` file being mistakenly empty after a single run of the program is:

$$P(y > 0) = 1 - (0.012)^x$$

where `x` is the number of files in a batch and `y` is the number of failures in a single run.

This problem is easily mitigated by running the decryption tool twice and crossing the results, reducing drastically the probability of having empty `JSON` files at the end. Finding out what might be the root cause of this is outside of the scope of this project.

5.12 Recall and Regions

As stated before, the option to export snapshots from Recall is only available inside the European Economic Area. As this could be an important limiting factor, we decided to

test if this could be circumvented. We did our tests first with the device set to Italy, then we performed a clean installation to set it to the United States. When performing a clean installation, Recall generates a new key so old snapshots were not compatible with the new setup. Also, we could confirm that the option to export snapshots was not available, nor did Recall provide a decryption key. We tried to change the current country in Windows Settings, but Recall bases itself on the country selected during the initial setup.

It does not matter where the device physically is, there is no check for that. If the user simply selected a country outside the European Economic Area during the initial setup, almost any analysis presented in this paper would not be possible.

5.13 Miscellanea

Here we present the results of tests that were too small to have a section of their own:

- Microsoft Store Applications: we tested to see if there was any difference in the metadata if the same application in the snapshot had been downloaded from the Microsoft Store or from another source.

  Result: The only difference in the metadata is that the field `IconUri`, which contains the path to the **.ico** file of the application, was correctly set in the Microsoft Store version instead of being empty.

- Data Persistence: we first reset the PC with the option to keep all of the data of the user, then we did a clean installation.

  Result: Even with the option of keeping the data of the user, all of Recall data was lost and it had to setup again. We were provided each time a new key, so the encryption had changed with each reset.

Chapter 6

Conclusion

We started this project questioning how useful could Recall be in a digital forensic investigation. We have seen that it can provide a detailed timeline of the actions of the user, with detailed information on the programs that were running. This is hardly the only method of retrieving such information, but it is relatively easy to do so, as it just requires the PIN of the account to access Recall. A PIN can be glanced while walking near a co-worker, typed in front of someone else without thinking about it or be simply naive (like 1234). A Copilot+ PC left unsupervised can be unlocked with the same PIN, and so a careless individual could have their history accessed by someone else, an history that can include sensitive data as we shown in Chapter 5. Depending on the use of the Copilot+ PC in question, it can range from annoying (a middling co-worker looking at work related stuff) to seriously dangerous (an abusive partner seeing your messages to a trusted friend). Forcing Recall to request every time the Biometric login would greatly enhance security, although (slightly) reducing the ease-of-use. A trade-off that we believe would be well worth it. On the forensic side, the biggest limiting factor is how restricted the pool of computers is where an investigator might find Recall helpful in their analysis:

- It must be a Copilot+ PC
- Recall have to enabled by the user
- The user must have set the initial location of the PC in a country inside the European Economic Area

With some confidence, we can say that the resulting subset is not going to be big enough to make a real impact:

- Copilot+ computers are still a luxury item compared to the average laptop

- Despite the improvements, the general public discourse is still wary of Recall and other AI tools, especially those that are seen as a potential security threat
- The regional setting excludes the majority of the world population, excluding users who set their computer to a country inside the EEA instead of their own

So, if the investigator has access to Recall and the decryption key, they can get a quick summarise of the activities the user did not want to hide and they can use the screenshots to better present the actions taken by the user to a less technically adept crowd. It will be an incomplete timeline, as Recall will not capture everything that happens, especially if the user is quick in its activities as shown in Section 5.5. The user might have also set Recall to filter out some programs which are likely going to contain critical or private information that they would not like to be captured in a snapshot. So the missing parts are potentially going to be activities that the investigator is looking for.

In the end, a useful tool if present, but nothing that an investigator should worry about adding to his repertoire of skills.

Bibliography

[Alb24] Albacore. *AmperageKit*. 2024. URL: https://github.com/thebookisclosed/AmperageKit (visited on 05/06/2025).

[Arn24] Pieter Arntz. *Microsoft Recall snapshots can be easily grabbed with TotalRecall tool*. 2024. URL: https://www.malwarebytes.com/blog/news/2024/06/microsoft-recall-snapshots-can-be-easily-grabbed-with-totalrecall-tool (visited on 01/23/2026).

[Bea25] Kevin Beaumont. *Microsoft Recall on Copilot+ PC: testing the security and privacy implications*. 2025. URL: https://doublepulsar.com/microsoft-recall-on-copilot-pc-testing-the-security-and-privacy-implications-ddb296093b6c (visited on 05/18/2025).

[Duc08] DuckDuckGo. *See how DuckDuckGo compares*. 2008. URL: https://duckduckgo.com (visited on 02/01/2026).

