Timestamp hiccups: Detecting manipulated filesystem timestamps on NTFS

Sebastian Neuner
SBA Research
Vienna, Austria
sneuner@sba-research.org

Artemios G. Voyiatzis
SBA Research
Vienna, Austria
avoyiatzis@sba-research.org

Martin Schmiedecker
SBA Research
Vienna, Austria
mschmiedecker@sba-research.org

Edgar R. Weippl
SBA Research
Vienna, Austria
eweippl@sba-research.org

ABSTRACT

Redundant capacity in filesystem timestamps is recently proposed in the literature as an effective means for information hiding and data leakage.

Here, we evaluate the steganographic capabilities of such channels and propose techniques to aid digital forensics investigation towards identifying and detecting manipulated filesystem timestamps.

Our findings indicate that different storage media and interfaces exhibit different timestamp creation patterns. Such differences can be utilized to characterize file source media and increase the analysis capabilities of the incident response process.

CCS CONCEPTS

•Applied computing →System forensics; Evidence collection, storage and analysis; Investigation techniques; •Security and privacy →*File system security;*

KEYWORDS

Information security, information leakage, steganography, digital forensics, NTFS, filesystem

ACM Reference format:
Sebastian Neuner, Artemios G. Voyiatzis, Martin Schmiedecker, and Edgar R. Weippl. 2017. Timestamp hiccups: Detecting manipulated filesystem timestamps on NTFS. In *Proceedings of ARES '17, Reggio Calabria, Italy, August 29-September 01, 2017,* 6 pages.
DOI: 10.1145/3098954.3098994

1 INTRODUCTION

Cybercrime is rising at unprecedented levels and threatens the function of our society [5]. Digital forensics are more and more important in investigating cases and provide evidence for committed crimes using or utilizing digital means.

Temporal analysis is very useful for digital forensics, especially for reconstructing the timeline of (digital) actions and events. However, the results of an analysis might not be reliable because timestamps of files and directories can be tampered by anti-forensic tools [3, 4].

Numerous steganographic techniques are proposed and analyzed in the research literature [9]. Storage or format-oriented steganographic techniques hide information in logical channels by utilizing redundant or unused fields in format specifications [6]. The topic of hiding data in filesystem metadata is already discussed in the late 1990s [1].

NTFS is the standard filesystem for Microsoft Windows operating systems. The rules for updating the NTFS timestamps are quite complex and depend on both the type of the file (e.g., a Microsoft Windows executable, a Microsoft Office document, or a directory) and the original source of the file (e.g., an external hard disk formatted with FAT32, a compressed archive, or a file from an UDF-formatted CD-ROM). Such subtle differences can be utilized to detect manipulated timestamps that (maliciously) aim to hinder the forensic investigation [2–4].

TOMS is a recently proposed steganographic system that offers stealthiness, robustness, and wide applicability [8]. TOMS hides information in the sub-second part of the NTFS timestamps that is not otherwise utilized in normal filesystem operations.

In this paper, we evaluate the steganographic capabilities of such channels and propose techniques to aid digital forensic investigations. More specifically, we study the following questions:

- How do different storage media and connection interfaces affect the timestamp patterns?
- How do different file creation approaches affect the timestamp generation patterns?
- How does the regular use of the filesystem affect the capacity of the channel over the sub-second part of the timestamps?
- How do practices followed in enterprise environment affect the channel capacity?

- How can we utilize these information to design appropriate detection techniques and mechanisms?

The remainder of the paper is structured as follows. Section 2 describes TOMS over NTFS timestamps in greater detail. Section 3 describes the methodology of our approach, the experiments performed, and the collected datasets. Section 4 evaluates collected information and proposes techniques to detect timestamp manipulation. Finally, Section 5 presents the conclusions of this paper and describes future directions of work.

2 STEGANOGRAPHY OVER FILE TIMESTAMPS WITH TOMS

NTFS treats everything as a file and maintains at least one Master File Table (MFT) entry that includes the last-modified, access, created, and MFT-entry modified (MACE) times. The timestamps are stored in the `FILETIME` structure. The structure contains a 64-bit value representing the number of 100-nanosecond intervals elapsed since January 1, 1601 (UTC) [7].

