ma2tl:
macOS Forensic Timeline Generator
Using mac_apt Analysis Results

Japan Security Analyst Conference 2022

Internet Initiative Japan Inc.
Minoru Kobayashi
Who am I?

Minoru Kobayashi

  Technical research, internal incident response
- External Activities
  Security Camp National Conference Speaker 2017-2019
  Japan Security Analyst Conference Speaker 2018/2020
  Black Hat USA 2018 Briefing Speaker
- Twitter: @unkn0wnbit
Table of Contents

1. Motivation
2. How to create a timeline using the analysis results of mac_apt
3. Implementation of ma2tl
4. Future work
5. Summary
Introduction
Introduction

- The contents of this presentation are all based on research and verification conducted on Intel Macs.
- There may be some differences in specifications on M1 Macs.
- However, in many respects, it could be diverted to investigate M1 Macs.
Motivation
The Need for Timelines in Forensics

◦ What to do after collecting artifacts
  ▪ Analyze OS and application artifacts with tools and create a timeline from the results.

◦ Purpose of creating a timeline
  ▪ Understand the situation (suspicious points) of the affected terminal.
  ▪ Organize the main activities of users, malware, and attackers based on timestamps.
    ▪ Execute programs, download files, mount volumes, set persistence, etc.
  ▪ Creating a timeline can reduce bias, leaps in thinking, and omissions in the research process.
Which tool to choose?

- **Plaso**
  - [https://github.com/log2timeline/plaso](https://github.com/log2timeline/plaso)
  - Automatic generation of super timelines
  - Capable of analyzing artifacts from various operating systems, including macOS.
  - Maintenance is active

- Plaso can generate super timelines 😊

% log2timeline.py --storage-file victim.plaso victim.E01
% psort.py -o l2tcsv -w victime.csv victim.plaso
It takes more than 4.5 days to process a disk image with a capacity of 30GB. (For default settings)
### Super Timeline by Plaso

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Source</th>
<th>Source Type</th>
<th>Type</th>
<th>User</th>
<th>Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>2021/8/10</td>
<td>16:30:06</td>
<td>FILE</td>
<td>File Start</td>
<td>Added Time</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2021/8/10</td>
<td>16:30:06</td>
<td>FILE</td>
<td>File Start</td>
<td>Metadata Modification Time</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2021/8/10</td>
<td>16:30:06</td>
<td>FILE</td>
<td>File Start</td>
<td>Creation Time</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2021/8/10</td>
<td>16:30:06</td>
<td>FILE</td>
<td>File Start</td>
<td>Added Time</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2021/8/10</td>
<td>16:30:06</td>
<td>FILE</td>
<td>File Start</td>
<td>Metadata Modification Time</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2021/8/10</td>
<td>16:30:06</td>
<td>FILE</td>
<td>File Start</td>
<td>Content Modification Time; Last Access Time</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

**Mostly filesystem events**
Problems with the super timeline generated by Plaso

° All filesystem metadata is parsed by default.
  ▪ The number of unimportant files in the analysis is much higher.
  ▪ One file metadata is split into four events (MACB).
° The system log is recorded as a line by line event.
  ▪ We want meaningful information about what happened, not just line-by-line events (we don’t need just log messages).
  ▪ Unified Logs are not analyzed.
° As a result, most of the super timeline is filled with file system and system log events.
° It takes too long to complete the analysis in the first place.
° This is not the kind of information a forensic analyst wants to see first.
The information the forensic analysts want and the investigation strategy

○ For malware infection investigation
  ▪ Persistence setting status
  ▪ Program execution history
  ▪ Volume (USB thumb drives or disk images) mount
  ▪ File Download

○ Make this kind of information the most basic timeline.
○ Flesh out the timeline by expanding the scope of the investigation or conducting a deeper investigation as needed.

○ We need a timeline to use as a basis for the investigation.
○ Creating a timeline with only the necessary activities from the analysis results of a tool focused on artifact analysis is more in line with the requirements.
Tools focused on artifact analysis

- There are two candidate analysis tools
  - AutoMacTC
    - https://github.com/CrowdStrike/automactc
    - Maintenance is stagnant.
  - mac_apt
    - https://github.com/ydkhatri/mac_apt
    - Maintenance is active.
Which tool should we use?

