Competence Centers for
Excellent Technologies

Time is on my side

Steganography in filesystem metadata

Sebastian Neuner, Artemios G.Voyiatzis, Martin Schmiedecker, Stefan Brunthaler, Stefan Katzenbeisser, Edgar R. Weippl

Motivation for our Work

- Raise awareness about hiding techniques for digital investigators
- Need for stealth communication
- Alternative channel (image steganography, audio steganography, ...)
- We did not know what the date 01.01.1601 is all about. Do you?

Steganography

Steganography

Hiding data in plain sight

Steganography – (our) Requirements

- Robustness:
  → Certain amounts of modifications allowed.

- Stealthiness:
  → The existance of an embedded message cannot be proven.

- Deniability:
  → ”What? Who said there is something hidden?”

Steganography – (our) Requirements

- Applicability:
  → The carrying medium should be widely used and offer enough capacity.

- Relying on Kerkhoffs Law:
  → Breaking Stealthiness should not reveal the message.

Steganography – Medium

What is the optimal medium to carry data?

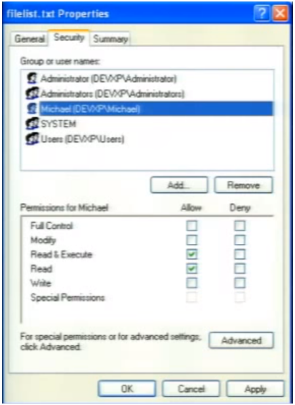
- Satisfying applicability...
  - In widespread use.
  - Offer enough ”random-looking” capacity to carry data.

Filesystem Metadata

Filesystem Metadata for Steganography?

Something like:

- ACL steganography shown by Michael Perklin in 2013 at BlackHat[1]
- Partially-stealth…



filelist.txt Properties

General | Security | Summary

Group or user names:

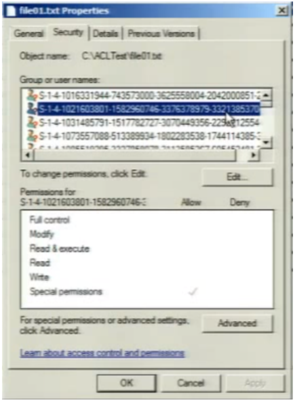
- Administrator (DEVXP\Administrator)
- Administrators (DEVXP\Administrators)
- Michael (DEVXP\Michael)
- SYSTEM
- Users (DEVXP\Users)

Add... | Remove

Permissions for Michael	Allow	Deny
Full Control	□	□
Modify	□	□
Read & Execute	☑	□
Read	☑	□
Write	□	□
Special Permissions	□	□

For special permissions or for advanced settings, click Advanced. | Advanced

OK | Cancel | Apply



file01.txt Properties

General | Security | Details | Previous Versions

Object name: C:\ACLTest\file01.txt

Group or user names:

- S-1-4-1016331944-743573000-3625558004-2042000851-
- S-1-4-1021603801-1582960746-3376378979-332138537
- S-1-4-1031485791-1517782727-3070449356-22 12554
- S-1-4-1073557088-513389934-1802283538-1744114385-

To change permissions, click Edit. | Edit...

Permissions for S-1-4-1021603801-1582960746-	Allow	Deny
Full control		
Modify		
Read & execute		
Read		
Write		
Special permissions	✓	

For special permissions or advanced settings, click Advanced. | Advanced

Learn about access control and permissions

OK | Cancel | Apply

Why Filesystem Metadata?

It satisfies a key requirement:
→ (Almost) Everyone uses it.

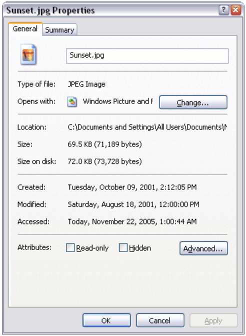
Huge code-bases (high possibility for steganographic channels).

Proposal

- A lot of modern filesystems provide nanosecond granularity
- Operating systems often only show up to minutes or seconds
- Let’s use the non-shown-bits as medium

Give it a try: Timestamp Granularity

GUIs mostly present time values up to minutes or seconds, but more granular storage



Sunset.jpg

Type of file: JPEG Image

Opens with: Windows Picture and F

Location: C:\Documents and Settings\All Users\Documents\M

Size: 69.5 KB (71,189 bytes)

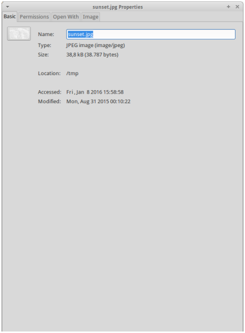
Size on disk: 72.0 KB (73,728 bytes)

Created: Tuesday, October 09, 2001, 2:12:05 PM

Modified: Saturday, August 18, 2001, 12:00:00 PM

Accessed: Today, November 22, 2005, 1:00:44 AM

Attributes: □ Read-only □ Hidden



Name: sunset.jpg

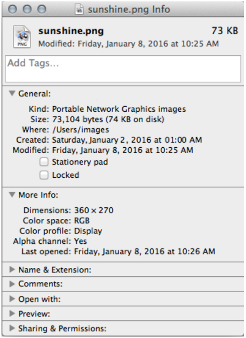
Type: JPEG image (image/jpeg)

