Threatray – Getting started

Session 1 – Intro and malware classification

THREATRAY

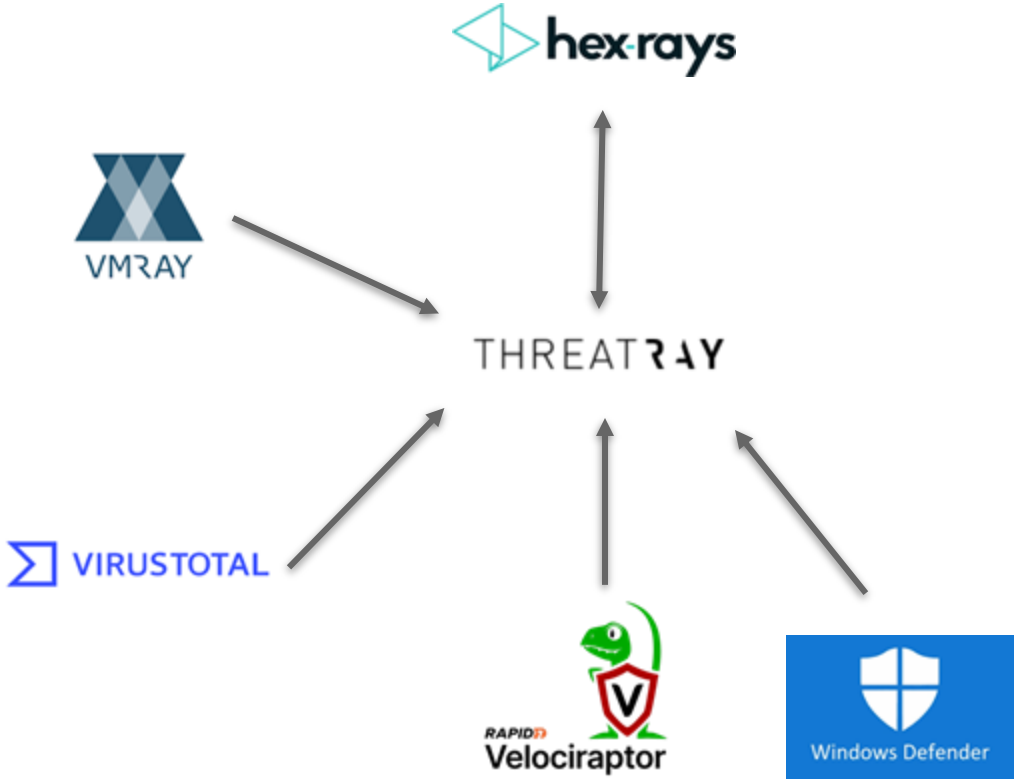
Content

- Threatray background
- File submission and the private organization repository
- Sandboxing view
- Binary intelligence view
- Threatray classification and detection verdict details
- Capability analysis

Threatray background

Threatray overview

- **Analysts Lack Deep-Dive Tooling**. Malware analysts defending against APTs and advanced threats are stuck with “surface-level” tools that remain slow, manual, and incomplete.
- **Code-First Analysis.** Threatray unlock the value of binary code deliver next gen malware defence capabilities beyond traditional hashes, metadata, and dynamic behaviour.
- **A novel type of product** dedicated to code similarity analysis that integrates with existing product stack
- **Truly novel capabilities.** Resilient classification, rapid variant discovery, and precise attribution — accelerating investigations that were previously impossible with existing tooling



The Core of Threatray's Technology

- **Code similarity analysis at scale and precision:** First platform to deliver on the promise of code similarity analysis (sometimes also called "code DNA analysis").

- **Code search engine:** A first-of-its-kind engine that makes binary code instantly searchable at scale. That is, we can match unknown / suspicious code against our database of 100's of millions of malware and benign code fragments.

- **Proven ML Models:** Highly tuned, battle-tested machine learning models are at the core of our engine.

Unique industry-leading capabilities

Classify and understand unknown code fast

Leading malware family classification — Accurately categorizes thousands of families across cybercrime, APTs, and C2 frameworks

Identifies new malware variants that bypass traditional solutions like YARA, AV, and VirusTotal

Industry-leading benign code classification

Code DNA for attribution

Hunting and intelligence

Code Pivoting – Instantly identify malware variants and clusters with point-and-click simplicity—no YARA rule writing needed

Traditional Pivoting – Seamlessly pivot on IPs, URLs, filenames, and other standard indicators

Accelerated IOC Discovery – Surface new indicators of compromise faster with intelligent correlation capabilities

Unified Intelligence Feeds – Integrate OSINT and raw malware feeds with your proprietary data and active cases in a single platform

Intelligence-driven reverse engineering

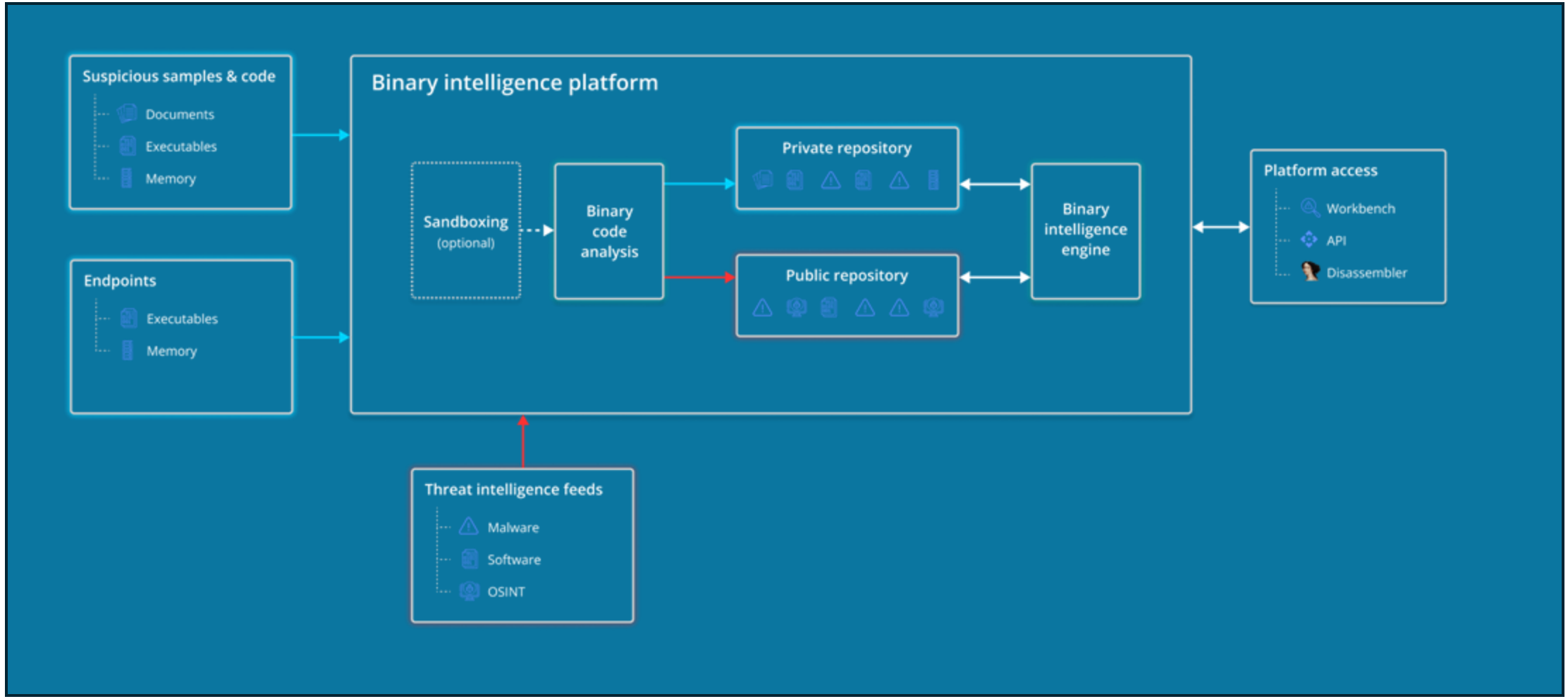
Quickly identify code that matters, identify malcode and library code at function level.

Streamline YARA rule development - Accelerate the creation and testing of precise YARA rules, saving significant analysis time.

Advanced code analysis and hunting — Use clustering and function-level analysis to identify shared code, track variants, and discover novel threats

THREATRAY

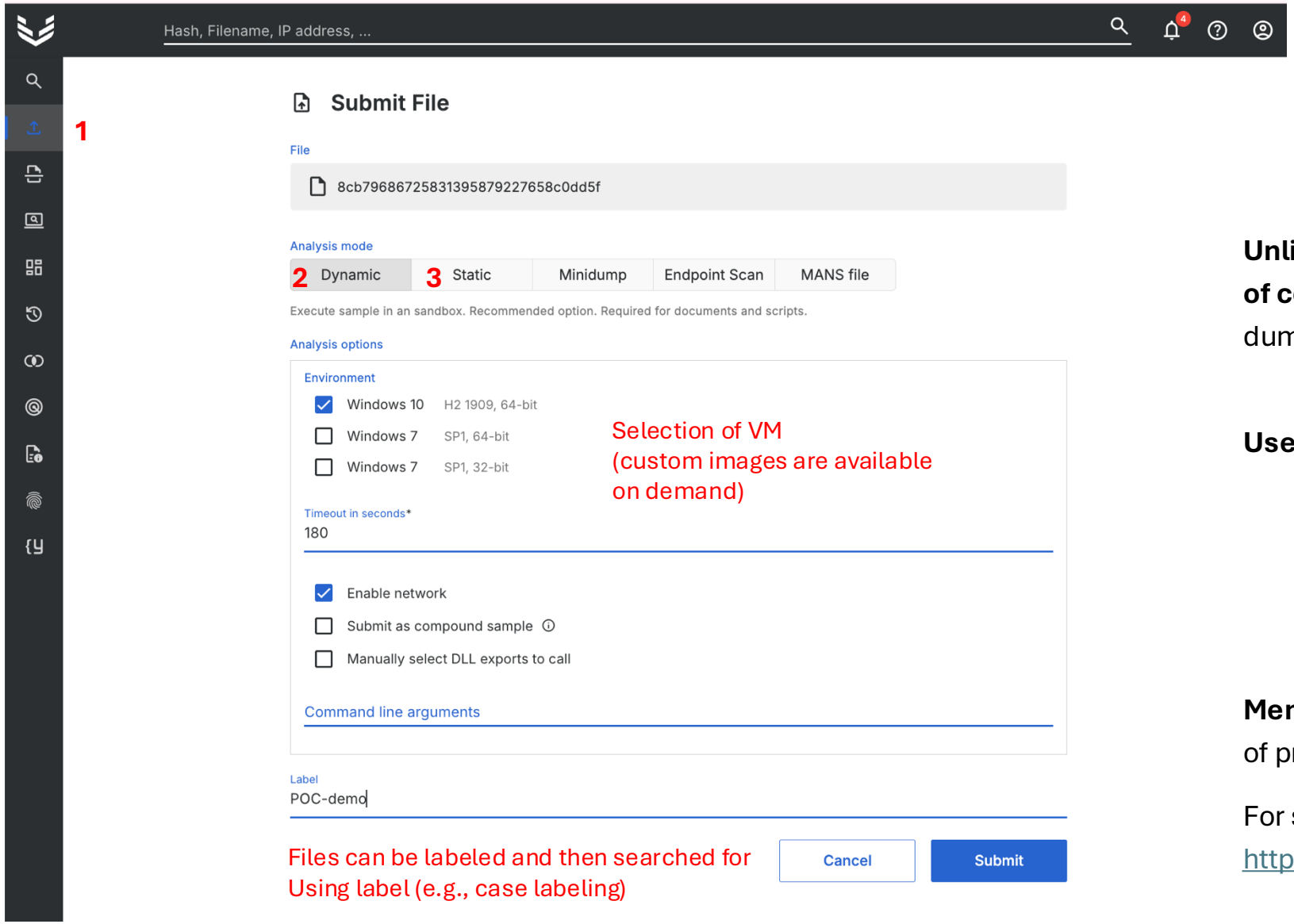
Threatray architecture



File Submission and Organization Repository (Private per Customer)

THREATRAY

File submission



Submission (1) is very similar to sandboxes

Normally use (2) dynamic analysis

- Especially for packed code (most of cyber crime malware)
- Scripts, documents etc.
- When you don't know

Unlike sandboxes Threatray can analyze any piece of code (not just executable code), like memory dumps, broken PE files etc. statically.

