

---

---

# Threat Categorization Based on Malware's C2 Communication

---

---

Key Words: Malware Analysis, C2, Threat categorization , Python, Fingerprinting, pcap

Group 6 – (KCST) Kerberos Cyber Security Team

# Team

---



Prof. Anand Handa  
(Mentor)



Nitesh Kumar  
(Mentor)



Mohammed Jawed  
(Contributor) - Founder



Sriram P  
(Contributor) Co-Founder

# Problem Statement

---

The threat landscape facing modern organizations is constantly evolving and becoming increasingly complex. With policies like BYOD and social data of individuals available on the social media, Passwords and 2FA are not going to stop the cyber attacks. Understanding the threat discovered in a corporate environment will help the infosec team to assess the impact of the attack and take measures accordingly. Cyber-attacks are becoming more sophisticated and complex, and it is becoming increasingly difficult to detect and block /prevent them.

One of the key challenges in effectively defending against cyber threats is the ability to accurately categorize and analyze potential threats.

In particular, understanding the Command and Control (C2) communications used by attackers is critical in identifying and responding to cyber attacks. Command-and-Control (C2) communication is a common technique used by attackers to control the infected hosts and steal sensitive information. It is crucial to identify C2 communication and categorize the network threats accurately to prevent and mitigate cyber-attacks.

**This project aims at looking into the networking concept of these C2 communicating malwares and tries to parse the network packets and classify the threats based on the unique communication pattern used by these malware families.**

**The rules also involve fingerprinting the TLS certificates used in the communication.**

# Executive Summary

---

The objective of the project is to classify network risks by examining the interaction between Command-and-Control (C2) and the compromised host within a company using either real-time pcap or archived pcap files. Analyzing the hazards identified within a corporate setting will enable the information security team to evaluate the influence of the threat on the organization's overall security stance.

The Core engine of this Project consists of three independent modules - Packet Processing, Rule Parser, and Rule Authoring - that work together to provide threat categorization by analyzing the C2 communication patterns and categorize the network threat into various categories such as BOKBOT, IcedID, Graftor, STRRAT, Cobalt Strike etc.

It includes a user-friendly interface built on ASP.NET Core and Bootstrap, making it easy for organizations to visualize and analyze their security data. The product also includes various APIs and a SQLite database, allowing for flexibility and scalability.

This Project is available for free on Github and DockerHub. Download and use it to grow and enhance your security infrastructure, and be sure to provide feedback to help the product evolve and improve over time. With ArkThor, you can take your organization's cybersecurity to the next level.

The project's outcome will provide valuable insights into the network threat landscape and help organizations take proactive measures to prevent and mitigate cyber-attacks.

In addition, the product is containerized, including the UI, APIs, core engine, and RabbitMQ. This allows for easy deployment and use in any organization.

# Objectives and Goals

---

## Objective:

To develop a state-of-the-art threat categorization engine based on C2 communication and that provides organizations with an advanced system for identifying and mitigating threats before they cause damage.

## Goals:

- Develop a scalable and easily deployable threat categorization engine based on the analysis of C2 communication in Pcap files.
- Provide a user-friendly interface that allows organizations to visualize their overall security posture and improve their threat detection capabilities.
- Include various APIs for flexibility and scalability, enabling organizations to customize and integrate the tool with their existing infrastructure.
- Ensure the tool stays ahead of malicious actors who seek to exploit vulnerabilities in an organization's digital infrastructure by incorporating cutting-edge technology and It should be Platform independent.
- Provide the tool for free on Github and DockerHub, making it accessible to organizations of all sizes and levels of expertise.

By achieving these goals, the objective of developing an advanced threat categorization engine that empowers organizations to stay ahead of emerging threats can be met.

# Methodology

## The ArkThor product is built using the following methodology:

1. **Core Engine Development:** The first step in the development process was to create the Core Engine. This involved designing and building three independent modules – Packet Processing, Rule Parser, and Rule Authoring. The Packet Processing module uses Scapy, an opensource library, to process packets. The Rule Parser module loads the ArkThor format rules and matches them with the output of the Packet Processing module. The Rule Authoring module contains rule components that can convert open source rules or human-authored rules.
2. **UI Development:** The user interface (UI) of the product is built using Asp DotNet Core, Javascript and Bootstrap. The UI includes a dashboard, various pages for displaying analysis results, and a measurement page that shows valuable KPIs in the form of pie charts, bar graphs, and knobs.
3. **API Development:** To provide flexibility and scalability, the product includes various APIs. These APIs allow users to interact with the product programmatically and access the data in the product's SQLite database.
4. **Database Management:** The product uses a SQLite database to store analysis records, configuration settings, and other data. The database is managed using SQLite commands and queries.
5. **Message Queue Integration:** The product uses RabbitMQ, a message queue system, to handle communication between different modules and components. This allows for efficient and reliable communication between different parts of the product.
6. **Containerization:** The product has been containerized using Docker, making it easy to deploy and use in any organization.
7. **Testing and Deployment:** The product is tested thoroughly to ensure that it meets the requirements and works as expected. It is then deployed to the production environment using Docker, which allows for easy deployment and management of the product.

Overall, this methodology ensures that the product is well-designed, flexible, and scalable, and that it provides valuable insights to organizations looking to improve their threat detection capabilities.

# Capstone Project Introducing

"As a result of this capstone project, we are proud to introduce **ArkThor**."

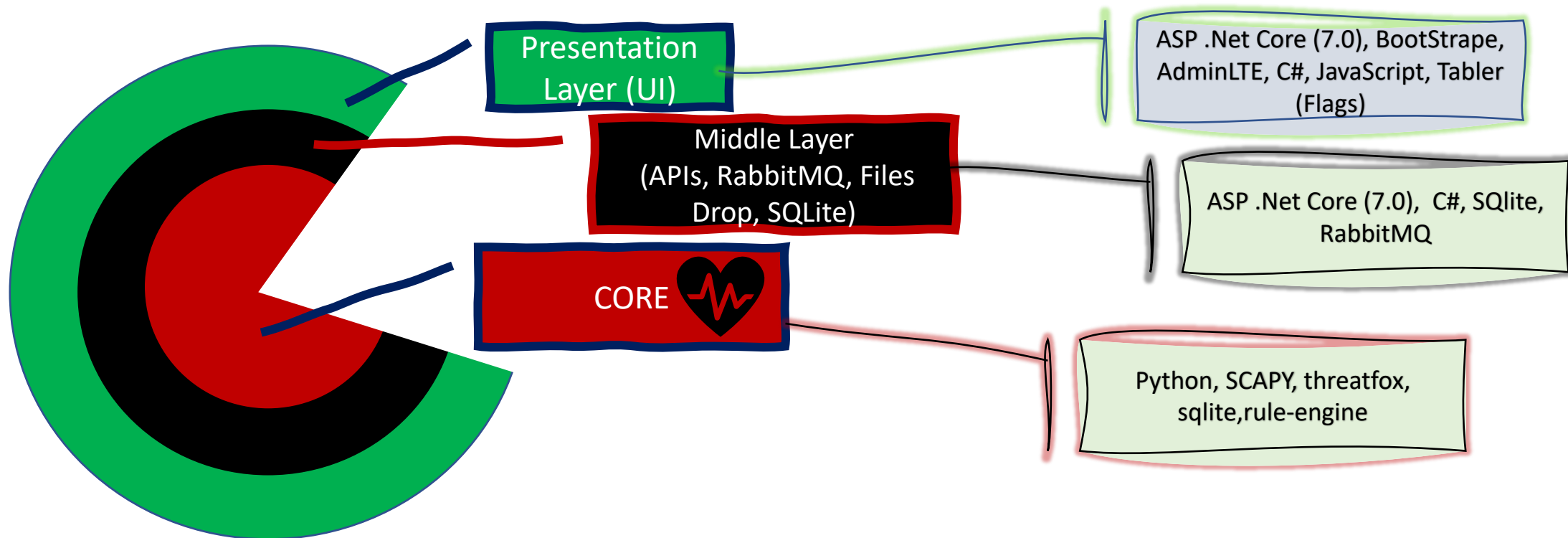


- "Ark" imply **safety** or **protection**
- "Thor" is associated with **strength** or **power**

"We aim to develop our capstone project into a fully functional product that can be brought to market and provide value to customers, rather than simply being a project for academic purposes."

