How Far Can We Push AI for Detection?

Martina Tivadar - #OFTW version 3.0 - London

- Macs are everywhere
- Not only are there more attacks, but they’re faster too
- The same way we can use AI to detect malicious actor can use it to create

Objective by the Sea v7.0

Part 1

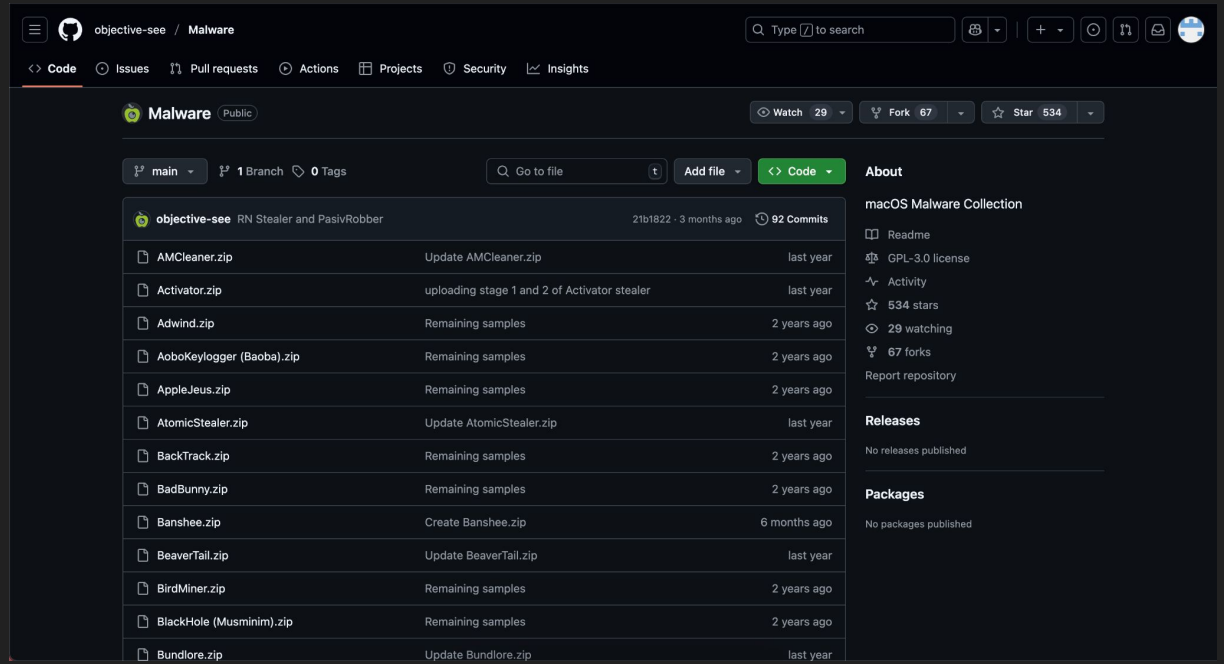
Let’s create a solution which detects macOS malware using LLMs

To do that we need some

data!!

