

AI + Security Level 1

Lab Practice

Module-8

Lab Practice-8.1:

Title: AI-Powered Threat Intelligence Using Open Source Data

Objective:

To utilize AI and machine learning techniques to gather, analyze, and interpret open-source threat intelligence data, enabling proactive threat detection and enhancing an organization's security posture.

Problem Statement:

With the abundance of open-source intelligence (OSINT) data available, organizations face challenges in effectively identifying and analyzing relevant threat information. Manual analysis is often too slow to keep up with emerging threats. By applying AI to OSINT, we can automate threat detection and gain actionable insights into potential security risks from various online sources.

Lab Practice-8.1:

Title: AI-Powered Threat Intelligence Using Open Source Data

Objective:

To utilize AI and machine learning techniques to gather, analyze, and interpret open-source threat intelligence data, enabling proactive threat detection and enhancing an organization's security posture.

Problem Statement:

With the abundance of open-source intelligence (OSINT) data available, organizations face challenges in effectively identifying and analyzing relevant threat information. Manual analysis is often too slow to keep up with emerging threats. By applying AI to OSINT, we can automate threat detection and gain actionable insights into potential security risks from various online sources.

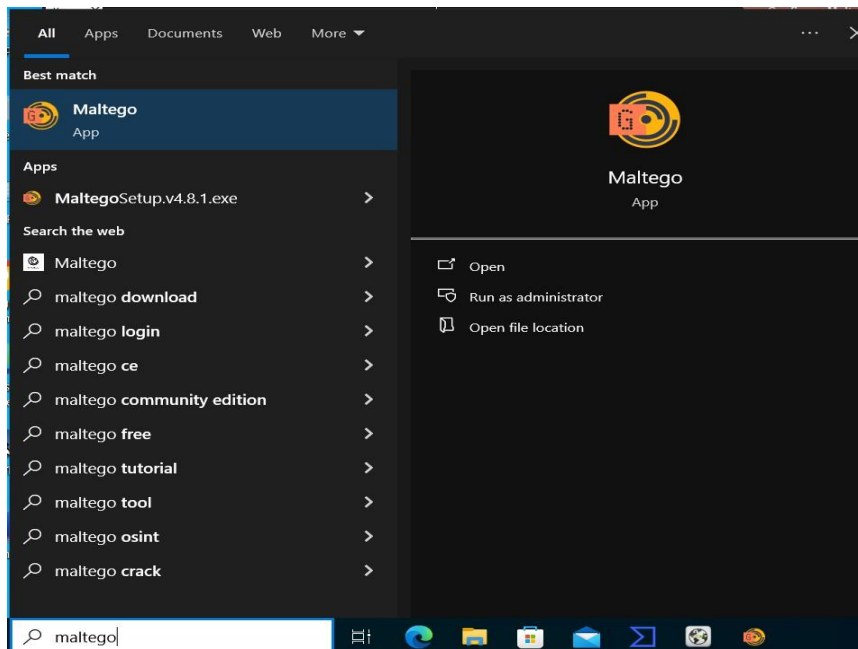
Required Tools:

- **Maltego CE (Community Edition):** A free tool for conducting small-scale OSINT investigations and visualizing threat intelligence data.
- **AI GPT tool (Security Testing:** <https://chatgpt.com/g/g-QOJkwOTCy-security-testing>)

