forensic sciences

MDPI

Article

A Digital Forensic View of Windows 10 Notifications

Patrício Domingues [1,†], Luís Andrade [2,†] and Miguel Frade [3,*,†]

[1] Instituto de Telecomunicações, CIIC, ESTG, Polytechnic of Leiria, Morro do Lena–Alto do Vieiro, 2411-901 Leiria, Portugal; patricio.domingues@ipleiria.pt
[2] ESTG, Polytechnic of Leiria, Morro do Lena–Alto do Vieiro, 2411-901 Leiria, Portugal; luis.m.andrade@outlook.com
[3] CIIC, ESTG, Polytechnic of Leiria, Morro do Lena–Alto do Vieiro, 2411-901 Leiria, Portugal
* Correspondence: miguel.frade@ipleiria.pt
† These authors contributed equally to this work.

Abstract: Windows Push Notifications (WPN) is a relevant part of Windows 10 interaction with the user. It is comprised of badges, tiles and toasts. Important and meaningful data can be conveyed by notifications, namely by so-called toasts that can popup with information regarding a new incoming email or a recent message from a social network. In this paper, we analyze the Windows 10 Notification systems from a digital forensic perspective, focusing on the main forensic artifacts conveyed by WPN. We also briefly analyze Windows 11 first release's WPN system, observing that internal data structures are practically identical to Windows 10. We provide an open source Python 3 command line application to parse and extract data from the Windows Push Notification SQLite3 database, and a Jython module that allows the well-known Autopsy digital forensic software to interact with the application and thus to also parse and process Windows Push Notifications forensic artifacts. From our study, we observe that forensic data provided by WPN are scarce, although they still need to be considered, namely if traditional Windows forensic artifacts are not available. Furthermore, toasts are clearly WPN's most relevant source of forensic data.

Keywords: digital forensics; Windows 10; Windows 11; push notifications; sqlite3

Citation: Domingues, P.; Andrade, L.; Frade, M. A Digital Forensic View of Windows 10 Notifications. *Forensic. Sci.* **2022**, *2*, 88–106. https://doi.org/10.3390/forensicsci2010007

Academic Editor: Marcus Rogers

Received: 28 December 2021
Accepted: 25 January 2022
Published: 31 January 2022

1. Introduction

The Windows Push Notification service (henceforth, WPN) appeared in the first release of Windows 10, although it was heavily based in Windows 8's Windows Notification Service [1]. As its name implies, it is a service that allows notifications to be delivered to a Windows 10 OS. Specifically, WPN allows applications to *push* notification data to the end user. Notifications can manifest themselves through three different graphical outputs: badges, tiles and toasts, plus a non-graphic mode, named *raw*. A badge is usually a tiny symbol, often a number that appears in one corner of the application's icon signaling a notification, for example, the number of new email messages. Although badges fit more naturally mobile OS such as iOS and Android, mostly because the limited screen space of these platforms is filled with applications' icons, badges are also used by Windows 10 in both the task bar and in the system tray. For example, the Microsoft Teams application resorts to a bottom-right badge in the application icon displayed in a Windows' task bar to report the network connectivity of the local computer to the Teams service. Other applications use it on the system tray. Examples include the Dropbox client software that notifies the user of an update operation on files by displaying a blue and white update symbol, and Microsoft's OneDrive that reports some statuses such as paused, synchronizing or lack of network connectivity as shown in Figure 1. These symbols are from a limited set, as they are part of the notification API. Likewise, a numeric badge can have a value between 1 and 99, with values above 99 represented as 99+. Figure 2 displays examples of badges as they appear in Windows 10 taskbar. Specifically, the left badge signals that Microsoft Teams has one new event to report (e.g., a newly received message), while the

center badge reports on a new message by the Facebook Messenger client. Finally, the right-most badge alerts that there are six events to process from Windows Your Phone, which can correspond to newly received SMS (Short Text Messages) or notifications from the coupled Android smartphone [2].

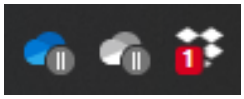

Figure 1. Windows 10s system tray badges status icons. From (**left–right**): Paused OneDrive, non-connected OneDrive, and Dropbox with one pending notification.

Figure 2. Examples of Windows 10s badges. From (**left–right**): Teams, Messenger, and Your Phone.

Tiles first appeared with Windows 8, when Microsoft decided to overhaul the interface of the OS, namely the start menu, introducing the *Metro* interface, along with a new type of applications called *Metro Apps*, guiding Windows 8 for touch interfaces [3]. Tiles are rectangular shapes in the screen, each one linked to an application, which can feed the tile with new content, hence the alternate designation of *live tiles*. In Windows 10, although Microsoft introduced new changes to the start interface, live tiles were kept, now being linked to *Universal Windows Platform* (UWP) applications. In this context, each tile is a live representation of its linked application. Examples of live tiles include Microsoft Photos, which rotatively displays, within the tile, photos existing in the device, or the Weather application which displays the weather forecast for the user's location. This is the case shown in Figure 3, where the weather prediction for the city of Leiria, Portugal is shown for three days. As we shall see later in Section 3.5, live tiles were removed from a Windows 11 interface.

Figure 3. A Windows 10 tile linked to the Weather application for the city of Leiria, Portugal.

Toasts are rectangular shaped boxes that pop up from the right bottom corner of the screen. An example of a toast is given in Figure 4. The name stems from the rogue similarity of toasts popping out of a toaster. Toasts are sometimes also designated as *banners*. By default, toasts are shown for five seconds, although this can be changed in Windows *Ease of Access/Display* up to a maximum of five minutes. There are a wide variety of toasts, some of them interactive, such as *Quick reply text box*, *Progress Bar*, and *Context menu actions*, among many others [4]. This class of interactive toasts are also known as *actionable* notifications. There are four different types of toast notifications: (i) *local*, (ii) *scheduled*, (iii) *periodic*, and (iv) *push*. Local toast notifications are produced by local applications. Examples include a mail client alerting the reception of a new email, or a social network application notifying a new message. Scheduled notifications are delivered by calendar-based applications that run within the browser, for example Google Calendar, to remind the user of an upcoming event. Periodic notifications are used by applications that deal with periodically changing data, such as weather or stock prices. Finally, a push notification originates from a cloud server. These cloud notifications are used, for example, by news applications to alert of a breaking news, or by a communication application such as Microsoft's Your Phone (https://www.microsoft.com/en-us/p/your-phone/9nmpj99vjbwv, accessed on 28 January 2022) to alert of an incoming call or message. Push mode

is the only one to support *raw* notifications that is a notification that goes directly from a cloud service to the application, without any graphical display. The application processes the notification, triggering the reaction it deems most appropriate, which can include toast notifications.

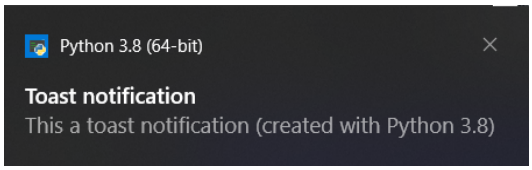

Figure 4. A Windows 10 toast notification triggered by a Python script.

An important element in the toast notification chain is the Windows Action Center. As the name implies, it acts as a central repositories for toast notifications, stocking toasts that can later on be reviewed by the user. Additionally, when the user is engaged on a task that should not be interrupted with toasts—e.g., displaying a full screen Powerpoint presentation—incoming toasts are silently redirected to the Action Center to avoid disturbing the user. Furthermore, applications can deliver the so-called *ghost* toasts that do not pop up on the screen, but instead go directly to the Action Center. An example of Windows 10 Action Center is shown in Figure 5, where besides a panel for fast access to settings (e.g., Night Light, Network, etc.), there is a toast message from the Dropbox application. Note that the `Focus Assist` section in the panel (bottom right) allows one to control how notifications are treated: when clicked once, the mode switches to `Priority only`, while an additional click activates the `Alarm only` mode. A third click loops back to the original mode of `Focus assist` Off. Windows Action Center display is activated by clicking in the message shape icon, which is highlighted by the squared red box at the bottom right of Figure 5. Another click and it gets closed.

