

E2E processing of malware samples using open source technologies

Pass the Salt - July 2025



DATADOG

whoami(we)



Matt Muir
Security Researcher



Fred Baguelin
Security Researcher

Agenda

01 What is Malhound?

02 Leveraging FOSS for malware analysis

03 Analysing Skidmap

04 Analysing a malicious model - live MalHound demo

05 Conclusion

What is MalHound?

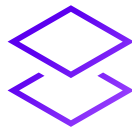
Problem Statement(s)



- We routinely collect and analyse large volumes of malware samples



- Traditional sandbox solutions are expensive and limited in a cloud-first world



- We need a mechanism for quickly extracting IoCs so we can deliver threat intel to customers

What do we currently have?



YETI

Open Source TIP

- Highly-extensible
- Helps having a contributor on the team 😊
- Ingests all honeynet data

What do we currently have?



Workload Protection Agent

eBPF Security Agent

- Excellent for monitoring Linux hosts and containers
- Works well in the environments our customers use
- Malware cannot easily evade kernel-level monitoring

What do we currently have?



HoneyNet

Multi-honeypot Infrastructure

- Heavily based on Docker
- Incoming feed of malware samples/malicious images etc
- Allows us to gather cloud-specific threat intel

What do we currently have?



AssemblyLine4

Malware Analysis Pipeline

- Developed by the Canadian CERT
- Processes queues of files and hands them off to other services for analysis