- In view of the maintenance status and functionality, I recommend "mac_apt".
- Why is maintenance so important?
  - macOS artifacts often change their file names and paths with version upgrades.
  - Using analysis tools that are not maintained will increase the number of artifacts that cannot be analyzed over time.
- Unified Logs parser is implemented.
  - Unified Logs records a lot of useful information, but only mac_apt has a parser implemented in OSS.
Motivation for creating the tool

- It is currently best to create a timeline that can be used as a template from the results of mac_apt analysis.
- To create a timeline from mac_apt analysis results, we need to refer to various tables.
  - A table will be created for the number of plugins used in the analysis.
  - Spotlight tables are cumbersome with many columns.
- Unified Logs contain useful information, but mac_apt does not analyze them according to the message content.
  - The message may change depending on the OS version upgrade.
  - It is complicated to do a lot of filtering manually.
  - Even with filtering, the output results may be large, and it may be difficult to visually check.
- I need a tool that automatically generates a forensic timeline!
Similar Tools

- In terms of organizing, displaying, and checking the results of `mac_apt` analysis, the following tools also exist:
  - `mac_int`
    - [https://burnhamforensics.com/projects/mac_int/](https://burnhamforensics.com/projects/mac_int/)
  - Building a Visualization Tool for `mac_apt`
    - [https://leahycenterblog.champlain.edu/2020/05/01/building-a-visualization-tool-for-mac_apt/](https://leahycenterblog.champlain.edu/2020/05/01/building-a-visualization-tool-for-mac_apt/)

- Different in the following ways
  - The main purpose of these tools is to check the results of `mac_apt` analysis in GUI, not to generate a timeline.
  - No maintenance is being performed at this time.
How to create a timeline using the analysis results of mac_apt
Analysis results of mac_apt

- **DB where mac_apt stores the analysis results**
  - `mac_apt.db`: Results of artifact analysis
  - `UnifiedLogs.db`: Parsed Unified Logs
  - `APFS_Volumes_<GUID>.db`: Parsed APFS metadata
  - `Export` and `SPOTLIGHT_DATA`: Folder where the artifact files exported from the disk image will be saved
There are as many tables as plugins used in the analysis.
You can apply filters equivalent to the log command.
You can check file timestamps, etc., but they are not formatted.
Timeline Creation Policy

◦ Focus the investigation on analysis results with time stamps.
  ▪ If timestamp is missing, refer to other tables or APFS_Volumes_xxxx.db

◦ Create a timeline for the following activities
  ▪ Persistence setting status
  ▪ Program execution history
  ▪ Volume mount
  ▪ File Download
Persistence Analysis (1/3)

- **mac_apt.db : AutoStart**
  - First, check the general user settings.
  - No timestamp was recorded.
Persistence Analysis (2/3)

- Check the timestamp of the auto-run configuration file and the executable in APFS_Volumes xxxx.db

```
SELECT * FROM Combined_Paths LEFT JOIN Combined_Inodes ON Combined_Paths.CNID = Combined_Inodes.CNID WHERE Combined_Paths.Path = "/path/to/file" LIMIT 1;
```
Persistence Analysis (3/3)

- Many autorun programs can be found in the folders listed on the right.
- Since macOS 10.15, the system volume and data volume have been split.
  - The system volume is mounted as read-only, so the risk of tampering is lower than before.
  - Starting with macOS 11, the system volume is also signed.
- Therefore, programs on the system volume can be excluded from the investigation at first.
### Program execution history analysis

- **mac_apt.db**: Spotlight Shortcuts
  - Applications executed via Spotlight will be recorded.

The string entered in Spotlight

<table>
<thead>
<tr>
<th>User</th>
<th>UserTyped</th>
<th>DisplayName</th>
<th>LastUsed</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Application name**

**Timestamp** (Only the date and time of the last execution.)

**Application path**
Volume mount analysis

- mac_apt.db : RecentItems
  
  We can see the volume name, but not the timestamp.

- mac_apt.db : FsEvents
  
  Modification date of the artifact file (Not the timestamp)

Filtering conditions for volume mounts (create a folder under Volumes/)

...
File download analysis (1/5)

- **mac_apt.db**: Quarantine
  - DB analysis results that store the same content as the `com.apple.quarantine` extended attribute that is assigned to files downloaded by web browsers, etc.
File download analysis (2/5)

- **mac_apt.db : Safari**
  - The destination of the file will be recorded.
  - Since there is no timestamp, the timestamp is inferred by linking the URL to the DataUrl in the Quarantine table.
  - Safari's download history is deleted after a day by default.