This ample capacity is a source of redundancy, given that such granularity is not used by any means in modern operating systems. TOMS was proposed to use this spare capacity to realize an information hiding channel of steganographic strength [8].

It is quite common for modern operating systems to disable updating the last access or even the last modified timestamp. This is for performance reasons and for increasing the useful lifetime of the storage medium, e.g., an SSD or USB flash drive. In consumer-grade usage scenarios, we can expect that the *last access* timestamp remains intact as well as the *creation* timestamp.

NTFS uses 24 bits to represent the sub-second part of the timestamps. Thus, six bytes per file can be used to create a hidden information channel. These bytes comprise the so-called TOMS elementary storage unit (ESU). The design of TOMS follows a four-layers approach: a storage container layer to handle the original message, an error correction layer for coping with errors, an encryption layer for hiding the presence of the message and the redundancy introduced by the error correction layer, and the ESU layer.

The interested reader can consult [8] for a detailed description of each layer and the steps to embed a message in the *creation* and *last access* timestamps of files stored in an NTFS filesystem and then recall it. Here, we are interested to explore if such information can be detected in first place due to the apparent uniformity of the timestamps and what is a realistic capacity for a TOMS steganographic channel. We describe in the next Section a series of experiments we designed and performed for this.

3 METHODOLOGY AND DATASETS

We designed three experiments to study the characteristics of the TOMS steganographic channel. We derived one dataset per experiment: a synthetic (artificially-generated) dataset, a consumer-grade dataset contributed by individual volunteers, and an enterprise-grade contributed by a collaborating company. We describe in the following the experiments and the collected information in greater detail.

3.1 Synthetic dataset

We used the same laptop computer running Microsoft Windows 10 64-bit build 1607 for all the steps described below. The laptop computer has an Intel i3 with 2.1 GHz and 4 GB RAM.

3.1.1 Storage media. We used five different storage media, namely a mechanical, spindle disk (HDD), a solid-state disk (SSD), two external hard disks (E1 and E2), and a USB flash drive (U1). The HDD is manufactured by Hitachi (model number: `HTS541010G9SA00`). It has a storage capacity of 100 GB and spins with 5,400 rpm (revolutions per minute). The SSD is manufactured by Micron (model number: `MTFDDAK256MAY`). It has a storage capacity of 256 GB. Both E1 and E2 disks drives were mechanical, spindle disks hosted in a separate case. E1 is manufactured by Western Digital (model number: `WD2500I032-001`). It has a storage capacity of 240 GB and spins with 5,400 rpm. An IDE connector is used to mount the disk inside the case. E2 is manufactured by Seagate (model number: `STDR2000200`). It has a storage capacity of 2 TB and spins with 5,400 rpm. A SATA connector is used to mount inside the case. Finally, U1 is manufactured by Kingston (model number: `DTR30G2`, Datatraveler). It has a storage capacity of 16 GB.

3.1.2 Connectors and interfaces. The two internal disks (HDD and SSD) were connected over a SATA bus. The two external disks were connected over a USB 2.0 (E1) and a USB 3.0 (E2) interface. A USB 3.0 port was used to connect the USB flash drive. We also used the HDD in a different setup (eS), connected to the system through an external SATA (eSATA) connector.

3.1.3 Manual file copy from external sources. After the initial setup of the operating system, several files were delivered using a USB flash drive. Some of those files were then installed to provide a basis for scripted file generation described in Section 3.1.4.

3.1.4 Scripted file generation. We used two scripts to generate files[1], one based on Python and one based on Microsoft Powershell. The Python script utilized the default packages, such as "`os`". The Microsoft Powershell script relied on no external packages.

Both scripts accept as an input parameters the path to create the files (e.g., `C:\tmp`); the number of files to create; and, optionally, a delay between each file creation. For the latter script, we suppressed the default behavior of printing the created files in the standard output (`stdout`), for performance (speed) reasons and for compliance with the behavior of the former script.