Use (3) static analysis for

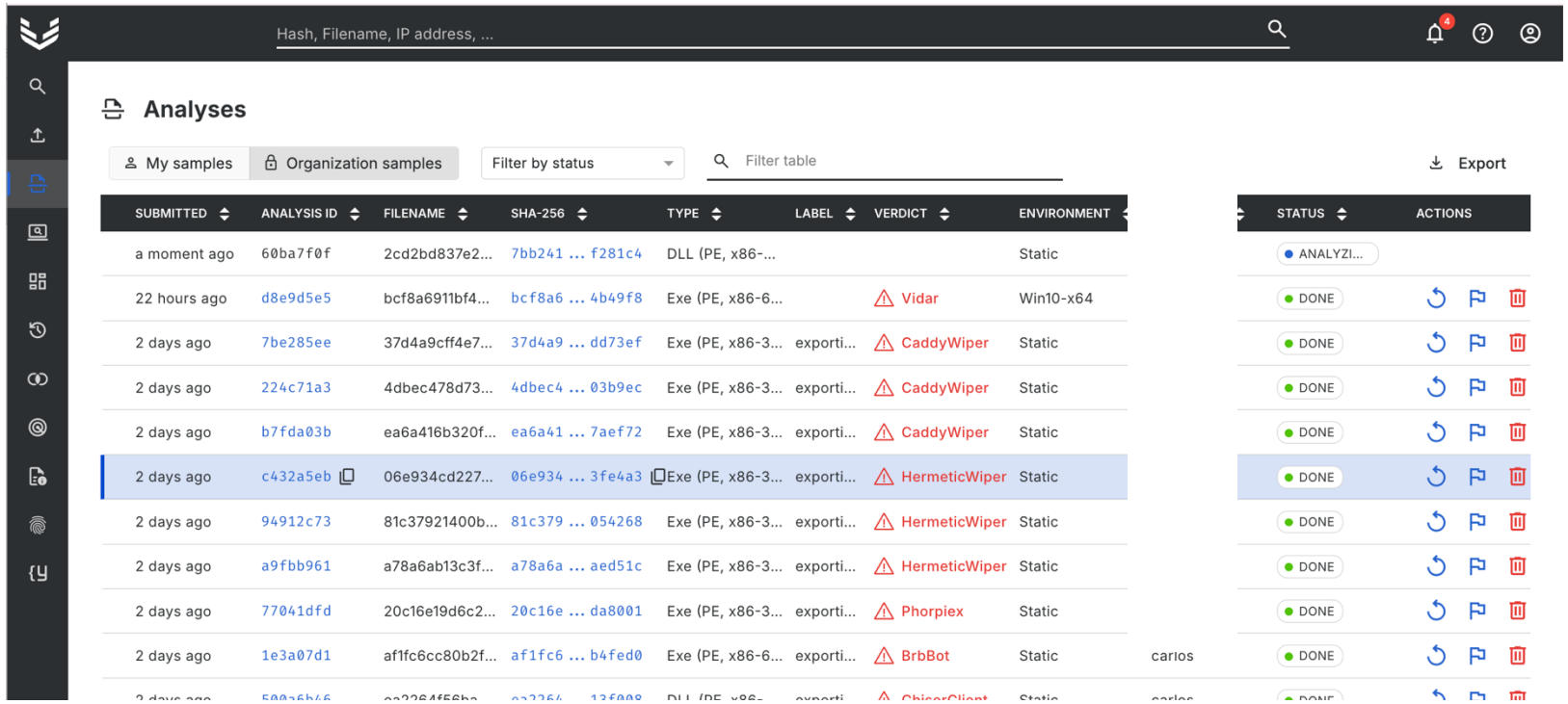
- Shellcode
- Raw memory dumps
- Non-packed code (often APTs)

Memory analysis: Threatray also supports the analysis of process mini-dumps and Trellix HX MANS files

For supported file types and more details check https://docs.threatray.com/docs/submitting-files

THREATRAY

Analyses – Your org's private repo



- The **Analyses view** (1) shows all the samples of an organization that have been submitted for analysis
- **"It is the view of the organization's strictly private malware repository"**
- The view contains samples that have been analyzed (status = DONE), samples that have submitted recently and are being analyzed (status = ANALYZING), etc.
- **We distinguish samples from analyses**. The same samples may be analyzed multiple times over time, resulting in multiple analyses of the same sample.
- We use SHA-256 hashes to uniquely identify samples, and we use the analysis-id to uniquely identify an analysis of a sample.

THREATRAY

Sandboxing in Threatray

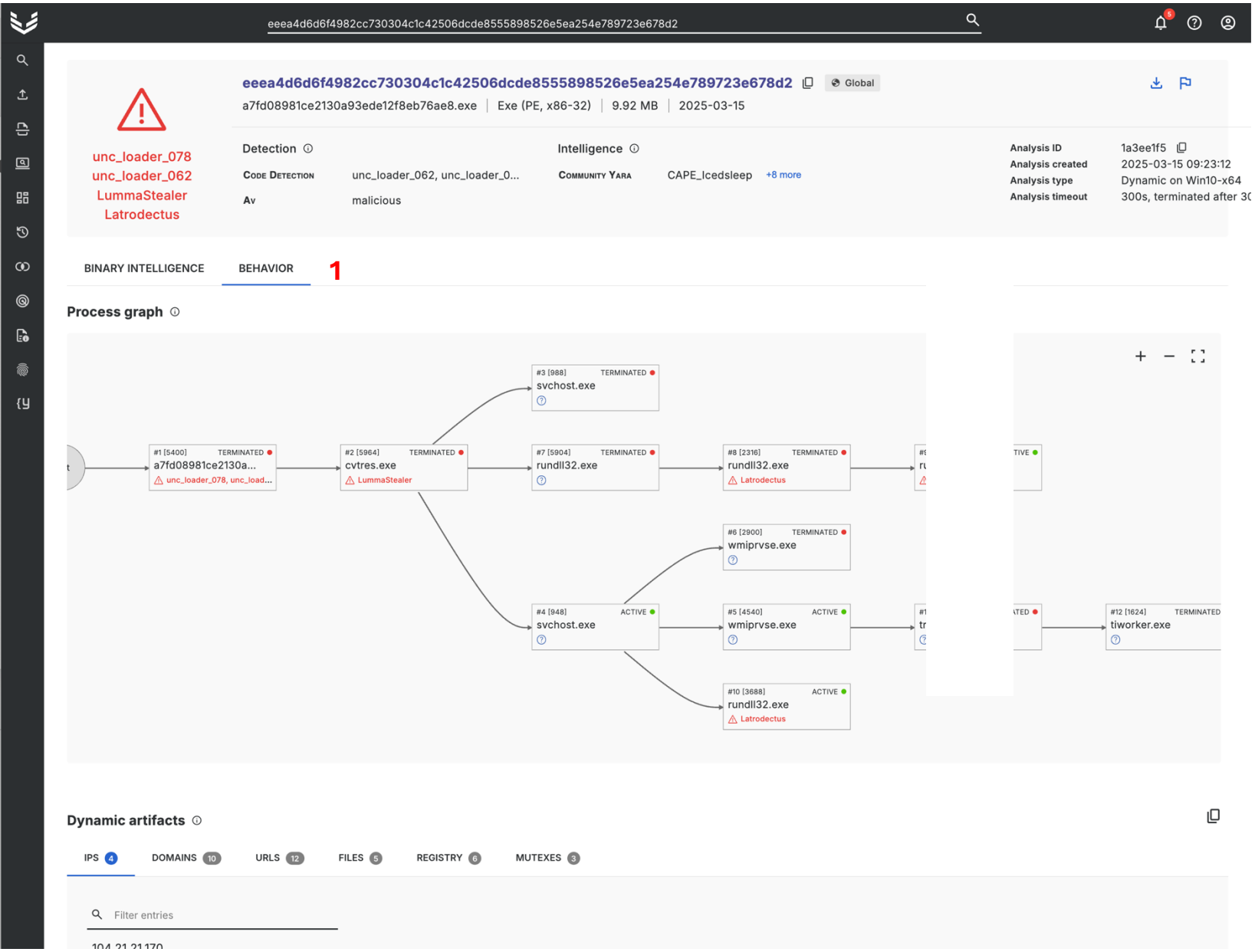
THREATRAY

Sandboxing in Threatray

- **Sandboxing =** dynamic / behavior
- **Threatray =** code / static + intelligence
- **Threatray sandbox**
  - State of the art
  - Excellent detonation rate and anti-anti-analysis
  - No-frills, basic sandboxing information (processes, command line arguments, behavior artifacts, e.g., domains, IPs, URLs, mutexes etc.)