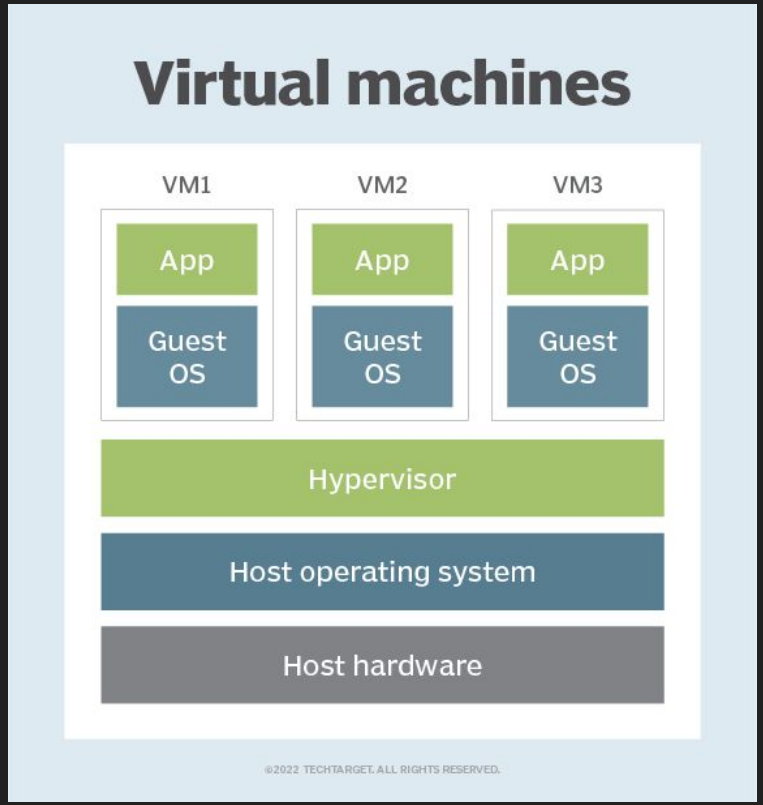
Data collection

github.com/objective-see/Malware



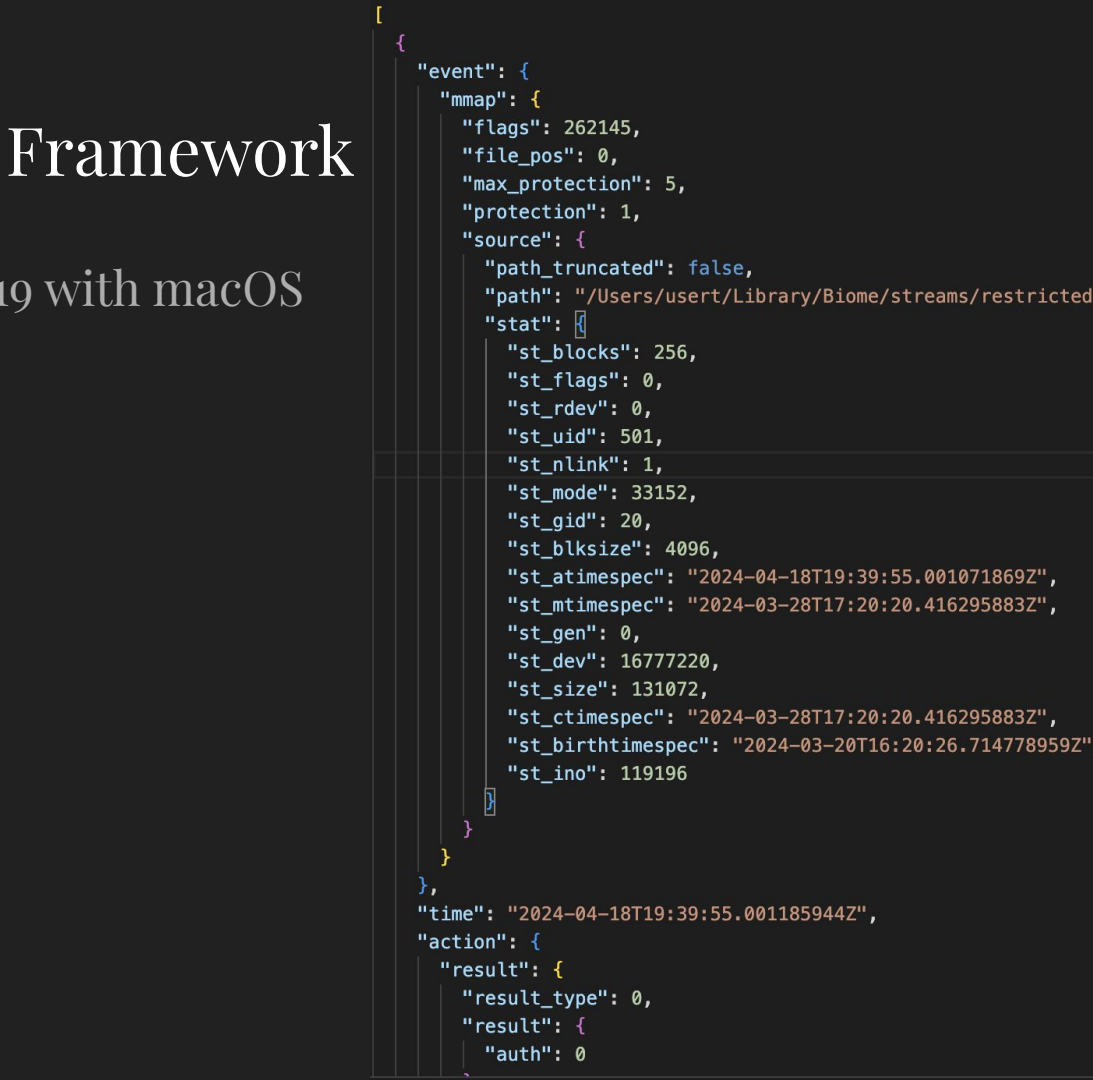
If you want to run malware on your computer:

Use a VM!



