Digital Forensic Artifacts of the Cortana Device Search Cache on Windows 10 Desktop

Patricio Domingues
School of Technology and Management
Polytechnic Institute of Leiria
Instituto de Telecomunicações
Portugal
Email: patricio.domingues@ipleiria.pt

Miguel Frade
School of Technology and Management
Polytechnic Institute of Leiria
Computer Science and Communication Research Centre (CIIC)
Portugal
Email: miguel.frade@ipleiria.pt

***Abstract*—Microsoft Windows 10 Desktop edition has brought some new features and updated other ones that are of special interest to digital forensics analysis. The search box available on the taskbar, next to the Windows start button is one of these novelties. Although the primary usage of this search box is to act as an interface to the intelligent personal digital assistant Cortana, in this paper, we study the digital forensic artifacts of the search box on machines when Cortana is explicitly disabled. Specifically, we locate, characterize and analyze the content and dynamics of the JSON-based files that are periodically generated by the Cortana *device search cache* system. Forensically important data from these JSON files include the number of times each installed application has been run, the date of the last execution and the content of the custom jump list of the applications. Since these data are collected per user and saved in a resilient text format, they can help in digital forensics, mostly in assisting the validation of other sources of information.**

***Keywords*-Windows 10, digital forensics, Cortana, device search cache, JSON**

I. Introduction

Microsoft Windows 10 was released at the end of July 2015 and has since accumulated a meaningful base of users, close to 20% of the whole desktop OS market at the time of this writing [1]. Windows 10 has already surpassed the number of users of Windows 8 [2]. The quick adoption is in part due to the persistent upgrade policy that allows owners of Windows 7 and 8.x to upgrade, free of direct costs, to Windows 10. Moreover, upgrades are facilitated and sometimes pushed through the *GWX* (Get Windows X) tool. In fact, to counteract the insistence of GWX to upgrade to Windows 10, regarded as annoyance by some users, some third parties have developed tools – Never 10 [3], GWX Control Panel [4] – to allow the user to control and block the upgrade to Windows 10. Nonetheless, the rapid adoption rate of Windows 10 means that it is poised to become a major OS for desktop systems in the foreseeable future, and therefore an important OS for digital forensics that cannot be ignored by digital forensic investigators and researchers. As it has occurred in previous major releases, the Windows 10 Desktop edition brings some novel features, such as the new Edge browser, the Cortana intelligent personal digital assistant and the message notification functionality. Other useful elements for digital forensics like Jump lists [5] and the System Resource Usage Monitor (SRUM) were changed [6].

Intelligent personal digital assistants have invaded computing devices, mostly mobile ones, in recent years benefiting from advances in speech recognition and artificial intelligence [7]. Intelligent personal digital assistants have been used, among other things, to place a call, select music, send text messages or get driving directions. The most interesting features of these assistants is their context awareness and the ability to interact with humans through voice. The main digital assistants are Apple's Siri[1], Google Now[2] and Microsoft's Cortana[3]. The context awareness, namely the localization of the pair device/user is particularly suited for mobile devices, since these devices, especially smartphones, are practically always on and kept close to their users. The ability to interact through oral communication is key for the success of these virtual assistants on mobile devices since it circumvents the limitations of keyboard-based input and allows for a hands-free approach.

Although intelligent personal digital assistants are geared towards mobile devices, they are available for desktop systems. This is the case for Cortana that is included in the Desktop edition of Windows 10. Despite Cortana being enabled by default when Windows 10 is installed, the fact that Cortana scans emails and calendars among other sensitive items, scares privacy-oriented users, resulting in some deactivation of Cortana on the desktop OS. More importantly, support for Cortana is still restricted to a few regions and languages. For example, despite Cortana being supported in the English version of Windows 10 Desktop, the digital assistant features are only available in Australia,

[1] http://www.apple.com/ios/siri
[2] http://www.google.com/landing/now/
[3] https://www.microsoft.com/en/mobile/experiences/cortana/

978-1-5090-0990-9/16 $31.00 © 2016 IEEE
DOI 10.1109/ARES.2016.44

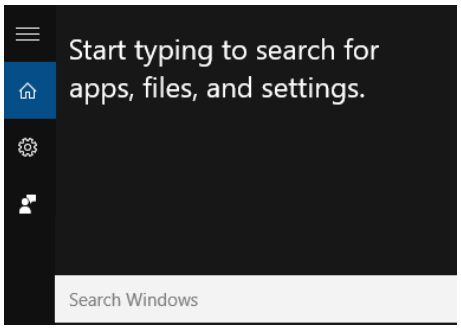

Figure 1. Cortana Search Box in Windows 10

Canada, India, UK and USA. This way, users of the English version of Windows 10 Desktop located outside of these countries have no access to Cortana.