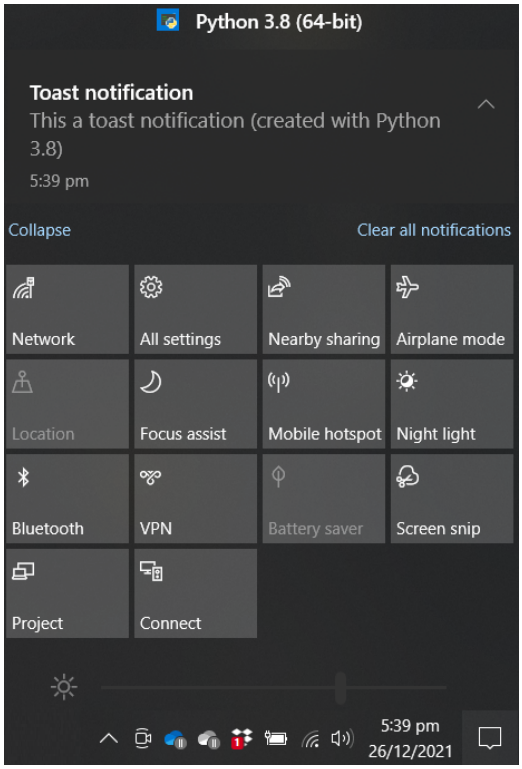

Figure 5. Windows 10 action center.

The main motivation to study Windows notifications from a digital forensic perspective is that notifications can harbor data that might be relevant to forensic investigations of Windows 10 devices and user(s). For example, an email client toast notification might hold

data and metadata—sender, date/time—about incoming emails. Although the amount of data are limited, it might be useful when common forensic artifacts have provided no valuable data, or the machine was cleaned with anti-forensic software [5,6]. Additional details about anti-forensic software are presented in Section 3.1.

In this paper, we analyze the WPN service from a digital forensic perspective: the goal is to identify and extract the forensic artifacts when a post mortem analysis is performed in a Windows 10 machine. The main contributions of this paper are: (i) Identification of forensic artifacts linked to WPN in both Windows 10 and Windows 11; (ii) development of an open source Python 3 script able to extract WPN's forensic artifacts; and (iii) implementation of a Jython extension module for the forensic software Autopsy. This module interacts with the Python 3 script to collect and display WPN's artifacts within Autopsy.

The remainder of this paper is organized as follows: Section 2 presents some related work regarding the Windows Notification system, while Section 3 details the services, files and databases that comprise WPN. Section 4 presents the Notification Analyzer software, comprising an autonomous Python 3 script and Autopsy's specific Jython module. In Section 5, experimental results highlighting extracted and recovered data are presented. Section 6 describes other sources of notifications with a potential interest for digital forensics, namely Windows event logs and caches of browsers. Finally, Section 7 concludes the paper and traces venues for future work.

2. Related Work

We now review related work, focusing on two main items: (i) works that deal with Windows' WPN, and (ii) tools aimed at recovering removed data from SQLite 3 databases.

2.1. Windows Notifications

Skulkin and de Courcier briefly analyze Windows 10 Notifications [7]. They observe that the first release of Windows 10 kept notification-related data in a file called `appdb.dat`. Changes to the `wpndatabase.db` file (a SQLite 3 database) occurred in Windows 10 anniversary edition, version 1607 launched in July 2016. The `wpndatabase.db` database summarily analyzed by the authors had five tables, while Windows 10s version 21H1 has eight tables. In their analysis of the table `Metadata`, Skulkin and de Courcier indicate that some of the fields of the table correspond to the number of notifications seen by the user. For instance, for the field `toast:maxCount`, the authors point out that the value 20 means that the corresponding user has effectively seen 20 toasts. By changing the `toast:maxCount` value, we were able to verify that this field corresponds instead to the maximum number of toast notifications that can exist at a given time in the notification center. In their analysis of the main differences between Windows 8.1 and the first release of Windows 10 at the level of digital forensic artifacts, Hintea et al. succinctly describes Windows 10 Notification Center [1]. They only note that toasts' notifications are stored in XML format in the file `appdb.dat`, giving no further data regarding Windows Notification Center. In his blog, Khatri details the structure of the `appdb.dat`, pointing out that the file hosts a binary database with the `DNPW` as the first four bytes [8]. Khatri also provides a Python 3 script that creates a CSV representation of data that exists in `appdb.dat` file. Maloney resorts to the Python 2 script `walitean.py` to transform the Write-Ahead Log (WAL) file `wpndatabase.db-wal` in a SQLite 3 database[9]. Github's user Kacos2000 (https://github.com/kacos2000/Win10/blob/master/Notifications/, accessed on 28 January 2022) has a branch dedicated to Windows 10s notifications. The Github repository provides two SQL queries and several power shell scripts that process data from the WPN's databases.

TZworks' `wpn` (https://tzworks.com/prototypes/wpn/wpn.users.guide.pdf, accessed on 28 January 2022) is a command line tool to parse WPN databases. According to the documentation, the tool also attempts to recover data and provides several output formats. As `wpn` is a commercial tool, which requires a valid license, we could not test it. ArtiFast (https://www.forensafe.com/artifast.html, accessed on 28 January 2022) is a forensic software from Forensafe that targets Windows OS. It lists Windows 10 notifications

as one of the artifacts processed by the tool (https://www.forensafe.com/resources/ArtiFast-Supported-Win-Artifacts.pdf, accessed on 28 January 2022). Again, as it is a commercial tool, we did not test it. Conversely, our Windows Notification Analyzer (WNA) software is available under GPL 3.0 license (https://github.com/labcif/YPA, accessed on 28 January 2022). Bilogrevic et al. study the management of push notifications in the Google Chrome web browser [10]. They report that, based on Google Chrome telemetry, the permission grant rate to deliver Google Chrome notifications to the desktop is only 10% for Windows, meaning that very often users do not accept notifications coming from the web browser. The authors also present a novel and less intrusive notification interface API that might increase the grant approval rate.

2.2. SQLite 3 Recovering Tools

SQLite 3 is the predominant database in the computer world, mostly due to its omnipresence in mobile applications [11]. Thus, the existence of several tools to recover SQLite3 removed data is no surprise. In this work, we solely focus on open-source solutions.

DeGrazia developed the `sqlparse.py` Python 2 script that recovers strings from free blocks of a SQLite 3 database, outputting CSV-formatted data [12]. As stated by its author, Daniels, the `undark` tool is a command line application that goes through a whole SQLite database file and dumps all data that it can find [13]. It is written in the C language. The `wal_crawler.py` Python 3 script parses the WAL file to recover non-directly accessible data from a SQLite 3 database ([14], chapter 12). The `bring2lite` tool analyzes the SQLite 3 main database file and the WAL file to recover data. As noted by the authors, this allows the Python 3-based software to yield a high rate of data recovery [15].

Pawlaszczyk and Hummert propose the FQlite software, which relies on carving algorithms and techniques to recover deleted data from SQLite databases [16]. The software analyzes both the database file and the associated WAL file. `FQlite` is written in Java and provides two running interfaces: a command-line one and a graphical user interface (GUI). Both interfaces output recovered data in a CSV format.