Apple Endpoint Security Framework

- first released in October of 2019 with macOS Catalina
- part of the operating system
- TCC events (march 2025)
  Transparency, Consent, and Control



We can capture events like:

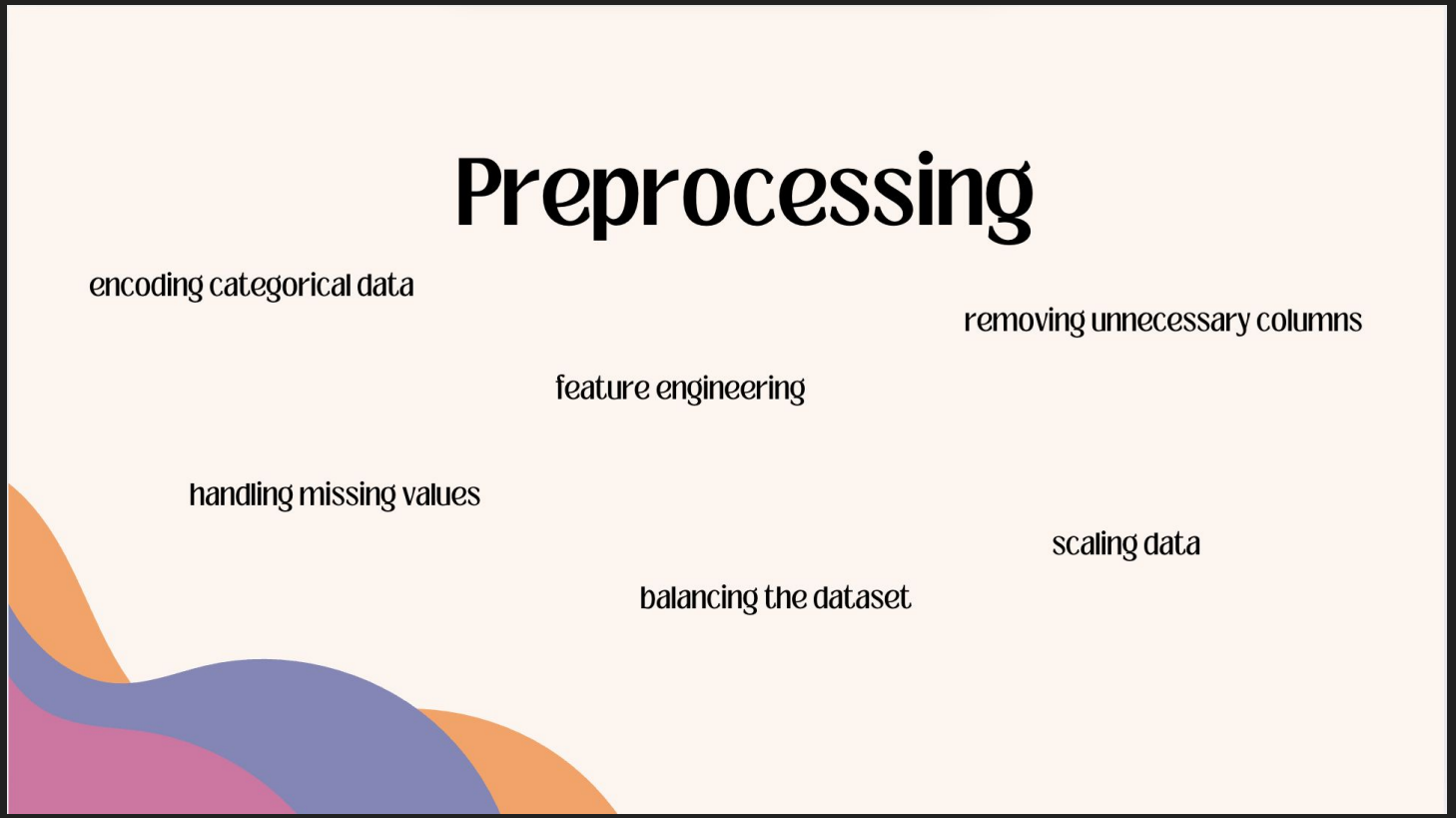
- When programs launch (and what they are)
- When files are opened, created, or deleted
- When disks or volumes are mounted
- Signals sent between processes
- Code-signing information for running software

```
sudo eslogger exec create rename unlink tcc_modify … > events.json
```

Also collected benign data

Data preprocessing

Classic machine learning models



LLMs

Problem: The logs are too big for API to handle (token limit).