3.1.5 Automated file generation process. We proceeded with the file generation as follows. First, we formatted the storage media and created an NTFS volume (filesystem) on them. Then, we did a fresh install of the operating system from a readily-available Microsoft Windows 10 image available on a spare external USB drive. We repeated the same procedure at the start of each experiment described below. This approach ensured a common starting point for all the experiments and a ground truth for the files of the operating system.

The format step in the beginning is a necessary step to ensure that the MFT is reset and all experiments start from the same point. Indeed, the MFT is a special file on NTFS-based filesystems. It

[1]The scripts are available at https://www.sba-research.org/ares2017hiccups/

contains a record for every file and directory ever contained in the filesystem. As such, a growing-in-size MFT impacts the read and write time of the files, due to the longer access time to the MFT. Hence, the more files are created by its iteration of script execution, the longer the access time becomes.

The following list summarizes the experiments we ran:

- One set of 100,000 files created with the Python script on the HDD and SSD with a random (uniform) delay of 0.1 to 1.0 second between each file creation. The same using the Microsoft Powershell script.
- Two sets of 100,000 files each created with the Python script on the HDD, SSD, E1, E2, U1, and eS with no delay between each file creation, i.e., as fast as possible. The same using the Microsoft Powershell script.

3.1.6 File timestamp extraction. We developed a Python script based on the `stat` subpackage of `os` to extract the file timestamps from each file on the filesystem. We extracted the creation (C), last accessed (A), and last modified (M) timestamps of each file and directory with a granularity of 100 nanoseconds. The outputs of the script were saved on an external disk for later processing. Once logs were collected, the storage media under test was wiped out, formatted, and the process concluded.

3.2 Consumer dataset

The second dataset was derived by four individuals that volunteered to participate in our study through personal invitations. We explained to the volunteers the aim of our study and shared with them the source code of the Python script used to extract the file timestamps for review before getting their consent. To protect the innocent, the script parsed through the SHA3 hashing algorithm both the filename and its path; the information collected are the hash output and the three timestamps (C, A, and M) for each file.

In the first round, the volunteers executed the timestamp extraction Python script on their personal computers and laptops running a version of the Microsoft Windows operating system and hosting at least one NTFS filesystem. We label these partitions as *consumer-grade*. The partitions span diverse uses, such as playing computer games, IT freelancing, leisure activities (e.g., Internet surfing and movie watching), backup storage for valuable personal information (e.g., photographs and long documents), and mobile computing.

The script outputs were written in text files and the volunteers shared back these files. The volunteers contributed timestamp information for ten NTFS partitions and 2.5 million files in total.

In the second round, we contacted the volunteers again after ten weeks and asked them to perform the same steps again. All the participants responded within one week. The combined logs provided information for 2.6 million files in total, i.e., a 100,000 increase in the number of files.

3.3 Enterprise dataset

The third dataset was contributed by a collaborating company that agreed to participate in our study. The company policy dictates common rules for each computer (e.g., automated installation of licensed software packages) and centralized administration by authorized personnel.

Using the same script as in the case of the individual volunteers, the company personnel responsible for its IT infrastructure extracted and shared with us the timestamps for more than 22 million files of 70 different computers and NTFS partitions.

In this enterprise environment, we assume that the file timestamps for a large number of files across different computers will be the same, as the files come from the same installation media.

4 ANALYSIS

Our analysis is based on the three types of datasets we collected (synthetic, consumer, and enterprise). We focus on the following parameters: effect of the underlying storage technology and creation techniques, effect of the regular use of the filesystem, and effect on large-scale installations.

For the sake of readability, hereafter we use the term "timestamp" to refer to the 24-bit sub-second part of each file's 64-bit "full timestamp". Recall that NTFS stores this parts of information with a granularity of 100 nanoseconds. Hence, there are in principle 10 million unique *timestamps*.

4.1 Storage technology and scripted creation

We analyzed the extracted creation (C) timestamps for the six storage media used to produce our synthetic dataset. Since the files are not accessed or modified afterwards, we do not discuss in the following the access (A) and last-modified (M) timestamps. They are the same as the creation timestamp.

The first step relates to the delayed creation of 100,000 files in the two internal disks (HDD and SSD). It took one minute and 35 seconds for the Python script to create all these files, and 31 seconds less for the Powershell script. Table 1 summarizes the average and standard deviation number of occurrences of each unique timestamp included in this part of the synthetic dataset.