[Ehr24] Michelle Ehrhardt. *Microsoft Has Delayed Copilot+'s "Recall" Feature Yet Again*. 2024. URL: https://lifehacker.com/tech/microsoft-delays-copilot-recall-feature-again (visited on 01/23/2026).

[Gud25] Daniel Ju Gudge Timotius Margo. *RecallSnapshotExport*. 2025. URL: https://github.com/microsoft/RecallSnapshotsExport (visited on 01/21/2026).

[Hag24] Alex Hagenah. *TotalRecall*. 2024. URL: https://github.com/xaitax/TotalRecall (visited on 05/06/2025).

[Kha24] Imad Khan. *Microsoft's Copilot Embraces the Power of OpenAI's new GPT-4o*. 2024. URL: https://www.cnet.com/tech/services-and-software/microsoft-copilot-embraces-the-power-of-openais-new-gpt-4-o/ (visited on 05/06/2025).

[Lea24] Microsoft Learn. *Recall Overview*. 2024. URL: https://learn.microsoft.com/en-us/windows/apps/develop/windows-integration/recall/ (visited on 01/28/2026).

[Lea25a] Microsoft Learn. *Click to Do overview*. 2025. URL: https://learn.microsoft.com/en-us/windows/apps/develop/windows-integration/click-to-do?tabs=csharp (visited on 02/20/2026).

[Lea25b] Microsoft Learn. *Enable relaunching your content from Recall*. 2025. URL: https://learn.microsoft.com/en-us/windows/apps/develop/windows-integration/recall/recall-relaunch (visited on 02/20/2026).

[Lea25c] Microsoft Learn. *Manage Recall*. 2025. URL: https://learn.microsoft.com/en-us/windows/client-management/manage-recall#information-for-developers (visited on 01/23/2026).

[Lea25d] Microsoft Learn. *Trusted Platform Module Technology Overview*. 2025. URL: https://learn.microsoft.com/en-us/windows/security/hardware-security/tpm/trusted-platform-module-overview (visited on 02/23/2026).

[Lea26] Microsoft Learn. *Windows registry information for advanced users*. 2026. URL: https://learn.microsoft.com/en-us/troubleshoot/windows-server/performance/windows-registry-advanced-users (visited on 02/23/2026).

[Lun25] Joshua Lund. *By Default, Signal Doesn't Recall*. 2025. URL: https://signal.org/blog/signal-doesnt-recall/ (visited on 01/23/2026).

[Sah25] Shivan Kaul Sahib. *Brave blocks Microsoft Recall by default*. 2025. URL: https://brave.com/privacy-updates/35-block-recall/ (visited on 01/23/2026).

[Sea19] Seabreg. *Regshot*. 2019. URL: https://github.com/Seabreg/Regshot (visited on 12/14/2025).

[Sup24a] Microsoft Support. *Filtering apps, websites, and sensitive information in Recall*. 2024. URL: https://support.microsoft.com/en-us/windows/filtering-apps-websites-and-sensitive-information-in-recall-a4c28bee-e200-4a4a-b60d-c0522b404a5b (visited on 02/01/2026).

[Sup24b] Microsoft Support. *Retrace your steps with Recall*. 2024. URL: https://support.microsoft.com/en-us/windows/retrace-your-steps-with-recall-aa03f8a0-a78b-4b3e-b0a1-2eb8ac48701c (visited on 05/18/2025).

[Tea25] Windows Insider Program Team. *Releasing Windows 11 Build 26100.3902 to the Release Preview Channel*. 2025. URL: https://blogs.windows.com/windows-insider/2025/04/10/releasing-windows-11-build-26100-3902-to-the-release-preview-channel/ (visited on 05/18/2025).

[War24a] Tom Warren. *Microsoft's all-knowing Recall AI feature is being delayed*. 2024. URL: https://www.theverge.com/2024/6/13/24178144/microsoft-windows-ai-recall-feature-delay (visited on 01/23/2026).

[War24b] Tom Warren. *Microsoft's Recall AI is creepy, clever and compelling*. 2024. URL: https://www.theverge.com/2024/12/12/24319609/microsoft-recall-hands-on-notepad (visited on 01/23/2026).

[Wes24] David Weston. *Update on Recall security and privacy architecture*. 2024. URL: https://blogs.windows.com/windowsexperience/2024/09/27/update-on-recall-security-and-privacy-architecture/ (visited on 01/31/2026).

[Zac24] Phill Moore Zach Stanford Yogesh Khatri. *Forensic Applications of Microsoft Recall*. 2024. URL: https://cybercx.com.au/blog/forensic-applications-of-microsoft-recall/ (visited on 05/06/2026).