Keep only essential data

- Focus on key fields
- Discard low-value details

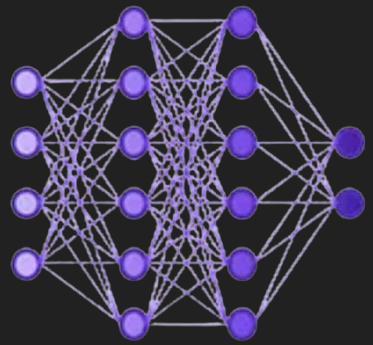
Prioritize recent activity

- Include the most recent events first, staying within the token limit
- Keep latest actions, where malicious behavior is most likely to occur

Large Language Models

LLMs recognize patterns by turning words into numbers and using neural networks to learn which words and phrases often go together.

Objective for the We is



word	probability
awesome	0.99
fun	0.95
...	...

The models tested

- **o4-mini** – smaller, faster, reasoning
- **GPT-4o** – fast, general-purpose
- **o1** – deep reasoning, slower

ChatGPT models are cloud-based, but we can run models locally with tool like LMstudio. 

Prompt engineering

A technique used to effectively communicate with LLMs and get the desired output.

6 prompt types

- **Zero-shot** – when we don't give any extra instruction to the LLM
- **One-shot** – we give the model an example
- **Few-shot** – we give the model a couple of examples so it sees the pattern
- **Chain-of-thought** – give the LLM step by step instructions how to think
- **Negative** – we tell the model what not to do
- **Hybrid** – a combination of two or more types

Zero-shot prompt

```
You are analyzing JSON logs collected using Apple’s Endpoint Security
Framework API. These logs describe system-level events such as
process executions and file operations.

Your task is to determine if the JSON file indicates malicious
activity.

Respond with:
1. "Yes" or "No" — whether malicious activity is present.
2. A very short explanation explaining your reasoning.

Here is the JSON to analyze:
```

One-shot prompt

You are analyzing JSON logs collected using Apple's Endpoint Security Framework API. These logs describe system-level events such as process executions and file operations.

Your task is to determine if the JSON file indicates malicious activity.

Use the following sign of compromise as a reference:
- A process spawning from `/tmp`, `/var/tmp`, or `/Users/Shared`, especially if unsigned or unknown.

Respond with:
1. "Yes" or "No" — whether malicious activity is present.
2. A very short explanation explaining your reasoning.

Here is the JSON to analyze:

Few-shot prompt

. . .

```
Use the following signs of compromise as reference:
- Unsigned processes launching from suspicious locations like `/tmp`,
`/var/tmp`, `/Users/Shared`, or hidden directories.
- Processes injecting code into other processes.
- A process rapidly creating or modifying many files.
- Execution of tools commonly abused by malware (e.g., `osascript`, `curl`,
`bash`, `python`) with obfuscated or remote arguments.
- Unexpected persistence mechanisms (e.g., LaunchAgents created by unknown
apps).
```

. . .

Chain-of-thought prompt

. . .

```
To do so, follow this step-by-step reasoning:
1. Identify which processes are involved and where they were executed from.
2. Check if the process is signed or known.
3. Check if any known suspicious patterns appear (code injection, persistence,
excessive file access).
4. Cross-check the behavior with common malware tactics.
5. Make a judgment if the behavior shown is consistent with malicious
activity.
```

. . .

Negative prompt

. . .

Your task is to **avoid** falsely labeling benign activity as malicious.

Do not label anything as malicious unless you have a strong reason to do so based on the data.

Respond with:
1. "Yes" or "No" — **only answer "Yes"** if the behavior clearly indicates compromise.
2. A very short explanation explaining your confidence.

Hybrid (few-shot and negative) prompt

. . .

Your task is to determine if the JSON file indicates malicious activity. Base your decision on known signs of compromise, **but avoid over-flagging benign activity.**

Use these signs of compromise:
- Unsigned or unknown processes running from suspicious locations
- Code injection or suspicious process relationships
- Use of known living-off-the-land binaries (`curl`, `osascript`, etc.)
- Attempts to establish persistence via system launch agents or daemons

Reason step by step and be conservative in labeling suspicious activity.

Respond with:
1. "Yes" or "No" — **only answer "Yes"** if the behavior clearly indicates compromise.
2. A very short explanation explaining your confidence.

