Linux incident response - malicious timestamp manipulation

Taz Wake | Halkyn Consulting Ltd | Originally published 2023-12-12

Introduction

Timestamp manipulation, a tactic often used by attackers to conceal their activities, involves altering file and system timestamps. It has been used maliciously pretty much since the dawn of computer attacks. The most commonly seen Linux tool (touch, talked about below) has been around since about 1979. The fact is i almost always present in an environment makes it the overwhelmingly most common choice for attackers. This post will look at some elements of what it does, and what it doesn't change. We will also look at configuring auditd for better monitoring of timestamp changes.

Hopefully, this will give defenders, responders and other security professionals a head start in detecting threat actors.

Tools for Timestamp Manipulation

Note:

For reference, in the examples below, we will look at modifications on the timestamps of this file:

```
sansforensics@siftworkstation: ~/Desktop
$ echo "This is just a test" > testfile
sansforensics@siftworkstation: ~/Desktop
$ stat testfile
  File: testfile
  Size: 20              Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 1310735     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/sansforensics)   Gid: ( 1000/sansforensics)
Access: 2023-12-12 19:24:35.515565150 +0000
Modify: 2023-12-12 19:24:35.515565150 +0000
Change: 2023-12-12 19:24:35.515565150 +0000
 Birth: 2023-12-12 19:24:35.515565150 +0000
```

Creation of "testfile" showing all four timestamps set to the time of creation.

Basic touch Command use

In Linux, touch (normally /bin/touch or /usr/bin/touch) is a built-in utility for modifying file timestamps. It can alter three types of timestamps: the access time (**atime**), modification time (**mtime**), and change time (**ctime**). The syntax for touch is straightforward:

- -a: Alters the access time.

- -m: Changes the modification time.

- -t: Specifies the timestamp (format: [YYYYMMDDhhmm.ss]).

For example:

```
sansforensics@siftworkstation: ~/Desktop
$ touch -a -m -t 199912312359.59 testfile
sansforensics@siftworkstation: ~/Desktop
$ stat testfile
  File: testfile
  Size: 20              Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 1310735     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/sansforensics)   Gid: ( 1000/sansforensics)
Access: 1999-12-31 23:59:59.000000000 +0000
Modify: 1999-12-31 23:59:59.000000000 +0000
Change: 2023-12-12 19:26:53.123855384 +0000
 Birth: 2023-12-12 19:24:35.515565150 +0000
```

Running touch against the test file.

However, touch cannot intentionally modify the inode change time (ctime), which is automatically updated by the system when the file's inode information changes and it doesn't modify the file creation (btime) timestamp.

In our example, the new ctime tells us the time the touch command was run and because the atime and mtime are before the creation of the file, it is a strong indicator that timestamp manipulation has taken place.

It is also worth noting, that this use of the command line is only accurate to the second. As a result, the manipulated timestamps show a fractional nano-second value of all zeros.

A slightly more effective use is to provide a date string with the -d argument. This allows you to set the timestamp to 1 nanosecond accuracy but still doesn't support changing the ctime or btime values. The syntax for this is:

```
sansforensics@siftworkstation: ~/Desktop
$ touch -d "1999-12-31 23:59:59.999999999" testfile
sansforensics@siftworkstation: ~/Desktop
$ stat testfile
  File: testfile
  Size: 20              Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 1310735     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/sansforensics)   Gid: ( 1000/sansforensics)
Access: 1999-12-31 23:59:59.999999999 +0000
Modify: 1999-12-31 23:59:59.999999999 +0000
Change: 2023-12-12 19:37:05.617299388 +0000
 Birth: 2023-12-12 19:24:35.515565150 +0000
```

Touch -d setting the nanosecond values of the A/M timestamps

Other Tools

Unlike Windows, it isn't that common to see attackers use other tools. One alternative approach, however, is to use **debugfs** to modify the timestamps at a lower level within the filesystem. However, this is not consistently effective and really does depend on the OS configuration.

Example: Modifying a File's Timestamp with debugfs

- **Identify the File's Inode Number** First, you need to find the inode number of the file whose timestamp you want to modify. You can do this using the ls command with the -i option. Replace filename with the name of your target file. This command will return the inode number of the file.