In this paper, we explore the digital forensic potential of Cortana when the intelligent personal digital assistant is either not installed or explicitly deactivated. Under this configuration, Cortana only provides the *search bar* that is available on the task bar, next to the start button, and through the *Windows key* + *q* keyboard shortcut. We show that under this minimal configuration, Windows 10 Desktop periodically collects data regarding the usage of all installed applications dumping these data to JSON-formatted files. These data are collected regardless on how the application is launched. Indeed, starting the application from the start menu, from a shortcut or directly by clicking on the executable yield collected data. Although data are limited to one JSON record per application with a few fields, the fact that the data are associated to a login, updated frequently and carries a timestamp can be useful to confirm and enhance data acquired from other forensic sources.

The main motivation for this study is to analyze the potential for digital forensics of the Cortana-based device cache search mechanism, taking advantage of this novel mechanism of Windows 10 desktop. Although the persistence of the data collected by Cortana's device cache search is reduced since the data are periodically replaced by newer ones, it is nonetheless another possible source of data that can be used to harvest meaningful digital forensic evidences. To the best of our knowledge, no previous work exists related to the device search cache of search box and its possible use for digital forensic mining. This is no surprise as Windows 10 was only released at the end of July 2015.

This paper is organized as follows. Section II reviews related work, while Section III presents the experimental setup. Section IV presents the Cortana organization, i.e., the directory layout and the files that support the Cortana application, as well as the JSON records and data that are saved to the device search cache. Section V analyzes the dynamics of file creation and deletion in the device search cache main directory. Finally, section VI concludes the paper and discusses possible future work.

II. Related Work

We are not aware of any previous work regarding the usage of the Cortana device search cache for digital forensics. On an online presentation [8] that summarizes the main novelties of Windows 10 Desktop for digital forensics, Brian Muir briefly describes the two databases associated to Cortana: *IndexedDB.edb* and *CortanaCoreDb.dat*. Both databases use the Extensible Storage Engine (ESE) to store and manage data. He mentions that *CortanaCoreDb.dat* contains a rich set of data (locations, messages, contacts), but only when Cortana is i) enabled and ii) the language/country pair of the OS is one that is supported by the intelligent personal assistant software. Our work focuses on digital forensic artifacts solely from Cortana device search cache on systems where there is, simultaneously, i) no support for the digital intelligent personal assistant features and ii) Cortana is explicitly deactivated through the system settings.

Chivers and Hargreaves studied the Windows Search service in Windows 7 from the digital forensic perspective [9]. As the authors state, the Windows Search service indexes files, emails, programs and internet browsing history of all the users of a personal computer [9]. The authors focus mostly on the possibility of carving data from the deleted records of the database, for digital forensic purposes. The Windows Search service also resorts to the ESE database engine, storing data and metadata in the *Windows.edb* file and associated auxiliary files (log files, etc.). When available, the Windows Search service holds a wealth of data. However, the service is sometimes deactivated by the users due to nuisances it might cause such as spike on CPU and disk usage, as well as the large amount of storage that it might use, and for SSD drives, the additional wear that it can provoke to the drive. Additionally, since ESE is a rather complex binary format, recovery of data from partial chunk of data can be rather difficult or even impossible. For example, Chivers reports, that, for the ESE database used by the private browsing mode of Internet Explorer 10, 40% of the pull-the-plug seizure experiments resulted in an unrecoverable database [10]. Thus, while Cortana device search cache covers much fewer files than the Windows Search service, its usage of JSON-based files enhances the probability of carving usable data, even when only a portion of the whole content is available.