Size: 38,8 kB (38.787 bytes)

Location: /tmp

Accessed: Fri , Jan 8 2016 15:58:58

Modified: Mon, Aug 31 2015 00:10:22



sunshine.png 73 KB

Modified: Friday, January 8, 2016 at 10:25 AM

Kind: Portable Network Graphics images

Size: 73,104 bytes (74 KB on disk)

Where: /Users/images

Created: Saturday, January 2, 2016 at 01:00 AM

Modified: Friday, January 8, 2016 at 10:25 AM

□ Stationery pad

□ Locked

Dimensions: 360 × 270

Color space: RGB

Color profile: Display

Alpha channel: Yes

Last opened: Friday, January 8, 2016 at 10:26 AM

More Granular Storage – A Study

What level of timestamp granularity do ”modern” filesystems offer?

Filesystem	File timestamp	Size	Granularity
NTFS	creation	64 bits	100 ns
	access	64 bits	100 ns
	modification	64 bits	100 ns
	modif. of MFT entry	64 bits	100 ns
ext4	creation	64 bits	1 ns
	access	64 bits	1 ns
	modification	64 bits	1 ns
	attribute modif.	64 bits	1 ns
btrfs	creation	64 bits	1 ns
	access	64 bits	1 ns
	modification	64 bits	1 ns
	attribute modif.	64 bits	1 ns

More Granular Storage – A Study

What level of timestamp granularity do ”modern” filesystems offer?

Filesystem	File timestamp	Size	Granularity
ZFS	creation	64 bits	1 ns
	access	64 bits	1 ns
	modification	64 bits	1 ns
	attribute modif.	64 bits	1 ns
FAT32	creation	40 bits	10 msec
	access	16 bits	1 day
	modification	32 bits	2 sec
HFS+	creation	32 bits	1 sec
	access	32 bits	1 sec
	modification	32 bits	1 sec
	attribute modif.	32 bits	1 sec
	backup	32 bits	1 sec
ext3	access	32 bits	1 sec
	modification	32 bits	1 sec
	attribute modif.	32 bits	1 sec

Putting it all together: Time is on my Side

Timestamp–Basics NTFS

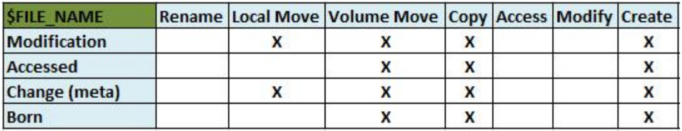
(Our PoCs target NTFS from Win Vista on → later...)

- MACE (Modified, Access, Creation, Modified MFT entry)
- Each 64bits

  → 24bits of that describe the nano seconds
- Number of 100 nano seconds since 1.1.1601

Timestamp–Basics NTFS

Before Vista (XP...):



$FILE_NAME	Rename	Local Move	Volume Move	Copy	Access	Modify	Create
Modification		X	X	X			X
Accessed			X	X			X
Change (meta)		X	X	X			X
Born			X	X			X

Timestamp–Basics NTFS

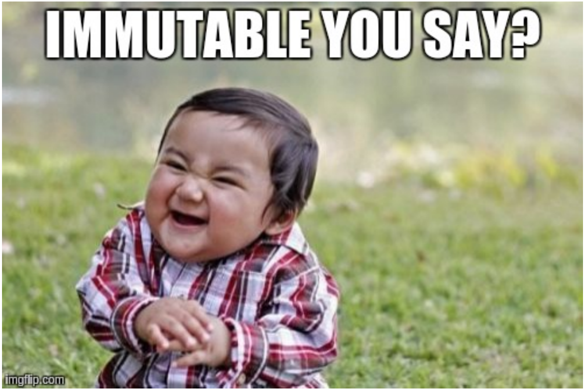
From Vista on...

- By default: NtfsDisableLastAccessUpdate set to 1
  → Immutable access time
- (ext4 mount option ”noatime”)

Timestamp–Basics NTFS

From Vista on...

- By default: NtfsDisableLastAccessUpdate set to 1
  $\rightarrow$ Immutable access time
- (ext4 mount option ”noatime”)



Time is on my side–PoC *

Embed information in the creation (C) and access (A) nano-timestamp-parts of files' metadata (MFT's filename attribute)

- Python
- NTFS
- Variable error correction
- Encryption
- Kerkhoffs Principle!