Our WNA software integrates all of these SQLite 3 recovery tools, with the sole exception being FQLite due to a limitation that does not allow it to run under Windows OS (https://github.com/pawlaszczyk/fqlite/issues/1, accessed on 28 January 2022). The rationale behind this approach is the reduced lifetime of notifications, since the maximum existence of a notification in the database is three days. This means that notifications, and particularly toasts which carry the most valuable forensic data, have a reduced lifespan, and hence, it is important to carve the database and the associated WAL file for removed data.

3. WPN's Services, Executables and Data Repositories

In this section, we analyze the main components of WPN, namely the Windows services and the executables that implements WPN, as well as the data repositories. First, we briefly describe Materials and Methods.

3.1. Materials and Methods

To study WPN and its components, we used two regular laptops: one with Windows 10–1909 and another one with Windows 11–21H2. To detect activity linked to WPN, we resorted to Windows Sysinternals' Process Monitor as it allows for monitoring in real-time Registry and file systems operations of processes and threads. We also resorted to the HxD Hex Editor to identify and interpret binary content, as well as Windows Sysinternals' Strings utility. To inspect the SQLite 3 databases, we used the Db Browser for SQLite application. The Registry analysis was performed with Windows own Regedit. We used the `jsonform` option of the versatile SFK (Swiss File Knife) versatile utility to format JSON content. To interact with WPN to produce graphical output, namely toasts, we resorted to `winrt` and `win10toast` Python 3 modules.

The methodology involved the generation of notifications and the subsequent analysis. For this purpose, several applications were used, namely Microsoft's Your Phone, as it can

easily be triggered to produce notifications. Other applications were Microsoft Outlook and web browsers such as Brave and Google Chrome. After triggering notifications, the hosting Windows system was analyzed, with focus in the WPN's associated directories.

The open source anti-forensic tool BleachBit version 4.4.2 (https://www.bleachbit.org/, accessed on 28 January 2022) that supports Linux and Windows, was also used to study its impact on Windows notifications. In its default configuration, BleachBit does not support the notification system, and thus no artifacts are removed. Moreover, when additional settings are enabled through the `Winapp2.ini` extension, (https://github.com/MoscaDotTo/Winapp2, accessed on 28 January 2022) the action `Notification` of BleachBit still does not remove the SQLite3 databases of WPN services. However, actions exist to clean the data storage notifications from applications such as Windows Edge or Google Chrome. In the case of the two browsers, these actions delete the levelDB database that holds the notifications. An example of those actions is depicted in Figure 6.

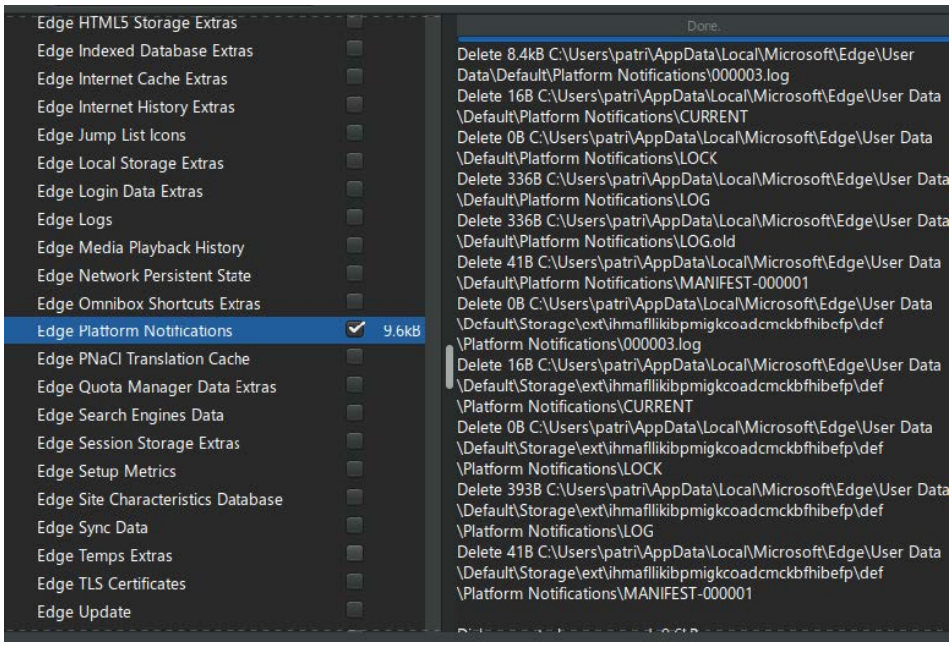

Figure 6. Example of Bleachbit extra actions to delete the levelDB database that holds the notifications of the Edge browser.

3.2. Services and Executable Files

WPN relies on two services: (1) a system service, referred to as a *Windows Push Notification System Service* and (2) a user service known as *Windows Push Notification User Service*. The system service has the following description text: *This service runs in session 0 and hosts the notification platform and connection provider which handles the connection between the device and WNS server*. Table 1 lists the executable file, the command line parameters and the dynamic link libraries (DLL) directly related to WPN that are loaded by the system service. This service handles notifications related to Windows OS functionalities such as Windows Defender, Windows Update and Bluetooth device pairing, to name just a few.

Table 1. WPN system service.

Command line	`svchost.exe -k netsvcs -p`
DLLs	`wpnservice.dll`, `wpncore.dll`, `wpnprv.dll`

The user service is described as follows in the Windows 10 service area: *This service hosts a Windows notification platform which provides support for local and push notifications. Supported notifications are tile, toast and raw*. Table 2 shows the executable path and its command line parameters, as well as the WPN related DLL loaded by the service. This

service is linked to the current logged in user and handles the delivery of notifications to the end user.

Table 2. WPN user service.

Command Line	`svchost.exe -k UnistackSvcGroup -s WpnUserService`
DLLs	`wpnuserservice.dll`, `wpncore.dll`, `wpnapps.dll`, `wpnclient.dll`

3.3. Databases

For each of the WPN service processes, there is a SQLite3 database whose main filename is `wpndatabase.db`, and has two associated files `wpndatabase.db-shm` and `wpndatabase.db-wal`. Specifically, there is one database for the system notification and another one for each user's account on the machine. Paths for both the system and per-user accounts are shown in Table 3.

Table 3. Paths for system and per-user account WPN data repository.

Scope	Path of Data Repository
System-wide	`\%WINDIR\%\System32\config\systemprofile\AppData\Local\` ↪ `Microsoft\Windows\Notifications`
Per user	`\%LOCALAPPDATA\%\Microsoft\Windows\Notifications`

The system WPN database files are kept in `C:\Windows\System32\config\systemprofile\AppData\Local\Microsoft\Windows\Notifications`. The `per user` WPN database files are held in the `\%LOCALAPPDATA\%\Microsoft\Windows\Notifications` directory, where `\%LOCALAPPDATA\%` is an environment variable that maps to `C:\Users\username\AppData\Local\`, with `username` being the login name of the user and considering that Windows is installed in the `C:` drive. This means that there is a database for each user account, and thus data kept in the database can be linked to individual accounts. This user attribution is of special importance in digital forensics in systems that can have multiple users, as it is the case of Windows machines.

As the `wpndatabase.db` databases are identical in their structure, next, we focus on the user's database, as it is the one that has the most meaningful forensic data, such as Outlook emails and calendar notifications. Each `wpndatabase.db` database has eight tables, whose names are listed in Table 4.

Table 4. Tables of `wpndatabase.db`.