# Project Core Idea

- "The entire project is comprised of three distinct layers: a platform-independent layer that is scalable and built using microservices. It is also designed to be easy to deploy and relies entirely on an open-source technology stack."
- "The inner layer can be used directly, without any dependencies on the upper layer."



Organization Don't buy products, they buy Solution to their Problems....



# ArkThor to Grow Organically

---

"Not only is this a capstone project, but it also represents a collaborative achievement of the CSCD course. Our aim is to foster organic growth of this project through open-source principles and community contributions."

"After consulting with our mentor and gaining approval to allow organic growth, we have made this project available on GitHub a open source project for people to contribute. In addition, a public version can be viewed as the ArkThor sample on Azure Cloud."

## **1. GitHub repo Url**

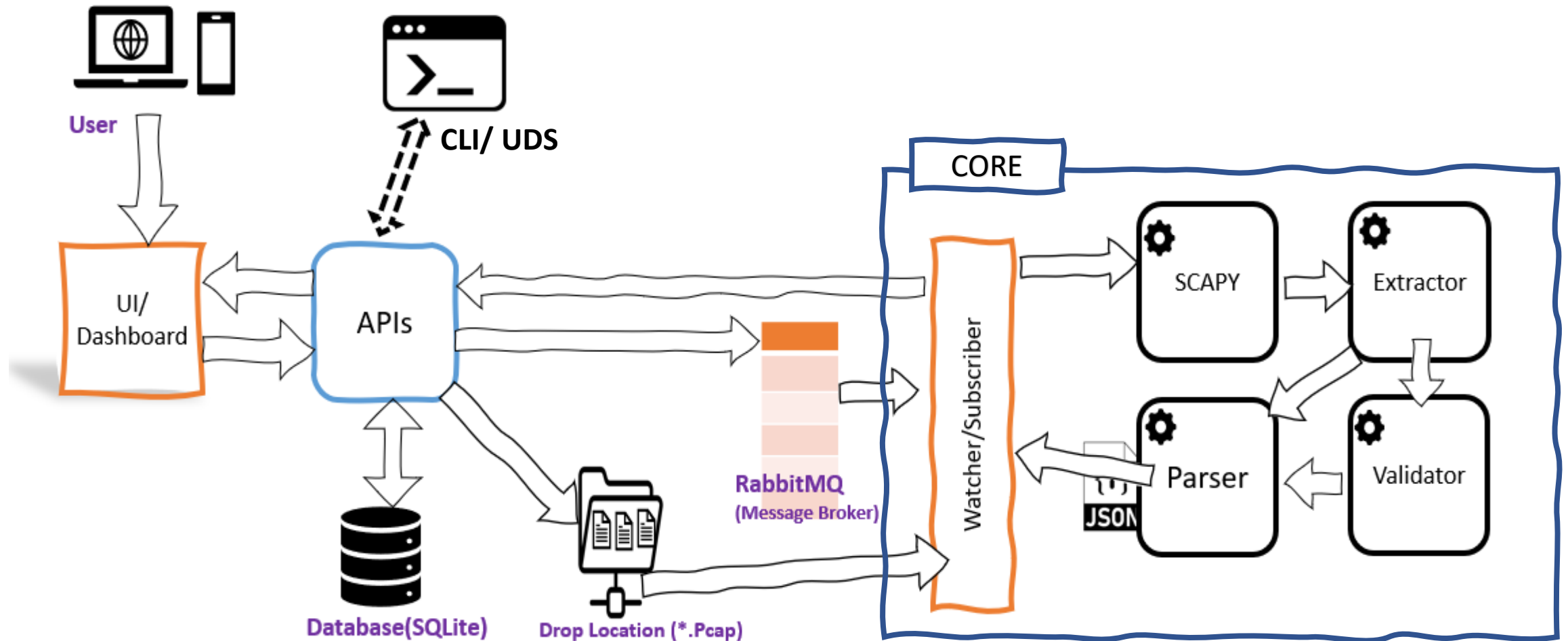
<https://github.com/JawedCIA/ArkThor>

## **2. Public Version**

<https://arkthor.azurewebsites.net/>



# ArkThor Architect Workflow Diagram

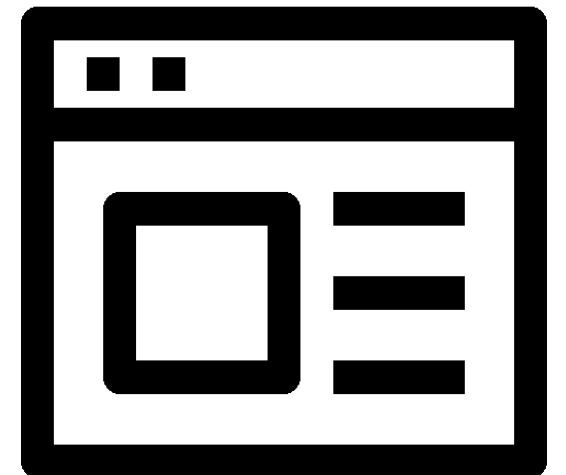


# Front Engine - UI

- **The topmost layer of the ArkThor product.**
- UI Layer provides end-users with a comprehensive set of information, enabling them to gain valuable insights into the threat landscape and take proactive measures to prevent and mitigate cyber-attacks.
- This layer is built on ASP.NET Core with Bootstrap and JavaScript, and its user interface design is inspired by AdminLTE.
- The user interface layer directly interacts with users, transmitting uploaded information to the layer immediately below (APIs) for further processing.
- The user interface layer also retrieves information from the APIs layer and presents it to end-users in both visual and textual formats.

**Currently, the UI Layer include the following features:**

1. **Dashboard**
2. **Analysis Records**
3. **Statistics – Measurement**
4. **Live Tracking - ArkthorBoard**



# Front Engine – UI - Dashboard

"The ArkThor product features a dashboard that serves as the landing page."

This dashboard provides users with a comprehensive view of their measurements, allows them to search through the ArkThor internal database for threats, and upload files for threat categorization.

Upon upload, the file is internally checked against criteria such as file extension (\*.pcap, \*.pcapng), file size limit (128 MB), and file signature. If the check is successful, the file is handed over to the ArkThor APIs.

Furthermore, users can gain insight into the top 10 analysis records, each with unique properties, from here user can navigate to view more in-depth file analysis information.

ArkThor Dashboard (Sample For Public View Only)

Threat Categorization: (Safety and Strength)

Admin

Dashboard

Analysis Records

Statistics

Track Live Status

MISCELLANEOUS

Rules

Knowledge Base

Agent Pools

SUBSCRIBE

FAQ

1 #File Queued

9 #File Submitted

5 #File Analyzed

5 #Threat Categorizes

Threat Categorization

File(s) Threat Search

This is a Threat Categorization engine (ArkThor) based on Malware's C2 Communication of uploaded pcap file, Created and developed by Group-6 (Cohort-6, 2022-2023) of IITK CSDC program.

Select pcap file (Max.Size limit - 50 MB) Choose Analyze

Subscribe

Track Live Analysis

Threat Categorization Analyses based on C2 Communication (Top 10 Records)

ID	SHA-256	FileName	Uploaded Date	Status	Threat Category	Analysis Date	Current Stage
93C0	93C0F2AAE464004EAD4DC476FC97695AD72F6115C ED28AE9B014B07082D4AF8	2020-12-31-traffic-analysis-quiz-03.pcap	2023-04-04 07:37:10	Cancelled		2023-04-03 14:06:01	
34F7	34F77C61A58BE8816F842639A8B76178462CF6350899 097MF8E7AC825F972797	MD-NoThreat.pcap	2023-04-04 07:36:50	Done	NO THREAT	2023-04-02 14:06:01	
AB5A	AB5AFB0838C594EC00254624FA769D4C9BF1E7876BF 5ED0D10DCB49E2EFF9E94	2023-03-08-IcedID-with-BackConnect-and-VNC-traffic.pcap	2023-04-04 04:56:01	Done	BOMBOT	2023-04-04 04:56:01	
7A06	7A063B8B680DCF4AF8FE3FEAAE873853A24ED287E4FD D390ECC70331D30784233	2020-12-31-traffic-analysis-quiz-05.pcap	2023-04-03 21:37:17	Queued			
A86B	A86B11EE443EE59D95E45908DF761235C5D240128D6 D195233D065A857777E88	2022-01-07-traffic-analysis-exercise.pcap	2023-04-01 20:47:16	Inprogress			
D986	D9867E18BCA13CF6135874FD81D167403C7CB68BA4 22679948DB914E9959890D	2022-03-21-traffic-analysis-exercise.pcap	2023-04-01 20:47:11	Done	SUSPICIOUS	2023-04-03 04:56:01	

Using ArkThor Left navigation users can access other ArkThor product functionalities".