- **Threatray contains a sandboxing component** for two reasons
  - maximize code collection (unpacking, downloading stages) for Threatray deep code analysis, “the more code the better”
  - collection of behavior artifacts for our intelligence capabilities
- **We don’t use any behavior heuristics for detection.** (all classification and detection is coming from Threatray’s unique code detection engine)

Sandboxing

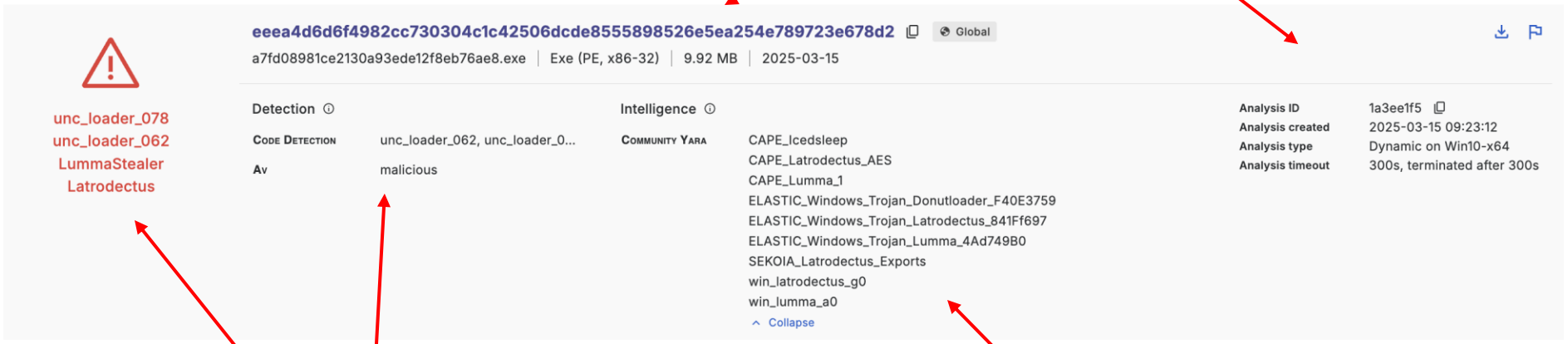


- **Go to “behavior view” (1) to see sandboxing information**
- The behavior reports contains the following elements.
- **Header** summarizes verdict (classification, detection), 3rd party YARA classification, and meta information
- **Process graph** showing which processes running, Threatray classification / detection per process
- **Dynamic artifacts** =IPs, domais, URLs, files, registry, mutexes
- **Dynamic capabilities** shows malware capabilities like obtained from behaviour logs.
- **Process tree with command line arguments.**

THREATRAY

Sandboxing – Header

Meta information

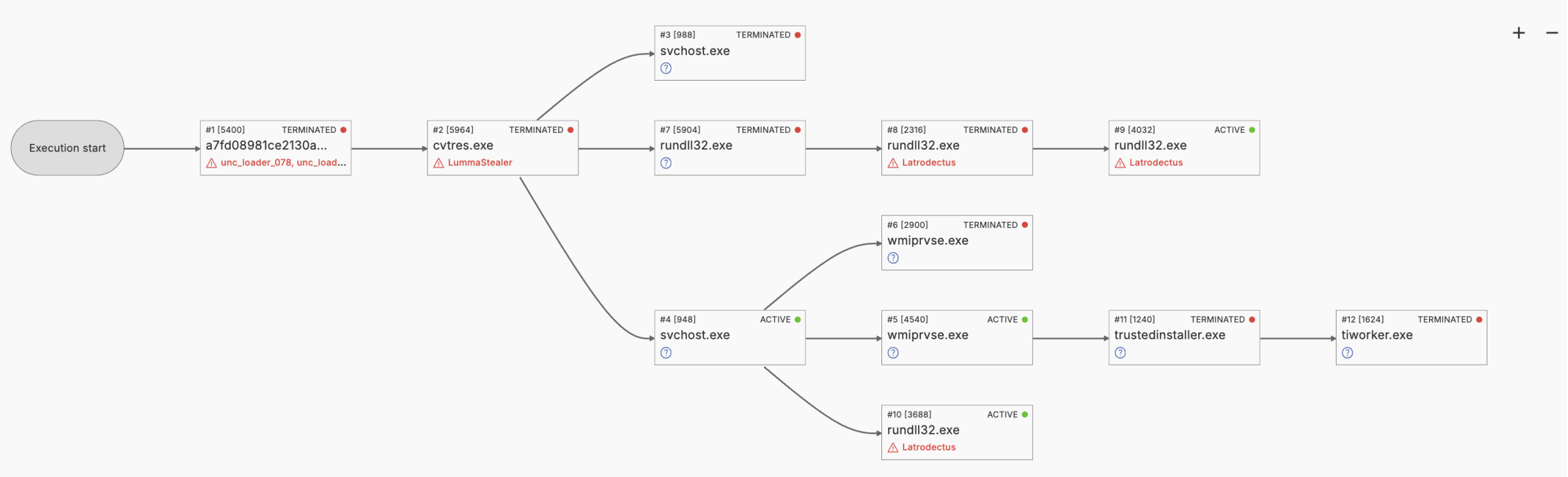


Classification and detection information

3rd party YARA rules from community and Nextron

THREATRAY

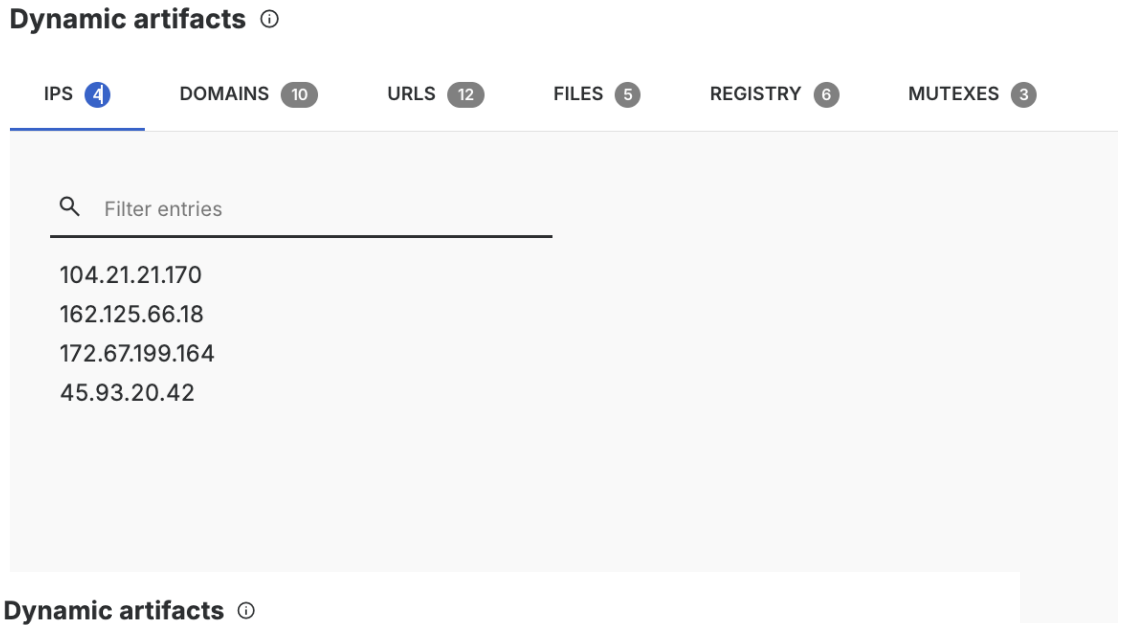
Sandboxing – Process Graph

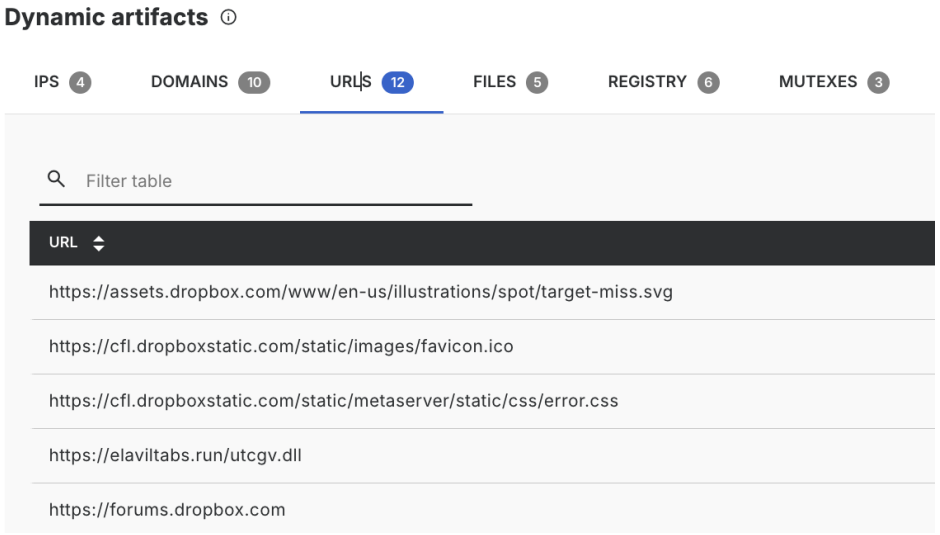


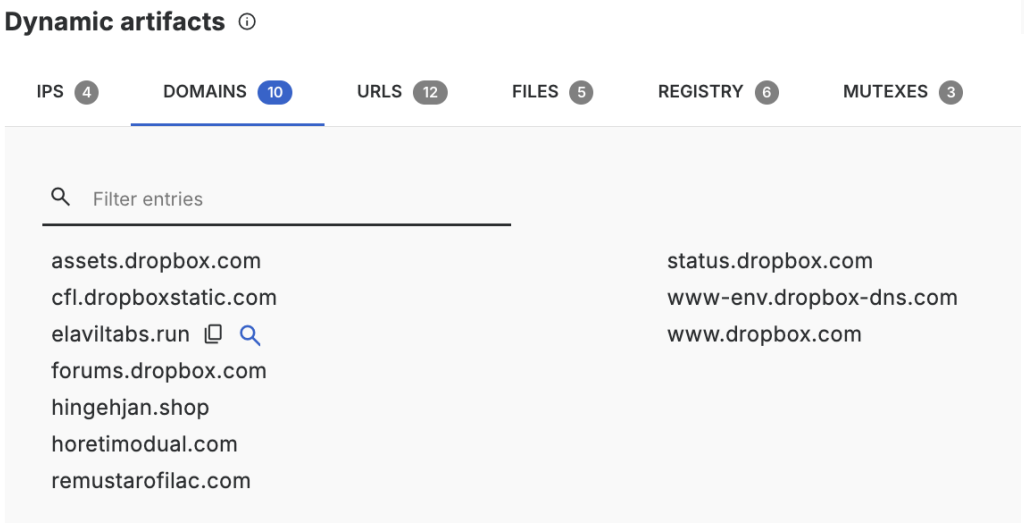
- Threatray tries to attribute the code found within each process to malware (or "goodware" / benign families).
- Question mark (?) means, we could not find any code in that process which is attributable by Threatray
- We see that Threatray's is tracking all the components of an infection chain (loader, different malware stages) – Loader > Latrodectus > LummaStealer

