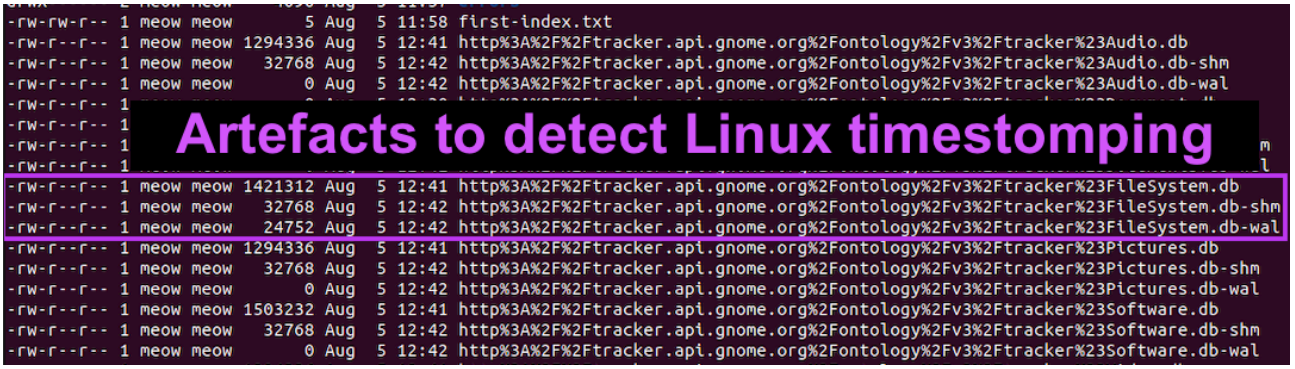
Detecting Linux Anti-Forensics: Timestomping

By inversecos

Published: 2022-08-05 · Archived: 2026-04-05 22:43:11 UTC



Threat actors can modify the timestamps on malicious files to evade detection. This technique has been used time and time again across various APT and opportunistic threat groups. How do you detect this when auditd, bash history and EDR does not exist? I notice basically all writeups on timestomping detection on Linux focus solely on looking at command-line usage of "touch". I'm here to offer you a different method.

I will cover TWO techniques to perform timestomping and offer a different method of detection that doesn't rely on auditd, command line logging or EDR. Why should you care? Because inexperienced analysts may notice a specific malicious file being created outside of an attack window and consider that the compromise timeframe may be even earlier than what it is. Further, the analyst may completely disregard a file as an attacker-based tool if it is completely outside of the timeframe. I'm here to help :)

This blog is structured in four sections:

- Timestomping on Linux (two methods)
- Detection using Tracker3
- Detection using AuditD
- Final thoughts on detection

TIMESTOMPING ON LINUX

There are two methods you can approach timestomping on Linux. These are:

1. Using the "touch" command. I don't recommend this technique for reasons that I'll cover.
2. Modifying the system date and then creating the attacker file.

Just as a quick refresher, this is how timestamps work in Linux:

(M) Modify – Updated whenever the file contents are changed.
(A) Access – Updated when the file contents are accessed (usually via script). Accessing a file via GUI does not always update the access time.
(C) Change – Metadata change time for the file i.e. file ownership change.
(B) Birth – Date the file was created. This is based on the operating system time and exists on EXT4.

Method 1. Timestomping using Touch (bad!)

This is done by running the command "touch -a -m -t <timestamp> <filename>". There are two downsides to using this technique. The first being it's been written about to death and most people have detections for this method. The second downside is that the "change" time cannot be modified using this method.

As you can see in the screenshot below – the change time is *very* different from the modify and access time. This is because the change time will only be modified based on the system date of the operating system. Without auditd or command line logging, this already looks a bit odd. This is because the change time is changed based on the system date of the operating system.

```
meow@meow-virtual-machine:~/.cache/tracker3/files$ touch -a -m -t 200001011103.09 meow.txt
meow@meow-virtual-machine:~/.cache/tracker3/files$ stat meow.txt
  File: meow.txt
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: 803h/2051d      Inode: 527834      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/    meow)   Gid: ( 1000/    meow)
Access: 2000-01-01 11:03:09.000000000 +1100
Modify: 2000-01-01 11:03:09.000000000 +1100
Change: 2022-08-05 12:52:18.583541634 +1000
```

Having the change date being later than the modify and access date does not mean a file was necessary timestomped. However, it does indicate that changes were made to the metadata for the file around the "attack" period (if it fits in the timeline). I cover proper detections for this in the next section.