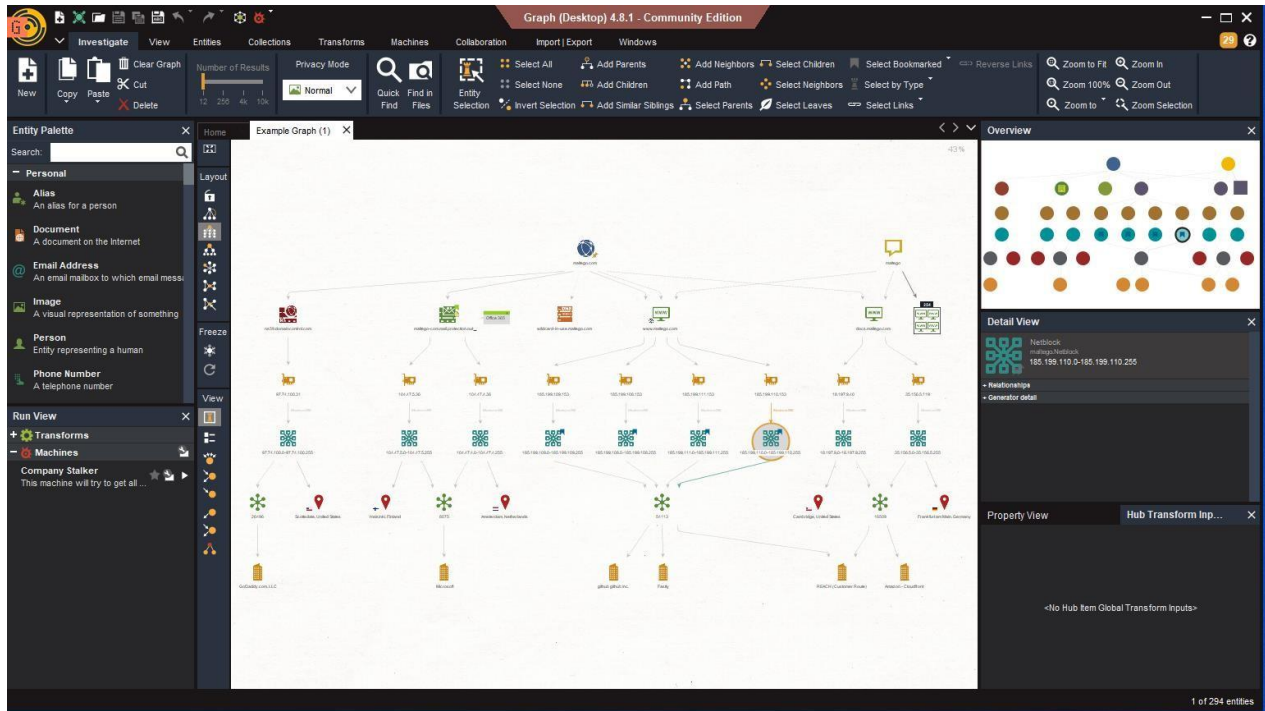
Step by step approach (Detailed Manner)

Step-1: Run Maltego CE and Gather information

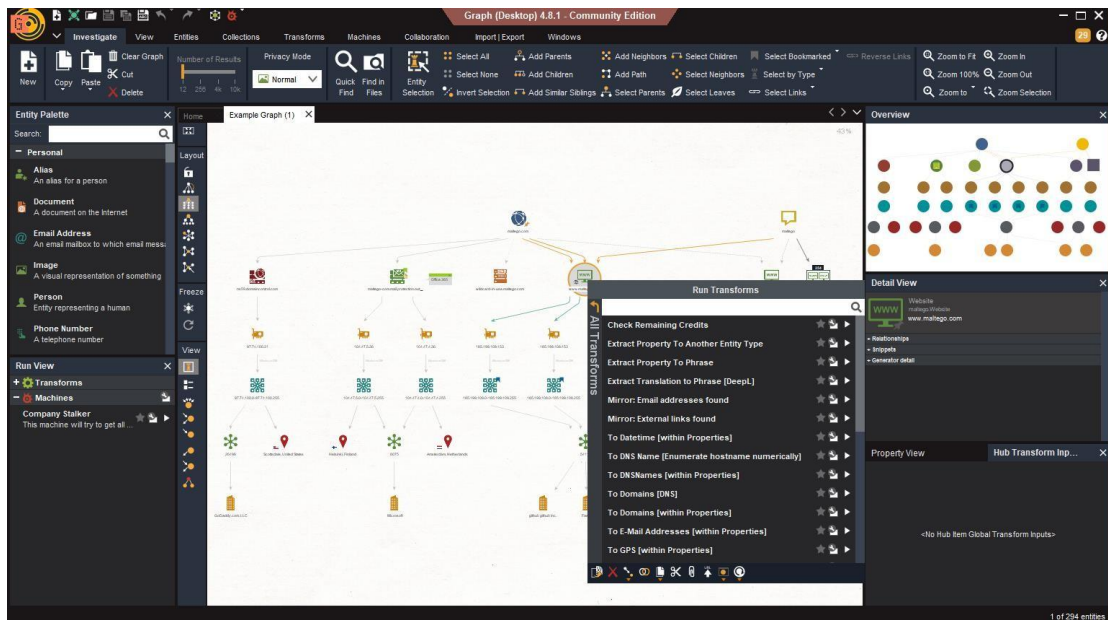
1. Search for Maltego in search bar and open it