💡 Maybe we can unite these
technologies 💡

Leveraging FOSS for Malware Analysis

Assemblyline4

Front-end to pipeline

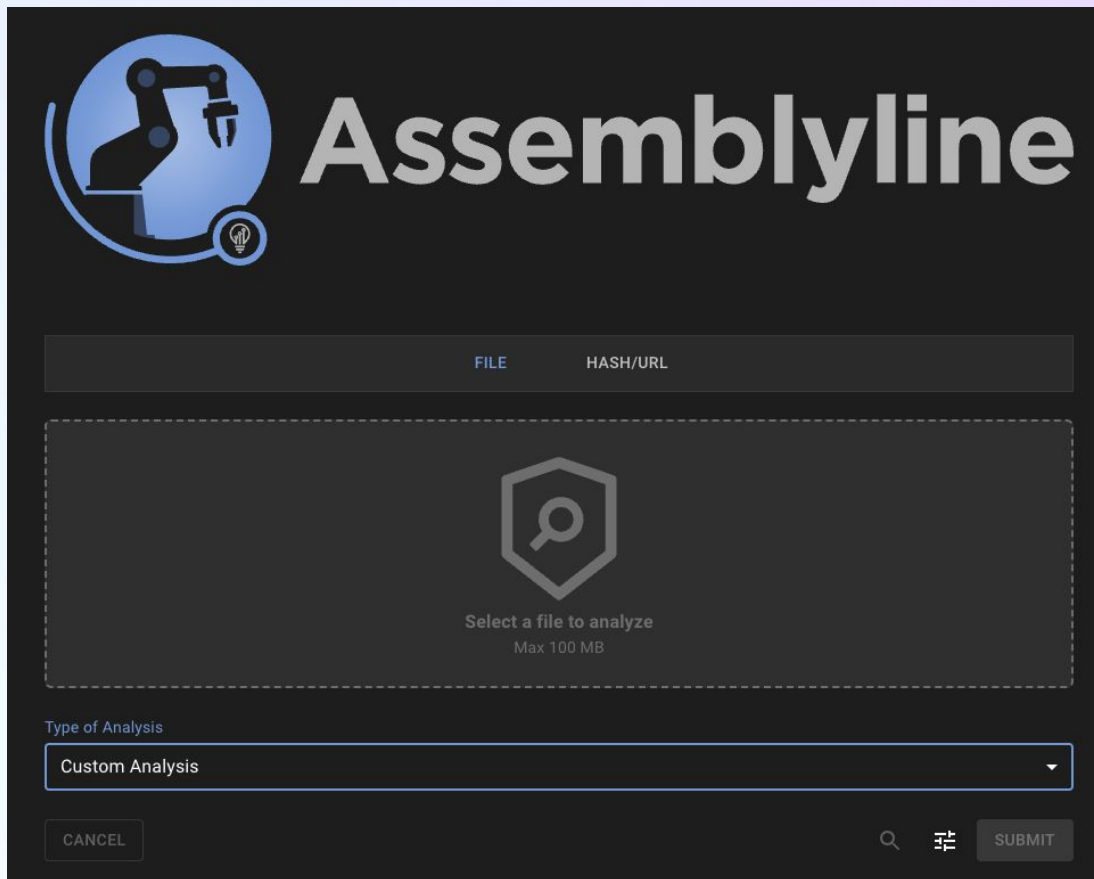
- Samples/models submitted via a web UI

Services

- Jobs in the pipeline
- Static/dynamic
- Written in Python

Conveyor-belt design

- Services executed in series
- Unique ID added to file, identify during processing



The screenshot shows the Assemblyline4 web interface. At the top left is a logo featuring a blue robot arm holding a document, with a lightbulb icon below it. To the right of the logo, the word "Assemblyline" is written in a large, bold, white sans-serif font. Below the header, there is a dark gray rectangular area with two tabs: "FILE" and "HASH/URL". The "FILE" tab is selected. In the center of this area is a large dashed rectangular box containing a light gray shield icon with a magnifying glass inside. Below the shield icon, the text "Select a file to analyze" is displayed, followed by "Max 100 MB". Below this dashed box is a label "Type of Analysis" and a dropdown menu currently showing "Custom Analysis". At the bottom left of the interface is a "CANCEL" button. At the bottom right are a search icon, a filter icon, and a "SUBMIT" button.

**Could we use AL4 and our
eBPF agent in combination as
a “sandbox”?**

MalHound

Detonate malicious samples

from ELF to models or docker images

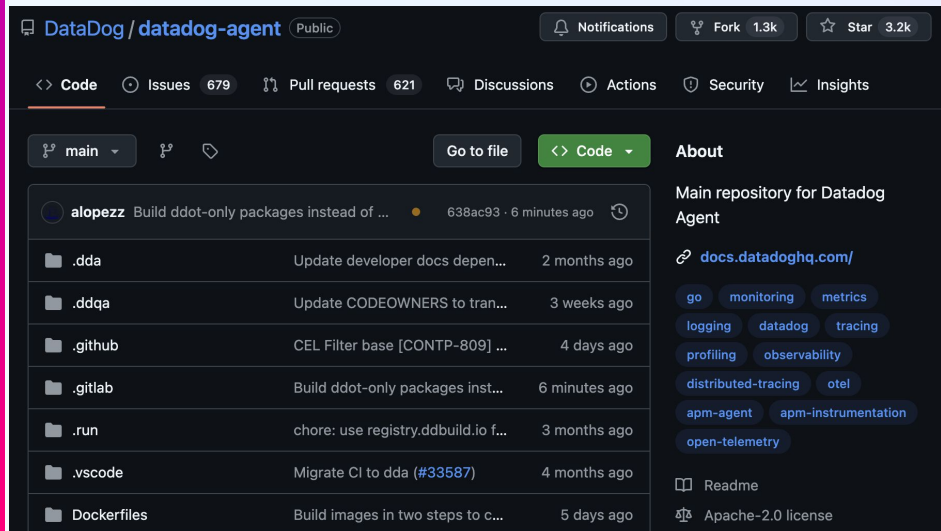
Rely on docker containers as execution unit