In the case of the Python script on the HDD, the distribution is quite uniform (average 1.0) and there are 100,000 unique full timestamps. In the case of the Powershell script on the HDD, the situation is slightly different. There are multiple repeating timestamps and only 66,397 unique full timestamps. This number is significantly low. There are even more than 4,300 cases where three files share the exact same full timestamp.

The situation is similar in the case of the SSD. The distribution is quite uniform for Python and all the 100,000 files have a unique full timestamp. Again, the Powershell script results in repeating timestamps and there are only 64,668 unique full timestamps. There are even more than 6,800 cases where three files share the exact same *full timestamp*.

We conclude from the above that the storage media and connectors do not affect the timestamp distribution. However, it appears that the scripting language used to generate the files *does* affect the produced timestamps.

The second step relates to the creation of two sets of 200,000 files in each of the six available setups described in Section 3.1.2. In general, the creation of the files was faster using Powershell rather than Python, as summarized in Table 2. It took a Powershell close to one minute to generate the 200,000 files of each set. This is quite an interesting observation: it is almost the same time needed in the first step to generate the 100,000 files. Python exhibited the same

Table 1: Average and standard deviation of the occurrences of unique timestamps in the synthetic dataset (delayed creation)

Storage	Python	Powershell
HDD	1.00 (0.02)	1.51 (0.62)
SSD	1.00 (0.02)	1.55 (0.68)

Table 2: Time in minutes to create a set 200,000 files for the synthetic dataset (no delay)

Storage	Python	Powershell
HDD	1:35	1:04
SSD	1:34	1:01
E1	0:59	1:01
E2	1:01	1:00
U1	0:59	0:24
eS	0:58	0:57

Table 3: Average and standard deviation of the occurrences of unique timestamps in the synthetic dataset (creation without delay)

Storage	Python	Powershell
HDD	12.95 (2.09)	1.99 (0.37)
SSD	13.52 (1.62)	1.97 (0.39)
E1	3.93 (2.04)	1.80 (0.50)
E2	17.31 (2.80)	1.68 (0.56)
U1	3.46 (0.94)	1.50 (0.52)
eS	31.00 (6.13)	2.15 (0.52)

performance as before for both HDD and SSD. It exhibits similar performance to Powershell in the case of the externally-connected disks E1, E2, and eS. The latter is also worth-mentioning, as the storage media is the same as HDD; changing just the connector from internal SATA to external eSATA, it takes 33% less time to create the files. The case of U1 is also interesting in that for Python it takes the same time (about one minute) like the other externally-connected. However, it takes less than half a minute for Powershell to create the files on it.

We proceed with an analysis of the 400,000 creation timestamps for each of the six storage media used to produce our synthetic dataset. Table 3 summarizes the average and standard deviation number of occurrences of each unique timestamp included in this part of the synthetic dataset. The situation now is quite different compared to that summarized in Table 1. The Powershell script results in an average number of occurrences that is close to 2.0 and a standard deviation less than 0.6. On the other hand, both the average and the standard deviation of occurrences is significantly high in the case of the Python script, ranging from about 3.5 for the USB flash drive (U1) to 31 for the hard drive connected over an eSATA connector (eS). We also observe that the same device has an average of almost 13 when connected through the internal SATA interface (row "HDD"). Figures 1-4 visualize these striking differences in the distribution of timestamps for the four cases of HDD and eS using Python and Powershell.

The aforementioned information suggests that an analysis of the distribution of the (sub-second) creation timestamp part may reveal both the scripting language and the storage media type and connector used when the files were created. This can be useful to detect if a set of files were originally created on the disk under investigation or were transferred to it through other means (e.g., copy from another media, which might disclose a data leakage).