THREATRAY

Sandboxing – Dynamic Artifacts



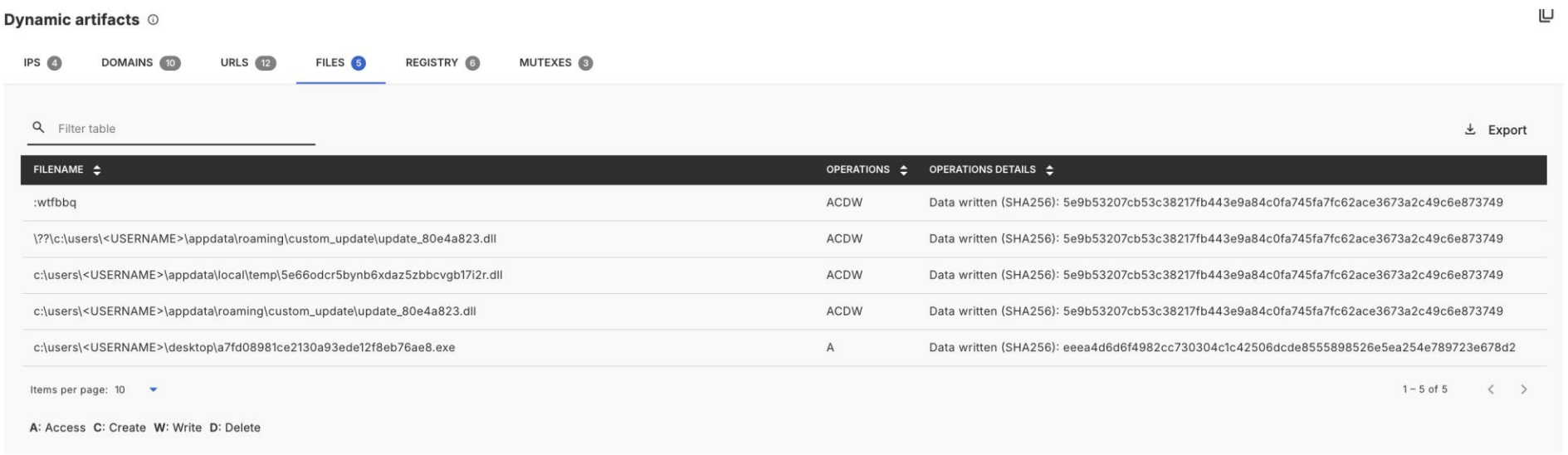




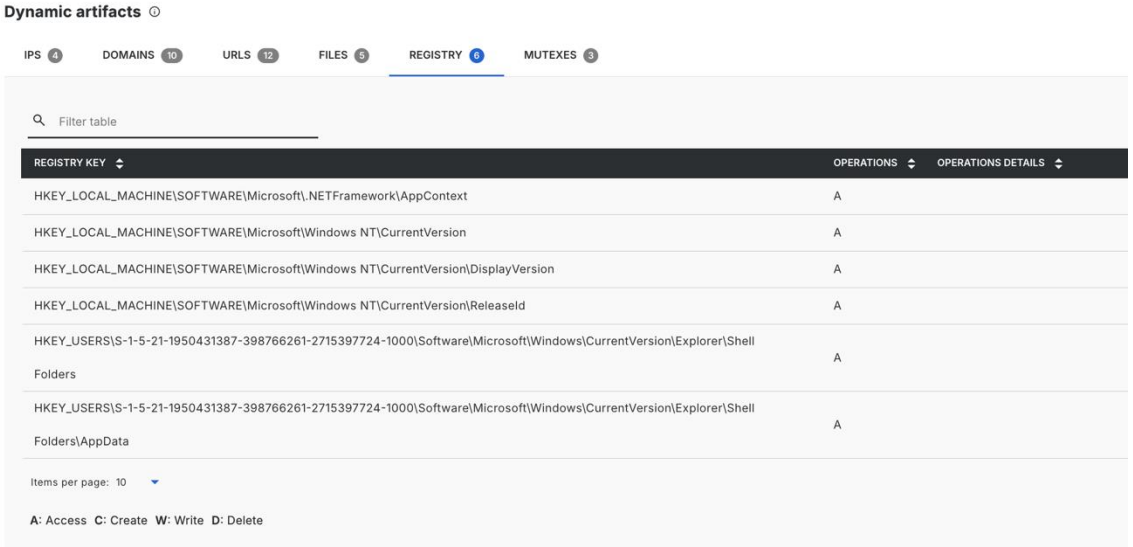
Network artifacts

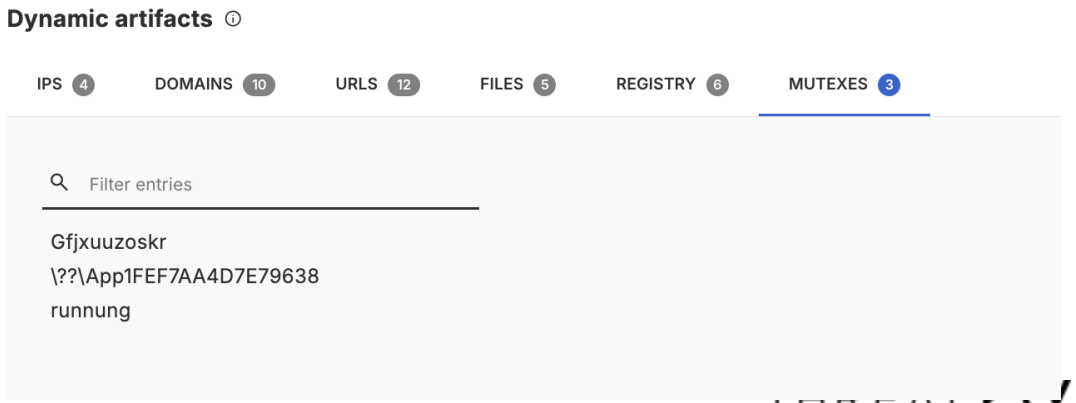
THREATRAY

Sandboxing – Dynamic Artifacts

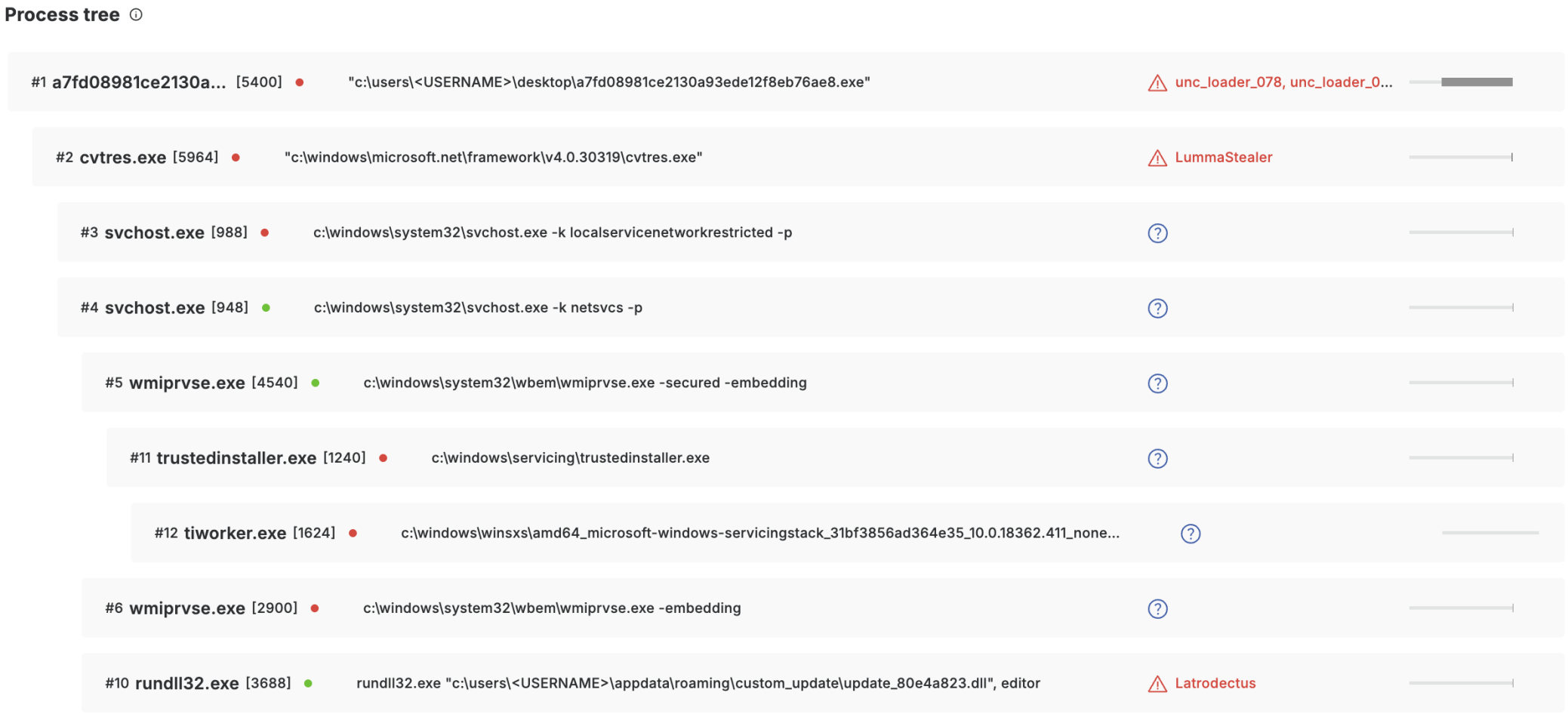


Host artifacts



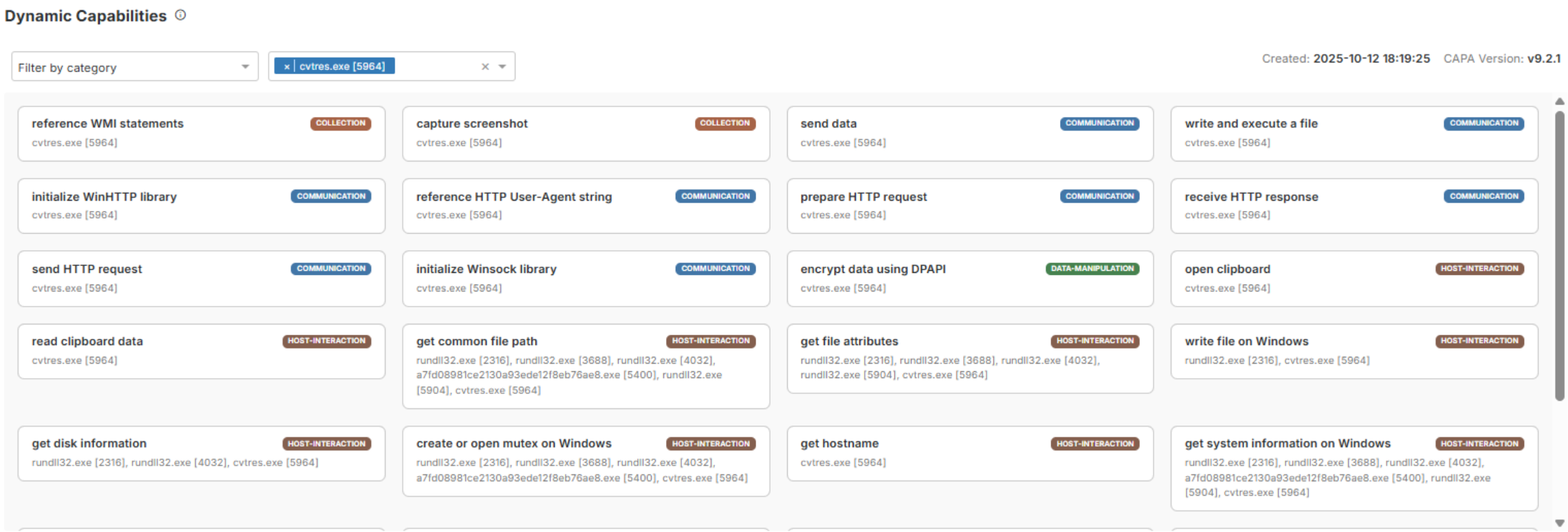


Sandboxing – Process tree and command lines

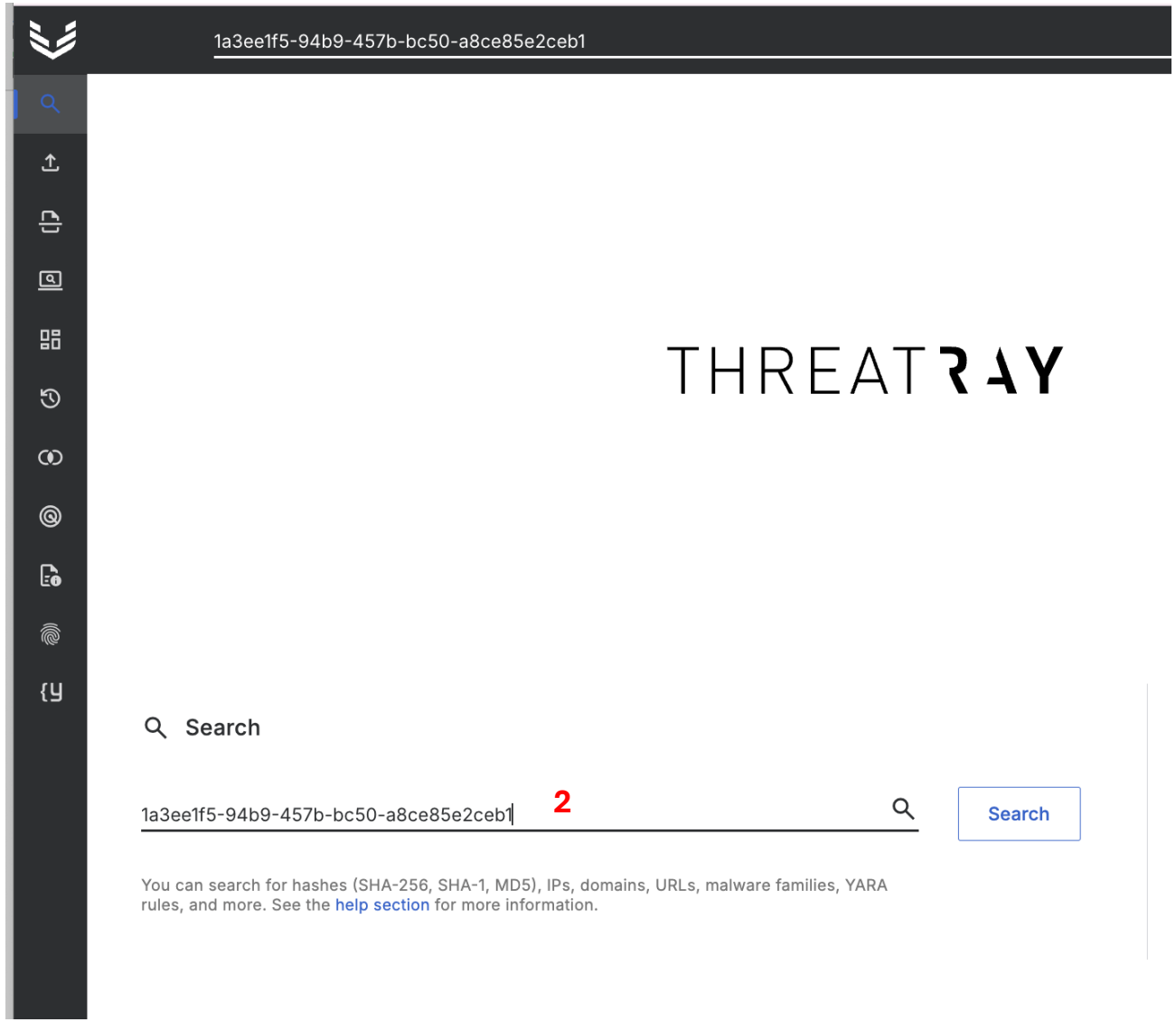


THREATRAY

Dynamic capabilities



How to replay this analysis in your Threatray instance



- Go to the search view (1) and paste the unique analysis ID (2) into the search bar

  The unique analysis ID for the Latrodectus / LummaStealer infection chain is

  1a3ee1f5-94b9-457b-bc50-a8ce85e2ceb1

- The analysis is in the global Threatray repository and accessible to all our customers (see next session).

THREATRAY

The binary intelligence view

The binary intelligence view

- **The binary intelligence view contains many of Threatray's unique and advanced code analysis capabilities about unknown / suspicious code.**

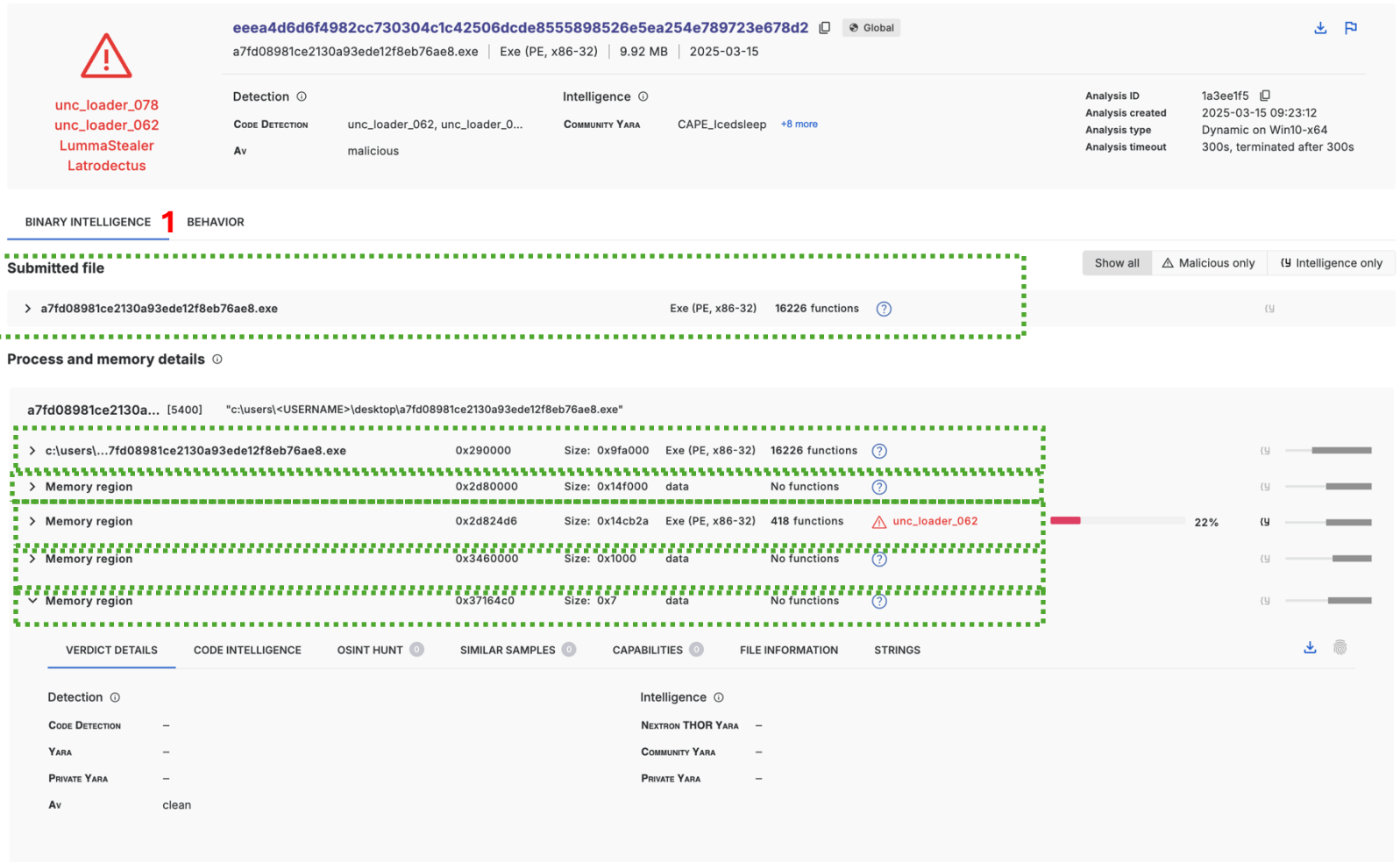