If you're interested in existing write-ups about detection for the touch command – it's straight forward and generally command-line based. You can read up about these here:

Elastic - https://www.elastic.co/guide/en/security/current/timestomping-using-touch-command.html

Abusing Touch - https://www.thegeekstuff.com/2012/11/linux-touch-command/

Method 2. Timestomping via Modifying the system date

The issue with the "touch" method is that it doesn't change the birth or change time as these are based on the operating system date. As such, a cleaner method to avoid the use of the "touch" command is to modify the system date, drop the files and then revert the system date. Most modern detections for this attack technique only solely detect on the use of the "touch" command *wink wink*.

As you can see in the screenshot below, I've updated the system time to the year 2000 and disabled NTP. Post "compromise" you can enable NTP again.

```
root@meow-virtual-machine:/home/meow/Desktop# timedatectl set-time '2000-02-01'
Failed to set time: Automatic time synchronization is enabled
root@meow-virtual-machine:/home/meow/Desktop# timedatectl set-ntp 0
root@meow-virtual-machine:/home/meow/Desktop# date -s "1 JAN 2000 18:00:00"
Sat 01 Jan 2000 18:00:00 AEDT
root@meow-virtual-machine:/home/meow/Desktop# timedatectl
               Local time: Sat 2000-01-01 18:00:06 AEDT
           Universal time: Sat 2000-01-01 07:00:06 UTC
                 RTC time: Fri 2022-08-05 02:38:05
                Time zone: Australia/Sydney (AEDT, +1100)
System clock synchronized: no
              NTP service: inactive
          RTC in local TZ: no
root@meow-virtual-machine:/home/meow/Desktop#
```

DETECTION METHOD 1: Tracker3

Tracker3 indexes the filesystem on GNOME and acts like a search engine. It's enabled by **default** on newer versions of Ubuntu. It tracks and indexes files and documents in various database files. These are stored in the directory “/home/<user>/.cache/tracker3/files/”. These files are broken down in the following DB files:

- Audio.db – Audio files
- Documents.db – Documents
- FileSystem.db – Files (this is the main one we are interested in)
- Pictures.db – Picture files
- Software.db – Application related files
- Video.db – Video files
- Meta.db – Index of all files in the environment

On top of all these files, there are also corresponding SHM and WAL logs. These are the shared memory file and write-ahead logs. The updates will first be committed to the WAL and then written to the DB. As such, for an analyst, it’s critical that you also consider these WAL logs as a source of interest and investigation. If you have ever performed mobile forensic investigations, you will already be aware of the pivotal role of analysing WAL in mobile investigations ^_^

```
drwx------ 2 meow meow    4096 Aug  5 11:57 errors
-rw-rw-r-- 1 meow meow       5 Aug  5 11:58 first-index.txt
-rw-r--r-- 1 meow meow 1294336 Aug  5 12:41 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23Audio.db
-rw-r--r-- 1 meow meow   32768 Aug  5 12:42 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23Audio.db-shm
-rw-r--r-- 1 meow meow       0 Aug  5 12:42 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23Audio.db-wal
-rw-r--r-- 1 meow meow       0 Aug  5 12:30 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23Document.db
-rw-r--r-- 1 meow meow 1294336 Aug  5 12:41 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23Documents.db
-rw-r--r-- 1 meow meow   32768 Aug  5 12:42 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23Documents.db-shm
-rw-r--r-- 1 meow meow       0 Aug  5 12:42 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23Documents.db-wal
-rw-r--r-- 1 meow meow 1421312 Aug  5 12:41 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23FileSystem.db
-rw-r--r-- 1 meow meow   32768 Aug  5 12:42 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23FileSystem.db-shm
-rw-r--r-- 1 meow meow   24752 Aug  5 12:42 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23FileSystem.db-wal
-rw-r--r-- 1 meow meow 1294336 Aug  5 12:41 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23Pictures.db
-rw-r--r-- 1 meow meow   32768 Aug  5 12:42 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23Pictures.db-shm
-rw-r--r-- 1 meow meow       0 Aug  5 12:42 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23Pictures.db-wal
-rw-r--r-- 1 meow meow 1503232 Aug  5 12:41 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23Software.db
-rw-r--r-- 1 meow meow   32768 Aug  5 12:42 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23Software.db-shm
-rw-r--r-- 1 meow meow       0 Aug  5 12:42 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23Software.db-wal
-rw-r--r-- 1 meow meow 1294336 Aug  5 12:41 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23Video.db
-rw-r--r-- 1 meow meow   32768 Aug  5 12:42 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23Video.db-shm
-rw-r--r-- 1 meow meow       0 Aug  5 12:42 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23Video.db-wal
-rw-rw-r-- 1 meow meow      10 Aug  5 12:42 last-crawl.txt
-rw-rw-r-- 1 meow meow      11 Aug  5 12:41 locale-for-miner-apps.txt
-rw-rw-r-- 1 meow meow       0 Jan  1  2000 meow.txt
-rw-r--r-- 1 meow meow 2826240 Aug  5 12:41 meta.db
-rw-r--r-- 1 meow meow   32768 Aug  5 12:42 meta.db-shm
-rw-r--r-- 1 meow meow       0 Aug  5 12:42 meta.db-wal
-rw-rw---- 1 meow meow       0 Aug  5 12:42 .meta.isrunning
-rw-rw-r-- 1 meow meow  225654 Aug  5 11:57 ontologies.gvdb
```