<table>
<thead>
<tr>
<th>Type</th>
<th>Name_or_Title</th>
<th>URL</th>
<th>Date</th>
<th>Other_Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOWNLOAD</td>
<td>FakeTest2-bash.dmg</td>
<td><a href="http://...download/FakeTest2-bash.dmg">http://...download/FakeTest2-bash.dmg</a></td>
<td>NULL</td>
<td>/Users/macforensics/Downloads/FakeTest2-bash.dmg</td>
</tr>
</tbody>
</table>
File download analysis (3/5)

- **mac_apt.db : Chrome**

![Image of a table with columns for Type, Name of Title, URL, Data download URL, Data start timestamp, Data completion timestamp, Local file path, and Origin URL. Each row represents a different download, with details for each column provided.]

- **Data download URL**
- **Download start timestamp**
- **Local file path**
- **Origin URL**
- **Download completion timestamp**
File download analysis (4/5)

- **mac_apt.db : SpotlightDataView**

  - **Local file path**
  - **Data download URL**
  - **Download completion timestamp**
File download analysis (5/5)

- No artifacts are left behind when files are downloaded with the macOS standard command "curl".
  - It leaves traces of the curl execution itself, but does not tell us where it was accessed.
- In such cases, other investigations such as malware analysis are also necessary.
Information confirmed from mac_apt analysis results (1/2)

- **Persistence setting status**
  - We know the autorun configuration file and the program to be autorun.
  - We can also see the timestamps of the above files.

- **Program execution history**
  - We know which applications were executed via Spotlight.
  - We know when it was last executed.
  - There is no other execution history with timestamps.

- **Volume mount**
  - We know the name of the volume you mounted.
  - The exact date and time of the mount is unknown.

- **File Download**
  - We know the date and time of the download, the URL from which it was downloaded, and the file path to which it was saved.

Not enough information.
Information confirmed from mac_apt analysis results (2/2)

◦ The information in mac_apt.db alone is clearly not enough to create a timeline
◦ Any other data we should investigate?
"UnifiedLogs.db"
UnifiedLogs.db is a goldmine (1/5)

- Unified Logs contains information necessary to create a timeline, such as program execution history and volume mount history, which are not left in other artifacts.
- A veritable gold mine for macOS forensics
- But for some reason, I almost never see articles or blogs that explain this kind of information.
UnifiedLogs.db is a goldmine (2/5)

- Even commercial products parse Unified Logs, but do not perform analysis based on message content.
- Database load time, filtering time, etc. are also slower than processing UnifiedLogs.db.
UnifiedLogs.db is a goldmine (3/5)

Filtering is possible, but the process is not very fast.

UnifiedLog will also be parsed.
UnifiedLogs.db is a goldmine (4/5)

- **log command**

% log show --predicate 'FILTERING CONDITION' --start 'YYYY-MM-DD hh:mm:ss' --end 'YYYY-MM-DD hh:mm:ss'

- **Filtering Keywords**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eventType</td>
<td>The type of event: activityCreateEvent, activityTransitionEvent, logEvent, signpostEvent, stateEvent, timesyncEvent, traceEvent and userActionEvent.</td>
</tr>
<tr>
<td>eventMessage</td>
<td>The pattern within the message text, or activity name of a log/trace entry.</td>
</tr>
<tr>
<td>messageType</td>
<td>For logEvent and traceEvent, the type of the message itself: default, info, debug, error or fault.</td>
</tr>
<tr>
<td>process</td>
<td>The name of the process the originated the event.</td>
</tr>
<tr>
<td>processImagePath</td>
<td>The full path of the process that originated the event.</td>
</tr>
<tr>
<td>sender</td>
<td>The name of the library, framework, kernel extension, or mach-o image, that originated the event.</td>
</tr>
<tr>
<td>senderImagePath</td>
<td>The full path of the library, framework, kernel extension, or mach-o image, that originated the event.</td>
</tr>
<tr>
<td>subsystem</td>
<td>The subsystem used to log an event. Only works with log messages generated with os_log(3) APIs.</td>
</tr>
<tr>
<td>category</td>
<td>The category used to log an event. Only works with log messages generated with os_log(3) APIs. When category is used, the subsystem filter should also be provided.</td>
</tr>
</tbody>
</table>
UnifiedLogs.db is a goldmine (5/5)