Automates all the burden of old school sandbox

Use Datadog open source agent

To profile processes activities (files, netflows, syscalls, ...) and applications

```
2025-02-07 15:41:27,178 - detonator - INFO - Container cws-agent already running (id: 9f2c41cd59b).
2025-02-07 15:41:27,220 - detonator - INFO - Starting malhound execution...
2025-02-07 15:41:27,220 - detonator - INFO - Bootstrapping BaseSandbox sample detonation
2025-02-07 15:41:27,306 - detonator - INFO - Container cws-agent already running (id: 9f2c41cd59b).
2025-02-07 15:41:57,307 - detonator - INFO - Initializing detonator container
2025-02-07 15:41:58,968 - detonator - INFO - Container c340e3d3ae7f769b4e88204dd08aa0f7b0145dffafe164d8e09c39b5a6
2025-02-07 15:41:59,260 - detonator - INFO - Container c340e3d3ae7f769b4e88204dd08aa0f7b0145dffafe164d8e09c39b5a6
2025-02-07 15:41:59,261 - detonator - INFO - Bootstrapping container: c340e3d3ae7f769b4e88204dd08aa0f7b0145dffafe164d8e09c39b5a6
2025-02-07 15:41:59,261 - detonator - INFO - Pulling image ubuntu:latest
2025-02-07 15:42:00,771 - detonator - INFO - Pulling from library/ubuntu
2025-02-07 15:42:01,179 - detonator - INFO - Pulling fs layer 5a7813e071bf...
2025-02-07 15:42:02,082 - detonator - INFO - Verifying Checksum
2025-02-07 15:42:02,082 - detonator - INFO - Download complete for ubuntu:latest. Extracting...
2025-02-07 15:42:04,972 - detonator - INFO - Pull complete
2025-02-07 15:42:05,103 - detonator - INFO - Digest: sha256:72297848456d5d37d1262630108ab308d39ec7ed1c3286a32fe0
2025-02-07 15:42:05,148 - detonator - INFO - Status: Downloaded newer image for ubuntu:latest
2025-02-07 15:42:07,520 - detonator - INFO - Container c340e3d3ae7f769b4e88204dd08aa0f7b0145dffafe164d8e09c39b5a6
2025-02-07 15:42:08,010 - detonator - INFO - Starting container c340e3d3ae7f769b4e88204dd08aa0f7b0145dffafe164d8e09c39b5a6
2025-02-07 15:42:09,034 - detonator - INFO - Container c340e3d3ae7f769b4e88204dd08aa0f7b0145dffafe164d8e09c39b5a6
running.
2025-02-07 15:42:10,348 - detonator - ERROR - Error while running pre-run script: b'\nWARNING: apt does not have
ate;\n'
2025-02-07 15:42:10,431 - detonator - INFO - Stopping activity dump for 5dcd54a9008553cd4c7e954d3c551ceddd0546t
2025-02-07 15:42:11,769 - detonator - INFO - Waiting 5 seconds for dump termination...
2025-02-07 15:42:26,770 - detonator - INFO - Listing activity dumps...
2025-02-07 15:42:28,081 - detonator - INFO - Running command ./c340e3d3ae7f769b4e88204dd08aa0f7b0145dffafe164d8e0
39b5a6d0d7cb
2025-02-07 15:42:28,081 - detonator - INFO - Let container c340e3d3ae7f769b4e88204dd08aa0f7b0145dffafe164d8e09c39
2025-02-07 15:42:28,081 - detonator - INFO - [SAMPLE OUTPUT START]
```