  Binary intelligence view is complementary and not replacing the behavior / sandbox view. Think of behavior view and binary intelligence view as "ying & yang" of malware analysis.

- **The focus is on unknown / suspicious "code blocks" found during an analysis.**

  A code blocks can be **binary files on disk** or **code that we find in memory** (e.g., from unpacking, malware stages that are downloaded from the Internet, dropped components etc.)

- **For each code block Threatray shows the same type of information and provides the same type of analysis capabilities.**

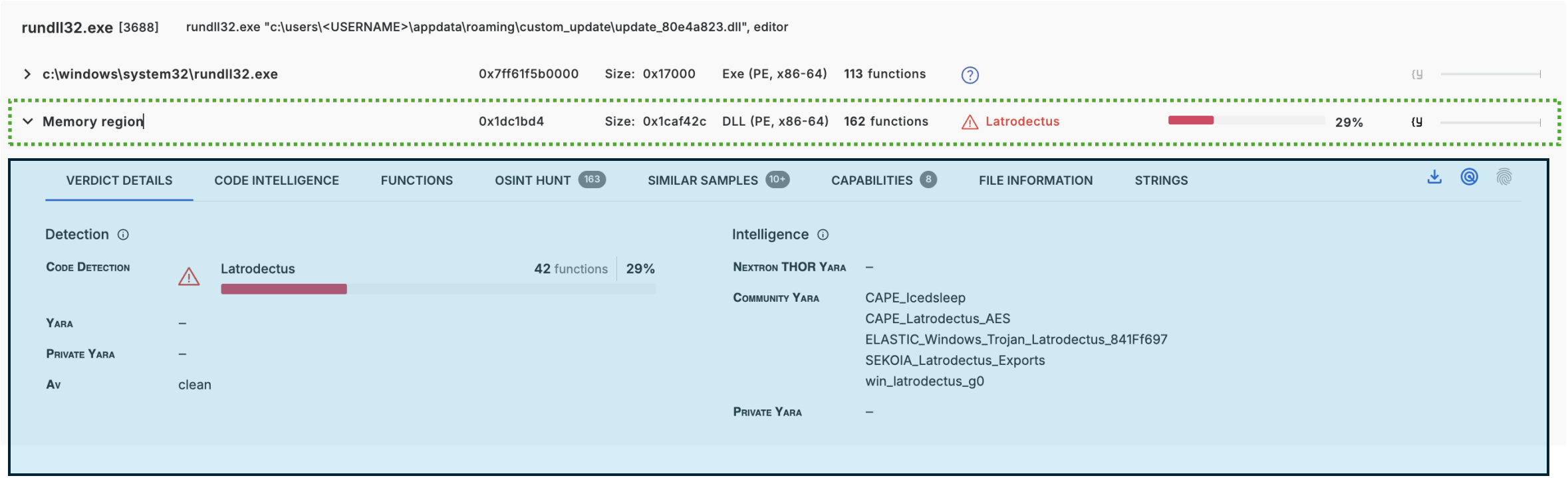
Binary intelligence view



- **Go to “binary intelligence view” (1)**
- **The header** is the same as in the behavior / sandbox view – after all it is the same analysis, just a different view.
- **The submitted file (2)** is analyzed statically (i.e., without sandboxing).
- **The process and memory detail view (3)** shows all processes and their virtual memory contents, whenever there is unknown / suspicious code in process memory.
- **The static file and each memory of file element in a process are “code blocks”** (green dashed boxes)

