Persistent Storage Analysis – Part I

Needles in a Haystack

Tobias Dussa
WP8-T1

Webinar, May 2022

Public

Game Plan

- The general approach to what to do to analyze persistent data once you have grabbed it.
- Showcase some simple sample cases:
  - Ways to access image files,
  - how to look *for* and look *at* suspicious files,
  - recovering hidden or lost data.
- … and some things to watch out for.
- Questions/discussion/open mike session.

STOP! A Word of Warning

We cannot and do not provide any legal counseling!

- If you know or suspect that there will be legal steps taken, talk to a lawyer first.
- Depending on your local legislation, there is a very real possibility that you inadvertently destroy evidence.

Preparatory Remarks and Intro
(This Should Feel Familiar)

The Bigger Picture

So there is some sort of persistent storage that you would like to analyze.

What *exactly* is the situation?

- What kind of storage? What size? Where?
  We will assume a local full-disk image file.
- What is the objective of the analysis?
  Finding malware, recovering data, … ?
- How “safe” do you want to play this out?

General Observations – Structure

- Mass storage devices typically come with some sort of structure, for example:
  - Partitions,
  - slices,
  - logical/RAID volumes,
  - file systems,
  - virtual images.
- Data can be hidden in any of these layers (on purpose or by happenstance).
- … but also “in plain sight.”

General Observations – Structure

- Each layer generally contains “slack space” that is not accessible from “higher” layers.
- Examples:
  - Deleted files,
  - unallocated space in volume groups,
  - space between non-aligned partitions boundaries.
- Also, there may be interesting metadata in “lower” layers.
- All in all, it is advisable to ~~get~~ analyze the data from as low a layer as possible.

General Observations – Size

- Mass storage comes in a huge range of sizes, from (realistically) just a few gigabytes up to several petabytes or more.
- This raises two potential problems:
  - How long it takes to ~~acquire~~ analyze data, and
  - where to store the ~~acquired~~ analyzed data.

DFN DEUTSCHES FORSCHUNGSNETZ DFN CERT

GÉANT

General Observations – Safety Level

- Somewhat related to the objective.
- Questions to ask:
  - Is it acceptable to lose (access to) the acquired data?
    Depends.
  - Is it acceptable to alter (parts of) the acquired data?
    Depends.
  - Is it acceptable to give others access to the acquired data (unintentionally)?
    Almost certainly not.
- Usually, forensics call for a fairly high degree of safety.

Let’s Dive In:
Initial Screening

The Easy Way to Access an Image File

- Obviously: Just mount the image. ;-)
- Key advantage: No additional tools necessary.
- The devil is in the detail:
  - Generally, avoid mounting forensic images read-writable:

    ```
    mount -r ${IMAGE} ${DIR}
    ```

    In fact, it is recommended to make the original image *read-only* and/or *immutable*:

    ```
    chmod a-w ${IMAGE} && chattr +i ${IMAGE}
    ```

  - What about partitions and the like?

Dealing With Partitions

- The easiest and most direct way:
  `kpartx -arv ${IMAGE}`
- More basic:
  - Find available partitions:
    `fdisk -l ${IMAGE}`
  - Mount the partition manually:
    `mount -r -o offset=$((512*${STARTSECTOR})) ${IMAGE} ${DIR}`
- Or copy the partition into a separate image file:
  `ddrescue -i ${OFFSET} -o 0 ${IMAGE} ${PART}`

First Things First

Any additional information helps in the analysis:

- “There are connections going to known-bad Command-and-Control systems.”
- “Process 1234 is running at 100 % CPU load but looks weird.”
- “Lots of other HPC systems report rootkits located in **`~/.mozilla/plugins/.aa`**.”

Investigate these (possible) leads first.

Going Through the Haystack

- Basic sanity checks (if available):
  - `debsums | fgrep FAILED`, `rpm --verify --all`
- Browse for suspect files, for example:
  - Binaries with the SUID bit set (`find / -perm -4000`),
  - directories/files starting with `.␣`, `..␣` or `...` (`find / -name ". *"`),
  - shell history files in interesting places (`find / -name ".bash_history"`),
  - SSH keys in interesting places (`find / -name ".ssh"`),
  - passwords set for interesting (read: system) accounts.