- Unified Logs format

- In fact, it is displayed as a single line.
Investigating Unified Logs (1/13)

- Program execution history (1)
  - Application Bundle (1)
  - macOS 10.15

```
% log show --info --debug --predicate 'sender == "LaunchServices" AND eventMessage beginswith "LAUNCHING:0x"'
```

Filtering the log data using "sender == "LaunchServices" AND eventMessage BEGINSWITH "LAUNCHING:0x""

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Thread</th>
<th>Type</th>
<th>Activity</th>
<th>PID</th>
<th>TTL</th>
<th>Sender</th>
</tr>
</thead>
<tbody>
<tr>
<td>2021-07-26</td>
<td>0x77b0f8</td>
<td>Default</td>
<td>0x0</td>
<td>78164</td>
<td>0</td>
<td>Evernote: (LaunchServices)</td>
</tr>
<tr>
<td>2021-07-27</td>
<td>0x61b6f</td>
<td>Default</td>
<td>0x0</td>
<td>482</td>
<td>0</td>
<td>Electron: (LaunchServices)</td>
</tr>
<tr>
<td>2021-07-29</td>
<td>0x102e4d</td>
<td>Default</td>
<td>0x0</td>
<td>498</td>
<td>0</td>
<td>Dock: (LaunchServices)</td>
</tr>
<tr>
<td>2021-07-29</td>
<td>0x10362b</td>
<td>Default</td>
<td>0x0</td>
<td>29622</td>
<td>0</td>
<td>open: (LaunchServices)</td>
</tr>
</tbody>
</table>

Sender is "LaunchServices" and the message starts with "LAUNCHING:0x".

Startup source

Launched application
Investigating Unified Logs (2/13)

- **Examples of startup source**
  - Finder
  - Dock
  - Spotlight
  - loginwindow
    - Applications that were executed when the "Reopen windows when logging back in" checkbox was checked in the logout dialog box and the user logged in again.
    - The application specified in "Login Items" under "Users & Groups".
  - open
    - If you run the application with the open command
Program Execution History (2)
- Application Bundle (2)
- macOS 11.0.1 - 12.0.1

The message changes to "LAUNCH: 0x".

Startup source

Launched application (Application Bundle ID)
Investigating Unified Logs (4/13)

- **Behavior in macOS 11 and later (1)**
  - Applications executed with the open command will not be recorded.
  - Bugs in macOS 11.6 (?)
    - Logging does not occur unless the startup source is "loginwindow", "SystemUIServer", or "SoftwareUpdateNotificationManager".
    - macOS 11.6.1 and later are back to the same specifications as 11.5.2 before.
    - Release notes for macOS 11.6.x have not been released, so details are unknown.
Behavior in macOS 11 and later (2)

- When the startup source is "loginwindow".
  - Applications that are subject to "Reopen windows when logging back in" will be logged as "Type = Info" and will only be logged in memory.
  - The message contains "launchInStoppedState=true"
  - Applications specified in the "Login Items" section of "Users & Groups" are logged as "Type = Default" and will remain logged even after reboot.

Filtering the log data using "sender == "LaunchServices" AND composedMessage BEGINSWITH "LAUNCH:"

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Thread</th>
<th>Type</th>
<th>Activity</th>
<th>PID</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022-01-14 08:21:21.918761+0900 0x21efb</td>
<td>Info 0x0</td>
<td>4067 0</td>
<td>loginwindow: (LaunchServices) [com.apple.launchservices:open]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Log of application executed with "Reopen windows when logging back in".
Investigating Unified Logs (6/13)

- Behavior in macOS 11 and later (3)
  - The first time you run an application downloaded from the Internet, Gatekeeper will be checked.
  - The log will be recorded with "Type = Info" (recorded in memory only).
  - The message contains "launchInQuarantine == true".

```bash
% log show --info --debug --predicate 'sender == "LaunchServices" and eventMessage beginswith "LAUNCH: 0x"' --start '2022-01-14 13:00:00'
```