Time is on my side–PoC 1

Save a metadata file

- Produce a metadata file, containing the location of all modified files
- Error corrected payload is encrypted
- Metadata file is also encrypted (with a different algorithm)
- Drawback: Obviously a file with random data is lying around

Time is on my side–PoC 2

Oblivious Replacement

- Take the data
- Produce error correcting codes
- Hide an index byte in the creation timestamp
- Hide the length indicators
- Encrypt the stuff
- Embed it

Time is on my side–PoC 2

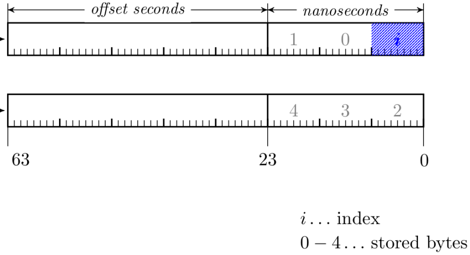
```
struct _ntfs_inode {
  uint64 mft_no;
  MFT_RECORD *mrec;
  ntfs_volume *vol;
  unsigned long state;
  FILE_ATTR_FLAGS flags;
  uint32 attr_list_size;
  uint8 *attr_list;

  uint32 nr_extents;
  union {
    ntfs_inode **extent_nis;
    ntfs_inode *base_ni;
  };

  uint64 data_size;
  uint64 allocated_size;

  ntfs_time creation_time;
  ntfs_time last_data_change_time;
  ntfs_time last_mft_change_time;
  ntfs_time last_access_time;

  uint32 owner_id;
  uint32 security_id;
  uint64 quota_charged;
  uint64 usn;
};
```



Time is on my side–Thoughts

- The index is needed to recover the correct order of the files
- The amount of error correction is variable but influences the possible capacity
- Speaking of capacity:

  → PoC 1 is able to use 48bits payload, where PoC 2 just 40 bits (index byte)

  → The more error-correction, the more capacity is needed (the more errors are recoverable)

Time is on my side–Capacity

Example for PoC2 (oblivious replacement)

- Creation: 3bytes / Access: 3bytes
  - Minus: 1byte per file (index)
  - Minus: Every 255th file contains the length of the whole data
  - Minus: Error correction

Time is on my side–Capacity Win8

Freshly installed Win8 → roughly 160k files

- Theoretical payload: 48bits * 160k: 960KB
- Real payload: (40bits * 160k) - (160k / 255 * 5) - (15% error correction)

  → ∼ 680kb hard payload

Time is on my side–Capacity Win8

Freshly installed Win8 → roughly 160k files

- Theoretical payload: 48bits * 160k: 960KB
- Real payload: (40bits * 160k) - (160k / 255 * 5) - (15% error correction)

  → ∼ 680kb hard payload

Impressive?

Mhmm...not really... **BUT...**

...we offer encryption

...we offer error correction

...we offer order recovery

...we offer stealthiness

Stealth?

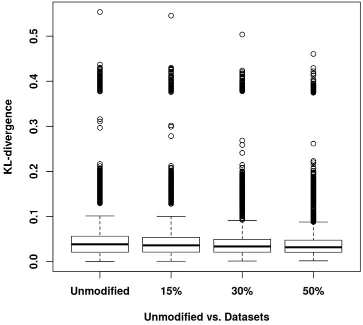
By relying on the requirement of encryption to look like random data, our embedded data looks like random data.

Stealth → statistically undetectable

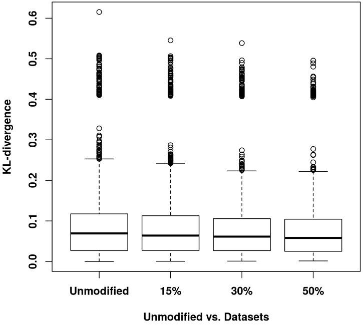
Undetectable?

Measured with Kullback–Leibler divergence (”measure of the difference between two probability distributions”[2])

Creation Timestamp



Access Timestamp



[2] https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence

Corpus

How and where did we measure Stealthiness?

- Synthetic data set
- Real-world data set

Synthetic Data Set

- Python script
- NTFS-3g
- 117 million files
- 50% no delay, 50% random delay between one and two seconds

Real-World Data Set

- 70 NTFS volumes in research lab
- Average: 290k files and 40k directories
- In total: 22.26 million files and directories

Attacks on our System

Attacks

- Denial of Service:

  → You have to know that the data is there

  → No information gain

  → (Re-)Set all timestamps

- Accidental reset

  → File gets deleted and re-created

Attacks

- Timestamp storages:

  MFT filename attribute ←→ MFT standard information

- 1:1 copy

  → Compare before and after embedding

Conclusion

→ Study on which filesystems are usable

→ Feasible

→ Low capacity

Future Work

→ Implications on the Windows $LogFile

→ Extend the PoC's to ext4

→ Fix minor bugs and release the PoC's at:

  → `https://www.sba-research.org/dfrws2016`

Thank you for your attention...

Contact:

Sebastian Neuner
sneuner@sba-research.org
PGP: 0x5E82F701

🐦 @sebastian9er

Image References

https://blogs.sans.org/computer-forensics/files/2010/10/ts_change_rules_gui1.jpg
http://cdn.meme.am/instances/32090244.jpg https://imgflip.com/i/18mv6s
http://philbaumann.com/wp-content/uploads/2009/01/Twitter_bird_logo_2012.png
http://img4.wikia.nocookie.net/__cb20121008041422/thehungergames/images/b/bd/I_has_a_question.jpg