# Front Engine – UI - Records

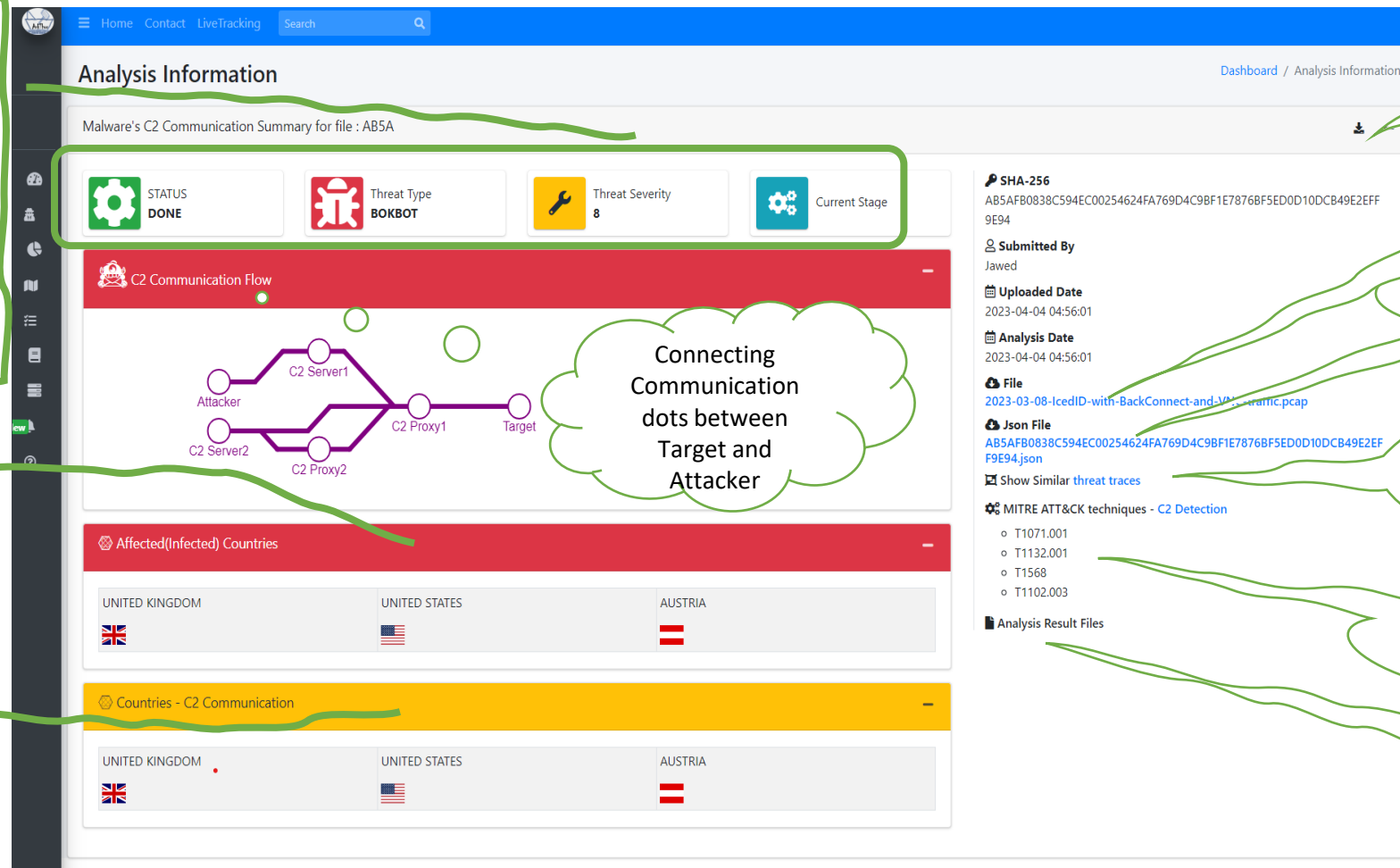
ArkThor users can access a comprehensive analysis report of the analyzed file from this page, which is accessible via the Dashboard, Records, or ArkThor Live Tracking Board.

The analyzed file contains critical information, including the

- Final status,
- Threat category type,
- Threat severity.

Affected Countries Names with Country Flag.

C2 Communication Countries with Names with Country Flag.



Download displayed information as PDF file.

Uploaded Pcap File

Final Analyzed JSON file

Navigate to File Records having similar threat category

MITRE ATT&CK techniques used

Intermedial Logs files

# Front Engine – UI - Statistics

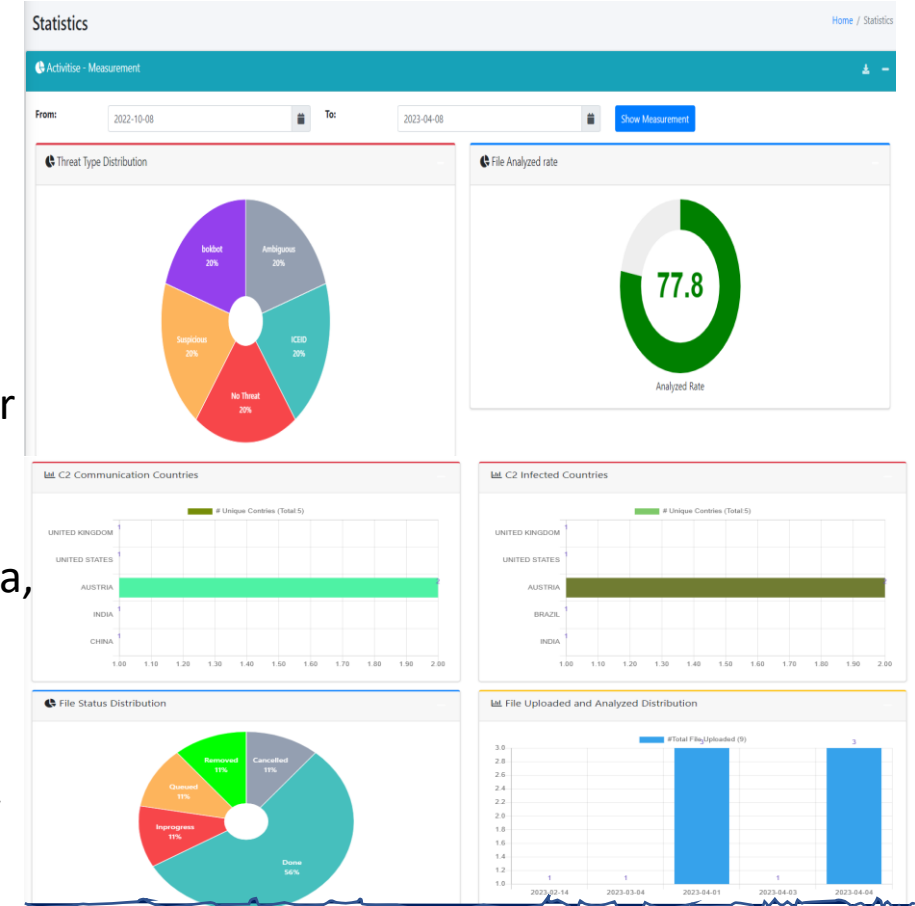
In the ArkThor product, we have developed a Statistics or Measurement page that showcases valuable Key Performance Indicators (KPIs) through the use of Pie Charts, Bar Graphs, and Knobs.

These visual aids help provide insightful and meaningful data to organizations based on the analysis records available in the ArkThor database. By presenting data in a clear and concise manner, users can quickly interpret and identify patterns and trends that can help them make informed decisions.

The data available on the Statistics or Measurement page in the ArkThor product is sourced from the ArkThor APIs, but organizations are not limited to only using these visualizations.