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Thread</th>
<th>Type</th>
<th>Activity</th>
<th>PID</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022-01-14 13:17:44.405335+0900</td>
<td>0x4786</td>
<td>Default</td>
<td>Dock: (LaunchServices) [com.apple.processmanager:front-35286506] LAUNCH: 0x0-0xe0ebb com.apple.DiskImageMounter launched with launchInQuarantine == true, so not starting the application.</td>
<td>1124</td>
<td>0</td>
</tr>
<tr>
<td>2022-01-14 13:18:44.148002+0900</td>
<td>0x50ea</td>
<td>Info</td>
<td>Dock: (LaunchServices) [com.apple.launchservices:open] LAUNCH: 0x0-0xe0ebbb com.apple.DiskImageMounter launched with launchInQuarantine == true, so not starting the application.</td>
<td>1124</td>
<td>0</td>
</tr>
<tr>
<td>2022-01-14 13:19:19.907199+0900</td>
<td>0x523c</td>
<td>Info</td>
<td>Dock: (LaunchServices) [com.apple.launchservices:open] LAUNCH: 0x0-0xf10f1 com.ridiculousfish.HexFiend launched with launchInQuarantine == true, so not starting the application.</td>
<td>1127</td>
<td>0</td>
</tr>
<tr>
<td>2022-01-14 13:21:21.389996+0900</td>
<td>0x5472</td>
<td>Default</td>
<td>Finder: (LaunchServices) [com.apple.launchservices:open] LAUNCH: 0x0-0xf10f1 com.ridiculousfish.HexFiend launched with launchInQuarantine == true, so not starting the application.</td>
<td>1127</td>
<td>0</td>
</tr>
</tbody>
</table>
```

Run the downloaded application (the first time).
Run the downloaded application (the second time).
Investigating Unified Logs (7/13)

- Program Execution History (3)
  - If there is no application bundle ID (1)
    - The application bundle ID is recorded as "(null)".
    - macOS 11 or later

```
% log show --predicate 'eventMessage beginswith "LAUNCH: 0x"' --start '2022-01-12'
Filtering the log data using "composedMessage BEGINSWITH "LAUNCH: 0x""
Skipping info and debug messages, pass --info and/or --debug to include.

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Thread</th>
<th>Type</th>
<th>Activity</th>
<th>PID</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022-01-12 03:57:14.516187+0900 0x1693</td>
<td>Default</td>
<td>0x0</td>
<td>Finder: (LaunchServices)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2022-01-12 03:57:25.281130+0900 0x1d25</td>
<td>Default</td>
<td>0x0</td>
<td>Finder: (LaunchServices)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>[com.apple.processmanager:front-35286506]</td>
<td>LAUNCH: 0x0-0x50050 (null) starting stopped process.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Program Execution History (4)

- No application bundle (2)
  - Identify applications recorded as (null)

Filtered by process name "lsd", message "Non-fatal error enumerating", and time just before (null) was recorded

Just before (null) is recorded (within about 0.1 seconds?)

application path
Investigating Unified Logs (9/13)

Program Execution History (5)

- Unsigned programs allowed to run by Gatekeeper
- It also logs the mounting of unsigned DMGs.
- Logged only on first run.
- macOS 10.15 - 12.0.1

```
% log show --info --debug --predicate 'category == "gk" and eventMessage BEGINSWITH "temporarySigning"'
Filtering the log data using "category == "gk" AND composedMessage BEGINSWITH "temporarySigning""
Timestamp                  Thread TYPE Activity PID TTL
2021-08-10 16:41:11.730226+0900 0x1dc9 Default 0x0 212 0
tsypolicyd: (Security) [com.apple.securityd:gk] temporarySigning type=3 matchFlags=0x0
path=/Users/macforensics/Downloads/FakeTest2-bash.dmg
2021-08-10 16:41:26.286794+0900 0x206c Default 0x0 212 0
tsypolicyd: (Security) [com.apple.securityd:gk] temporarySigning type=1 matchFlags=0x0
path=/Volumes/FakeTest2-bash/FakeApp.app/Contents/MacOS/FakeApp
```

Programs with execute permission or mounted DMGs
Investigating Unified Logs (10/13)

- Program Execution History (6)
  - adhoc signed program
  - macOS 10.15 - 12.0.1

```bash
% log show --predicate '((process == "kernel" and eventMessage beginswith "AMFI: " and eventMessage contains " adhoc ") or (process == "amfid" and eventMessage contains "signature"))'
```

Filtering the log data using "(process == "kernel" AND composedMessage BEGINSWITH "AMFI: " AND composedMessage CONTAINS " adhoc ") OR (process == "amfid" AND composedMessage CONTAINS "signature")"

Skipping info and debug messages, pass --info and/or --debug to include.

```
2022-01-19 16:06:09.001258+0900 0x3753 Default 0x0 kernel: (AppleMobileFileIntegrity) AMFI: '/Users/macforensics/Downloads/SysJoker/types-config.ts' is adhoc signed.
2022-01-19 16:06:09.002729+0900 0x1c8c Default 0x0 amfid: /Users/macforensics/Downloads/SysJoker/types-config.ts signature not valid: -67050
```

The executed program has an adhoc signature.