Table's Names	
`HandlerAssets`	`NotificationData`
`HandlerSettings`	`NotificationHandler`
`Metadata`	`TransientTable`
`Notification`	`WNSPushChannel`

In our testing, the tables `NotificationData` and `TransientTable` remained empty. As the table `NotificationData` has solely three fields—`NotificationId`, `Key` (text), and `Value` (text)—appropriate to store key/value pairs, we hypothesize that it might be a leftover from previous versions of Windows. As the name appears to suggest, the `TransientTable` might be used for transient data of the notification system. The `Metadata` table holds some configuration values. An example is given in Table 5. Note that the record `currentNotificationId` is the current notification ID and gives an indication of the number of notifications processed so far (89591 for the given example). Records suffixed with *maxCount* define limits. For example, `toast:maxCount` regulates the maximum number of toasts that are kept in a Windows Action Center.

From a digital forensic perspective, the table `Notification` holds the most meaningful data. This table has 14 fields, listed in Table 6. Next, we solely focus on fields with relevant value for digital forensics.

Table 5. Example of the content of the `Metadata` table.

Key	Value
tile:maxCount	5
toast:maxCount	20
badge:maxCount	1
toastCondensed:maxCount	80
DB_INITIALIZED	1
OutlookQuirkEnabled	1
TilesBandwidthTotal	0
CurrentNotificationId	89591

Table 6. Fields and data types of table Notification (default values are within parentheses).

Field Name	Datatype
Order	integer (primary key)
Id	integer
HandlerId	integer
ActivityId	GUID
Type	text [tile,badge,toast,toastCondensed]
Payload	BLOB
Tag	text
Group	text
ExpiryTime	int64
ArrivalTime	int64
DataVersion	int64
PayloadType	text ('Xml')
BootId	int64 (0)
ExpiresOnReboot	boolean (FALSE)

The `HandlerID` field holds the ID of the application that handles the notification. This ID corresponds to the `RecordID` of table `NotificationHandler`. Examples of handlers, extracted from the `NotificationHandler`, are shown in Table 7. It should be noted that there is a large number of handlers. In the studied systems, we found over 238 handlers defined in the `NotificationHandler` table. The number of handlers depends on the installed applications. In the given case, 143 notification handlers were created by Windows 10, as their creation time is the same as Windows 10 OS installation time or Windows 10 last version update. The creation time of a notification handler is kept in the field `CreatedTime` of the `NotificationHandler` table. Other notification handlers are created by UWP applications which are installed on the system.

Table 7. Partial view of the NotificationHandler table.

RecordId	PrimaryID	HandlerType
38	Windows.SystemToast.WindowsTip	app:system
149	Microsoft.Office.OUTLOOK.EXE.15	app:desktop
238	Microsoft.SkyDrive.Desktop	app:desktop

Another field of table `NotificationHandler` is `Type`. It is a text field that represents the type of notification, which can hold one of the following qualifiers: `badge`, `tile`, `toast` and `toastCondensed`. Data that comprise the notification are kept in yet another text field of the `Notification` table: `Payload`. These data are in a XML-like format, and the fields they harbor depend on the source application. A shortened example of a toast

from the Facebook Messenger Windows UWP application is shown in Listing 1. From the payload, one can extract that the message comes from `Jane Doe` and has the content "`Almost done!`". Toast payloads are definitely the most interesting ones as they can come from messaging applications such as Facebook Messenger, Microsoft Outlook, and Google Calendar, just to name a few. This can yield valuable information, as an email notification from Microsoft Outlook holds the sender and the initial part of the subject. The same goes for Facebook Messenger toasts that identify the sender and hold the initial part of the message. Obviously, the amount of data are limited, and as stated earlier, its usefulness is limited and restricted to scenarios where other sources of forensic data are not available. On the contrary, `toastCondensed` are of low use, as they have a NULL payload, at least in the corresponding entry in the table.

Listing 1. Shortened example of a toast from the Facebook Messenger Windows UWP application.

```
<toast launch="messenger">
    <visual>
        <binding template="ToastGeneric">
            <text hint-maxLines="1">Jane Doe</text>
            <text>Almost done!</text>
            <image src="https://scontent.flis9-1.fna.fbcdn.net/v/t1.0-1/p100x100/(...)">
             placement="appLogoOverride" hint-crop="circle"/>
        </binding>
    </visual>
    <audio src="ms-appx:///Assets/incoming_message.m4a" silent="false"/>
</toast>
```

There are two important date/time fields in table `Notification`: `ArrivalTime` and `ExpiryTime`. Both are expressed in Microsoft Filetime 64-bit format, that is, as the number of 100-nanosecond intervals elapsed since 1 January 1601, 0:00 UTC. The former refers to the arrival of the notification, while `ExpiryTime` field indicates when the notification is set to expire. By default, a notification is set to last for a maximum of three days, but this can be set to a much lower value by the source application.

3.4. The Wpnidm Directory

Within the notification data directories, there is a subdirectory named `wpnidm`. This directory exists in the system-wide notification and in the per-user account data repositories. In the analyzed systems, although the system-wide `wpnidm` directory was empty, the per-user `wpnidm` held JPG, PNG and IMG images.

These images are from toast notifications and tiles. For example, to notify of a new message, the Facebook Messenger application for Windows adds a thumbnail of the profile photo of the Facebook account of the sender (https://www.microsoft.com/en-us/p/messenger/9wzdncrf0083, accessed on 28 January 2022). This thumbnail, as well as the full sized profile image, is kept in the `wpnidm` directory. The name of all files kept in the `wpnidm` directory is an at most 8-wide hexadecimal number (e.g., `a3ac0253.jpg`), which is a 32-bit number. We could not figure out how the filenames are created, although we ruled out that they do not correspond to the CRC32 digest hash of the content of the respective file. Interestingly, some of the profile photos are repeated several times, with different and unrelated names (e.g., `a3ac0253.jpg` and `e1979169.jpg`). After some research, we found out that the content of the `wpnidm` directory is mapped into the registry, more precisely at the following key path: `\HKEY\_USERS\S-1-5-21-XX-XX-XX-SID\Software\Microsoft\Windows\CurrentVersion\PushNotifications\wpnidm`. Under this key, there is one subkey for each file that exists in the `wpnidm` directory. Each subkey has the same name as the file, minus the extension. Within this subkey, several data fields exist. These data fields are listed jointly with a brief description in Table 8. For some of the entries, the *Url* entry points to an online version of the image file, accessible without authentication. However, for some other entries of the same image, attempts to access the *Url* yield a *URL signature expired*. We hypothesize that an image duplicate is created when the URL for the previously

valid image expires, this behavior explaining the existence of several duplicates of images. The content of a wpnidm key is shown in Figure 7.

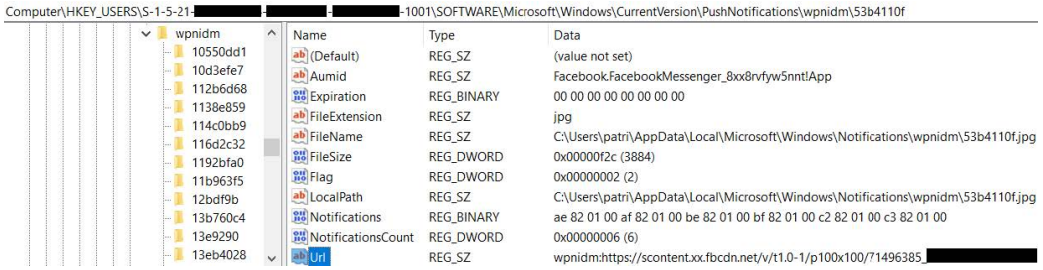

Figure 7. Content of a registry's wpnidm key.

Table 8. Fields and content of a wpnidm subkey.