Khatri studies the forensic implication of the System Resource Usage Monitor (SRUM) [6]. SRUM is a resource usage monitoring system that was made available for the first time in Windows 8. SRUM periodically collects metrics related to the usage of certain resources of the system per process. Metrics include full process I/O data (bytes read and written), CPU cycle utilization, network utilization (data uploaded/downloaded) among other data. Some of these data are available to the user under the *AppHistory* tab

in Windows's task manager. The data for SRUM are first collected to the SOFTWARE registry hive, and then are, on an hourly basis, written to an ESE database named *SRUDB.dat*. The data are kept for a minimum of 30 days, with some metrics such as network usage being retained for a period as long as 90 days [6]. From the perspective of digital forensics, SRUM data can be used, for instance, to associate to a given process and user security ID (SID), situations of data exfiltration, where a high volume of data is uploaded, or to pinpoint the usage of an anti-forensic tools such as the well-known *CCleaner* [11].

Singh et al. analyze jump lists in Windows 10. The authors report on the modifications of jump lists, namely the file format used by jump lists in Windows 10 [5]. They also study the resilience of the data provided by jump lists to anti-forensic tools and tactics. The *Jump lists* feature appeared in Windows 7. For a regular user, some of the functionalities of jump lists are directly available via the right mouse menu by clicking on the icon of an application that supports jump lists [12], [5]. The services available through jump lists can range from the list of most recently accessed files, the list of most frequently accessed files and quick access to tasks, such as opening a new tab or accessing a given service. From the digital forensics perspective, the value of jump list lies within the plethora of data recorded in jump lists files, which can be mined with special digital forensics tools [13]. There are two types of jump lists: *automatic* and *custom* jump lists. As the name implies, automatic jump lists are created and maintained by the operating system, while custom jump lists need to be explicitly created and maintained by the applications that aim to support the jump list service. Besides Microsoft applications, few others support custom jump list. Examples of applications that support custom jump lists include the VLC multimedia application [4], as well as some browsers, such as Google Chrome and Firefox. Note that an application can create a simple custom jump list by resorting to the `SHAddToRecentDocs` function from the Windows API to open a file [14].

Since jump lists are kept by the OS on a user basis – jump lists files of a user are kept in a directory under the user's profile – they can pinpoint the digital activity of a given user account. The importance of jump lists within the context of this paper, is that, as we shall see later on, custom jump lists is one of the artifacts recorded in the device search cache files.

The so-called *UserAssist* is a key that has existed in the registry [15] since Windows XP. The key – `HCU\Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist` – holds some subkeys that contains statistical data regarding the executions of applications in the local system. Data for an application are recorded as long as the application is launched from Windows Explorer, either directly from the executable or through a shortcut. This is also the case for *portable applications*. Data recorded under the *UserAssist* key include the number of executions and a 64-bit timestamp with the date/time of the last launch of the application. The NirSoft application *UserAssistView* [16] is one of the tools that displays the data in a human readable format. The data available under the *UserAssist* key are significantly less detailed than the ones provided by Cortana's device search cache. Additionally, some anti-forensic tools erase the data kept by the *UserAssist* key. This is the case for the CCleaner [11] application. Nonetheless, as usual in digital forensics, the data from the *UserAssist* key, when available, can be of important value, especially when they contain data regarding the usage of (non-installed) portable applications.

Collie studies the digital forensics data that can be extracted from the *IconCache* database file [17]. The IconCache database is a Windows maintained file that caches the icons of applications. It exists since Windows XP. The IconCache database can be successfully used to detect the presence of executable files within USB connected devices that were used on the analyzed computer [17], [18]. However, since Windows populates the IconCache database with the icons of executable files, whether or not they are invoked, care needs to be taken in the interpretation of results [17]. The Cortana device search cache also deals with icons, since it creates, in a directory under the user's profile, a copy of all the icons of the registered applications that exist in the system. Moreover, the clone icon file receives the name of the application ID. However, these files are erased when the corresponding application or shortcut is removed from the system, and thus have limited usefulness for digital forensics.

III. Experimental Setup

The observations and results presented in this paper were collected from a Windows 10 Desktop, 64-bit English version 1511, build 10586.128, labeled as *Windows 10 Education*. The OS was freshly installed on a 240 GB SSD drive, with a i5-2410M @2.30 GHz CPU and 4 GB of RAM. The OS was set for Portuguese (Portugal) under the *Region and Country* configuration, with the Locale ID 2070, corresponding to *pt-PT* regional settings.

The OS was installed with the Cortana assistant application explicitly deactivated. Nonetheless, as we show in this paper, even when not enabled, Windows 10 still has an active Cortana module – *search box* and the associated *device cache search* – that collects data with forensic values on a per user basis, data which are periodically dumped to JSON-based files that are kept under a set of directories, as we shall see next.