The only thing you need to know is that tracker3 uses SPARQL and it does retain some deleted data. If you are interested in the documentation for tracker3 commands you can run, take a look at this link: https://gnome.pages.gitlab.gnome.org/tracker/docs/commandline/.

Now, the most important DB file here is the FileSystem.db file. Every time a file is created it will create two entries in the FileSystem.db table:

- Nfo:FileDataObject
- Nie:DataObject

Both entries will track the creation time of the file as well as the time this entry was inserted into the database. Please note that the "insertion" time (idk the way to refer to this lol) occurs shortly after the file is created and is based on the system time.

The screenshot below shows the original table creation for "FileDataObject". As you can see it tracks the following timestamps:

- fileCreated (third value)
- fileLastAccessed (second variable)
- fileLastModified (second last variable)

```
CREATE TABLE IF NOT EXISTS "nfo:FileDataObject" (ID INTEGER NOT NULL PRIMARY KEY, "nfo:fileLastAccessed" INTEGER, "nfo:fileCrea
ted" INTEGER, "nfo:fileSize" INTEGER, "nfo:permissions" TEXT COLLATE TRACKER, "nfo:fileName" TEXT COLLATE TRACKER, "nfo:hasHash
" INTEGER, "nfo:fileOwner" INTEGER, "nfo:fileLastModified" INTEGER, "tracker:extractorHash" TEXT COLLATE TRACKER);
```

For the table "DataObject" you can also see the original table creation below. You can see that it tracks one interesting timestamp:

- nie:created (second last variable).

This essentially tracks when a file was created / inserted into this table. This happens almost simultaneously.

```
CREATE TABLE IF NOT EXISTS "nie:DataObject" (ID INTEGER NOT NULL PRIMARY KEY, "nie:url" TEXT COLLATE TRACKER UNIQUE, "nie:byteS
ize" INTEGER, "nie:lastRefreshed" INTEGER, "nie:created" INTEGER, "nfo:belongsToContainer" INTEGER);
```

Now let's delve into some examples of using this artefact to perform detection.

1. Detecting touch command timestomping

In this example I used touch to timestomp a file named "meow.txt". As you can see the modify and access time have been changed to 2000.

```
meow@meow-virtual-machine:~/.cache/tracker3/files$ stat meow.txt
  File: meow.txt
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: 803h/2051d      Inode: 527834      Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/    meow)   Gid: ( 1000/    meow)
Access: 2000-01-01 11:03:09.000000000 +1100
Modify: 2000-01-01 11:03:09.000000000 +1100
Change: 2022-08-05 12:52:18.583541634 +1000
```

The dirty and quick way to query the tracker3 db is through using sqlite3. The command I am running is “sqlite3 <filename> .dump | grep <filename>”

```
meow@meow-virtual-machine:~/.cache/tracker3/files$ sqlite3 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23F
ileSystem.db .dump | grep "meow.txt"
INSERT INTO "nfo:FileDataObject" VALUES(1036,946710001,1659665327,0,NULL,'meow.txt',NULL,NULL,946710001,NULL);
INSERT INTO "nie:DataObject" VALUES(1036,'file:///home/meow/Desktop/meow.txt',0,NULL,1659665327,607);
```

From above you can see that the three timestamps I have highlighted in pink correlate to the:

- fileCreated time
- fileLastModified time
- nie:Created time

What’s important to note here is that when a file is created, an entry is immediately created in both the FileDataObject and DataObject tables. Therefore, the nie:Created time correlates with the actual file creation time.