Field Name	Description
`FileExtension`	JPG, PNG, IMG
`Url`	URL of image/thumbnail prefixed with `wpnidm:`
`FileName`	Full path of file
`FileSize`	file size (bytes)
`Flag`	32-bit value. Always 2.
`LocalPath`	Full path of file
`Aumid`	application ID (e.g., Facebook)
`Expiration`	(always zero)
`NotificationsCount`	Number of notifications with the image
`Notifications`	Variable 12-byte hexadecimal value

3.5. Notifications in Windows 11

We now briefly review both notification services, System and per-user account, in Windows 11. For this purpose, we analyze Windows 11 Home, version 21H2, build 22000.318, which was installed in a computer as an upgrade of Windows 10.

At the user level, there is one major change: Windows no longer has live tiles in the start menu. This is one of the modifications of the so-called *start menu* overhaul, which, in Windows 11, is unmovable at the bottom center of the screen. Another modification is the now rounded corner in Windows 11's windows. Figure 8 shows a toast notifications with a clickable URL. Another major change is related to the user interface of Action Center, which is now less cluttered than it was in Windows 10, as shown in Figure 9.

Internally, we noticed only a single major change in the `wpndatabase.db` databases: the addition of a table called `TimedNotification`. However, as this table remained empty in our testing environment, we could not figure out its real purpose, besides the indication given by its name. Table 9 lists the fields and datatypes of the `TimedNotification` table. Interestingly, support for tiles was left untouched in the `wpndatabase.db` files, and in fact there are still applications that resort to tiles, such as Bing.Weather and Microsoft Office OneNote.

Figure 8. A Windows 11 Toast with a clickable URL.

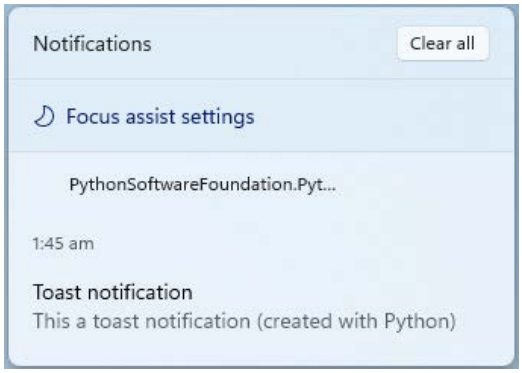

Figure 9. Windows 11 action center.

Table 9. Table TimedNotifications of database `wpndatabase.db` (Windows 11).

Field	Datatype
TimeEventId	GUID
BBIworkId	GUID
WnfStateName	INT64
HandlerId	INT64
NotificationType	INT
Url	Text

4. Notification Analyzer Program

To automate the process of extracting and analyzing the `wpndatabase.db` database, we developed the `WindowsNotificationsAnalyzer` program. This program comprises two elements: (i) a Python command line script named `NotifAnalyzer` that can be executed as an autonomous script, and (ii) a Jython script tailored to run as a module within the digital forensic software Autopsy [17,18]. This Jython script runs, parses and integrates the `NotifAnalyzer` output into the Autopsy's graphical interface. As stated earlier, the Autopsy module is called `Windows Notification Analyzer`.

4.1. The NotifAnalyzer Script

`NotifAnalyzer` is a Python 3 script that receives, through command line arguments, the path for a `wpndatabase.db` database, and produces as output a JSON file with the main data from the database, namely from the `Notification` and `NotificationHandler` tables. The path for the `wpndatabase.db` to analyze is given through the `-p/--path <path>` command line option, while the path and name for the output JSON file are specified through the option `-j/--json <path>`. As an aside feature, `NotifAnalyzer` produces itself a notification toast when it terminates its execution as shown in Figure 10. This toast is only produced if the option `-t/--toast` is given at the command line and the `win10toast` (https://pypi.org/project/win10toast/, accessed on 28 January 2022) Python module is installed on the local machine.

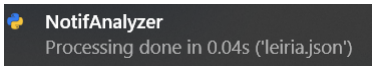

Figure 10. Toast signaling NotifAnalyzer's end of execution.

4.2. Windows Notification Analyzer Module

Autopsy (https://www.sleuthkit.org/autopsy/, accessed on 28 January 2022) is an open source software that runs within a Java Virtual Machine (JVM). It combines several functionalities and forensic tools within a graphical user interface to assist digital forensics practitioners in forensic examinations. Some of the integrated tools are The Sleuth Kit to access unmounted filesystems [19], `ffmpeg` (https://ffmpeg.org/, accessed on 28 January 2022) to interact with videos, `tesseract` (https://github.com/tesseract-ocr/tesseract,

accessed on 28 January 2022) to provide optical character recognition (OCR) and the Image Magick (https://imagemagick.org/, accessed on 28 January 2022) suite to process and display images. Autopsy is a well-known software within the digital forensic community. It can be extended through modules that can be developed either in Java or in Jython. Jython is a Python environment that runs within a JVM instance, offering the speed of development of Python and the wealth of functionalities of the JVM. Modules written for Jython run within the JVM used by Autopsy.

To assist digital practitioners, we developed the Jython-based module WNA. The `WNA` module resorts to the `NotifAnalyzer` Python 3 script to access a JSON-formatted representation of the meaningful data of each of the `wpndatabase.db` under analysis. The JSON data are then further analyzed and imported to Autopsy by loading them to Autopsy's blackboard, and creating entries in the *Extracted Content* result tree. The entries created by the module are prefixed with the *WPN:* string. Figure 11 illustrates the content tree produced by the execution of the WNA module over the notification database of user `Leiria`. The numbers within curve parentheses count the number of lines of content extracted by WNA from the database. For example, there were 42 rows extracted from the `Notification` table as pointed out by *User Leiria – Notifications (42)*. The last five entries shown in Figure 11 come from applying recovering algorithms to the database and are addressed in Section 4.3.

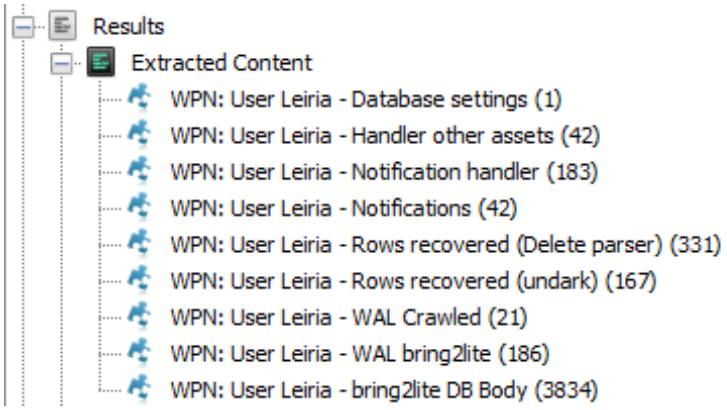

Figure 11. Example of the extracted content tree produced by the WNA module.

4.3. Recovering Records

From the digital forensic perspective, a major issue with notifications is the limited lifetime of the data. Indeed, as stated earlier, toast notifications, which are the most relevant from a digital forensic perspective, have a maximum lifespan of three days. We also noticed that all toast entries in the `Notification` table are deleted when the logged on user selects the *clear all notifications* option through `Action Center`. To circumvent the high turnaround of data, `NotifAnalyzer` which analyzes and extracts notifications from the SQLite3 `wpndatabase.db` databases—one per user account—resorts to various utilities to attempt to recover records. The fact that the `wpndatabase.db` databases have the *write ahead log* enabled also helps in the recovery of records. This way, we are able to extend the timespan of notification data, although, depending on the recovering method, recovered data can come with significant levels of noise.