docker



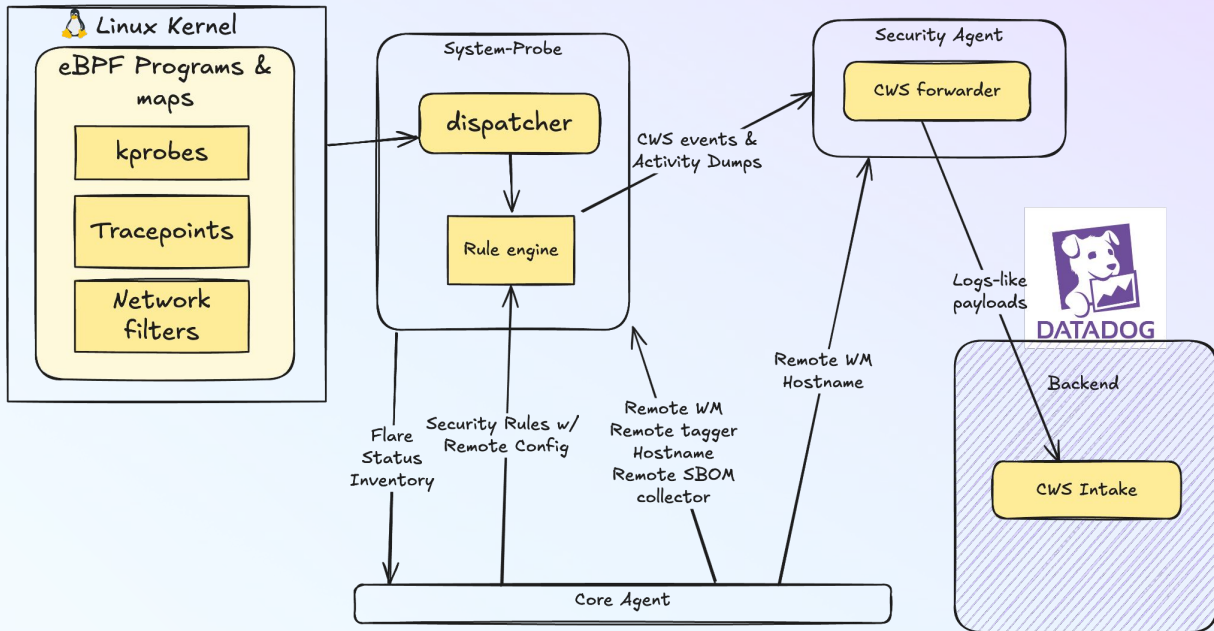
Datadog agent 101

Low level tracers

- eBPF (kernel land)
- System-probe (userland)