If preferred, organizations can utilize other available tools such as Kibana, Grafana, or other data visualization platforms to analyze the data available in the ArkThor database. The goal of ArkThor is to provide a comprehensive and flexible cybersecurity solution that can integrate with existing systems and tools, and the inclusion of APIs and the ability to export data is designed to facilitate this flexibility.

Therefore, organizations can choose the best approach for their unique needs and use the data as they see fit.



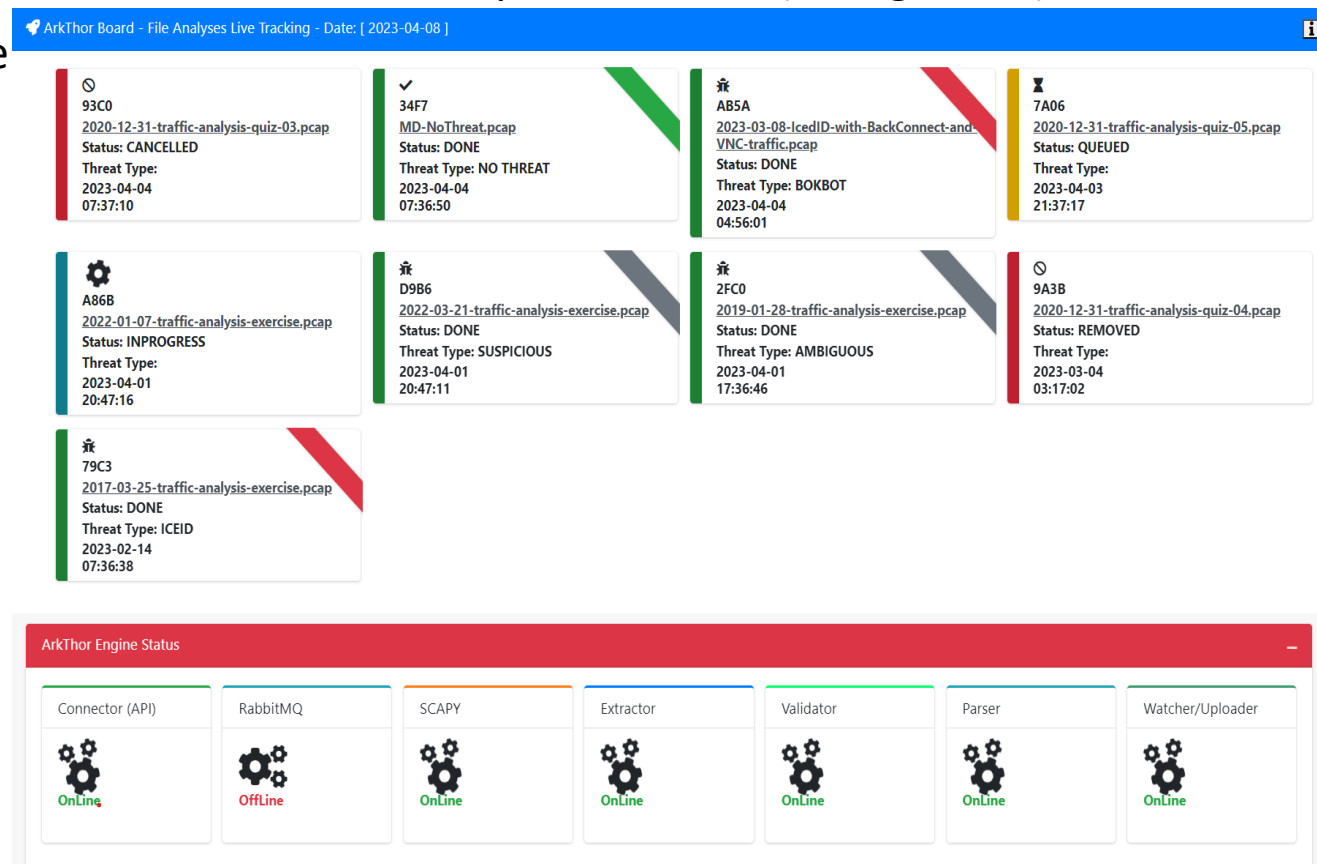
# Front Engine – UI - Board

The ArkThor product also features a dedicated page called the "ArkThor Board," which provides users with valuable insights into the analysis of files using the ArkThor system. This feature was developed with the idea that organizations may want to display all relevant information on a big screen for easy viewing and monitoring. The ArkThor Board is designed to show analysis information in real-time, with automatic refresh every 180 seconds (configurable).

To ensure the best visual appearance and usability, the ArkThor Board includes various indicators and visualizations that make it easy for users to quickly interpret and understand the data being presented, whether on a big screen or a mobile device.

The inclusion of these features enhances the overall value and usefulness of the ArkThor product, allowing organizations to stay on top of their cybersecurity posture with ease and efficiency.

*Additionally, The ArkThor Board also includes information on the current status of different ArkThor Engine's modules. This information helps users to understand whether the system is online or offline and whether any specific modules require attention.*





# Front Engine – APIs

In ArkThor, there are several APIs that have been developed using ASP.NET Core and C#. These APIs allow for the receipt of Pcap files, which are then stored in a database. Additionally, a physical copy of the Pcap file is created and stored on a share location. Finally, the SHA256 of the Pcap file is stored on RabbitMQ for CORE engine.

The APIs are available in several different types, including POST, GET, and PUT. These APIs are designed to work with ArkThor and include functionality such as uploading JSON results, uploading Pcap files, uploading support files, updating status and threat type, retrieving measurements, creating file records, and more.

By leveraging these APIs, users can easily integrate ArkThor into their existing workflows and gain access to its powerful analysis capabilities.

## FileUpload

POST	/api/FileUpload/UploadFileOutPutJson
POST	/api/FileUpload/UploadFileForAnalysis
POST	/api/FileUpload/UploadSupportingFile
GET	/api/FileUpload/GetSupportFiles

## ArkThor.API 1.0 OAS3

<http://localhost:33900/swagger/v1/swagger.json>

### FileRecord

PUT	/api/FileRecord/UpdateThreatType
PUT	/api/FileRecord/UpdateCurrentStage
PUT	/api/FileRecord/UpdateStatus
PUT	/api/FileRecord/UpdateAnalyzedDate
PUT	/api/FileRecord/UpdateSeverity
GET	/api/FileRecord/GetAllFileRecord
GET	/api/FileRecord/GetDashboardCount
GET	/api/FileRecord/GetFileRecordByUploadedDate
GET	/api/FileRecord/GetFilesDistributionBasedOnAnalyzededDate
GET	/api/FileRecord/GetC2CountriesDistributionBasedOnUploadedDate
GET	/api/FileRecord/GetC2InfectedCountriesDistributionBasedOnUploadedDate
GET	/api/FileRecord/GetFilesDistributionBasedOnUploadedDate
GET	/api/FileRecord/GetThreatDistributionBasedOnUploadedDate
GET	/api/FileRecord/GetStatusDistributionBasedOnUploadedDate
GET	/api/FileRecord/GetSimilarThreatRecords
GET	/api/FileRecord/GetTOPFileRecord
GET	/api/FileRecord/GetByHash
POST	/api/FileRecord/CreateFileRecord



# Front Engine – APIs – Database

ArkThor utilizes a SQLite database to store all pertinent information on uploaded Pcap files, which it treats as assets.

The system presently includes two separate databases:

1. **FilesRecord**, which stores both the information on the uploaded Pcap file and its corresponding analyzed results, and
2. **SupportFile**, which contains information on all intermediate analyzed results files associated with each Pcap file.