[4] http://www.videolan.org

IV. Cortana Organization

There are two main types of content related to Cortana: i) binary-based which comprises the executable and DLL files that implements Cortana's functionality; ii) the configuration and collected data files. Next, we separately describe each type.

A. Executables and DLLs

The executable files and most of the DLLs that support Cortana are located in the directory `Microsoft.Windows.Cortana_cw5n1h2txyewy\` which is itself a subdirectory of the system folder `C:\Windows\SystemApps\`. For the sake of simplicity, we assume that system drive, given by the environment variable `%SystemDrive%`, corresponds to `C:`, as it is usually the case. Note that `cw5n1h2txyewy` is a hash that represents the PublisherID of Microsoft Windows, that is, *CN=Microsoft Windows, O=Microsoft Corporation, L=Redmond, S=Washington, C=US*. Other Cortana's related DLLs exist in the `C:\Windows\System32\` directory. Examples include *windows.cortana.pal.desktop.dll* and *Cortana.Persona.dll*. The search box functionality of Cortana is provided by one of the executable that exists in the `Microsoft.Windows.Cortana_cw5n1h2txyewy` directory: *SearchUI.exe*. The *SearchUI.exe* application is run with the argument `-ServerName:CortanaUI.AppXa50dqqa5gqv4a 428c9y1jjw7m3btvepj.mca`.

For digital forensics, an interesting property of the Cortana device search cache lies in its persistence. Indeed, it runs even when Cortana has been configured to be off. Additionally, attempts to terminate its supporting process – SearchUI.exe – are void, since it is quickly respawn afterwards. Informal evidences show that a working approach to disable the Cortana device search cache is to i) to take ownership of the folder `Microsoft.Windows.Cortana_cw5n1h2txyewy\`; ii) terminate the SearchUI.exe process; and then iii) quickly rename the folder whose ownership was claimed in step i). Note that this approach also disables the search box and thus hinders the usability of the system.

B. Configuration and Collected Data

For the considered setup, the data collected by Cortana's device search cache are stored under the `C:\Users\USER\AppData\Local\Packages\ Microsoft.Windows.Cortana_cw5n1h2txyewy\` directory, where `USER` corresponds to the login name of the user.

The data of interest to this study are located under the subdirectory `LocalState`. This directory `LocalState` hosts two subdirectories: `DeviceSearchCache` and `AppIconCache`.

1) The AppIconCache directory: As the name suggests, the `AppIconCache` directory stores a copy of the icons of the applications that are installed on the system, as well as of the shortcuts that are also created by the installation of the applications. For example, for the Dropbox application[5], two icons are created: one for the Dropbox application itself and another one that corresponds to the link http://www.dropbox.com, which is created in the start menu. Note that the icon for the web link is the one of the web browser defined as default by the user.

Each icon file is named with the *AppID* and corresponds to a 32×32 BMP color format. For the studied OS release, the files of the icons had the names `{5DD50ED5-10A3-4F90-B736-77185DBDEE12}` for the Dropbox application and `{663E949C-C7E7-4E25-BFDA-126AD8FDA47C}` for its URL link. The icon of the link was the icon of Google Chrome (default browser of the experimental machine), although under a different file name: Google Chrome icon's file name was `{95ac0abf-fd09-4624-a05d-34181627b2a1}`.

The icon files are of limited forensic values. Although the presence of an icon could be interpreted as proof of the existence of the application, the fact is that other applications could have the same icon. Regarding the various timestamps of the icon files, the date/time of last access corresponds to the last refresh done by the system to Cortana-based data, while the file creation date/time corresponds to the installation date of the application. Since an icon file is removed when the application itself is uninstalled, usage of icon files to detect a once installed application requires the recovery of deleted files and thus might not be feasible, yielding no clear advantages over other methods.

2) The DeviceSearchCache directory: The directory `DeviceSearchCache` contains the periodically created JSON-formatted file that holds the data regarding applications and associated shortcuts installed on the system. Periodically data are dumped to a text file named *AppCacheTIMESTAMP.txt*, where *TIMESTAMP* represents the value of the current timestamp expressed in Microsoft's 64-bit FILETIME format [19]. For instance, *AppCache131053178802735320.txt* corresponds to the file saved on Sat, 16 Apr 2016 22:04:40 UTC.