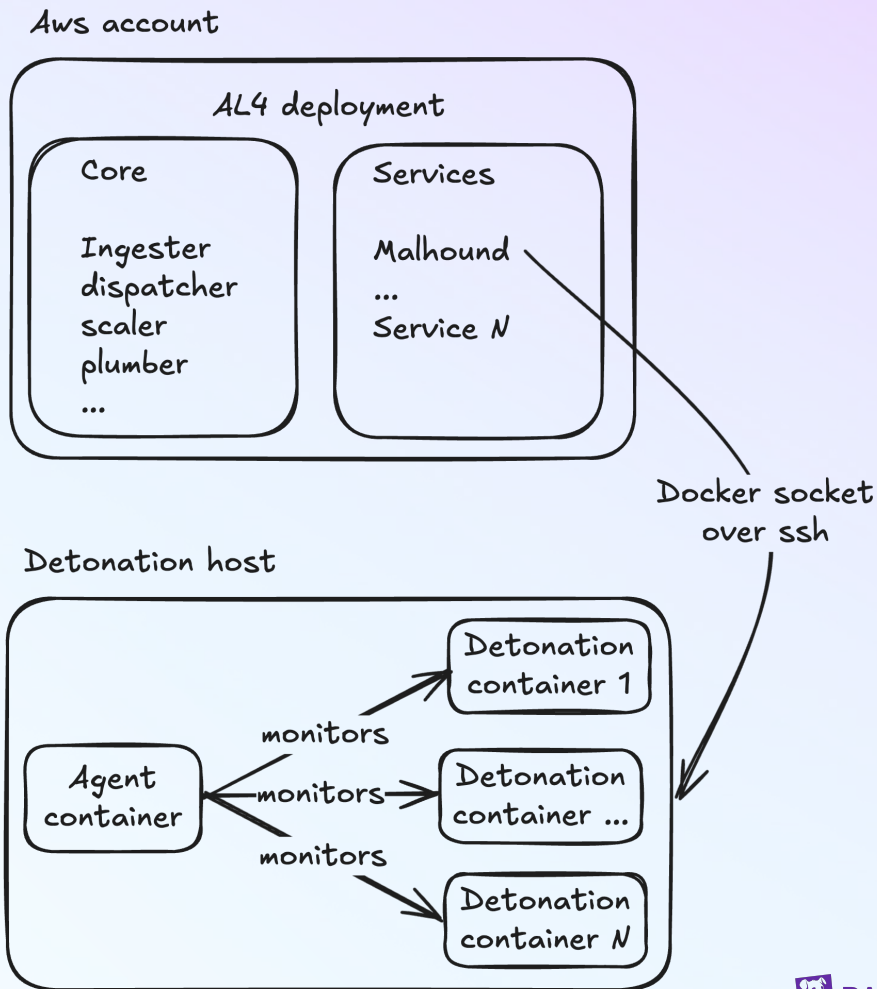
Application tracers

- Profile application for several programming languages
- Support runtime injection for Python or Nodejs for example
- Compatible with OTel



Uniting AL4 & Malhound

- Rely on AL4 to handle storage / analysis / UI / scalability
- Add Malhound as a dynamic analysis service
- Renders Malhound results in AL4:
 - IOCS
 - Process tree
 - Syscalls table
 - Files table
 - Netflows table



How does this look to an analyst?

Custom submission parameters

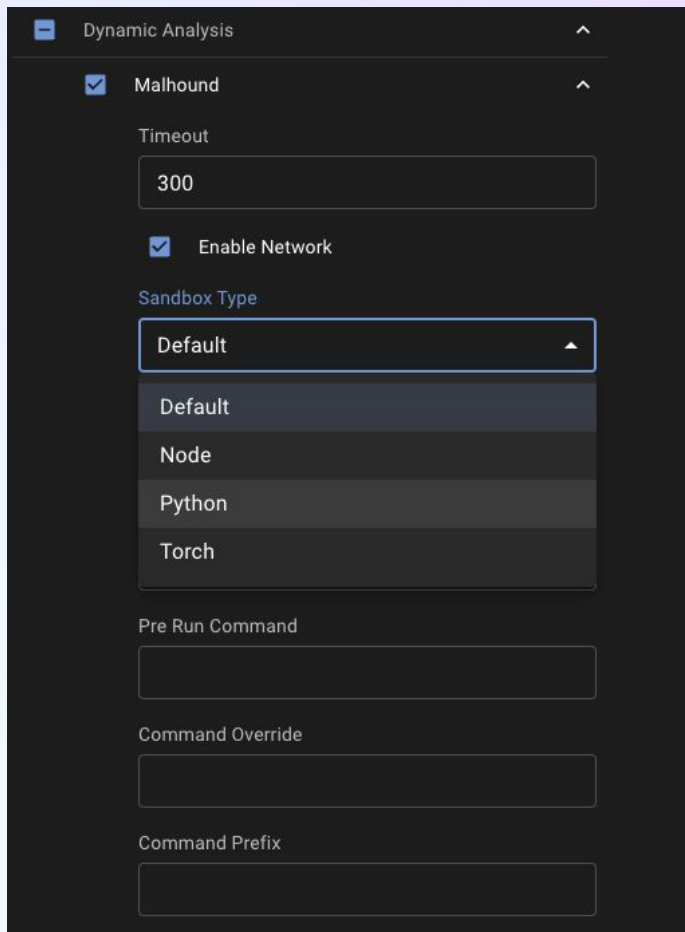
- Execute sample with argument
- Override container start command
- Configure networking/execution timeout
- Apply custom policies and filtering expressions

The screenshot shows the 'Dynamic Analysis' submission form in a dark-themed interface. At the top, there are two tabs: 'FILE' and 'HASH/URL', with 'HASH/URL' being the active tab. Below the tabs, the 'Hash/URL to Analyze' field contains the SHA256 hash '20b4d8894898768c7c3516db97dee9646508a...' with a 'SHA256' label. A note below states 'A valid URL requires a scheme, i.e. https://www.example.com'. The 'Type of Analysis' dropdown is set to 'Custom Analysis'. Under 'Select the following external sources', both 'MalwareBazaar' and 'VirusTotal' are checked. At the bottom left are 'CANCEL' and 'SUBMIT' buttons, with a search icon and a filter icon between them. On the right side, the 'Dynamic Analysis' section is expanded, showing several configuration options: 'Mailhound' is checked, 'Timeout' is set to '300', 'Enable Network' is checked, 'Sandbox Type' is set to 'Default', 'Detonation Image' is set to 'Detonator:Latest' with a refresh icon, 'Workdir' is set to '/', 'Pre Run Command' is empty, 'Command Override' is empty, 'Command Prefix' is empty, 'Command Arguments' is empty, and 'Post Run Command' is empty.