Furthermore, the `wpndatabase.db` database has a configuration that does not impair recovery of records. Indeed, `Auto Vacuum` is set to `None`, and `Secure Delete` is not enabled, meaning that records are not wiped out when their states changes to *deleted*. Additionally, `wpndatabase.db` operates in Write Ahead Log mode (WAL), with auto checkpoint set to 1000, which is the default value of a SQLite3 database. All of this increases the probability of recovering a larger amount of records [20]. The main properties of the `wpndatabase.db` related to record recovery are listed in Table 10.

The four methods used by `WNA` to attempt to recover records are listed in Table 11. The first two methods act solely on the main file of the database, that is, `wpndatabase.db`,

while the third method—`bring2lite` [15]—carves both DB and WAL files. Finally, the last method—`WAL crawler`—parses the WAL file, that is, the file `wpndatabase.db-wal`, exploring the fact that the database is set in *write ahead* mode. We could not use `FQLite` as it fails to create output when ran under Windows due to the use of ":" in the file name, a character that cannot be used to name files in Windows.

Table 10. Main properties of `wpndatabase.db`.

Property	Value
Auto Vacumm	None
Journal Mode	WAL
Journal size Limit	−1 (unlimited)
Secure Delete	Off
WAL Auto Checkpoint	1000

Table 11. Data recovering methods used by the WNA module.

Method	Operates on
Delete parser [12]	DB file
Undark [13]	DB file
bring2Lite [15]	DB & WAL files
WAL crawler [21]	WAL file

As can be seen in Figure 11, the volume of data recovered by the used methods are important, especially by the DB-based `bring2lite DB Body`, as the recovery process yielded 3834 lines. However, this method is also the one with the lowest signal-to-noise ratio. Nonetheless, despite the noise, recovered data can still provide important information, such as valuable notifications. Additionally, as Autopsy indexes the content through the SOLR search platform [22], even mingled data can still be recovered by practitioners' searches.

The `WNA` module allows digital forensic practitioners to select which recovering methods, if any, they want to apply to the analysis of the WPN database(s). The selection of recovering method is done through configuration boxes shown right before the execution start of the module. These boxes are shown on the right side of Figure 12.

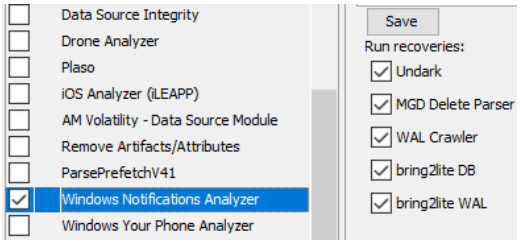

Figure 12. WNA's interface to select recovering methods within Autopsy.

5. Experimental Results

To assess the value of WPN in digital forensic contexts, we ran the WNA module over two Windows 10 forensic images. The first image comes from a machine with daily usage from a single user with Windows 10 Pro, version 1909 (10.0.18363.1139). The second image is a Windows 10 Enterprise, version 2004 (10.0.19041.1) with two created local accounts but with practically no usage, since the forensic image was collected within hours of the system installation, as the goal was to look for WPN in a practically fresh installation. Henceforth, we identify the 1909 image as `W10-1909`, while the other image is labeled as `W10-2004`.

For both the forensic images, the WNA module was ran within Autopsy, and the results were then analyzed. As stated earlier, WPN has one SQLite3 database per user account plus another system-wide database.

5.1. System's WPN Database

Table 12 displays WNA results when ran over the system's database of the analyzed forensic images. No notifications of any type—badge, tile or toast—were found in the system's `wpndatabase.db` files. Although the recovery procedure returns a large number of rows—7752 for `bring2Lite` in the `W10-2004` machine—none has real value for digital forensics. In fact, the rows hold one of three classes of elements [23]: (1) data from the tables; (2) SQL expressions used to build the database (create statements, etc.); (3) data related to channel URI and token needed by the application or service to push notification through the channel.

Table 12. Main results of WNA for system's wpndatabase of W10–1909 and W10–2004.

Name	W10–1909	W10–2004
Notification handlers	46	49
Notifications	0	0
badge	-	-
tile	-	-
toasts	-	-
Delete parsers (DB)	4	0
undark (DB)	3	0
WAL crawled	7343	7752
bring2lite (WAL)	7343	7752
bring2lite (DB)	1184	NA

Examples of the three classes are shown in Table 13. The last row contains two `Filetime64` timestamps: `132810120411300000` and `132810120441332000`, corresponding, respectively, to 2021-11-10T10:00:41+00:00 and 2021-11-10T10:00:44+00:00.

Table 13. Example of recovered content from System's `tabase.db`.

Type	Content
1	`1,c:storage:toast,0,`
2	`index,sqlite_autoindex_Metadata_1,Metadata,21,NULL,`
3	`3899 (...), 147, 132810120411300000, 132810120441332000,` ↪ `https://db5p.notify.windows.com/?token=AwYA, (...)`

The number of `notification handlers`, as reported in Table 12, is slightly larger for `W10-2004`: 49 vs. 46. Of these, 45 entries are common to both systems, while four are unique to `W10-2004` and another one solely exists in `W10-1909`. Interestingly, scanning the `NotificationHandler` table for added entries might be a fast method to detect new system services or at least services that can produce notifications, when comparing two versions of Windows 10. The differences detected in the `NotificationHandler` between the two forensic images are shown in Table 14. Note that the entry which is exclusive of *W*101909 is related to the Your Phone UWP application, which was in use in the system.

Table 14. Entries of the system's `NotificationHandler` table that only exists in one of the forensic image.

Image	Content
W10 1909	`Microsoft.WindowsPhoneCCPService_8wekyb3d8bbwe`
W10 2004	`Microsoft.Windows.InputSwitchToastHandler`
W10 2004	`Windows.SystemToast.BackupReminder`
W10 2004	`Windows.SystemToast.ServiceInitiatedHealing.Notification`
W10 2004	`Windows.SystemToast.StorSvc`

5.2. User's WPN Database

We now analyze the content of users' linked `wpndatabase.db` databases. The results are shown in Table 15, where the column `Valuable` reports how many of the items have user-related data and thus might have some forensic value. Most of the non-valuable items are simply data from the `HandlerSettings` table, such as `[48,u's:tile',1]`, while items classified as valuable are notification messages from applications. Valuable items collected by applying recovery methods available in WNA, shown in the last five rows of Table 15, are essentially toast messages. The absence of data from the `W10-2004` system is due to the low usage of the machine, as it only had a very short period of usage after the installation of W10-2004.

Table 15. Main results of WNA for user's `Wpndatabase.db` of W10-1909 and W10-2004.

	W10 1909		W10 2004	
Name	**Raw**	**Valuable**	**Raw**	**Valuable**
Notification handlers	184	NA	143	NA
Notifications (total)	82	59	14	0
Notifications (badge)	3	0	0	–
Notifications (tile)	21	1	13	1
Notifications (toasts)	58	58	1	0
Delete parsers (DB)	211	14	17	0
undark (DB)	134	4	11	0
WAL crawled	12	0	0	–
bring2lite (WAL)	415	16	0	–
bring2lite (DB)	4226	12	3572	8

The main factor that contributes for the low ratio `valuable/non-valuable` seen in Table 15 for the `W10-1909` system is the reduced number of messages that exists at any given time in the `wpndatabase.db` database. This is due to the limited lifetime of messages—maximum three days—and the reduced number of notifications, mainly toast messages, as excessive notifications can be a nuisance to users. All of this means that the `wpndatabase.db` never grows significantly, and when an entry is deleted, the freed space is used by another notification. This makes recovering data less efficient.

6. Other Sources Holding Notifications Data

In this section, we analyze two other potential sources for notifications forensic data: Windows Event Log, and Browsers.