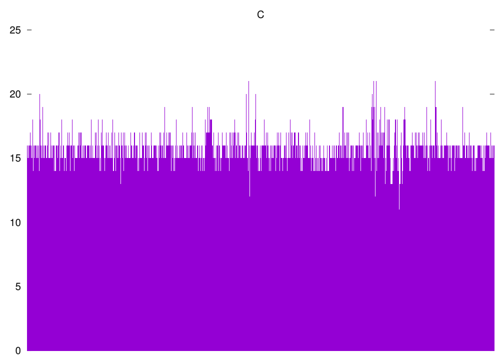

Figure 1: Synthetic, HDD, no delay, Python

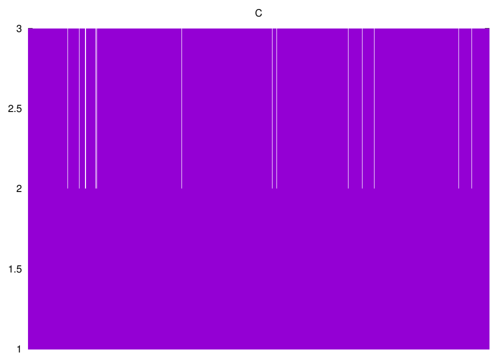

Figure 2: Synthetic, HDD, no delay, Powershell

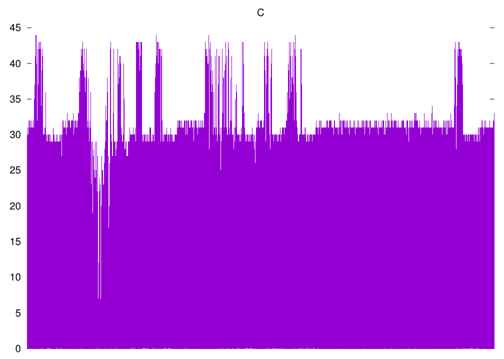

Figure 3: Synthetic, eSATA, no delay, Python

4.2 Regular use of the filesystem

The first part of our analysis, described in Section 4.1, justified that is it feasible to generate a large numbers of files in a short time.

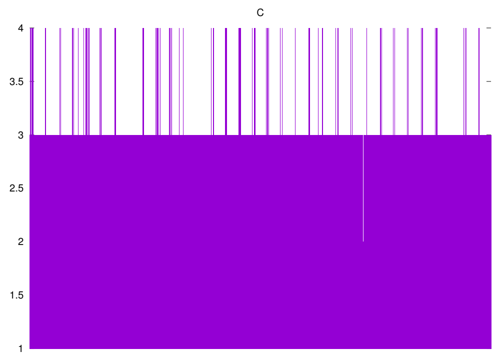

Figure 4: Synthetic, eSATA, no delay, Powershell

Table 4: Number of unique timestamps (bins) of files in the consumer-grade dataset filesystems

id	Round 1		Round 2	
	C-bins	A-bins	C-bins	A-bins
S1	66,146	83,907	74,897	92,662
S2	57,920	92,405	104,538	68,224
T1	67,787	103,290	76,182	113,017
T2	132,149	138,917	625,753	766,984
T3	127,086	135,902	127,086	135,902
T4	270,206	317,427	270,273	317,503
M1	385,088	384,961	292,000	317,906
M2	206,461	287,538	194,620	297,097
A1	184,367	207,896	88,782	113,356
A2	-	-	20,745	21,811

This does allow the fast creation of a steganographic channel to hide information.

In the second part of the analysis, we study the effect of the day-to-day use of a filesystem on the timestamps. We use the consumer-grade dataset for this analysis. There are information for ten NTFS volumes (filesystems).

Table 4 summarizes the number of unique create (C) and last-access (A) timestamps observed (C-bins and A-bins respectively) in each volume of the consumer dataset at the beginning of the experiment (Round 1) and after ten weeks (Round 2). There are some interesting observations to further discuss in the next paragraphs.

A first observation is that there is a *thirtyfold* difference in size among the ten volumes in the number of C-bins, as demonstrated in the case of A2 and T2 in Round 2. This indicates different usage patterns for the volumes (e.g., storing only the operating system files and using it a working space).

A second observation is that number of unique timestamps heavily fluctuated between the two rounds of the experiment. There are NTFS volumes that remain almost intact (e.g., T3 and T4), volumes that rapidly expand (e.g., S2 and T2), or shrink (e.g., M1 and A1).