Detonation Environments

Environment is configurable

- Support for ELF, Python and Node scripts along with PyTorch models
- Guarddog - requirement to analyse heavily-obfuscated nodeJS and Python scripts
- Can easily modify container to introduce new detonation environments



The image shows a configuration window titled "Dynamic Analysis" for a tool called "Malhound". It contains several settings:

- Malhound** (checked)
- Timeout**: A text input field containing the value "300".
- Enable Network** (checked)
- Sandbox Type**: A dropdown menu currently showing "Default". The dropdown list is open, showing four options: "Default", "Node", "Python", and "Torch".
- Pre Run Command**: An empty text input field.
- Command Override**: An empty text input field.
- Command Prefix**: An empty text input field.



Malhound AL4 output

Rely on activity dumps

- Displays process tree
- Can test detection engineering rules
- Displayed in Datadog for slide but output from MalHound is all JSON
- Easily trace malware's behaviour
- Similar to a sandbox, but tracing done via eBPF or a profiler

The screenshot displays the Malhound web interface, which is a dark-themed dashboard for analyzing malware submissions. The left sidebar contains navigation links: Assemblyline, Submit, Submissions, Alerts, Search, Dashboard, Manage, Administration, and Help. The main content area is divided into several sections:

- File Identification:** A table showing various file hashes and their corresponding file names. The file name is a long alphanumeric string.
- File Frequency:** A table showing the first and last seen times of a file, along with its count.
- Submissions Summary:** A section showing the submitter (4x admin) and the submission details.
- Service Results:** A section showing the execution tree of the submission.

The execution tree shows the following process flow:

- 292057 **sh** (PID 292057) with command `/usr/bin/dash`
- 292233 **bash** (PID 292233) with command `/usr/bin/bash`
- 292240 **python** (PID 292240) with command `/usr/bin/python3.12 -c import torch; torch.load('./9cd3d63baa139794dbc415d3d4848844702c43b001a835598d4af578f36e484f', weights_only=False)`

Analysing Skidmap

Skidmap Overview



Cryptojacking malware targeting Linux

- Originally discovered in 2019 by [Trend Micro](#)



Hides mining activities using malicious kernel modules (rootkits)

- `getdents()` hooking - hide dropped files
- Fakes CPU and network usage statistics to conceal mining



Constantly evolving family, new variants frequently discovered

- [Trustwave](#) variant in 2023 - Redis targeting
- We notice Skidmap activity in our own honeypot in Q2 2025

Skidmap Analysis

selinuxdefconed

- UPX-packed x86-64 ELF
- Primary payload for 2025 campaign
- Multiple embedded payloads
- Malformed UPX header
- upx_dec works for the “parent” but not the rest
- Could extract each payload and unpack but let’s use MalHound!