The executed program has an invalid signature.
Investigating Unified Logs (11/13)

- Program Execution History (7)
  - Deny execution by security policy
  - macOS 10.15

```bash
% log show --predicate 'eventMessage contains "Security policy would not allow process"'
Filtering the log data using "composedMessage CONTAINS "Security policy would not allow process""
Skipping info and debug messages, pass --info and/or --debug to include.
Timestamp                       Thread     Type        Activity             PID    TTL
2022-01-12 02:35:12.569186+0900 0xa980     Default     0x0                  0      0    kernel: (AppleSystemPolicy) Security policy
would not allow process: 822, /Users/macforensics/Downloads/floss
```

Program refused to run

- macOS 11.0.1 - 12.0.1

```bash
% log show --info --debug --predicate 'eventMessage contains "Security policy would not allow process"'
Filtering the log data using "composedMessage CONTAINS "Security policy would not allow process""
Timestamp                       Thread     Type        Activity             PID    TTL
2021-08-20 17:26:24.667681+0900 0x1b6ba    Default     0x0                  0      0    kernel: (AppleSystemPolicy) Security policy
would not allow process: 2954, /Users/macforensics/Downloads/floss
```

You can search using the same criteria as macOS 10.15, but the message formatting will be slightly different.
Investigating Unified Logs (12/13)

- **Volume Mount (1)**
  - macOS 10.15 - 12.0.1
  - HFS+

```bash
% log show --info -debug --predicate 'process == "kernel" AND (eventMessage CONTAINS[cd] "mounted" OR eventMessage CONTAINS[cd] "unmount")'
```

Filtering the log data using "process == "kernel" AND (composedMessage CONTAINS[cd] "mounted" OR composedMessage CONTAINS[cd] "unmount")"

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Thread</th>
<th>Type</th>
<th>Activity</th>
<th>PID</th>
<th>TTL</th>
<th>Volume name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022-01-08 01:06:05.705926+0900</td>
<td>0x5d2a6</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>kernel: (HFS) hfs: mounted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Script Debugger</td>
<td></td>
<td></td>
<td>Script Debugger 8.0 on device disk4s2</td>
</tr>
<tr>
<td>2022-01-08 01:06:12.082076+0900</td>
<td>0x5d4e9</td>
<td></td>
<td></td>
<td>0</td>
<td>0</td>
<td>kernel: (HFS) hfs: unmount</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Script Debugger</td>
<td></td>
<td></td>
<td>initiated on device disk4s2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The message contains "mounted" or "unmount".

**Volume name**

**mounted**

**unmount**
Investigating Unified Logs (13/13)

- Volume Mount (2)
  - macOS 10.15 - 12.0.1
  - APFS (same filtering conditions as HFS+)

% log show --info -debug --predicate 'process == "kernel" AND (eventMessage CONTAINS[cd] "mounted" OR eventMessage CONTAINS[cd] "unmount")'

Filtering the log data using "process == "kernel" AND (composedMessage CONTAINS[cd] "mounted" OR composedMessage CONTAINS[cd] "unmount")"