6.1. Windows Log Events

Windows Log Events is the well-known log systems of Windows. It uses the `EVTX` log format [24] to record events in several files. All files of Windows Log Event are kept in the directory `c:\Windows\system32\winevt\logs`. In the Windows 10 systems that we analyzed, out of the more than 350 event log files, only two had the word *notification* in their names, as shown in Table 16. Moreover, only one of these two files—`Microsoft-Windows-PushNotification-Platform\%4Operational.evtx`—had data. In fact, in regularly used Windows 10 machines, this log file is often at its maximum default size, which is 1 MiB, a sign of its high usage. The other log file is `Microsoft-Windows-PushNotification-Platform\%4Admin.evtx`, but in all analyzed Windows 10 machines, this file was empty.

Table 16. Filenames of notification-related event log files.

Event Log Filename	**Content?**
`Microsoft-Windows-PushNotification-Platform\%4Admin.evtx`	Empty
`Microsoft-Windows-PushNotification-Platform\%4Operational.evtx`	Yes

Extraction of meaningful data from this log file is challenging, as recorded events come from different sources, data have wide diversity of formats. A much-trimmed example is given in Listing 2. It is a message from Microsoft's Your Phone application [2]. The given example has an explicitly named payload of 2886 bytes. This payload is a binary blob. When decoded, it reveals a text with carriage return/line feed (CRLF, i.e., `\r\n`) line separators, shown in Listing 3, with a header and a JSON-formatted content. The JSON content hosts another payload in the appropriately named field `innerPayload`. This inner payload is base64 encoded, and, when decoded, it reveals a mix of format. As can be seen in Listing 4, the content includes UTF-8 (e.g., `com.microsoft.phonecontent`), UTF-16 (e.g., `b.a.t.t.e.r.y`, recognizable due to the spaced letters when interpreted as ASCII) and binary fields (not shown). Note that the inner format of event messages of the notification is much dependent on the source application, requiring special decoding for each type of message/application. More importantly, logged messages have low forensic value as the content is mostly for logging purpose, with no user data.

Listing 2. Entry from the notification event log (edited).

```
<Events>
   <Event>
      <System>
         <Provider Name='Microsoft-Windows-PushNotifications-Platform'>
            (...)
            <EventID>1268</EventID>(...)
            <EventData>
               <Data Name='Verb'>NFY</Data>
               <Data Name='Bytes'>2886</Data>
               <Data Name='Payload'>4E4659203020574

            (...)
            </Data>
```

Listing 3. Decoded payload.

```
NFY 0 WNS\NOTIF.1044479.2886
Time: 2021-11-01T22:43:05Z
Channel: 1;2480582478643571433
Type: wns/raw
Expiry: 2021-11-01T22:43:36Z
Msg-Id: 2FFA92518E571E1
Mirroring: Allowed
TTL: 31
Ack: true
MS-CV: ZO43+QOf7KSkj7wk.0.2.2.1.1.1.1.1.1

{
   "cdpNotificationTypeId":1,
   "notificationTypeId":0,
   "commands":[
      {
         "requestId":"45(...)",
         "sourceUserDeviceThumbprint":"c+(...)=",
         "destinationUserDeviceThumbprint":"+p(...)E=",
         "commandTypeId":100,
         "commandTypeText":"CDP",
         "innerPayload":{
            "cdp":"AgE(...)ADQE="
         },
         "isPayloadHeldback":false,
         "timeToLive":"2021-11-01T22:43:36.6156757Z",
         "correlationId":"1(...)",
         "correlationVector":"ZO(...)"
      }
   ]
}
```

Listing 4. Decoded inner payload.

```
(...)
com.microsoft.phonecontent
(...)
b.a.t.t.e.r.y.C.h.a.r.g.i.n.g.....
b.a.t.t.e.r.y.P.e.r.c.e.n.t...
b.a.t.t.e.r.y.P.o.w.e.r.S.a.v.e
i.p.4.A.d.d.r.e.s.s..New
192.168.1.101.
i.p.4.S.u.b.n.e.t.M.a.s.k...
(...)
```

6.2. Browsers

Browsers can be another source of forensic data linked to Windows notifications. This happens when a web application running in the browser produces a notification and the user has previously authorized the application/browser to deliver toast notifications to Windows Action Center. Examples of applications able to deliver notifications include email clients such as `outlook.com` and `Gmail`, as well as calendar applications such as `Google Calendar`.

For browsers based on Chromium, such as Google Chrome, Edge, Brave and Vivaldi, these web applications generated notifications are kept in a LevelDB database, a key-value store engine [25], classified as an NO-SQL database. In LevelDB, keys and values have no format restrictions, as they are blobs of data.

A LevelDB database has several files, all kept in the same directory. The most relevant files are `xxxxxx.log` and `yyyyyy.ldb`, where `xxxxxx` and `yyyyyy` are hexadecimal numbers (e.g., `000003.log` and `000006.ldb`. These files contain the keys/values that is, data. First, the key/value pairs are written in a `.log` file. This goes on until the `.log` file reaches a given threshold, which is 4 MiB by default. When this occurs, the data held by the `.log` are transferred to a `.ldb` file, and a new `.log` is created. After a given number of `.log` conversion to `.ldb`, the `.ldb` file is consolidated, that is, keys are sorted and duplicate entries are eliminated. The comparison is performed by a user-given comparator [26].

Regarding data extraction, and considering that notifications have a substantial text part, data can be extracted from `.log` files through strings extractor utilities such as Microsoft Sysinternals' `strings`.

For Chromium-based browsers, the location of the correspondent notification directory is given in Table 17. Note that the variable `\%LocalAppData\%` corresponds to the user's local application data and maps to `C:\Users\USER\AppData\Local`, with `USER` corresponding to the login name of the user. An example of the content retrieved from a `.log` LevelDB file is shown in Listing 5.

Table 17. Location of Notification LevelDB directory for Chromium-based browsers.

Browser	Location
Brave	`\%LocalAppData\%\BraveSoftware\Brave-Browser\User Data\Default\Platform Notifications`
Chrome	`\%LocalAppData\%\Google\Chrome\User Data\Default\Platform Notifications`
Edge	`\%LocalAppData\%\Microsoft\Edge\User Data\Default\Platform Notifications`
Vivaldi	`\%LocalAppData\%\Vivaldi\User Data\Default\Platform Notifications`

Listing 5. Notification of a mail message extracted from a `.log` file.

```
SenderName"Patricio Domingues"
SenderEmail"patricio@[redacted]"
Subject"[Mail message] (subject)"
BodyPreview"This is a test message."
MessageId"AQ(...)"
NotificationType"NewMail"
```

7. Conclusions

Windows Push Notification is a Windows 10 service that delivers various types of notifications—badges, tiles and toasts—to applications and to logged on users. As notifications are kept in a per-user `wpndatabase.db` SQLite 3 database, although for a short timespan, it is possible to access the last notifications delivered to the respective user. Moreover, resorting to record recovering methodologies for SQLite 3 allows the recuperation of database records that might contain relevant artifacts for the case in examination.

The forensic value of notifications will depend on the installed applications and on the respective usage of these applications at the computer under analysis. For instance, a user that employs a messaging application such as Facebook Messenger UWP, and the user's notification database, might hold some relevant received messages that can help forensic practitioners.

The `NotifAnalyzer` and the associated `WNA` module for Autopsy parse all Windows 10s `wpndatabase.db` databases, showing within Autopsy data from the most relevant tables of the database—`Notification` and `NotificationHandler`. Additionally, by resorting to four different record recovery methods, the software allows for the recuperation of older records that can yield potentially valuable data for the forensic examination.