0000000000405060	20 D0 B4 00 00 00 00 00	28 D0 B4 00 00 00 00 00	77.....(d.....
0000000000405070	30 D0 B4 00 00 00 00 00	38 D0 B4 00 00 00 00 00	0d'.....8d'.....
0000000000405080	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
0000000000405090	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000004050A0	61 00 00 00 20 00 00 00	63 00 00 00 04 00 00 00	a...'.c.....
00000000004050B0	64 00 00 00 40 00 00 00	65 00 00 00 00 00 08 00	d...@...e.....
00000000004050C0	69 00 00 00 10 00 00 00	6A 00 00 00 00 40 00 00	i.....j.....@.
00000000004050D0	73 00 00 00 01 00 00 00	74 00 00 00 00 80 00 00	s.....t.....
00000000004050E0	75 00 00 00 02 00 00 00	41 00 00 00 80 00 00 00	u.....A.....
00000000004050F0	44 00 00 00 00 00 01 00	54 00 00 00 00 00 02 00	D.....T.....
0000000000405100	7F 45 4C 46 02 01 01 00	00 00 00 00 00 00 00 00	.ELF.....
0000000000405110	02 00 3E 00 01 00 00 00	60 10 3A 02 00 00 00 00	..>.....`.....
0000000000405120	40 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	@.....
0000000000405130	00 00 00 00 40 00 38 00	03 00 00 00 00 00 00 00	@.8.....
0000000000405140	01 00 00 00 06 00 00 00	00 00 00 00 00 00 00 00	
0000000000405150	00 00 40 00 00 00 00 00	00 00 40 00 00 00 00 00	..@.....@.....
0000000000405160	00 10 00 00 00 00 00 00	F8 10 8B 01 00 00 00 00	
0000000000405170	00 10 00 00 00 00 00 00	01 00 00 00 05 00 00 00	
0000000000405180	00 00 00 00 00 00 00 00	00 20 CB 01 00 00 00 00	
0000000000405190	00 20 CB 01 00 00 00 00	5D 04 6F 00 00 00 00 00	..'......].o.....
00000000004051A0	5D 04 6F 00 00 00 00 00	00 10 00 00 00 00 00 00	]o.....
00000000004051B0	51 E5 74 64 06 00 00 00	00 00 00 00 00 00 00 00	Q.....
00000000004051C0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000004051D0	00 00 00 00 00 00 00 00	00 00 00 00 00 00 00 00	
00000000004051E0	10 00 00 00 00 00 00 00	2D CE 56 BA 55 50 58 21	-.....UPX!
00000000004051F0	08 14 0E 16 00 00 00 00	28 09 6B 01 C8 35 6A 01	(k...j.....
0000000000405200	20 01 00 00 62 00 00 00	0E 00 00 00 1A 03 00 3F	..b.....?
0000000000405210	91 45 84 68 3D 89 A6 DA	8A CC 93 E2 4E D9 07 94	.E.h=...'......
0000000000405220	1E 01 82 A7 10 29 99 02	7A C8 52 11 70 9F AF 41	).z...p.....
0000000000405230	A0 03 33 E4 19 84 D5 0A	E8 7E DF D9 06 41 7A B9	..3.....Az.....
0000000000405240	A0 9B 7F 37 50 F9 77 D9	BA 25 93 33 22 85 22 23	..~7P.w3.%~".#"#
0000000000405250	1C AB 3F 19 E3 A4 5C BC	99 53 D3 2E 71 52 82 40	..?.....S..qR.@
0000000000405260	AF 6E 82 BE A9 83 2B C9	00 3C 81 DA 76 00 A4 C5	+...<.....
0000000000405270	00 00 E1 62 00 00 0E 49	09 00 1A 03 00 28 16 14	I.....(..
0000000000405280	9D 0D 37 73 36 8F 0B B1	14 EB B1 4D F9 3A F1 60	..7s6.....:..
0000000000405290	62 13 79 06 00 19 2B 51	B6 EB F6 68 F5 F4 B6 79	b.y...+Q.....
00000000004052A0	41 06 93 AD EE 42 F1 EA	7A 0D 5C BF 07 45 69 DC	A.....\..Ei.....
00000000004052B0	B8 46 B5 C4 C1 6E A1 6F	98 DA D6 1C 42 F5 5A 4E	.F...n.o....B...
00000000004052C0	D2 CA 60 24 BA 21 27 6C	2F 2C 99 B8 79 F4 99 2C	..\$.!'l/,...y...
00000000004052D0	E6 5F DF 85 B6 D9 99 16	0E 40 EE E2 1A ED 02 84	@.....
00000000004052E0	6E D2 2C A5 4C E3 AC 91	04 AD A0 29 E7 5C 51 D4	n...L哪'...')....
00000000004052F0	07 22 54 36 F7 FD 67 D6	6C 0A C9 F8 66 BD B8 99	..T6....l...f....