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Thread</th>
<th>Type</th>
<th>Activity</th>
<th>PID</th>
<th>TTL</th>
</tr>
</thead>
<tbody>
<tr>
<td>2022-01-08 01:04:48.91752+0900 0x5cfc7</td>
<td>Default</td>
<td>0x0</td>
<td>0 0</td>
<td>kernel: (apfs) apfs_vfsop_unmount:2441: disk1: unmounting volume com.apple.TimeMachine.2022-01-08-000409.local</td>
<td>0x0</td>
</tr>
<tr>
<td>2022-01-08 01:04:48.91778+0900 0x5cfc7</td>
<td>Default</td>
<td>0x0</td>
<td>0 0</td>
<td>kernel: (apfs) apfs_vfsop_unmount:2733: snapshot deletion completed on the livefs</td>
<td>0x0</td>
</tr>
<tr>
<td>2022-01-08 01:04:48.91782+0900 0x5cfc7</td>
<td>Default</td>
<td>0x0</td>
<td>0 0</td>
<td>kernel: (apfs) apfs_vfsop_unmount:2798: nx_num_vols_mounted is 5</td>
<td>0x0</td>
</tr>
<tr>
<td>2022-01-08 01:04:48.91788+0900 0x5cfc7</td>
<td>Default</td>
<td>0x0</td>
<td>0 0</td>
<td>all done.  going home. (numMountedAPFSVolumes 44)</td>
<td>0x0</td>
</tr>
<tr>
<td>2022-01-08 01:07:39.91784+0900 0x5d869</td>
<td>Default</td>
<td>0x0</td>
<td>0 0</td>
<td>kernel: (apfs) apfs_vfsop_mount:2234: disk5s1: mounted volume: FakeTest2-bash</td>
<td>0x0</td>
</tr>
<tr>
<td>2022-01-08 01:07:45.86595+0900 0x5d9f4</td>
<td>Default</td>
<td>0x0</td>
<td>0 0</td>
<td>kernel: (apfs) apfs_vfsop_unmount:2441: disk5: unmounting volume FakeTest2-bash</td>
<td>0x0</td>
</tr>
</tbody>
</table>

Local snapshots can be ignored.

Mount or unmount

Volume name

file system
3 Implementation of ma2tl
ma2tl implementation policy (1/2)

- **mac_apt to timeline → ma2tl**
- **Support for macOS 10.15 or later**
- **Automate the verification procedure for each of the activities mentioned above.**
  - If the analysis result has timestamps, create events from the main data.
    - **mac_apt.db : SpotlightShortcuts**
  - If the analysis result does not have timestamps, create events by associating it with a table of relevance.
    - **mac_apt.db : AutoStart + APFS_Volumes_xxxx.db**
    - **mac_apt.db : Safari + Quarantine**
  - Filtering UnifiedLogs.db to extract necessary information from messages
ma2tl implementation policy (2/2)

- Implement analysis plugins for each type of activity.
  - Activities may be recorded across multiple analysis results, and information needs to be integrated to be output as a timeline.
  - If you need a new activity, just add a new plugin

- Replace Unified Logs event messages with content whose meaning is easy to understand.

- Specify the timeline time range manually.
  - I don't want a super timeline, but a minimum timeline that can be used as a starting point for investigation.
  - Specify the range of dates and times that the forensic analysts are interested in.
Configuration of ma2tl

- Analysis results of mac_apt
  - mac_apt.db
  - UnifiedLogs.db
  - APFS_Volumes_xxxx.db

- [Plugins].
  - Program Execution
  - Persistence
  - File Download
  - Volume Mount

Invoke ma2tl

Output ma2tl result

- SQLite
- XLSX
- TSV
## Plugin implementation example: File download

### Table 1: File Download Events

<table>
<thead>
<tr>
<th>TimeStamp</th>
<th>AgentName</th>
<th>DataUrl</th>
<th>OriginUrl</th>
<th>Other_Info</th>
</tr>
</thead>
</table>

```python
# function to extract file download events
def extract_safari_quarantine_file_download(basic_info, downloaded_events):
    sql = 'SELECT Quarantine.TimeStamp, Quarantine.AgentName, Quarantine.DataUrl, Quarantine.OriginUrl, Safari.Other_Info FROM Quarantine
            INNER JOIN Safari ON Safari.Type = "DOWNLOAD" AND Quarantine.DataUrl = Safari.URL ' +
            'WHERE Quarantine.TimeStamp BETWEEN "{}" AND "{}" AND 
               Quarantine.AgentName = "Safari" ' +
            'ORDER BY TimeStamp', \'format\'(start_ts, end_ts)

    run_query = basic_info.mac_apt_dbs.run_query
    start_ts, end_ts = basic_info.get_between_dates_utc()

    for row in run_query(MacAptDBType.MACAPT_DB, sql):
        skin_flag = False
        ts = row[\'TimeStamp\']
        data_url = row[\'DataUrl\']
        origin_url = row[\'OriginUrl\']
        local_path = row[\'Other_Info\']
        agent = row[\'AgentName\']

        if event.data_url == data_url and event.local_path == local_path and get_timedelta(event.ts, ts) <= datetime.timedelta(seconds=1):
            skip_flag = True
            break

    if not skip_flag:
        downloaded_events.append(FileDownloadEvent(ts, data_url, origin_url, local_path, agent))

    return True
```

1) Configure the information required for the file download events from the Safari and Quarantine tables in mac_apt.db.