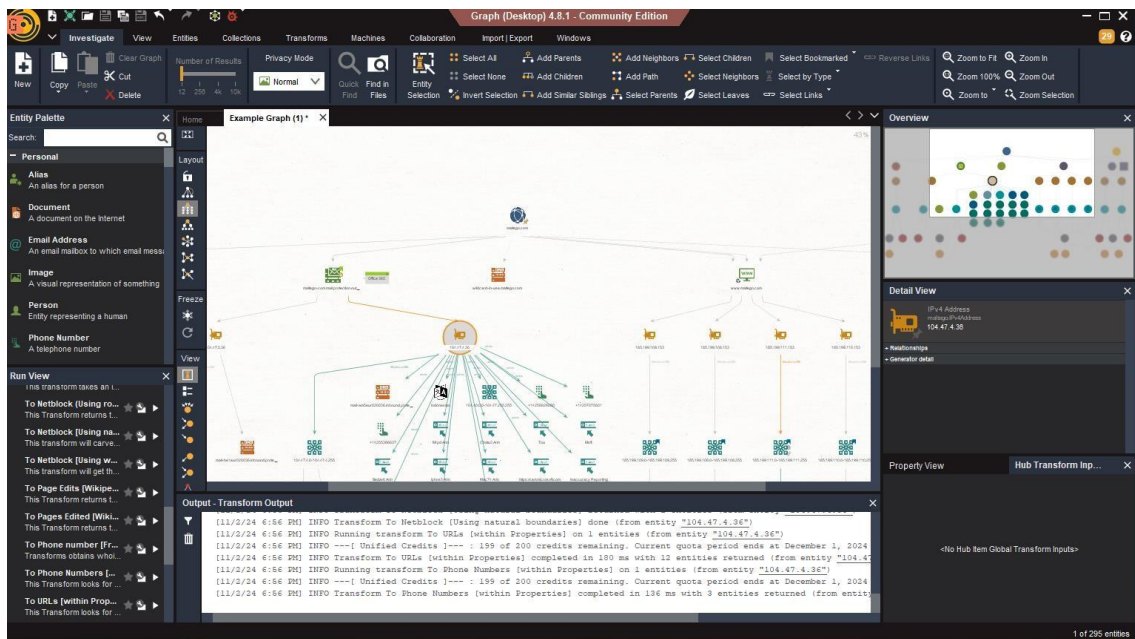
2. Give the valid URL for information Gathering such as www.nmap.org
3. Expand the domain entity of maltego.com by going to all transforms and running various transforms like DNS names, IP address, MX records, NS records, SOA, email addresses, Whois information to name a few. You see the Tree structure of www.maltego.com