The screenshot displays a SQLite database interface. On the left, the 'Structure' tab shows the 'FilesRecord' table with the following fields: HashValue (TEXT, NOT NULL, PRIMARY KEY, UNIQUE), FileName (TEXT), UploadedBy (TEXT), UploadedDate (NUMERIC), ThreatType (TEXT), Status (TEXT), Isold (INTEGER), Extension (TEXT), ContentType (TEXT), Size (TEXT), AnalyzedDate (NUMERIC), Data (BLOB), Severity (INTEGER), C2Countries (TEXT), C2Communication (TEXT), CurrentStage (TEXT), Extractor (TEXT), Validator (TEXT), Parser (TEXT), JsonData (BLOB), MITRE (TEXT), and infected\_countries (TEXT). In the center, the 'Databases' pane shows the 'ArkThor (SQLite 3)' database containing two tables: 'FilesRecord' and 'SupportFile'. On the right, the 'Structure' tab shows the 'SupportFile' table with the following fields: HashValue (TEXT, NOT NULL), FileName (TEXT), UploadedBy (TEXT), UploadedDate (NUMERIC), Extension (TEXT), ContentType (TEXT), Size (TEXT), Data (BLOB), Isold (INTEGER), ID (INTEGER, PRIMARY KEY, UNIQUE, NOT NULL), and an empty field. The SQLite logo and 'Database' text are visible in the top right corner.

```
CREATE TABLE FilesRecord (  
  HashValue TEXT NOT NULL PRIMARY KEY UNIQUE,  
  FileName TEXT,  
  UploadedBy TEXT,  
  UploadedDate NUMERIC,  
  ThreatType TEXT,  
  Status TEXT,  
  Isold INTEGER,  
  Extension TEXT,  
  ContentType TEXT,  
  Size TEXT,  
  AnalyzedDate NUMERIC,  
  Data BLOB,  
  Severity INTEGER,  
  C2Countries TEXT,  
  C2Communication TEXT,  
  CurrentStage TEXT,  
  Extractor TEXT,  
  Validator TEXT,  
  Parser TEXT,  
  JsonData BLOB,  
  MITRE TEXT,  
  infected_countries TEXT  
);
```

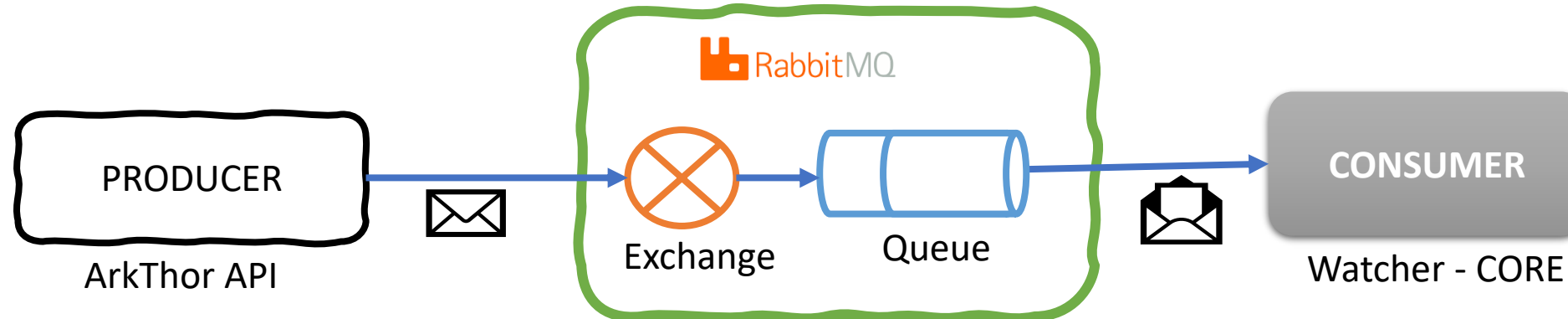
Databases

Filter by name

- ArkThor (SQLite 3)
  - Tables (2)
    - FilesRecord
    - SupportFile
  - Views

```
CREATE TABLE SupportFile (  
  HashValue TEXT NOT NULL,  
  FileName TEXT,  
  UploadedBy TEXT,  
  UploadedDate NUMERIC,  
  Extension TEXT,  
  ContentType TEXT,  
  Size TEXT,  
  Data BLOB,  
  Isold INTEGER,  
  ID INTEGER PRIMARY KEY UNIQUE NOT NULL  
);
```

# Front Engine – APIs – Message broker



RabbitMQ is a popular open-source message broker that can be used to manage message queues, route messages between applications, and distribute work across multiple systems. It provides a reliable and scalable messaging system that can be integrated with various programming languages and frameworks.

In this project, RabbitMQ is used as a message broker in this project to alert CORE about uploaded pcap file for analysis. However, in future we can use this to manage the communication between different module /components of the ArkThor. For example, the Pcap files can be processed by a component that reads the files and extracts relevant features, and then sends the feature data to another component that applies machine learning algorithms(Future work) for threat categorization. RabbitMQ can be used to manage the message queues between these components, ensuring that messages are delivered reliably and efficiently.

RabbitMQ can also be used to distribute work across multiple systems. For example, the machine learning algorithms(Future work) may require significant computational resources to process large volumes of data. RabbitMQ can be used to distribute the workload across multiple systems, ensuring that the processing is efficient and scalable.

# Core Engine

---

The core engine is made of 3 independent modules

- **Packet Processing module**

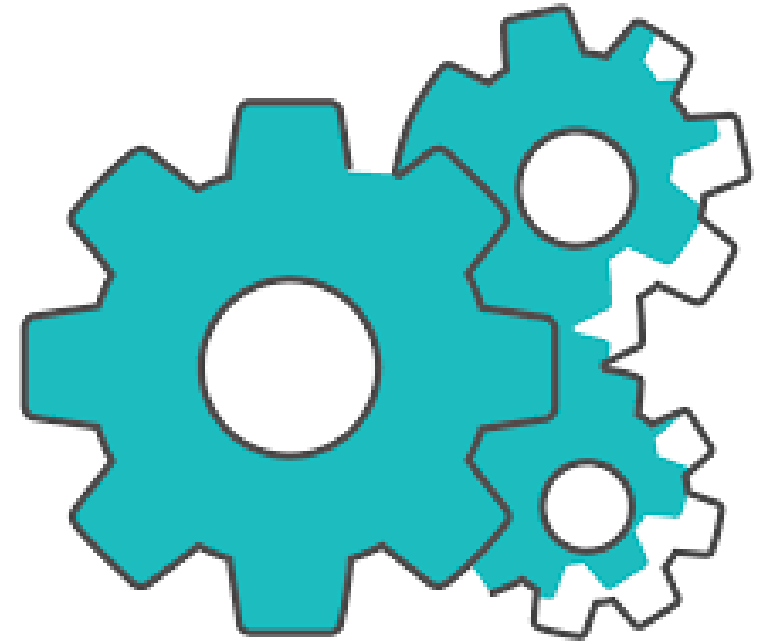
The packet processing module is built around Scapy, an opensource library.

- **Rule Parser module**

Loads the ArkThor format rules and matches them with the output of Packet Processing module