THREATRAY

Same analysis capabilities for every code block

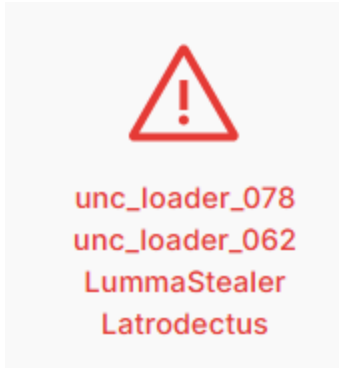


- Code block is green dashed box (in this example it is code found in process memory)
- Analysis capabilities for this code block are show with light blue background

THREATRAY

Threatray classification & detection verdicts, and 3rd party YARA

Detection / classification verdict

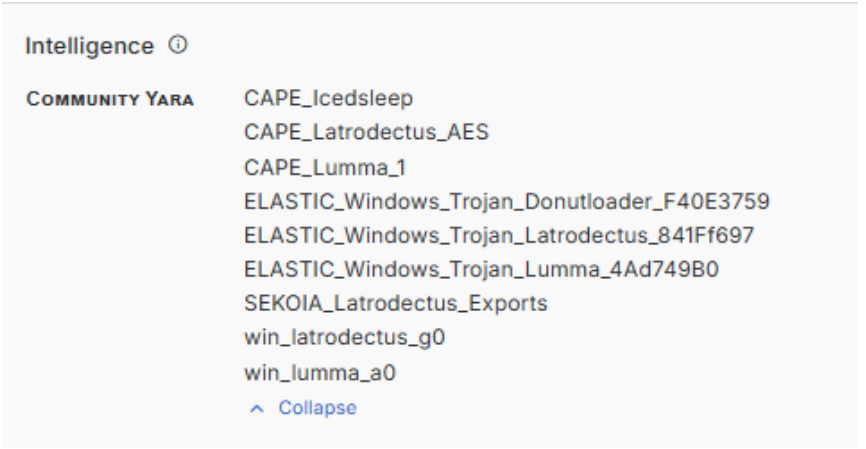


- **A Threatray detection and classification verdict is computed from multiple sources:**
  - Threatray code detection technology (based on code similarity)
  - An OEM antivirus engine
  - YARA rules curated by Threatray
  - YARA rules curated by users (optional)

- **Almost all family identification / classification is coming from Threatray's code detection technology.**

- **For family naming we follow the Malpedia naming convention** (https://malpedia.caad.fkie.fraunhofer.de/families) wherever possible. Yet, we track more and different families than Malpedia, and there for not all our family names can be tracked back to Malpedia.

3rd party YARA matches



- We use 3rd party YARA rules and scan every code block (i.e., memory and files) with those rules
- The 3rd party rules consist of open source YARA rules, which we curate.
- We are also using NEXTRON YARA rule set, through our technology partnership with NEXTRON

3rd party YARA matches

- The **Threatray verdict** (classification, detection) **and YARA matches** are shown in **multiple locations of our analysis reports.**

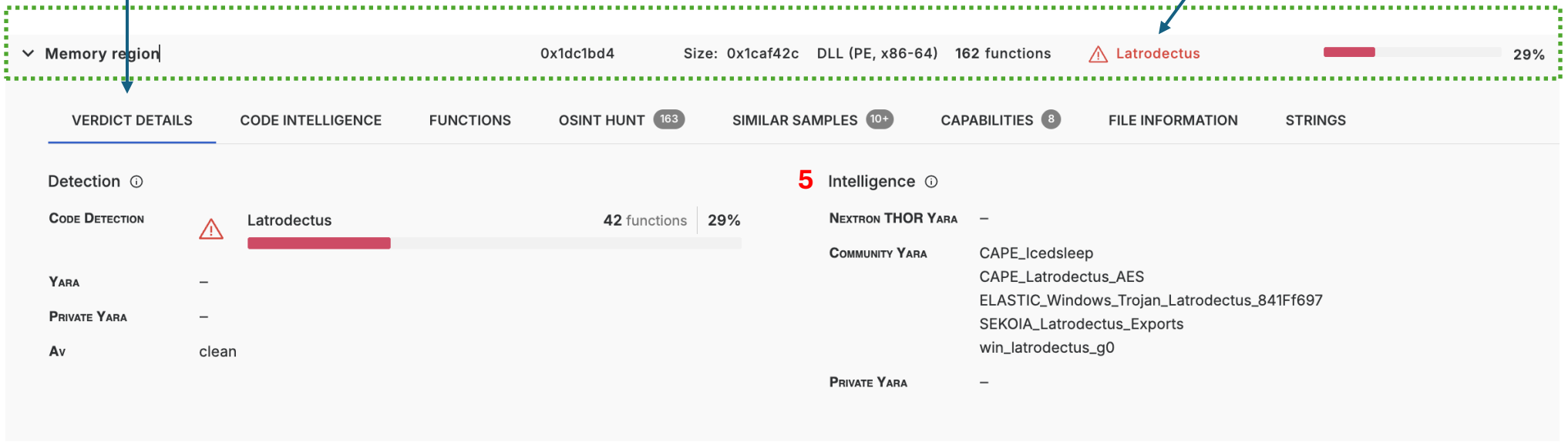
- **Where are verdicts and YARA shown?**
  - In the header, where we aggregate verdicts and YARA
  - For every code block, i.e., static file and in process memory

- **Verdict and YARA matches have always have the same structure and contain the same type of information (see next slide)**

Verdict and YARA – Example in process memory

Go to “verdict details” to understand why is Latrodectus

We classify the code block as Latrodectus



(1) Threatray code detection identifies Latrodectus
(2) No detection from Threatray curated YARA rules
(3) No detection from customer supplies private YARA rules
(4) We use an OEM antivirus (AV) which has no detection in this case

(5) We are using curated open-source community YARA rules and are also using the industry leading NEXTRON YARA rules (“3rd party YARA rules”)

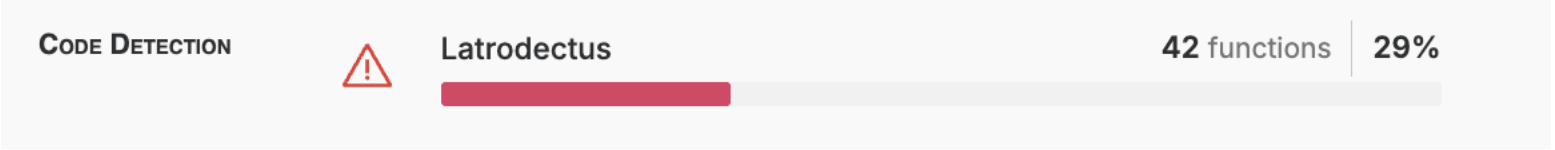
In this example we see some community rule matches.

THREATRAY

Threatray code detection – how does it work?

- ***Code detection* is our industry wide unique malware detection and classification technology. I**t is based on measuring code similarity of a piece of unknown code block against our huge database of malware code.

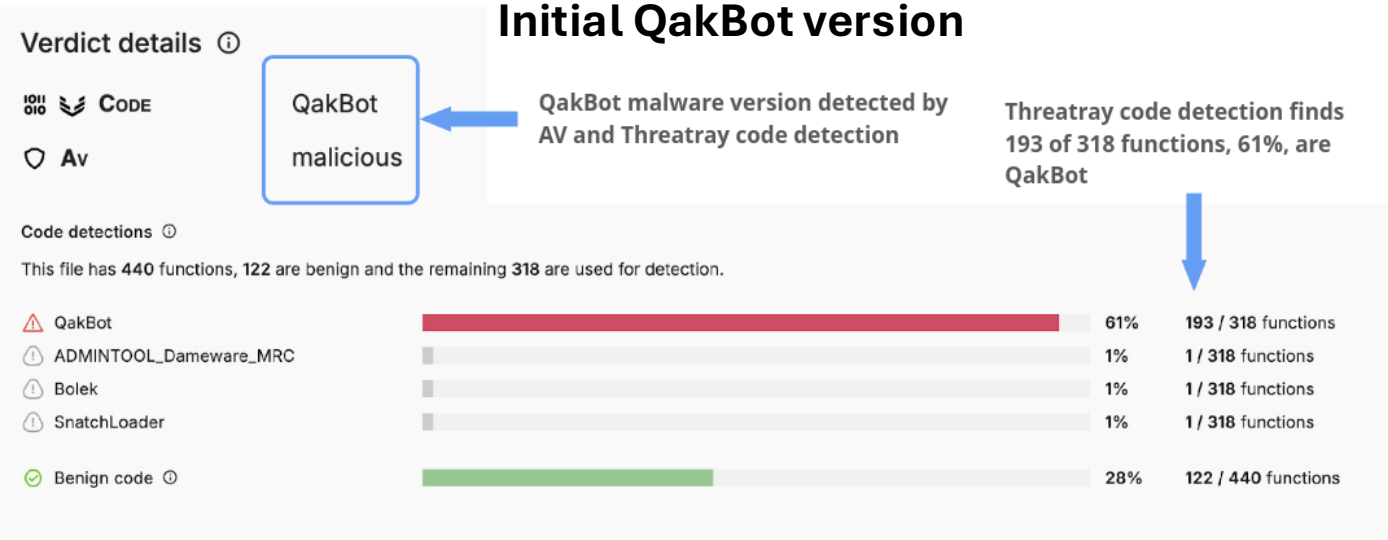
- Here we detect Latrodectus because 29% of the unknown code is very similar to Latrodectus code from our malware code database