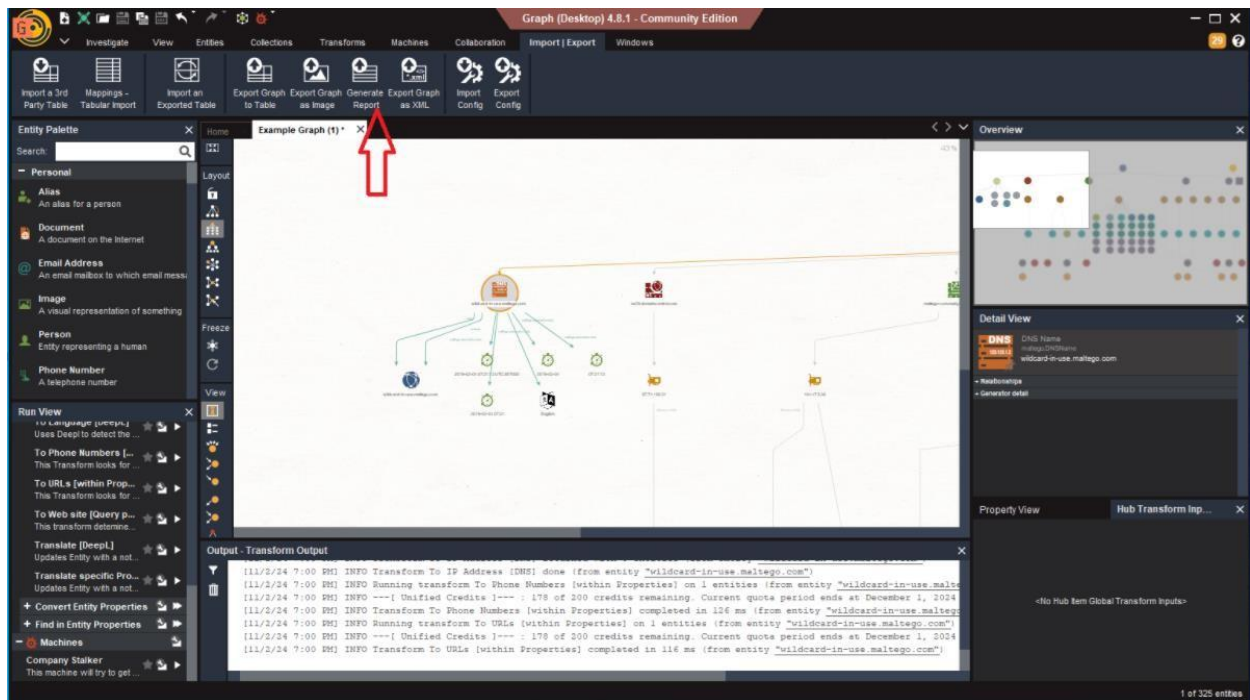
4. Right on any entity and apply all transforms.
 - **Entities:** Use Maltego's "Entities" (e.g., domains, IPs, registrant names, email addresses) that represent data types relevant to your investigation.
 - **Transforms:** Apply "transforms" to entities to gather deeper information, such as resolving a domain to IP addresses or identifying registrant details.
 - **Visualization:** Maltego visualizes relationships between entities. For example, it can show connections between suspicious IP addresses and domains, helping you to spot patterns.