- **Rule Authoring module**

Contains rule component that convert open source rules or human authored rules



# Core Engine – The Packet Processing

## Features

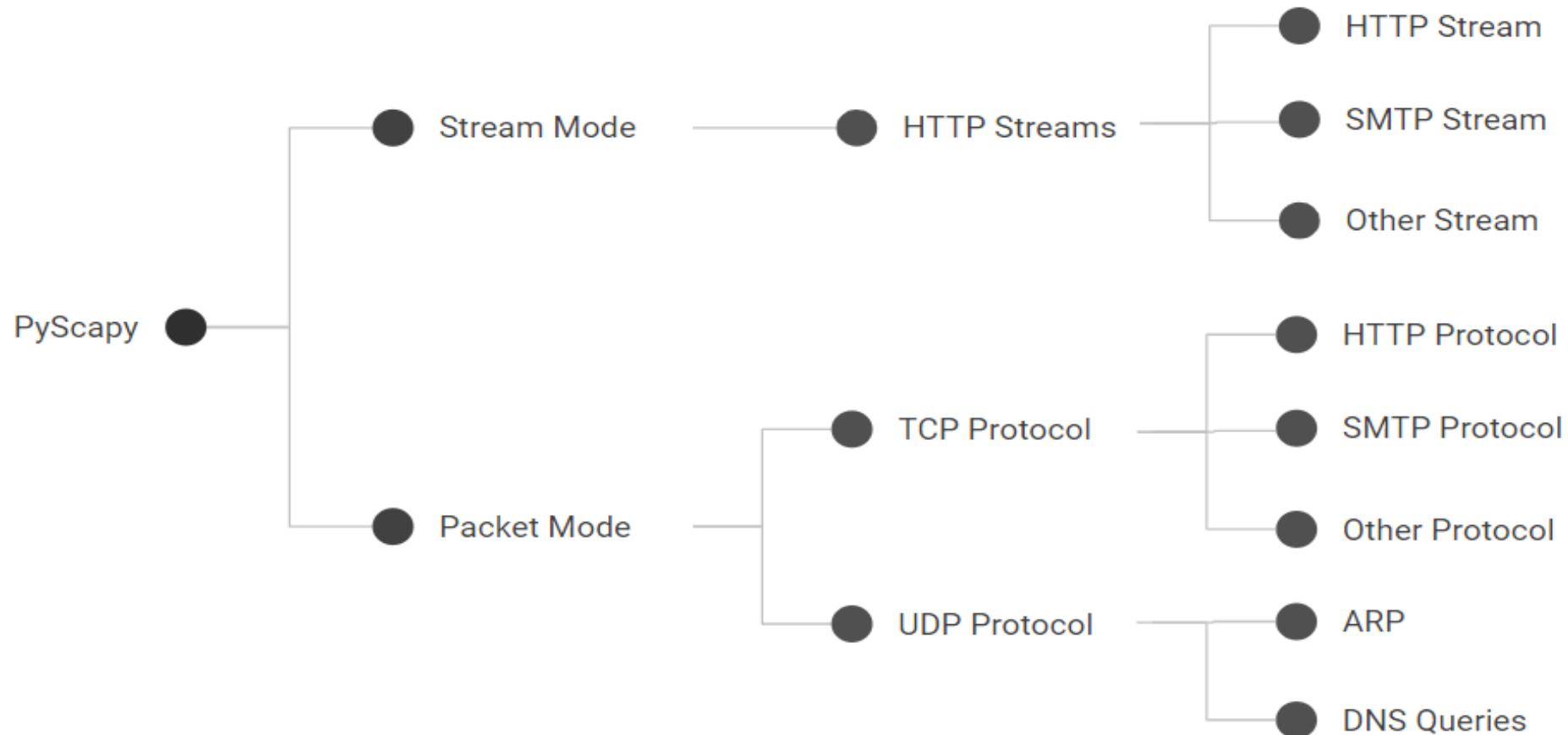
- Packet processing module extracts packet in both stream mode and packet mode
- It is designed to process both UDP and TCP packets
- It extracts both IPV4 and IPV6 packets
- Saves extracted artifacts as json files and stores them as artifacts
- The Artifacts can be used to further author ArkThor rules
- Relates DNS requests and Responses
- Support for HTTP 1.2 and HTTP 2.0
- TLS certificates are extracted and validates
- Raw files are stored and enables yara to be run over them
- Sniffs TLS 1.2 (Perfect Forward Secrecy) packets for known private keys
- Allows authoring of plugins for further data processing and extraction

### Scapy

Scapy is able to forge or decode packets of a wide number of protocols, send them on the wire, capture them, match requests and replies, and much more.

# Core Engine – The Packet Processing

## Data Availability



# Core Engine – The Packet Processing

## DNS Data

The packet processing engine Parses the DNS packet with the query sent to lifeinsurencequotes.xyz and the response IP of 146.19.230.208 is being parsed and stored to the dns artifact of the packet.

Wireshark · Follow UDP Stream (udp.stream eq 39) · 2023-03-08-IcedID-with-BackConnect-and-VNC-traffic.pcap

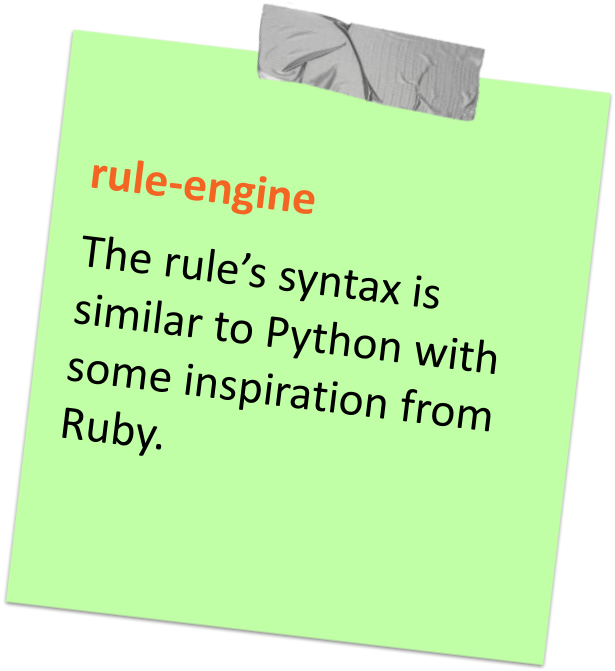
```
00000000 37 b1 01 00 00 01 00 00 00 00 00 00 13 6c 69 66 7..... lif
00000010 65 69 6e 73 75 72 61 6e 63 65 71 75 6f 74 65 73 einsuran cequotes
00000020 03 78 79 7a 00 00 01 00 01 .xyz.... .
00000000 37 b1 81 80 00 01 00 01 00 00 00 00 13 6c 69 66 7..... lif
00000010 65 69 6e 73 75 72 61 6e 63 65 71 75 6f 74 65 73 einsuran cequotes
00000020 03 78 79 7a 00 00 01 00 01 c0 0c 00 01 00 01 00 .xyz.... .
00000030 00 02 58 00 04 92 13 e6 d0 ..X..... .
```

```
{
  "lifeinsurancequotes.xyz.": {
    "dns_query_num": 1,
    "dns_query_type": "A",
    "response_code": 0,
    "answers": "146.19.230.208",
    "CN": "GB"
  },
}
```

# Core Engine – The Rule Parser

## Features

- Accepts single fields based matches
- Accepts equal comparison
- Accepts python regexes
- Comparison for datetime
- Uses open-source rule-engine Project



### **rule-engine**

The rule's syntax is similar to Python with some inspiration from Ruby.

# Core Engine – The Rule Parser

## Sample Rule

- A sample ArkThor rule that utilizes the domain name and the IP addresses as rule for classification. You have the flexibility to specify the MITRE associated with so ArkThor can give a summary of all the MITRE seen by the PCAP.

```
{
  "rule_name": "bokbot",
  "authored_timestamp": 1679144203,
  "rule": "domain == \"lifeinsurancequotes.xyz.\" or answers == \"146.19.230.208\"",
  "scope": "DNS",
  "severity": 2,
  "MITRE": [ "T1071.001", "T1132.001", "T1568", "T1102.003" ]
},
```