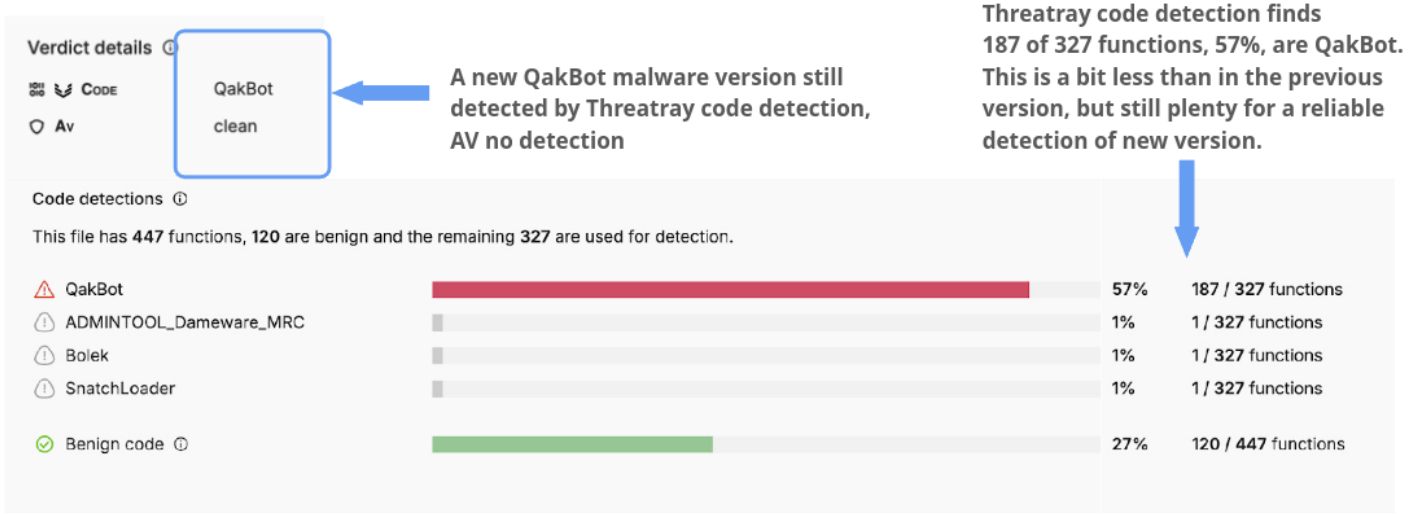
- **Code detections have key advantages for Threatray users:**
  - Very resilient to new malware variants => Threatray detects new variants, YARA, AV, sandboxes miss
  - We can track more malware families than others, since it is very easy for us to track new malware families
  - We can easily and very reliably identify non-maliciouis codes and tooling like LOLBins etc.

Threatray code detection resilience against new malware variants

Initial QakBot version



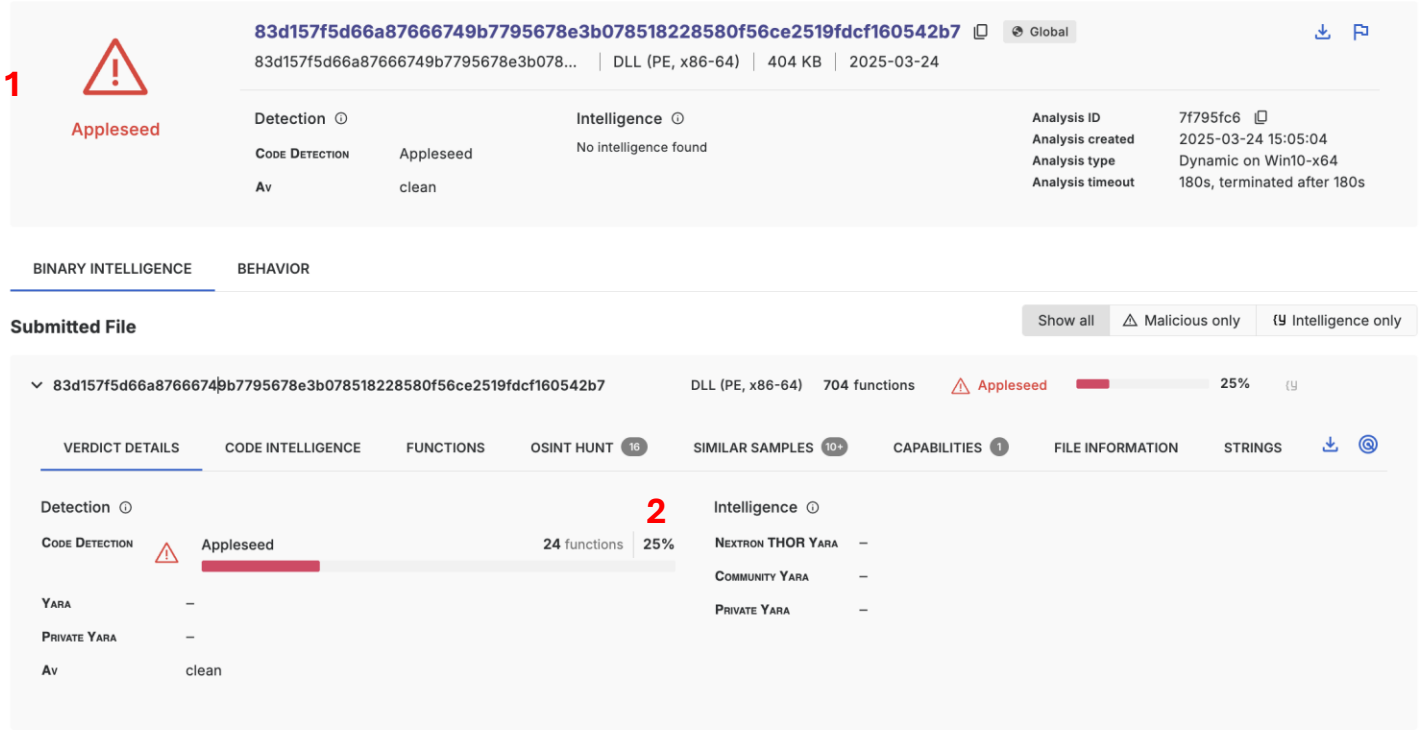
New QakBot version



- New version of QakBot detected only by Threatray, missed by AV and YARA
- Threatray code detection is resilient against new malware variants
- The reason is that in newer malware variants, the amount of known malicious code is decreasing — yet we can still measure it.

THREATRAY

NK APT example only caught by Threatray



- **This is an example of North Korean APT sample from the Appleseed family.**
- **It is a new version of Appleseed**, we have received the sample in spring 2025 and were the only technology able to classify the sample (1)
- The reason is our code detection technology, which has found 25% code reuse (2) between this new unknown sample and previous Appleseed sample which is 4 years old.