Each dump file contains a set of JSON records. Specifically, there is one JSON record per installed application, as well as for any shortcut existing on the *Windows Start Menu*, and in other directories, like the *Desktop* directory of the currently logged user. For example, the *inkscape* application, whose installer adds a link to the project homepage [6] in the start menu has two JSON entries on an *AppCacheTIMESTAMP.txt* file: one for the application and another one for

[5]http://www.dropbox.com
[6]http://www.inkscape.org

the homepage of the inkscape project.

The set of applications whose data is periodically dumped to a JSON-based *AppCacheTIMESTAMP.txt* file matches the list that can be accessed through the Microsoft's powershell commandlet *Get-StartApps* [20]. This command returns a list with the names and AppIDs of all the installed applications accessible to the current user [21]. The number of applications can be obtained by issuing the pipped command *Get-StartApps | measure* on powershell.

The data dumped to an *AppCacheTIMESTAMP.txt* file are far more complete than the output of the *Get-StartApps* command: every JSON record describing an application/shortcut is comprised of 17 fields. Table I shows the name of the fields, as well as their respective content, for the record of the *notepad* application collected at the test machine. Due to space limitation, the fields *ParsingName*, *AppID* and *EncodeTargetPath* are truncated. As stated earlier, the *DateAccessed* field represents the date/time, in Microsoft's FILETIME format, of the last access to the application. For example, the value 131053065494310000 corresponds to the date *Sat, 16 Apr 2016 18:55:49 +0000* UTC time. The *ParsingName* field corresponds to the application's *Application User Model ID* (AppUserModelID). If the application does not explicitly set its own AppUserModelID, the OS generates one, and the *ParsingName* field is set to *microsoft.autogenerated.CODE*, where *CODE* is the OS generated code.

Due to space restriction, the entry for the *Connected-Search.JumpList* is not shown in Table I. We briefly present the fields of the JumpList entry in Table II. This entry corresponds to another set of JSON records that captures the custom *jump list* entries for the application. In the particular case of the *notepad* application, the jump list entry holds, for each of the last opened files, the name, full path, date of access and a numeric field called *Points*. The *Points* field is an integer value for the entries related to the *notepad* application that appears linked to the number of times the file has been opened by the application. However, for other applications like *Microsoft Winword*, the *Points* field is a floating-point value like, for example, 4.8035187721252441. We ignore the meaning of a floating-point *Points* field.

A jump list entry corresponds to the application custom jump list. For instance, the jump list entry for the *Google Chrome* application has three further categories: *most visited*, *recently closed* and *tasks*. This corresponds to the jump list entries that are available when the jump list menu is accessed through the right mouse button on the google chrome icon on the task bar. Of the 178 entries of the JSON data file from the test machine, only 18 of them had a non-empty *JumpList* field. Some of these entries are Microsoft utilities such as Notepad, Paint, the Edge Browser, as well as Microsoft Office applications. Others are non-Microsoft applications, including the Google Chrome browser, the SumatraPDF PDF viewer and Mozilla Firefox.

Name	Type	Content
FileExtension	12	".exe"
ProductVersion	12	"N/A"
Kind	12	"program"
ParsingName	12	{1AC14E77-02E7-...}/notepad.exe
IsFlagged	5	1
TimesUsed	5	110
Tile.Background	5	16777215
AppId	16	{30CCF6CE-4BD9-4B88-...}
Arguments	12	"N/A"
Identity	12	"N/A"
Filename	12	"notepad"
JumpList	12	*JSON record*
ItemType	12	"Desktop"
DateAccessed	14	131053065494310000
EncodedTargetPath	12	{1AC14E77-02E7-...}/notepad.exe
SmallLogoPath	12	"N/A"
ItemNameDisplay	12	"Notepad"

Table I
JSON RECORD FOR NOTEPAD

Type	Type of entry (all observed entries had the value 1)
Name	Name of the file
Path	Full path of the file
Date	Date of the last access to the file (just the date, no time)
Points	Number (integer or floating-point)

Table II
FIELDS OF THE *JumpList* ENTRY.

Besides the *JumpList* entry, other interesting fields present in a JSON record of a device search dump file are *TimesUsed*, *DateAccessed* and *Arguments*. We briefly describe each of them.