- **Open the File System in** debugfs You'll need to open the file system containing the file in debugfs. This should be done with root privileges and while the file system is not mounted (or mounted in read-only mode to avoid file system corruption). An example of the command is below. Here, /dev/sdXN should be replaced with the appropriate device identifier and partition number of your file system.

- **Modify the Timestamp** Inside debugfs, use the stat command to check the current timestamps, and then use the set_inode_field command to modify them. In the example below, replace <inode_number> with the inode number obtained earlier and YYYYMMDDHHMM with the desired date and time in the format "year, month, day, hour, minute".

- **Close** debugfs After making the changes, exit debugfs by typing quit.

Problems with using debugfs to modify timestamps

1. **File System Mounted in Write Mode**: If the file system is mounted in write mode while you're making changes with **debugfs**, the file system's own operations might overwrite or disregard the changes made by **debugfs**. It's generally recommended to unmount the file system or mount it in read-only mode before using **debugfs** for modifications, but this is not going to be trivial for most attackers...
2. **File System Caching**: Linux file systems often use caching mechanisms. If you modify an inode's timestamp with debugfs while the file system is mounted, the kernel's cache might not be aware of these changes, and once the tool is closed, the cache may overwrite your changes with the cached values.
3. **File Access After Modification**: If the file whose atime has been modified is accessed after the modification (even by a background process or a file system check), the timestamp could be updated to the current time, thereby overwriting the modification.

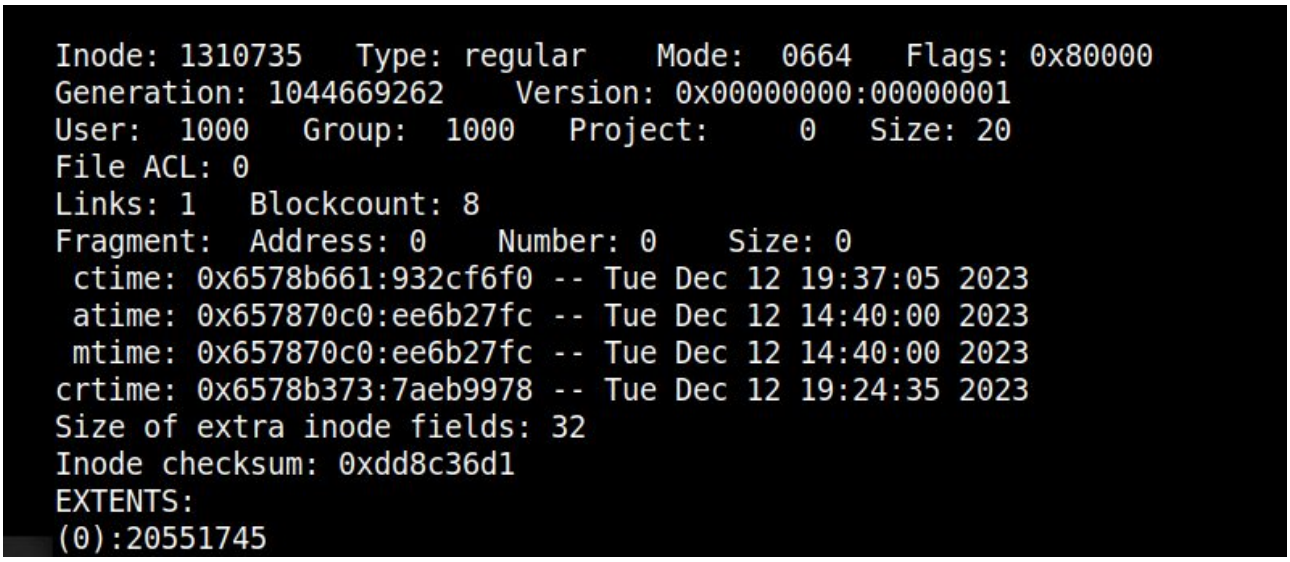
4. **File System Features or Settings**: Some file systems have features or settings that might affect how timestamps are handled. For example, the **noatime** mount option (default on many Linux file systems) tells the OS not to update atime when files are read, which could interfere with your attempts to manually set the timestamp.
5. **Incorrect Use of debugfs**: A small syntax error or incorrect inode number could result in the changes not being applied as expected.
6. **File System Integrity Tools**: Some systems have file system integrity monitoring tools (like AIDE or Tripwire) that could potentially revert changes made to file system metadata.
7. **Journaling File System**: If you're working with a journaling file system (like ext4), the changes made directly to the inode might not be committed to the journal, and thus, could be lost upon exiting **debugfs**.