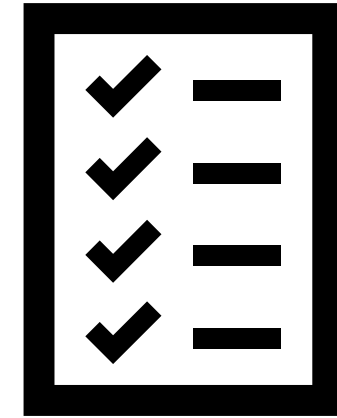


# Core Engine – The Rule Authoring

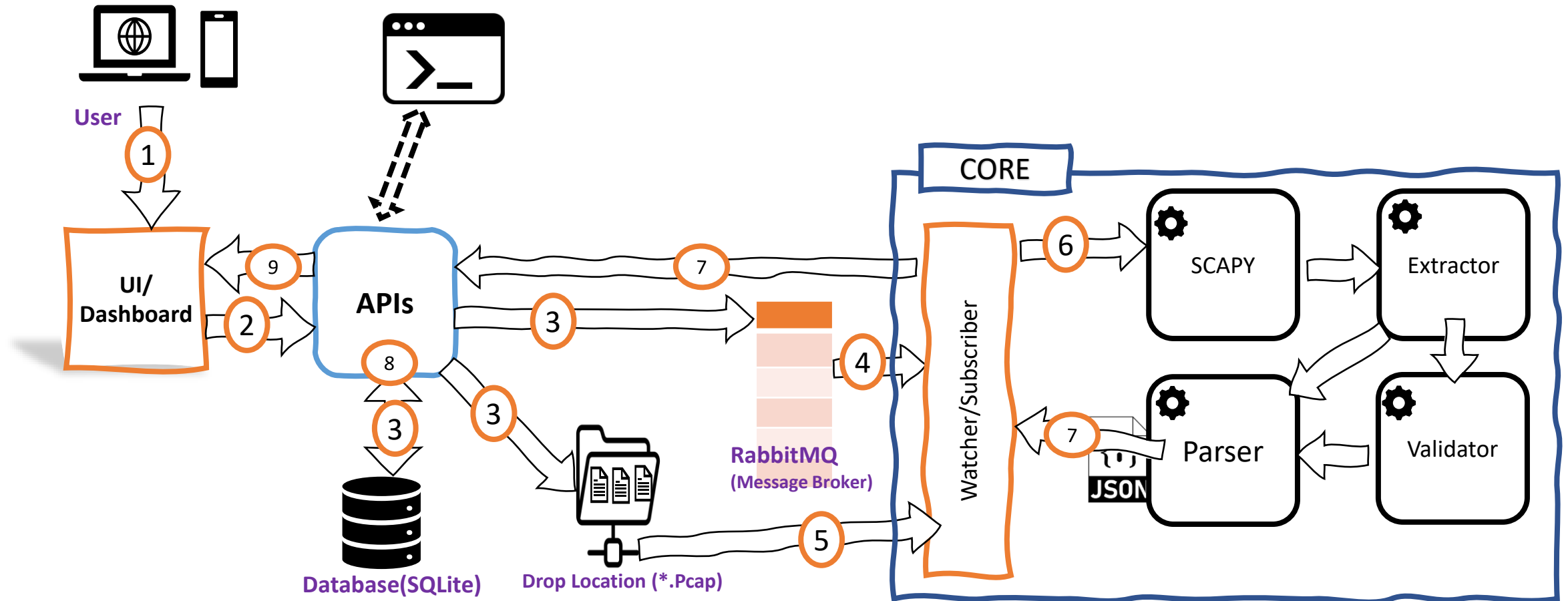
---

## Features

- Accepts single fields based matches
- Accepts equal comparison
- Accepts python regexes
- Comparison for datetime
- Uses opensource rule-engine Project



# ArkThor Connecting Dots ....



# End -to-End Working

## 1. User uploads Pcap file to the UI

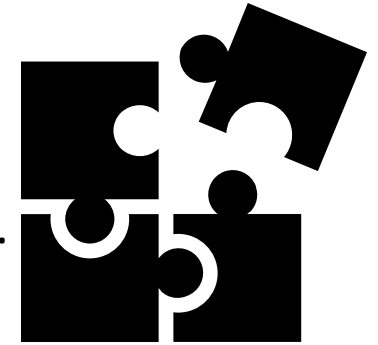
- The user selects a Pcap file from their local device and uploads it through the UI.

## 2. UI sends the Pcap file to the API

- Once the user has uploaded the file, the UI sends it to the API for storage and analysis.
- The API provides a RESTful endpoint that accepts Pcap files as input.

## 3. API saves the Pcap file in SQLLite database and on local drive

- Upon receiving the Pcap file, the API saves it in a shared directory for later use by CORE Engine. This directory can be specified in the configuration file of the API otherwise by default it will save at same location on API under “UploadedFiles” folder.
- The API also stores the file in a SQLite database table to keep track of all the files that have been analyzed. This table can contain information such as the file name, file size, upload date, UploadedBy, sha256 hash value, and the analysis status, C2 communication Countries, Threat type, Severity and much more data related to uploaded file.
- After saving the Pcap file, the API generates the sha256 hash value of the file and sends it to RabbitMQ as a message. This message serves as a notification to the core model that a new Pcap file is available for analysis.



# End -to-End Working

---

## 4. Core (Watcher Module) reads the sha256 from RabbitMQ

- The Watcher module is a separate application or process that runs in the background, continuously listening to the RabbitMQ message queue for new tasks.

## 5. Core (Watcher Module) picks the Pcap file for analysis

- Upon receiving a new message containing the sha256 hash value, the Watcher module retrieves the corresponding Pcap file from the shared directory by matching the hash values and pass on to SCAPY for threat categorization.

## 6. Core Engine analyzes the Pcap file for threat categorization

- All necessary checks for validating the pcap are done first, After validation, the pcap is loaded with scapy.
- Scapy runs first in stream mode to extract all the stream HTTP artifacts
- Scapy then runs in packet capture mode to extract UDP artifacts
- Once extracted, every components results are stored in their respective json
- The rule engine is now invoked by the watcher
- Rule engine then parses all the available rules and runs them over the generated json artifacts
- Results are aggregated and is given to aggregator which returns back with the valid threat family formulated by the rules.
- The analysis results are formatted as a JSON object that contains the Pcap file SHA256, the threat type, MITRE att&ck technique, and any other relevant information such as the severity score, the analyzed time, and so on.

# End -to-End Working

## 7. Core Engine submits the JSON object to the API as well as Status

- Once the analysis is complete, the core engine sends the JSON object containing the analysis results to the API through a RESTful endpoint.

## 8. The API saves the JSON object in a SQLite database table for later retrieval based on SHA256 of uploaded file

## 9. UI fetches the analysis results from the API and displays the results to the user.

- The UI retrieves the analysis results from the API through a RESTful endpoint by passing SHA256 of file.
- The UI parses the JSON object and displays the analysis results to the user in a user-friendly format, such as a table or a chart.
- The user can interact with the UI to view the analysis results of different Pcap files, filter the results based on different criteria, or export the results to a PDF file and view measurements.

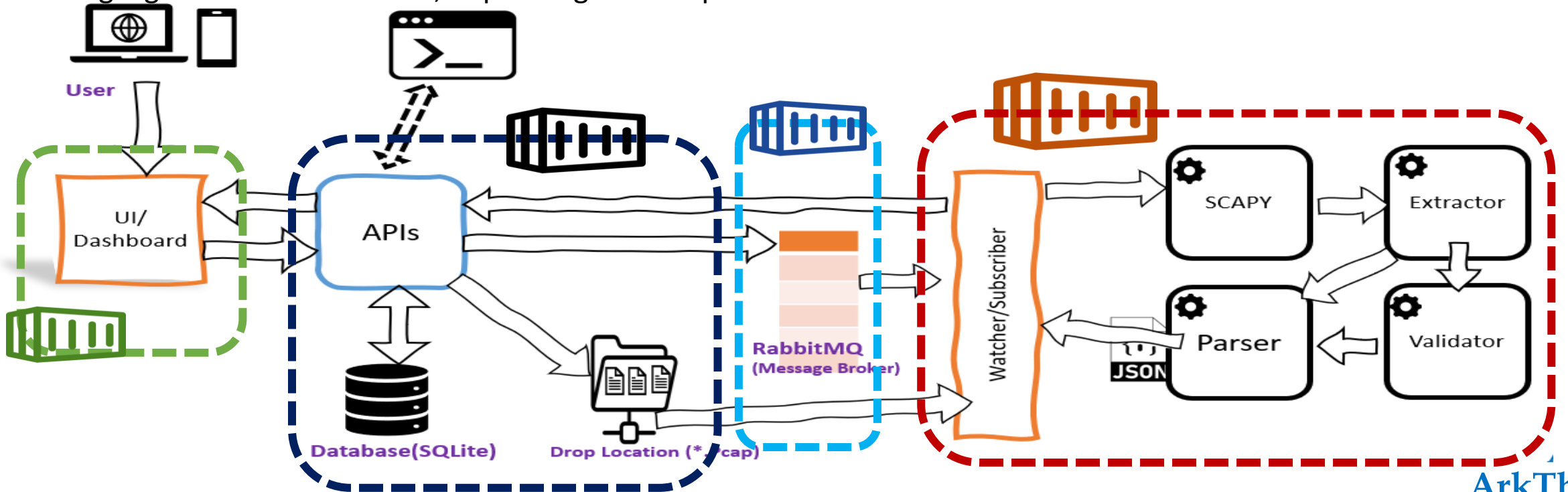
```
{
  "rule_name": "bokbot",
  "authored_timestamp": 67165768,
  "severity": 8,
  "MITRE": [
    "T1071.001",
    "T1132.001",
    "T1568",
    "T1102.003"
  ],
  "infected_countries": [
    "GB",
    "US",
    "AT"
  ],
  "c2_countries": [
    "GB",
    "US",
    "AT"
  ],
  "Status": "Done",
  "SHA256": "AB5AFB0838C594EC002 .....",
  "analyzed_time": 1680584161
}
```

# Containerize - ArkThor

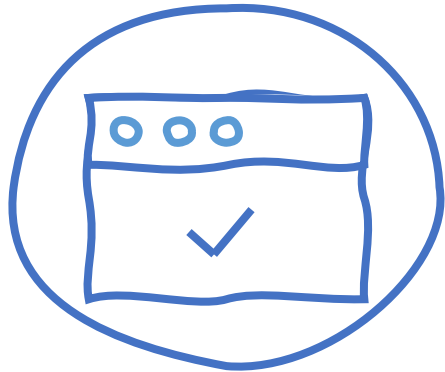
This workflow is designed to be scalable and fault-tolerant, meaning that it can handle large volumes of Pcap files and recover from failures or errors gracefully.