Let’s convert these times into something more legible:

```
meow@meow-virtual-machine:~/.cache/tracker3/files$ date -d @946710001
Sat 01 Jan 2000 18:00:01 AEDT
meow@meow-virtual-machine:~/.cache/tracker3/files$ date -d @1659665327
Fri 05 Aug 2022 12:08:47 AEST
```

This is a great way to detect timestomping forensically on Linux – by noting that the CREATION time is LATER than when the file was accessed and modified. This is the key detection here and results in a high-fidelity outcome :)

2. Detecting timestomping via updating system time

Now let’s use the second timestomping technique of changing the actual system time on the operating system to perform timestomping on a file named “bad.txt” and see if this can detect it. As you can see below, I changed the system time to the year 2000 and then dropped a bad.txt file and all the timestamps are written to my desired time.

```
root@meow-virtual-machine:/home/meow/Desktop# touch bad.txt
root@meow-virtual-machine:/home/meow/Desktop# stat bad.txt
  File: bad.txt
  Size: 0               Blocks: 0          IO Block: 4096   regular empty file
Device: 803h/2051d      Inode: 535256      Links: 1
Access: (0644/-rw-r--r--)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2000-01-01 18:00:01.408000057 +1100
Modify: 2000-01-01 18:00:01.408000057 +1100
Change: 2000-01-01 18:00:01.408000057 +1100
 Birth: 2000-01-01 18:00:01.408000057 +1100
```

Using this technique for detecting timestomping by assessing the Filesystem.db does NOT work for this technique. Because the Filesystem.db time is based on the overall operating system date time. As you can see from the screenshot below, the timestamp of 946710001 maps to our timestomped time of 2000-01-01.

```
meow@meow-virtual-machine:~/.cache/tracker3/files$ sqlite3 http%3A%2F%2Ftracker.api.gnome.org%2Fontology%2Fv3%2Ftracker%23F
ileSystem.db .dump | grep "bad.txt"
INSERT INTO "nfo:FileDataObject" VALUES(1039,946710001,946710001,0,NULL,'bad.txt',NULL,NULL,946710001,NULL);
INSERT INTO "nie:DataObject" VALUES(1039,'file:///home/meow/Desktop/bad.txt',0,NULL,946710001,607);
```

DETECTION METHOD 2: AuditD

I'm not going to go into too much detail here because this topic has been written about a lot. All that I have to say here is, you can set up the following rules to monitor:

- Usage of the touch binary
- Changes made to a directory (this is honestly hellish because… it raises the bigger question of… what directory are you monitoring? Do you monitor critical ones and risk missing out on some directories?) For this exercise I set it to /tmp which is the directory I am messing around with for this exercise's sake.

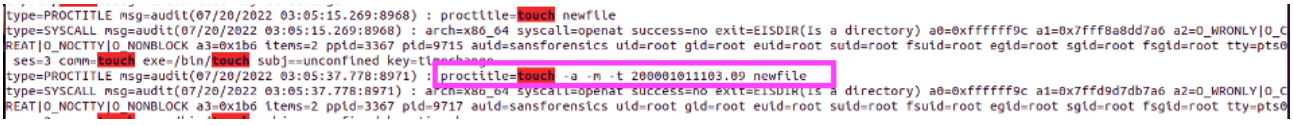
The screenshot below shows the two audit rules I created for this testing:

```
# timestomping
-w /usr/bin/touch -p x -k timestomp
-w /tmp -k timechange
```

Below is a screenshot of me doing a basic timestomp using the "touch" command:

```
root$ touch -a -m -t 200001011103.09 newfile
root$ stat newfile/
  File: newfile/
  Size: 4096            Blocks: 8          IO Block: 4096   directory
Device: 801h/2049d      Inode: 4718633     Links: 2
Access: (0755/drwxr-xr-x)  Uid: (    0/    root)   Gid: (    0/    root)
Access: 2000-01-01 11:03:09.000000000 +0000
Modify: 2000-01-01 11:03:09.000000000 +0000
Change: 2022-07-20 03:05:37.778047567 +0000
```

As you can see this has been caught by the audit rules:



FINAL THOUGHTS

So, what does this tell us? It tells us three interesting things:

- Timestomping by changing system time evades detections that focus on the "touch" command (which is the main detection I keep seeing everywhere).
- FileSystem.db will catch instances of timestomping where a client does not have command line logging, EDR or auditd enabled (also is a good method for detection in terms of forensics)
- Forensic detection using FileSystem.db in tracker3 fails when the system time change method is used for timestomping because the DB is updated using system time (this is understandable)

True detection of timestomping would need to focus on a combination of 3 aspects: command line logging (to catch instances of touch, datetime modification etc), analysis of FileSystem.db and the associated WAL and

finally, use of auditd to filter on binaries of interest / directories of interest.