- *TimeUsed*: records the number of times that the application/link was used by the studied user.
- *DateAccessed*: records, under a Microsoft's FILETIME format, the last UTC date/time that the application pointed out by the record was accessed.
- *Arguments*: string that holds the arguments used to launch the application. Note that these arguments are fixed, that is, they are the arguments defined on the shortcut used to launches the application. For example, this is the case for the *Cygwin64 terminal*, which is a shortcut created by the installation of the cygwin environment, and which defines the string *-i /Cygwin-Terminal.ico* as arguments. These arguments are recorded on the corresponding entry for *cygwin*.

3) The SettingsCache.txt file: The directory `DeviceSearchCache` hosts another JSON-formatted file: *SettingsCache.txt*. This file holds JSON records that appear related to applications which can be used to interact with the settings of the system. This is the case, for example, of the JSON record shown in Figure 2, which is associated to the control settings of the *auto play* mechanism. From the forensic point of view, this file appears to have no value,

```
{
  "System.ParsingName": {
    "Type": 12,
    "Value": "AAA_SystemSettings_(...)
  "System.Setting.PageID": {
    "Type": 12,
    "Value": "SettingsPagePC(...)" },
  "System.Setting.SettingID": {
    "Type": 12,
    "Value": "SystemSettings_(...)" },
  "System.Setting.GroupID": {
    "Type": 12,
    "Value": "SettingsGroupAutoplay" },
  "System.Comment": {
    "Type": 12,
    "Value": "Turn AutoPlay on or off" }
},
    (...)
```


Figure 2. A JSON record of the SettingsCache.txt file

since it does not store any user-defined configurations, nor any date/time values.

V. Creation and Deletion of Files in the Device Search Cache

Several events can trigger the creation of a new *AppCacheTIMESTAMP.txt* file in Cortana's device search cache directory. The events are: i) periodicity; ii) install/uninstall of an application; iii) add/remove of a shortcut on the desktop; iv) reboot/resume of the computer. We now describe each one of the triggering events.

Periodically, a new *AppCacheTIMESTAMP.txt* file is added to the device search cache directory. On our experimental setup, this occurred roughly every 15 minutes. Note that the periodic file creation occurs even if nothing system-wise has changed. In this case, the newly created file has the same content than the previous one(s).

Additionally, whenever an application is installed or, on the contrary, uninstalled, a new *AppCacheTIMESTAMP.txt* file is created, regardless of the time elapsed since the last creation of an *AppCacheTIMESTAMP.txt* file. The same occurs when a shortcut is added or removed from the user's desktop: this event triggers the creation of a new *AppCacheTIMESTAMP.txt* file. Finally, when a user logs in the machine after a reboot or a resume operation, a new *AppCacheTIMESTAMP.txt* file is created in the device search cache directory.

A reboot/restart operation triggers the removal of all the files of the device search cache directory, except for the *SettingsCache.txt* files. Besides the removal on reboot/restart, the device search cache directory is also cleaned periodically, with all *AppCacheTIMESTAMP.txt* files being deleted. In our experimental setting, the periodical cleaning occurred roughly every 8 hours, around 0h30, 8h30 and 16h30.

Regarding anti-forensics tools, our tests with the well-known CCleaner application [11] showed that the *AppCacheTIMESTAMP.txt* files are not deleted when CCleaner is run. We used version 5.17.5590 of CCleaner. In fact, the installation of CCleaner just triggered the dump of an up to date *AppCacheTIMESTAMP.txt* JSON file. Note however, that while the last accessed time and the number of times remains available after the execution of CCleaner, jump list data are lost. This is a consequence of CCleaner erasing the jump lists. This way, the *AppCacheTIMESTAMP.txt* files that are created following the execution of CCleaner no longer have the pre-CCleaner jump list data.

VI. Conclusion and Future Work

This paper reported on the digital forensics value of Windows 10 Desktop's device search cache module. The device search cache is linked to the search box that exists on the taskbar, next to the start menu button. It periodically dumps, on a JSON-based format, a text file that contains some metrics regarding the usage of all applications that are installed on the local system. Besides the periodic dumps, *AppCacheTIMESTAMP.txt* files are created whenever an application is installed/uninstalled, or when a shortcut is created/deleted on the user's desktop.