Evaluation

Tried every prompt type with every model

6 x 3 x all samples

Confusion matrix

True positive (TP)	**False negative (FN)**
False positive (FP)	**True negative (TN)**

True Positive – caught malware correctly

False Positive – false alarm (benign but flagged)

False Negative – missed malware

True Negative – correctly ignored benign log

Metrics

- **Accuracy** - overall correctness
- **Precision** - actual correctness
- **Recall** - how many malwares we caught
- **F1** - balanced score (precision + recall)

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}$$

Results

Zero-shot prompting worked best overall

It caught all malware (100% recall) and only a few false alarms

Chain-of-thought and few-shot prompts also did very well

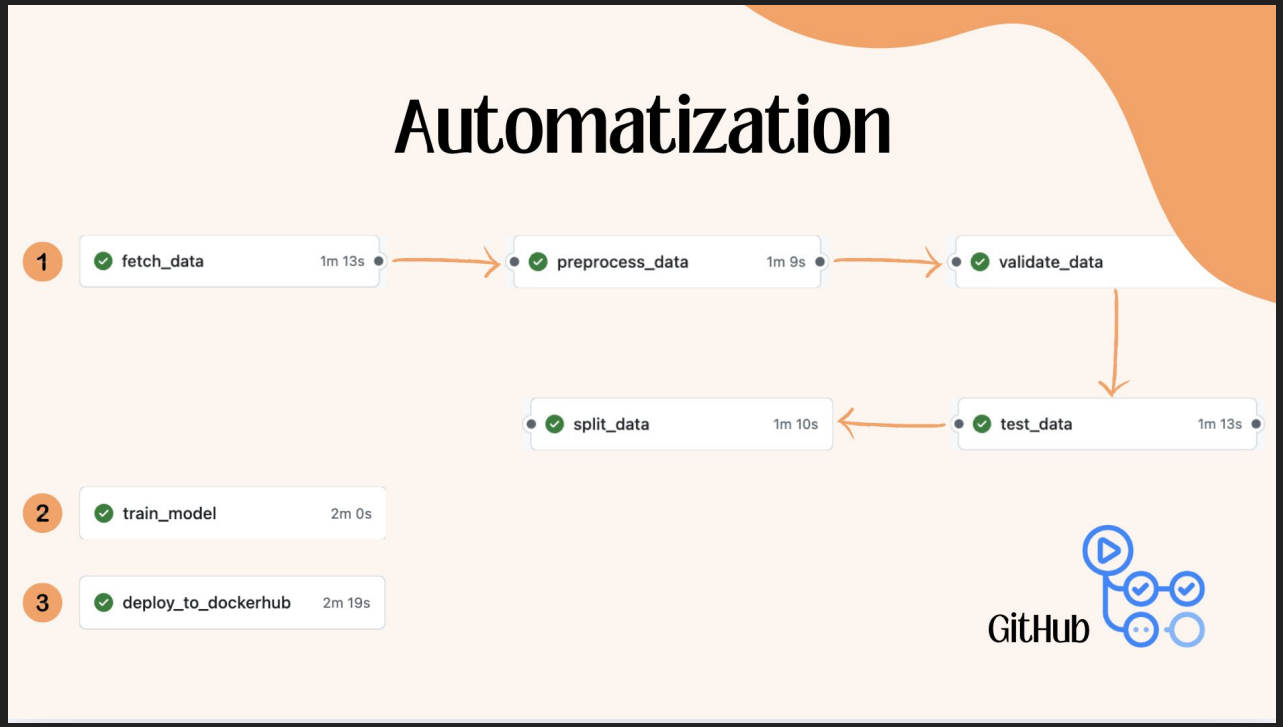
Encouraging the model to "think step by step" helps

One-shot and negative prompts did worse

Missed many attacks but were cautious (few false alarms)

How can we use this?

It can be an additional layer to our existing system




data.yaml
on: push
fetch_data
19s
preprocess_data
9s
analyze_data
15s

Takeaways

- Even simple prompts **("zero-shot")** worked very well here
- More guidance helps, but not always better than a simple question
- Cautious prompts lower false alarms but miss real attacks
- Bigger, more expensive models aren't always better: **GPT-4o** and **o1** gave same results, but **GPT-4o** was cheaper
- For best balance of accuracy and cost: Use **GPT-4o**
- For cheap bulk triage: Use **o4-mini** (accepting some misses)

Takeaways

- Don’t fully rely on AI
- Be careful with sensitive data

Thank you!

Q&A