Interviewing the Usual Suspects

- Log files (login and connection traces),
- audit traces,
- critical binaries (e. g., SSH server *and* client, web server):
  - SSH client binaries in particular are oftentimes used to grab passwords; drop files posing as kernel header files are not uncommon (XOR-encrypted) → `/usr/include/linux/*` might be interesting.
    (`dpkg-query -S ${FILE}`, `rpm -qf ${FILE}`)
  - SSH and web server binaries might contain backdoors.

Persistence of Malware

Malware often tries to stay active across system reboots. There are many ways to achieve this. Some of the most common places to look:

- Init process data (inittab, init scripts, systemd units, ...),
- cron jobs (user crontabs, scheduled scripts),
- modifications of ubiquitous services/binaries.

A Low-Effort Alternative to Mounting: `testdisk`

- Not necessarily pre-installed,
- scans for partition data,
- provides an interactive walk-through ability,
- can detect and (sometimes) recover deleted files (in file systems),
- comes with **`photorec`**, which will do full-disk carving for some file formats,
- on GitHub: **`https://github.com/cgsecurity/testdisk`**.

Deeper and wider: Timelines

The Idea of Timelines

- Generally speaking, timelines provide a single, consolidated, central view of what has happened over time on a given system.
- Potential data sources:
  - File system,
  - log files,
  - any other timestamped data.
- For the moment, we will look at file system timelining.

File System and Timestamps

In most file systems, every file “has” three timestamps:

- “M”: Modification time (content change),
- “A”: Access time (content read),
- “C”: Change time (status/metadata change).
- (Some filesystems also have “Cr”: Create time.)

Hence, this triple is often called “MAC” data.

Creating a Timeline With System Tools

Trivial to do on a mounted file system:

```
find ${DIR} -xdev -print0 \
    | xargs -0 stat -c "%Y %X %Z %A %u %g %n" \
    > timestamps.dat
```

Maybe pretty-print it with Nixon’s script:

```
timeline-decorator.py < timestamps.dat \
    | sort -n > timestamps.txt
```

This, of course, touches *every* directory, so all the access times are potentially screwed up.

More Hardcore Timeline Creation

The Sleuth Kit (**https://sleuthkit.org/**) provides a set of low-level analysis tools. These can, among many other things, collect timeline information:

```
tsk_gettimes -m ${IMAGE}.raw \
   > ${IMAGE}.tsktimeline
```

Again, some prettyprinting and massaging would be nice:

```
mactime -y -d -b ${IMAGE}.tsktimeline \
   | sed "/^0000-00-00T00:00:00Z/d" \
   > ${IMAGE}.tsktimeline.csv
```

Timeline Analysis Observations

- mtimes and atimes can be changed trivially, ctimes are harder to fake.
- ctimes often indicate file creation times.
- atimes record when binaries were executed.
- Many tools try hard to preserve atimes and mtimes (for instance, **tar**).

Timeline Analysis Pitfalls

- Every timestamp recorded in the filesystem is the *most current* timestamp. Earlier timestamps are **overwritten**. In particular, timestamps may well be tainted by the initial investigation.
- Cronjobs and the like can periodically screw up traces.
- The time zone of the system is critical information.

Wrap-Up

Recap

- Stabbing in the dark is very, very, very tedious.
- Every clue you can possibly get helps a lot.
- Automation helps a lot, but requires effort.

Dangerous Pitfalls

- Not having found evidence is not absence of evidence.
- Absence of evidence is not evidence of absense.
- The few traces present can be misinterpreted → jumping to conclusions is very easy.
- Any open-ended forensic analysis likely requires more effort than you are willing or able to spend → it helps to define the goal *beforehand*.

Thank you

Any questions?

www.geant.org

The research leading to these results has received funding from
the European Union's Horizon 2020 research and innovation programme under Grant Agreement No. 731122 (GN4-2).