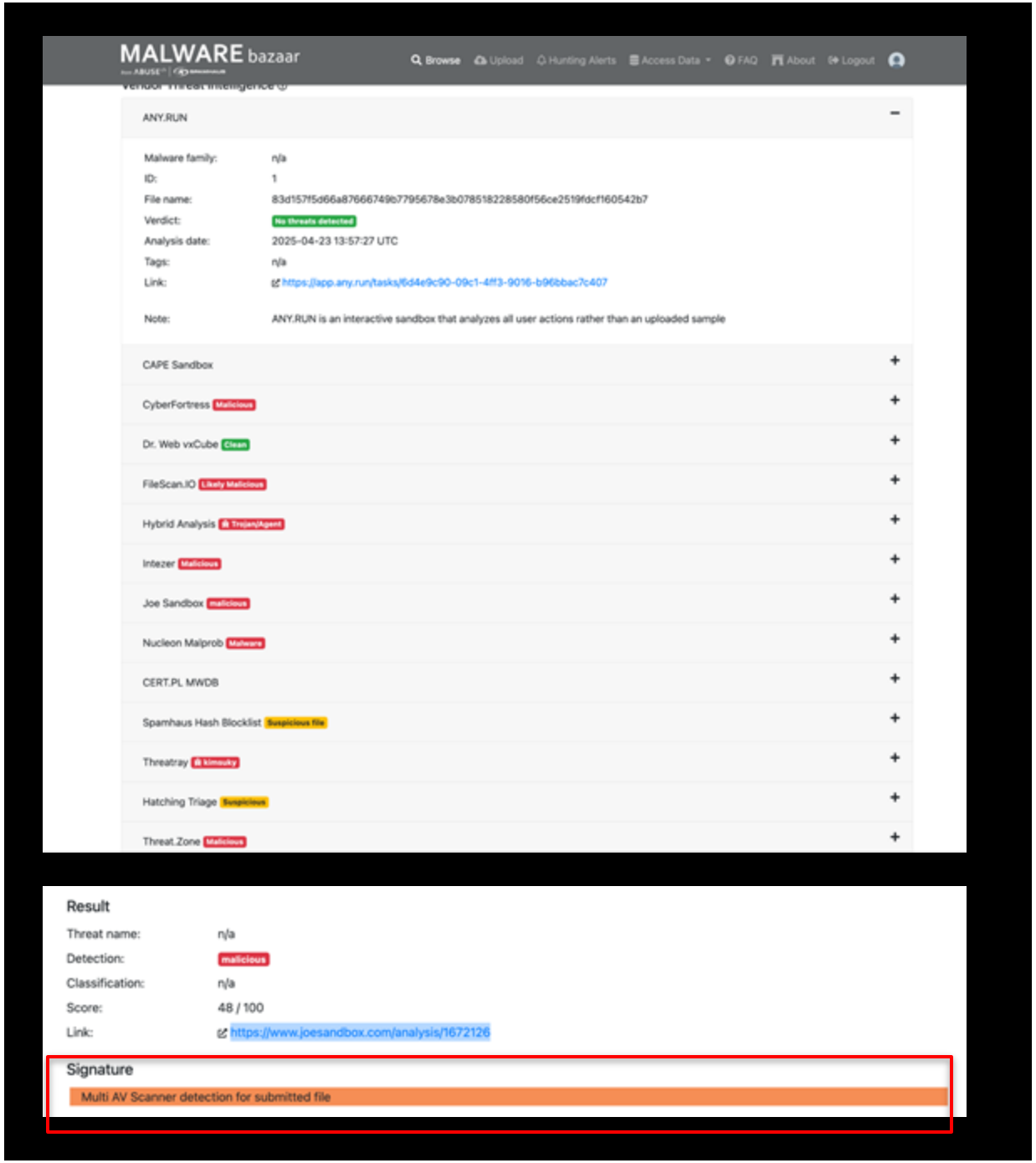
THREATRAY

NK APT example only caught by Threatray

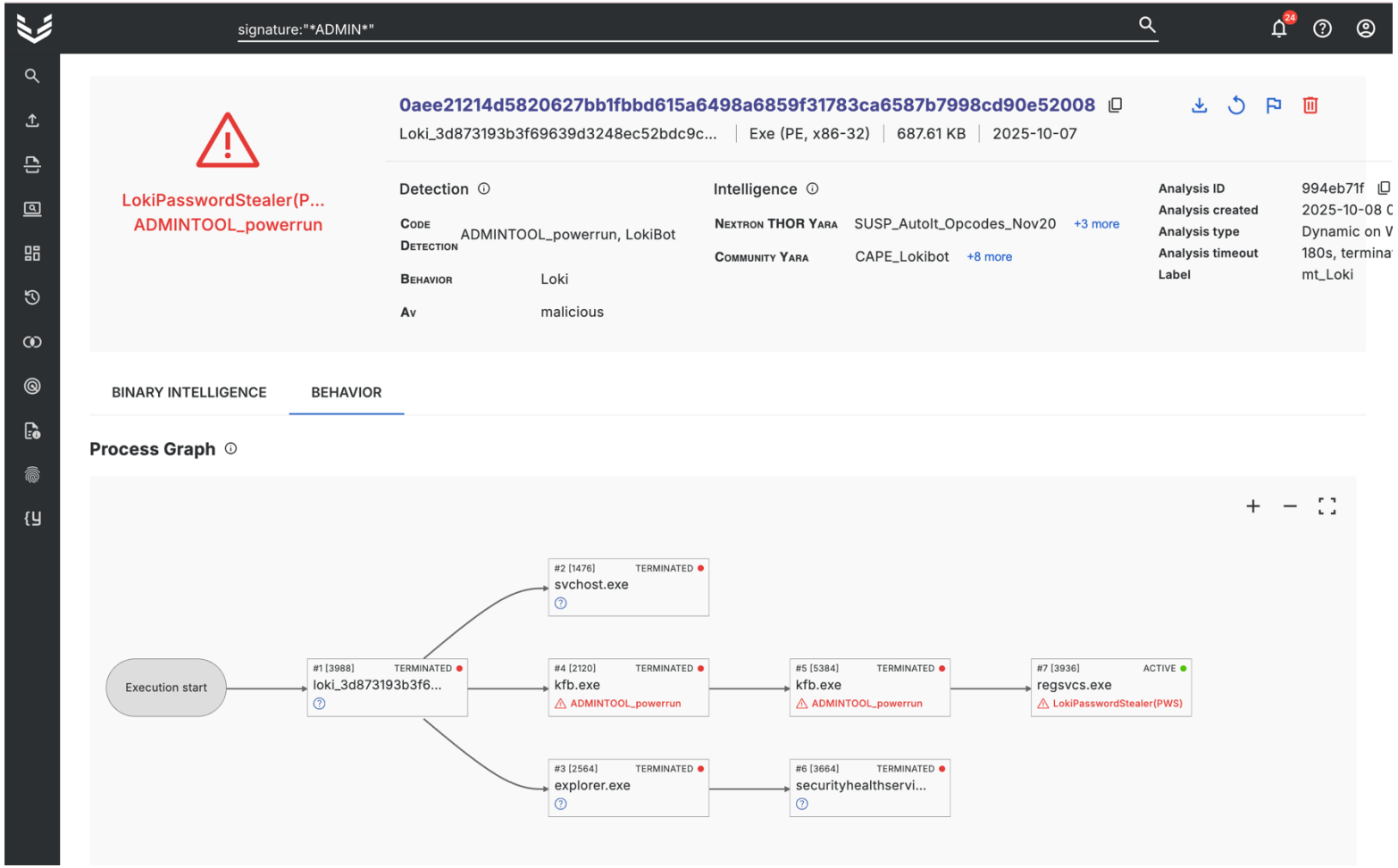
- **Traditional sandboxes fail** to classify new malware variants because of (1) brittle signature tech (2) lack of dedicated APT classification capabilities, leaving critical threats unidentified.

- We only see OEM AV detections which are part of most sandboxes, but no classification.

- See https://bazaar.abuse.ch/sample/83d157f5d66a87666749b7795678e3b078518228580f56ce2519fdcf160542b7/

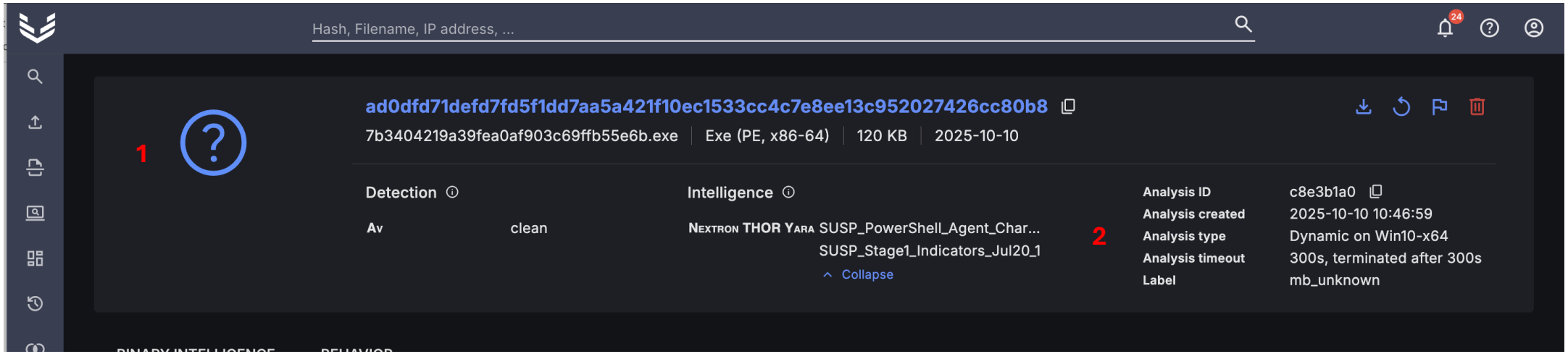


Tracking LOLBins and admin tools



- This is an example where an LOLBin / admintool is used as part of the infection chain
- We see that the attacker first deploys the Powerrun admin tool (1), and then LokiPasswordStealer malware (2).
- As you can see, we prefix LOLBins / admintools with “ADMINTOOL_...” in the verdict.

Our 3rd party YARA rules are NOT used for the classificationion / detection verdict



- **Our 3rd party YARA is treated as intelligence and NOT used for classification and detection**

- This example shows a sample that is neither detected nor classified by Threatray, hence the blue question mark (1).

  However, we see matches by NEXTRON Thor (2), which are very likely correct, since high-quality rules set.

- **We strongly advise to look at our 3rd party YARA matches** however, we just don't control those rules and therefore do not use them for detection and classification.

THREATRAY

How to replay your Threatray instance

Here are the analysis IDs for the examples shown:

- For the Latrodectus example, binary intelligence view etc. we have used 1a3ee1f5-94b9-457b-bc50-a8ce85e2ceb1
- For the sample with THOR YARA match but no Threatray detection: c8e3b1a0-2680-4492-a74d-374c27bb5d3f
- For the Appleseed APT example, we have used analysis 7f795fc6-0892-40d5-a5c2-5e1e38528b0e
- For the LOLbin / admin tool example we have used 994eb71f-c4c6-4c9f-b726-be6b9ab47332

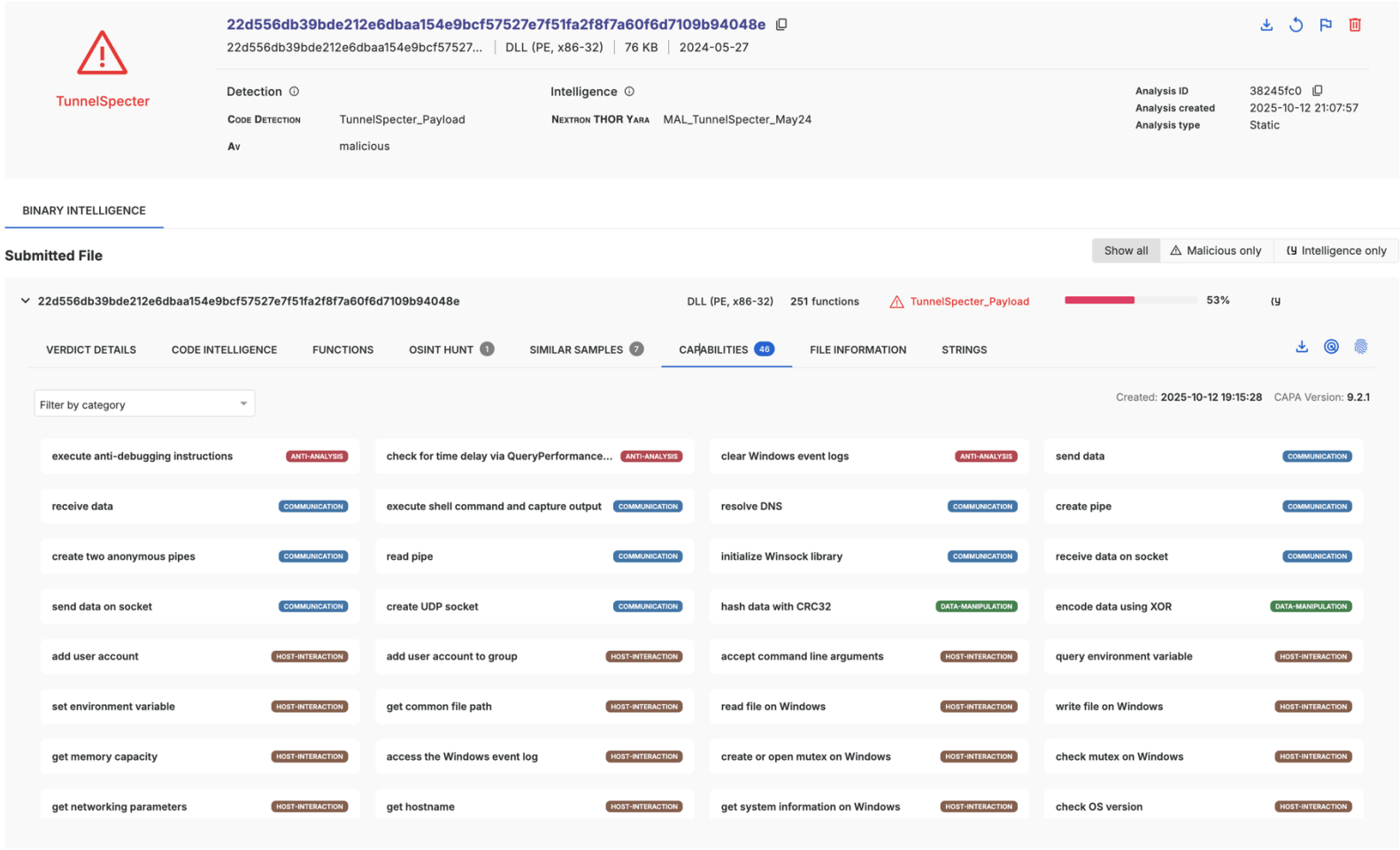
Capability analysis

Code capabilities

- **For every code block we extract capabilities of that code block using static analysis.**
- We use the Mandiant CAPA framework for doing so.
- This (static) code-based approach reveals capabilities that are present in code but not present in behavior capabilities.
- Static capabilities are especially useful when analysing samples without any classification information, as they greatly assist in the triage process.

Code capabilities

- This is an example showing a TunnelSpectre sample, a Chinese APT
- It is a DLL that we have analyzed statically only (without sandboxing)
- The capabilities show that this sample features anti-analysis, and host reconnaissance and command execution capabilities.



THREATRAY

How to replay your Threatray instance

Here are the analysis IDs for the examples shown:

- For the Code Capabilities examples using the TunnelSpectre APT we have used 38245fc0-9cef-43cd-9c5c-1a32b28dac0b