An example of the impact of caching is shown here

```
sansforensics@siftworkstation: ~/Desktop
$ ls -i testfile
1310735 testfile
sansforensics@siftworkstation: ~/Desktop
$ sudo debugfs -w /dev/mapper/ubuntu--vg-ubuntu--lv
debugfs 1.46.5 (30-Dec-2021)
debugfs:  set_inode_field <1310735> atime 202312121440
debugfs:  set_inode_field <1310735> mtime 202312121440
debugfs:  quit
sansforensics@siftworkstation: ~/Desktop
$ stat testfile
  File: testfile
  Size: 20              Blocks: 8          IO Block: 4096   regular file
Device: fd00h/64768d    Inode: 1310735     Links: 1
Access: (0664/-rw-rw-r--)  Uid: ( 1000/sansforensics)   Gid: ( 1000/sansforensics)
Access: 1999-12-31 23:59:59.999999999 +0000
Modify: 1999-12-31 23:59:59.999999999 +0000
Change: 2023-12-12 19:37:05.617299388 +0000
 Birth: 2023-12-12 19:24:35.515565150 +0000
```

Example part 1 - debugfs used to change the timestamps but this is not reflected in the filesystem yet

If you view the raw data again with **debugfs**, however, it shows that the timestamps have been modified and, for a period of time, both timestamps apply to the inode (depending on what tool is used to view them)



debugfs stat output

MITRE ATT&CK Framework Reference

In the context of the MITRE ATT&CK framework, timestamp manipulation aligns with the Timestomping technique (T1099), falling under the broader category of Defense Evasion. It is a technique used to modify the timestamps of files and directories to hide malicious activity.

Detection opportunities

System Logs

Linux systems typically do not log timestamp changes made by tools like touch. Therefore, detecting such modifications through standard system logs is challenging.

The Linux Journal

The Linux Journal, a subsystem of systemd, might provide indirect evidence of timestamp manipulation, such as records of commands executed or services started that could have performed the modifications. However, these too are limited in scope and detail.

Auditd Configuration for Monitoring Timestamp Changes

To enhance the monitoring of timestamp changes, auditd, the Linux Auditing System, can be configured to watch for specific system calls related to file modifications. Here's how to configure auditd:

1. **Add Audit Rules**: Create rules to monitor system calls like utimensat, which is invoked by touch. Add these lines to /etc/audit/audit.rules:

```

```

2. **Restart Auditd**: Apply the rules by restarting auditd:

```

```

3. **Monitoring**: Check for timestamp modification logs using ausearch:

Detection Advice

Detecting timestamp manipulation requires a holistic approach:

1. **Baseline Normal Activity**: Understand normal timestamp patterns in your environment.

2. **Use of File Integrity Monitoring Tools**: Tools like AIDE or Tripwire can detect changes in file timestamps.

3. **Regular Audits**: Periodic audits of file timestamps can reveal anomalies.

4. **Correlation with Other Indicators**: Combine timestamp data with other logs to identify suspicious activity patterns.

About Halkyn Consulting

Halkyn Consulting Ltd is a UK-based security and risk management consultancy providing 24/7 incident response, digital forensics, threat hunting, virtual security leadership (vCISO) and security compliance support to organisations worldwide. We deliver clear, practical, vendor-independent advice based on real-world threats — no hype, no upsell.

More free security resources are available at www.halkynconsulting.co.uk/security-resources.

Web: www.halkynconsulting.co.uk | **Email:** info@halkynconsulting.co.uk | **Tel:** +44 1244 940858