The use of RabbitMQ as a message queue ensures that messages are delivered in a reliable and efficient manner, while the use of SQLite database as a storage backend allows for easy management and retrieval of data.

(Future WORK) The core model can be trained and fine-tuned using different machine learning algorithms and datasets, depending on the specific use.

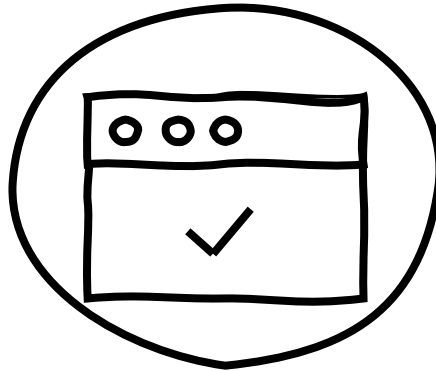


# Things to look forward



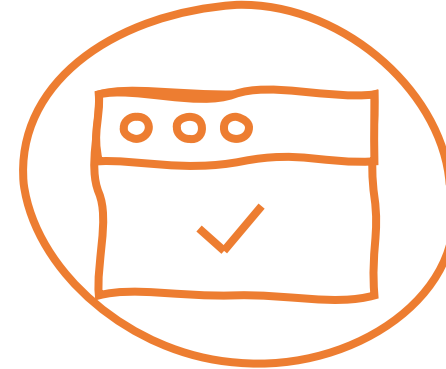
## CORE

- Live Dump of Wire
- Machine Learning Model and AI
- Provide Executive Summary in JSON format
- C2 Communication Nodes Details



## APIs

- Explore uses of other Databases and Messaging System for Vertical growth
- Include more measurement APIs for UI
- Allow notification using Email as well as SMS with Weekly reports.



## UI

- Include Search functionality based on SHA256 of file as well as based on Threat type
- Allow information from outside world
- Show Executive Summary of Analysis Summary
- Display C2 communication Nodes graph

# Conclusion

---

The capstone project that gave birth to the ArkThor product is a cutting-edge threat categorization engine that utilizes various moveable modules based on the analysis of C2 communication in Pcap files. The result is a state-of-the-art system that is user-friendly and provides organizations with the ability to visualize their overall security posture while improving their threat detection capabilities.

The ArkThor product includes various APIs that enhance its flexibility and scalability, and a SQLite database for data storage. The use of RabbitMQ allows for message queuing and reliable communication between different components, together with all components, making it a powerful tool for organizations of all sizes. The product has been designed to be easily deployable, with a containerized architecture that allows it to be used as a plug-and-analyze solution in any organization.

The ArkThor product has significant implications for the field of cybersecurity, providing organizations with an advanced system for identifying and mitigating threats before they cause damage. Its user-friendly interface and scalability make it an ideal solution for organizations of all sizes, while its cutting-edge technology ensures that it stays ahead of malicious actors who seek to exploit vulnerabilities in an organization's digital infrastructure.

Overall, the containerization of all components, combined with the powerful threat categorization engine and user-friendly interface, makes ArkThor a valuable tool for organizations looking to improve their threat detection capabilities and overall security posture.



# References

---

UI

- [AdminLTE](#)
- [ASP.Net Core](#)
- [Bootstrap](#)
- [Tabler \(Flags\)](#)
- [JavaScript](#)

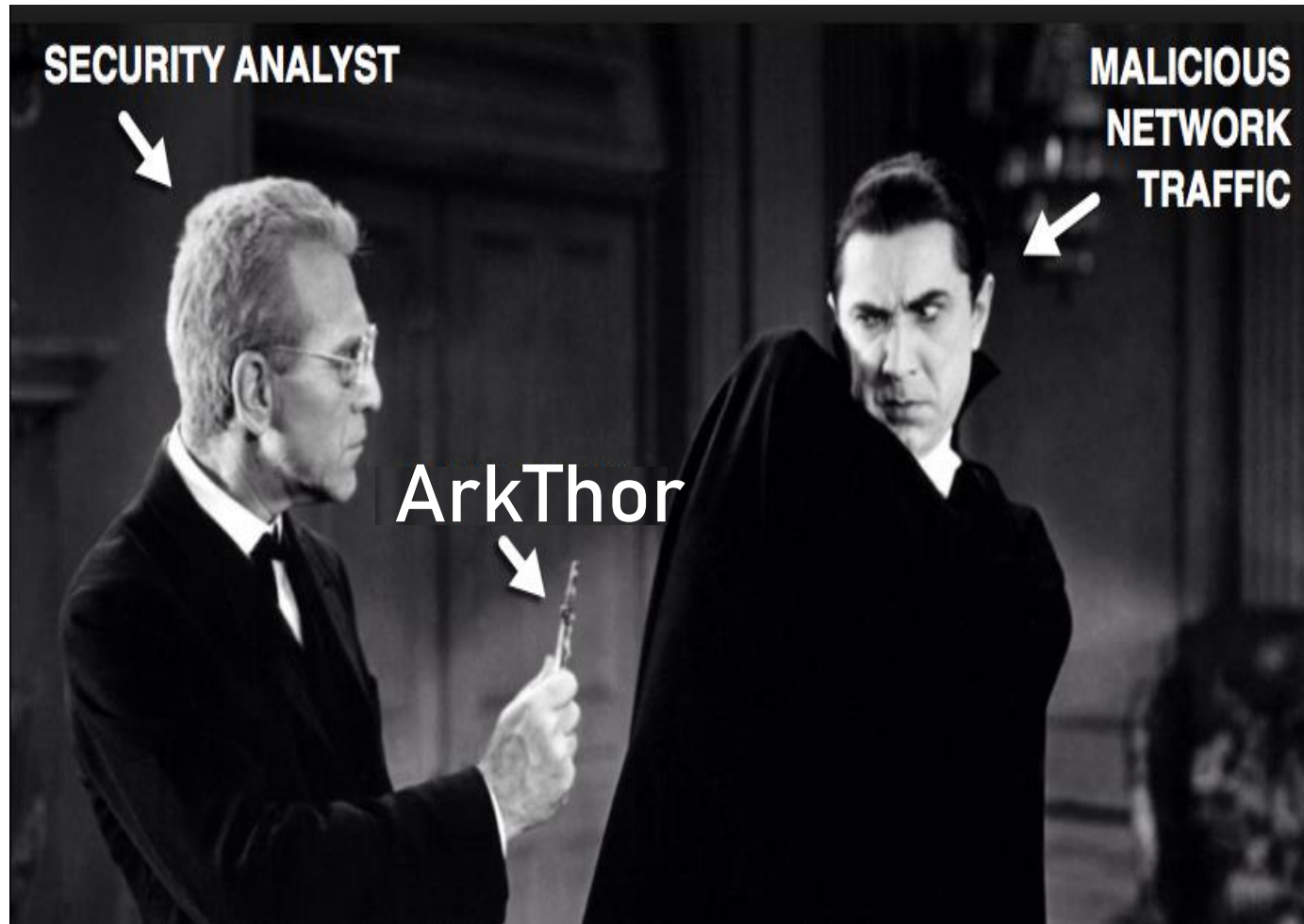
API

- [SQLite](#)
- [RabbitMQ](#)
- [ASP.Net Core](#)
- [C#](#)

CORE

- [Scapy](#)
- [Rule-Engine](#)
- [Threatfox](#)
- [python](#)

# Thank you!



Don't wait any longer, head over to Github and/or DockerHub and get your hands on the ArkThor product today!

This powerful tool is available for free and can help you improve your organization's security posture and threat detection capabilities.

Use it to grow and enhance your security infrastructure, and be sure to provide feedback to help the product evolve and improve over time.

Don't miss out on this opportunity to take your cybersecurity to the next level - download ArkThor today!

[Arkthor.help@gmail.com](mailto:Arkthor.help@gmail.com)

**ArkThor**