Skidmap Analysis

MalHound Raw JSON Output

- Quickly identify syscalls used by the malware
- Useful for binary triage
- Looks like the binary writes additional payloads

```
"syscall_nodes": [  
  { "image_tags": ["latest"], "syscall": 0 }, // read  
  { "image_tags": ["latest"], "syscall": 1 }, // write  
  { "image_tags": ["latest"], "syscall": 3 }, // close  
  { "image_tags": ["latest"], "syscall": 4 }, // stat  
  { "image_tags": ["latest"], "syscall": 5 }, // fstat  
  { "image_tags": ["latest"], "syscall": 9 }, // mmap  
  { "image_tags": ["latest"], "syscall": 10 }, // mprotect  
  { "image_tags": ["latest"], "syscall": 11 }, // munmap  
  { "image_tags": ["latest"], "syscall": 12 }, // brk  
  { "image_tags": ["latest"], "syscall": 16 }, // ioctl  
  { "image_tags": ["latest"], "syscall": 17 }, // pread64  
  { "image_tags": ["latest"], "syscall": 21 }, // access  
  { "image_tags": ["latest"], "syscall": 87 }, // unlink  
  { "image_tags": ["latest"], "syscall": 90 }, // chmod  
  { "image_tags": ["latest"], "syscall": 158 }, // arch_prctl  
  { "image_tags": ["latest"], "syscall": 218 }, // set_tid_address  
  { "image_tags": ["latest"], "syscall": 231 }, // exit_group  
  { "image_tags": ["latest"], "syscall": 257 }, // openat  
  { "image_tags": ["latest"], "syscall": 318 }, // getrandom  
  { "image_tags": ["latest"], "syscall": 334 }, // rseq  
],  
"network_devices": []
```

Key Insights for Malware Analysis

1. The malware process (PID 140643) shows significantly more syscall diversity - typical of malicious behavior
2. File access is tracked separately - you can correlate file access patterns with syscall usage
3. Syscalls 87 (unlink) and 90 (chmod) in the malware process suggest file manipulation
4. Syscall 257 (openat) indicates modern file opening operations

Skidmap Analysis

Additional payloads identified

- Sample is a loader that writes out the following files:
 - reviews
 - mldconfigs
 - kmod (masquerade)
- Hashes displayed for each
- Additional payloads self-extracted from main binary
- TTP overlap with 2023 variant

```
"image_tags": ["latest"],
"name": "reviews",
"is_pattern": false,
"file": {
  "uid": 0,
  "user": "root",
  "gid": 0,
  "group": "root",
  "mode": 33188,
  "ctime": "1750691192431070471",
  "mtime": "1750691192431070471",
  "mount_id": 2186,
  "inode": "2152064",
  "in_upper_layer": false,
  "path": "/usr/bin/reviews",
  "basename": "reviews",
  "filesystem": "overlay",
  "package_name": "",
  "package_version": "",
  "package_srcversion": "",
  "hashes": [
    "sha1:6306f546bbb787f7aad8c32089262ec984b9e2cc",
    "sha256:8dee19b985b6d50da3a44063c91c6d9dc91640f57e58e31bddcccc90aa5b54c8",
    "md5:6bdb3c97f821e6262626c93e71c4f27c"
  ],
  "hash_state": "DONE"
```


Detonating a Malicious Model - Live Demo

What's next



Open source Malhound and AL4 service

Soon™ 🙄



Improve integration between AL4 and Yeti

Yeti plugin to submit `command_line` to AL4

AL4 plugin to submit selected IOCs to Yeti



Support other tracers and container runtimes

Kunai, Tracee, Falco... And move from default docker runtime to kata container.

Takeaway



Relying on Open Source is important when dealing with sandboxes

You need to understand how things work to be sure you're not missing something.



Cloud environment is not well supported in the sandbox ecosystem, FOSS helps to build your own pipeline.

We needed support for packages, docker images, models, ...



FOSS community is always helpful!

AL4 community was super helpful and the project is very well thought.

Q&A