A third observation is that in all but two cases (M1 in Round 1 and S2 in Round 2), the are more A-bins than C-bins. It appears that many files share the same creation timestamp but there access times are modified later on. This is further supported by the evidence provided in Table 5. The average number of occurrences for the C-bins is greater than the one of A-bins for all cases. As the total number of creation and last-access timestamps are equal, this is an expected

Table 5: Average and standard deviation of the occurrences of unique timestamps in the consumer-grade dataset (aggregated)

id	Round 1		Round 2	
	C-bins	A-bins	C-bins	A-bins
S	3.80 (17.44)	2.73 (9.53)	3.60 (8.96)	2.56 (8.96)
T	2.87 (37.81)	2.40 (7.53)	2.21 (56.91)	1.80 (10.33)
M	2.36 (59.62)	2.08 (37.66)	2.47 (64.04)	1.96 (33.09)
A	2.02 (5.29)	1.79 (3.58)	3.07 (34.43)	2.45 (4.75)

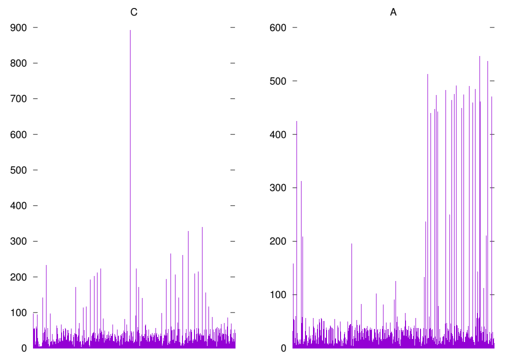

Figure 5: Histogram of C-bins and A-bins for S2

finding. However, the standard deviation is quite bigger than the average value. This is an indication that the distributions deviate from a uniform one (even for the reduced number of available bins) and there might be a significant number of outliers.

Our assumption is valid, as depicted in Figure 5 and Figure 6. The Figures depict the histogram of C- and A-bins for two NTFS volumes, namely S2 and A1. There are a lot of outliers values, reaching even 900 occurrences. Similar trends are observed in all volumes of our dataset. They are not reported here due to space limitations. One of the volunteers kindly agreed to share the file names and paths of the whole NTFS volume (identified as S1) for the purpose of this paper.

Our analysis indicates that a large number of files are part of software installations on the volume (e.g., an office productivity suite and a computer game). Such files are installed from compressed archives and/or DVD/CD-ROM media. As such, the original timestamps are preserved when copied to the NTFS volume. The original media do not support 100-nanosecond granularity[2], the "hiccups" in the histogram hence. These hiccups do not invalidate the TOMS steganographic channel. Rather, an attacker must carefully select a subset of files that exhibit a smooth, uniform distribution and avoid specific paths that come from installation media (e.g., the `C:\Program Files\` folder). Another approach would be to create a new set of files altogether for hiding information in their timestamps.

[2]The UDF filesystem supports microsecond granularity (Source: http://www.osta.org/specs/pdf/udf260.pdf).

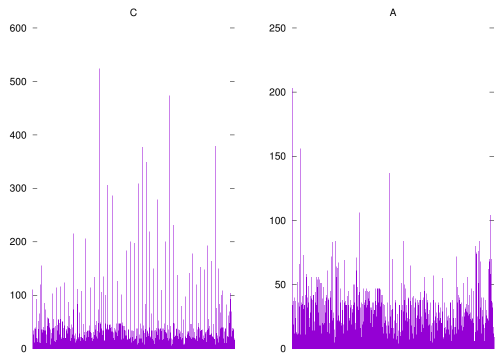

Figure 6: Histogram of C-bins and A-bins for A1

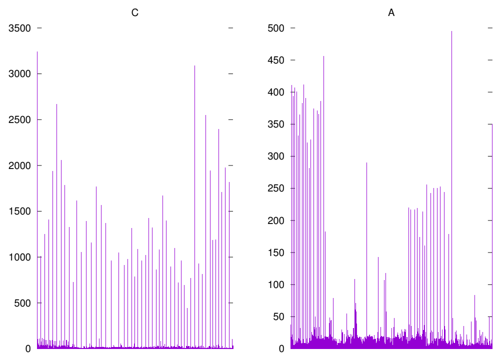

Figure 7: Histogram of C- and A-bins in enterprise volume 42

4.3 Enterprise environment

In the third and last part of the analysis, we study the effect of a homogeneous environment on the distribution of the file timestamps. We use the enterprise-grade dataset for this analysis. There are information for 70 NTFS volumes.