2) Extract the data.

3) Determine the duplication.

4) Add to timeline.
Plugin implementation example: Volume mount

1) Filter the logs of volume mounts.

2) Extracting volume name from messages using regular expressions
Execution example

- The path where the results of mac_apt analysis are stored.
- Output destination for ma2tl
- Time range of the timeline to be generated

```
% python ./ma2tl.py -i ~/Documents/test -o ../ma2tl_output/test -s '2022-01-13 11:00:00' -e '2022-01-13 11:59:59' ALL
```

Output path: /Users/macforensics/Documents/GitHub/ma2tl_output/test

MA2TL-INFO-Command line: ./ma2tl.py -i /Users/macforensics/Documents/test -o ../ma2tl_output/test -s 2022-01-13 11:00:00 -e 2022-01-13 11:59:59 ALL

MA2TL-INFO-Input path: /Users/macforensics/Documents/GitHub/forked/mac_apt_out/test

MA2TL-INFO-Running plugin FILE_DOWNLOAD
MA2TL.PLUGINS.FILEDOWNLOAD-INFO-Detected 1 events.

MA2TL-INFO-Running plugin PERSISTENCE
MA2TL.PLUGINS.PERSISTENCE-INFO-Detected 8 events.

MA2TL-INFO-Running plugin PROG_EXEC
MA2TL.PLUGINS.PROGEXEC-INFO-Detected 2 events.

MA2TL-INFO-Running plugin VOLUME_MOUNT
MA2TL.PLUGINS.VOLUMEMOUNT-INFO-Detected 2 events.
Example of analysis results

- Timeline of Script Debugger downloaded, installed, and run on macOS 11.5.2.

<table>
<thead>
<tr>
<th>Timestamp (UTC)</th>
<th>Activity Type</th>
<th>Activity Description</th>
<th>Plugin name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2020-01-01 00:00:00.000000</td>
<td>Launch Safari</td>
<td>Launch System Preferences from Script Debugger</td>
<td></td>
</tr>
<tr>
<td>2020-01-01 01:00:00.000000</td>
<td>Download a DMG</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 01:00:00.000000</td>
<td>Volume mount</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 02:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 03:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 04:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 05:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 06:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 07:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 08:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 09:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 10:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 11:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 12:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 13:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 14:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 15:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 16:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 17:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 18:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 19:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 20:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 21:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 22:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2020-01-01 23:00:00.000000</td>
<td>Program Execution</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Since it is macOS 11, the Script Debugger first-time startup artifacts are not left behind.
ma2tl demo
ma2tl demo scenario

1) Download a malware downloader disguised as a tool.

2) Run the downloader.

3) Download a malware.

4) Execute the malware and set persistence.
The malware was downloaded using curl, so it is not included in the timeline generated by ma2tl.

The path of the autorun program is not a standard folder.
Future work
Future work

○ Support for more mac_apt analysis results
  ▪ Analysis results with timestamp
  ▪ Analysis results showing timestamps in combination with APFS_Volumes_xxxx.db

○ Ongoing investigation of Unified Logs
  ▪ Application Execution
  ▪ Program refused to be executed by the system
  ▪ exFAT, NTFS, SMB volume mount

○ Optimize the timeline to be generated
  ▪ Eliminate duplicate events
  ▪ Expand the scope of events to include cautionary messages.

○ Maintenance
  ▪ Will newer versions of macOS still record log messages that ma2tl can recognize?
5 Summary
Summary

- Shared how to create a timeline from mac_apt analysis results and Unified Logs.
- Introduced the implementation and function of ma2tl.
  - Automatic generation of timeline from mac_apt analysis results and Unified Logs
  - More activities can be supported by plugins.
- ma2tl GitHub repository
  - https://github.com/mnrkbyys/ma2tl
Thank you for listening!

Any questions?
CREDITS for this presentation template and Icons

Special thanks to all the people who made and released these awesome resources for free:
- Presentation template by SlidesCarnival