The resilience of the device search cache is a plus for digital forensics. Indeed, disabling the device search cache requires a non-trivial intervention on the directory that hosts the *AppCacheTIMESTAMP.txt*. Additionally, the periodic creation of the JSON-based files augments the probability of being able to recover data through carving techniques, while resorting to a JSON text-based format means that partial data might still be useful.

Despite the limitations – data are exclusively from installed applications, the set of collected metrics is limited and the files are periodically deleted –, we believe that the device search cache can be useful to complement and validate other digital forensics sources.

As future work, we plan to study the device search mechanism on other version of Windows 10 Desktop and to assess the resilience of the service to other system cleaner and anti-forensics applications and techniques.

Acknowledgments

Financial support was partially provided in the scope of R&D Unit 50008, financed by the applicable financial framework (FCT/MEC through national funds and when applicable co-funded by FEDER - PT2020 partnership agreement). The authors would also like to thank the reviewers for their insightful comments that led to an improvement of this paper and also for pointing interesting directions for future work.

REFERENCES

[1] StatCounter, "StatCounter at GlobalStats: Operating Systems," *http://gs.statcounter.com/#os-US-monthly-201505-201604*, April 2016.

[2] W3schools.com, "Web statistics: OS platform statistics," *http://www.w3schools.com/browsers/browsers_os.asp*, April 2016.

[3] S. Gibson, "GRC - Never 10 application," *https://www.grc.com/never10.htm*, April 2016.

[4] "GWX Control Panel," *http://blog.ultimateoutsider.com/*, April 2016.

[5] B. Singh and U. Singh, "A forensic insight into Windows 10 Jump Lists," *Digital Investigation*, vol. 17, pp. 1–13, 2016.

[6] Y. Khatri, "Forensic implications of System Resource Usage Monitor (SRUM) data in Windows 8," *Digital Investigation*, vol. 12, pp. 53–65, 2015.

[7] K. M. Alhawiti, "Advances in artificial intelligence using speech recognition," *World Academy of Science, Engineering and Technology, International Journal of Computer, Electrical, Automation, Control and Information Engineering*, vol. 9, no. 6, pp. 1351–1354, 2015.

[8] B. Muir, "Windows 10 Forensics: OS Evidentiary Artefacts," *http://www.slideshare.net/bsmuir/windows-10-forensics-os-evidentiary-artefacts*, July 2015.

[9] H. Chivers and C. Hargreaves, "Forensic data recovery from the windows search database," *Digital Investigation*, vol. 7, no. 3, pp. 114–126, 2011.

[10] H. Chivers, "Private browsing: A window of forensic opportunity," *Digital Investigation*, vol. 11, no. 1, pp. 20–29, 2014.

[11] "Piriform Ltd - The CCleaner system cleaner," *http://www.piriform.com/ccleaner*, 2016.

[12] A. G. Barnett, "The Forensic Value of the Windows 7 Jump List," in *Digital Forensics and Cyber Crime*. Springer, 2011, pp. 197–210.

[13] G. S. Smith, "Using jump lists to identify fraudulent documents," *Digital Investigation*, vol. 9, no. 3, pp. 193–199, 2013.

[14] Microsoft, "SHAddToRecentDocs function / Win32 API," *https://msdn.microsoft.com/en-us/library/aa932918.aspx*, 2016.

[15] D. J. Farmer and V. Burlington, "A forensic analysis of the windows registry," *Champlain College Burlington, Vermont*, 2007.

[16] "NirSoft - UserAssistView browser for UserAssist Key," *http://www.nirsoft.net/utils/userassist_view.html*, 2016.

[17] J. Collie, "The windows IconCache.db: A resource for forensic artifacts from USB connectable devices," *Digital Investigation*, vol. 9, no. 3, pp. 200–210, 2013.

[18] C.-Y. Lee and S. Lee, "Structure and application of iconcache. db files for digital forensics," *Digital Investigation*, vol. 11, no. 2, pp. 102–110, 2014.

[19] C. Boyd and P. Forster, "Time and date issues in forensic computinga case study," *Digital Investigation*, vol. 1, no. 1, pp. 18–23, 2004.

[20] E. Wilson, *Windows PowerShell Step by Step*. Microsoft Press, 2015.

[21] Microsoft, "Get-StartApps commandlet / powershell," *https://technet.microsoft.com/en-us/library/mt188240.aspx*, 2016.