Our analysis revealed that a large number of files are, as in the case of the consumer-grade dataset, part of software installations from compressed archives and UDF volumes. There are stronger patterns now, as these installations are instrumented from a central location and from the same installation media.

Figure 7 depicts the distribution of the create (C) and last-access (A) timestamps for one randomly-selected volumes; all the 70 volumes exhibit the same patterns and are not reported here due to space constraints. The "hiccups" are evident once more but a TOMS-based attack is again possible.

We note that in the case of an enterprise environment, the attackers have a harder task to solve, as the channel capacity is further reduced. Indeed, the IT administrators can compare the distribution of timestamps of an investigated volume with many more available in the enterprise - this is not possible in the case of single consumer-grade volume as there is no "reference" volume available to compare against. Furthermore, company policies can reduce the number of volumes and paths where an attacker can place the files for the TOMS channel, further minimizing (but not eliminating) the risk of an attack.

5 CONCLUSIONS

TOMS utilizes the redundant capacity available in filesystem timestamps to build an information hiding channel of steganographic strength. We assessed the feasibility of building such channels using different storage media and connectors and evaluated the attainable channel capacity in artificial (synthetic), consumer, and enterprise environments.

We confirm that TOMS is a feasible threat. However, the attacker has to spend significant effort to hide the manipulations from a proper digital forensic investigation, either by disabling the TOMS encryption layer or by choosing those files that exhibit a uniform distribution of timestamps already. The capacity of the steganographic channel is further reduced in an enterprise environment with centralized administration, where different NTFS volumes can be compared with each other for irregularities in timestamp patterns.

As future work, we aim to extend the feasibility study of TOMS in other filesystems and devise tools that can automate timestamp monitoring and detection of such manipulations.

ACKNOWLEDGMENTS

The research was supported by the Austrian Research Promotion Agency (FFG) through the Bridge Early Stage grant P846070 (SpeedFor) and the COMET K1-centres programme (SBA2 and SBA-K1). We thank the reviewers for their insightful feedback and the volunteers for providing their filesystem data.

REFERENCES

[1] R. Anderson, R. Needham, and A. Shamir. 1998. The steganographic file system. In *Information Hiding*. Springer, 73–82.
[2] K.-P. Chow, F.Y.W. Law, M.Y.K. Kwan, and P.K.Y. Lai. 2007. The rules of time on NTFS file system. In *Second International Workshop on Systematic Approaches to Digital Forensic Engineering (SADFE 2007)*. IEEE, 71–85.
[3] X. Ding and H. Zou. 2010. Reliable Time Based Forensics in NTFS. (2010). Available on https://www.acsac.org/2010/program/posters/ding.pdf.
[4] X. Ding and H. Zou. 2011. Time based data forensic and cross-reference analysis. In *Proceedings of the 2011 ACM Symposium on Applied Computing*. ACM, 185–190.
[5] Europol. 2016. 2016 Internet Organized Crime Threat Assessment (IOCTA). (2016). Online at https://www.europol.europa.eu/content/internet-organised-crime-threat-assessment-iocta-2016.
[6] H. Khan, M. Javed, S.A. Khayam, and F. Mirza. 2011. Designing a cluster-based covert channel to evade disk investigation and forensics. *Computers & Security* 30, 1 (2011), 35–49.
[7] Microsoft Developer Network. 2017. File Times. (2017). Online at https://msdn.microsoft.com/en-us/library/ms724290/.
[8] S. Neuner, A.G. Voyiatzis, M. Schmiedecker, S. Brunthaler, S. Katzenbeisser, and E.R. Weippl. 2016. Time is on my side: Steganography in filesystem metadata. *Digital Investigation* 18 (2016), S76–S86.
[9] E. Zielińska, W. Mazurczyk, and K. Szczypiorski. 2014. Trends in steganography. *Commun. ACM* 57, 3 (2014), 86–95.