Another source of notification messages is Windows Event Logs, namely the `Microsoft-Windows-PushNotification-Platform\%4Operational.evtx` event log. However, the log contains mostly control messages, with few or no user data, and thus has minor forensic value. Additionally, messages are presented in hex format, and encoded in base64, with some of them having additional internal payload, also encoded. Encoding makes for a difficult indexation, hardening searches.

Web applications, such as email clients and calendars, running within browsers can also trigger notifications. For Chromium-based browsers, these notifications are kept in LevelDB and can be recovered. Depending on the web application, some data with forensic value can be retrieved.

Our analysis of the first release of Windows 11 Home edition reveals that WPN infrastructure remains the same, with the exception of the addition of the `TimeNotification` in the `wpndatabase.db` databases.

Due to the limited scope and timespan of the data, it is not expectable that Windows 10/11's notifications can yield a large volume of meaningful forensic artifacts. Nonetheless, in situations where other more traditional sources of data are not available or have failed to deliver useful data, notifications can still provide valuable artifacts.

As future work, we plan to add reporting to the module, using Autopsy's support, and keep accompanying the evolution of Windows 10/11's notification service to incorporate changes and evolution to the underlying databases.

Author Contributions: Conceptualization, P.D. and M.F.; methodology, P.D. and M.F.; software, L.A.; validation, P.D., M.F. and L.A.; formal analysis, P.D. and M.F.; investigation, P.D., M.F. and L.A.; resources, P.D. and L.A.; data curation, P.D. and M.F.; writing—original draft preparation, P.D.; writing—review and editing, M.F. and L.A.; visualization, M.F.; supervision, P.D. and M.F.; project administration, and funding acquisition, M.F. All authors have read and agreed to the published version of the manuscript.

Funding: This work was partially supported by CIIC under the FCT/MCTES project UIDB/CEC/-4524/2020, and EU funds under the project UIDB/EEA/50008/2020.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hintea, D.; Bird, R.; Green, M. An investigation into the forensic implications of the Windows 10 operating system: recoverable artefacts and significant changes from Windows 8.1. *Int. J. Electron. Secur. Digit. Forensics* **2017**, *9*, 326–345. [CrossRef]
2. Domingues, P.; Andrade, L.M.; Frade, M. Microsoft's Your Phone environment from a digital forensic perspective. *Forensic Sci. Int. Digit. Investig.* **2021**, *38*, 301177. [CrossRef]
3. Rui, H.; ZhiGang, J.; BaoLiang, W. Comparison of Windows Phone 8 and Windows 8. In Proceedings of the 2013 6th International Conference on Intelligent Networks and Intelligent Systems (ICINIS), Shenyang, China, 1–3 November 2013; pp. 55–57. [CrossRef]
4. Microsoft. Windows Developer—Toast Content. 2021. Available online: https://docs.microsoft.com/en-us/windows/uwp/design/shell/tiles-and-notifications/adaptive-interactive-toasts/ (accessed on 27 November 2021).
5. Conlan, K.; Baggili, I.; Breitinger, F. Anti-forensics: Furthering digital forensic science through a new extended, granular taxonomy. *Digit. Investig.* **2016**, *18*, S66–S75. [CrossRef]
6. AlHarbi, R.; AlZahrani, A.; Bhat, W.A. Forensic analysis of anti-forensic file-wiping tools on Windows. *J. Forensic Sci.* **2021**, *66*. [CrossRef] [PubMed]
7. Skulkin, O.; de Courcier, S. *Windows Forensics Cookbook*; Packt Publishing: Birmingham, UK, 2017.
8. Khatri, Y. Parsing the Windows 10 Notification Database. 2016. Available online: http://www.swiftforensics.com/2016/06/parsing-windows-10-notification-database.html (accessed on 1 Decemebr 2021).
9. Maloney, B. Windows 10 Notification WAL Database. 2018. Available online: https://malwaremaloney.blogspot.com/2018/08/windows-10-notification-wal-database.html (accessed on 1 December 2021).
10. Bilogrevic, I.; Engedy, B.; Porter, J.L., III; Taft, N.; Hasanbega, K.; Paseltiner, A.; Lee, H.K.; Jung, E.; Watkins, M.; McLachlan, P.; et al. "Shhh...be quiet!" Reducing the Unwanted Interruptions of Notification Permission Prompts on Chrome. In Proceedings of the 30th USENIX Security Symposium (USENIX Security 21), Virtual, 11–13 August 2021; pp. 769–784.
11. Chopade, R.; Pachghare, V.K. Ten years of critical review on database forensics research. *Digit. Investig.* **2019**, *29*, 180–197. [CrossRef]
12. DeGrazia, M. SQLite-Deleted-Records-Parser: Recovering Deleted Entries in SQLite Database. 2015. Available online: https://github.com/mdegrazia/SQLite-Deleted-Records-Parser (accessed on 8 November 2021).
13. Daniels, P.L. Undark—A SQLite Deleted and Corrupted Data Recovery Tool. 2021. Available online: http://pldaniels.com/undark/ (accessed on 8 November 2021).
14. Miller, P.; Bryce, C. *Learning Python for Forensics*, 2nd ed.; Packt Publishing: Birmingham, UK 2019.
15. Meng, C.; Baier, H. bring2lite: A Structural Concept and Tool for Forensic Data Analysis and Recovery of Deleted SQLite Records. *Digit. Investig.* **2019**, *29*, S31–S41. [CrossRef]
16. Pawlaszczyk, D.; Hummert, C. Making the Invisible Visible–Techniques for Recovering Deleted SQLite Data Records. *Int. J. Cyber Forensics Adv. Threat Investig.* **2021**, *1*, 27–41. [CrossRef]
17. Barr-Smith, F.; Farrant, T.; Leonard-Lagarde, B.; Rigby, D.; Rigby, S.; Sibley-Calder, F. Dead Man's Switch: Forensic Autopsy of the Nintendo Switch. *Forensic Sci. Int. Digit. Investig.* **2021**, *36*, 301110. [CrossRef]
18. Wu, T.; Breitinger, F.; O'Shaughnessy, S. Digital forensic tools: Recent advances and enhancing the status quo. *Forensic Sci. Int. Digit. Investig.* **2020**, *34*, 300999. [CrossRef]
19. Garfinkel, S.L. Automating disk forensic processing with SleuthKit, XML and Python. In Proceedings of the 2009 Fourth International IEEE Workshop on Systematic Approaches to Digital Forensic Engineering, Berkeley, CA, USA, 21 May 2009; pp. 73–84.
20. Liu, Y.; Xu, M.; Xu, J.; Zheng, N.; Lin, X. SQLite forensic analysis based on WAL. In *International Conference on Security and Privacy in Communication Systems*; Springer: Berlin/Heidelberg, Germany, 2016; pp. 557–574.
21. Miller, P.; Bryce, C. *Learning Python for Forensics: Leverage the Power of Python in Forensic Investigations*; Packt Publishing: Birmingham, UK, 2019.
22. Shahi, D. *Apache Solr*; Apress: New York, USA, 2015. [CrossRef]
23. Microsoft. Windows Push Notification Services (WNS) Rview. 2021. Available online: https://docs.microsoft.com/en-us/windows/uwp/design/shell/tiles-and-notifications/windows-push-notification-services--wns--overview/ (accessed on 1 December 2021).
24. Studiawan, H.; Sohel, F.; Payne, C. A survey on forensic investigation of operating system logs. *Digit. Investig.* **2019**, *29*, 1–20. [CrossRef]
25. Dent, A. *Getting Started with LevelDB*; Packt Publishing: Birmingham, UK, 2013.
26. Focus, F. After SQLite, What Next? A Must-Read Primer on LevelDB. 2020. Available online: https://www.forensicfocus.com/articles/after-sqlite-what-next-a-must-read-primer-on-leveldb/ (accessed on 13 November 2021).