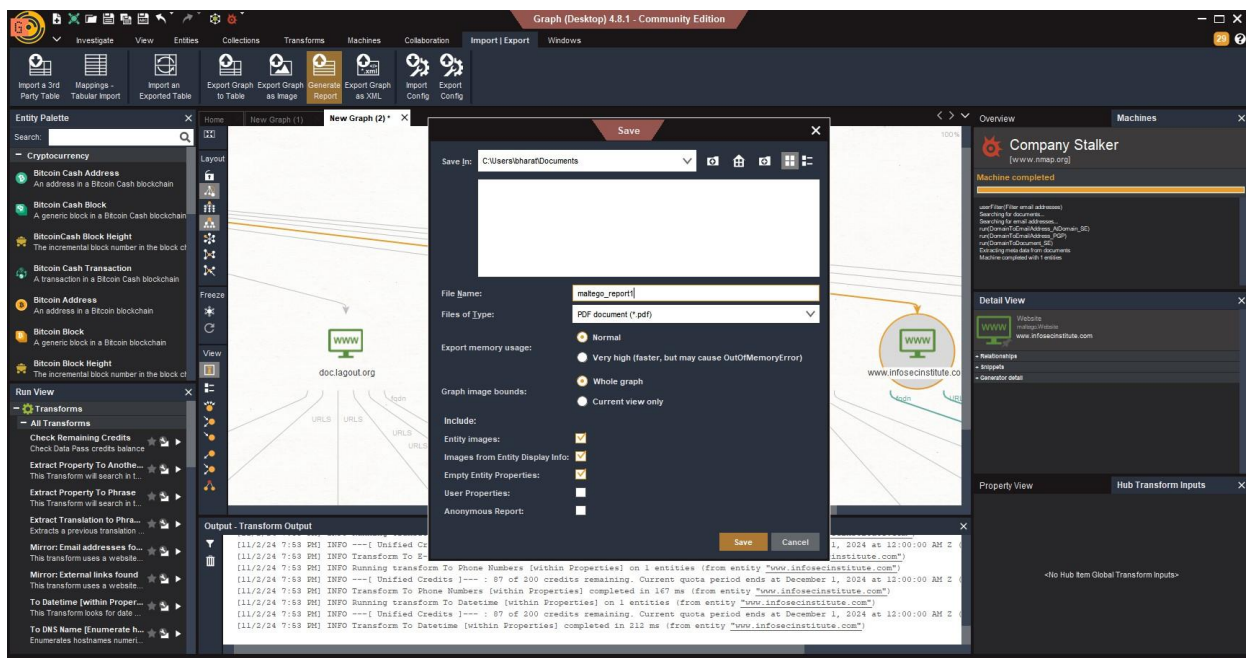
5. Apply all Transforms one by one and check it in visualization



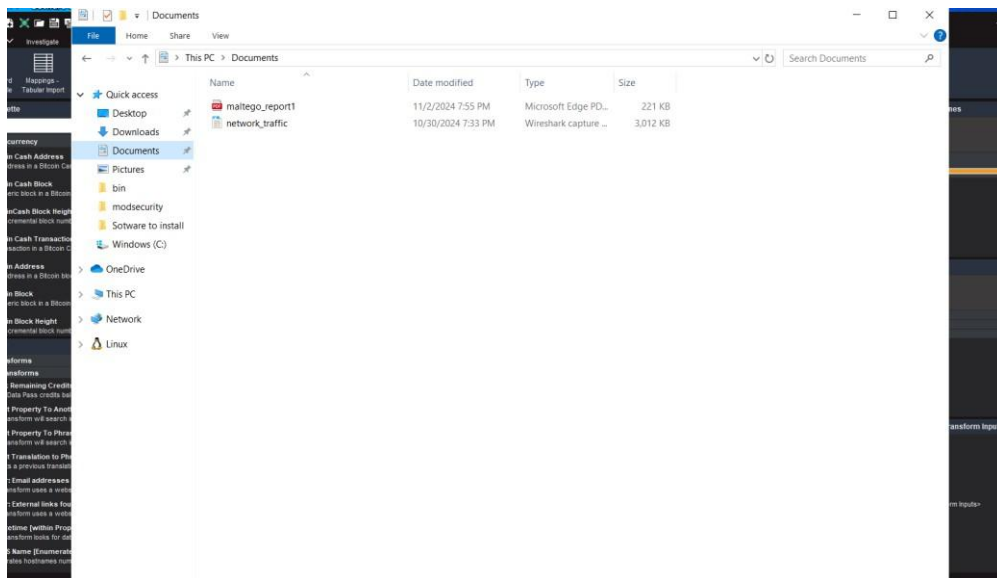
6. Generate the report



7. Generate the report for further analysis using AI tools



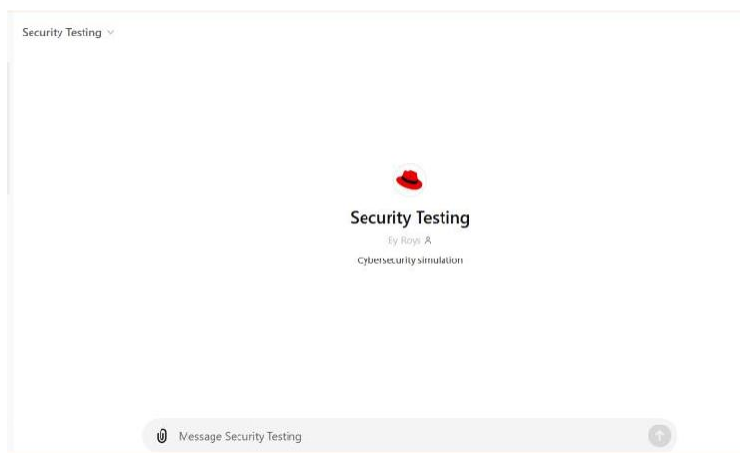
8. Check it in my document space, It looks like:



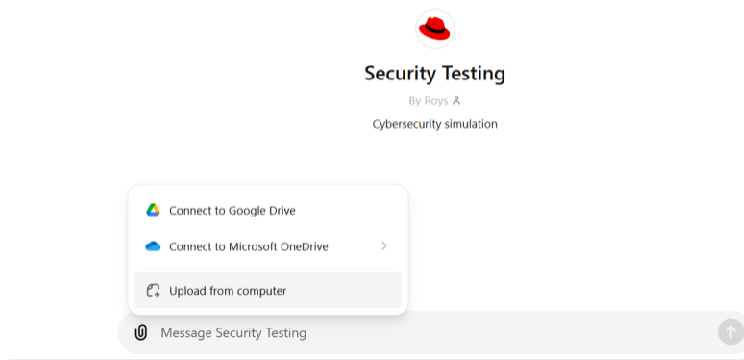
Step 3: Extract and Process the Data

Use Security Testing GPT: <https://chatgpt.com/g/g-QOJkwOTCy-security-testing>

This is the Screen of Security testing GPT



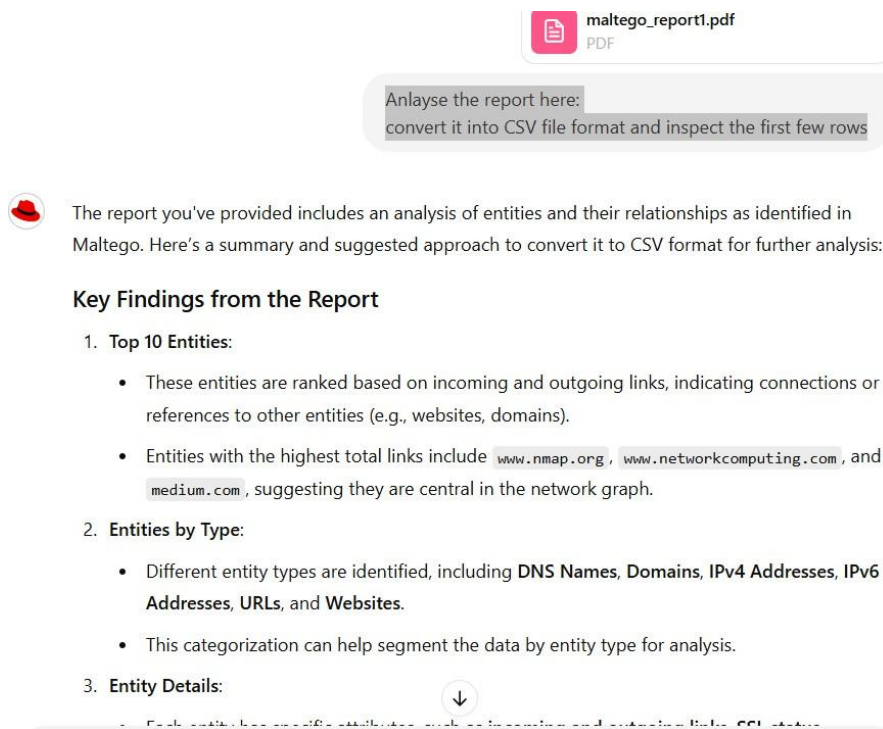
Upload the Report for analyzing the vulnerability.



3.1 : Data Export from Maltego

a) To Export Data from Maltego:

Prompt-1: Analyse the report here: convert it into CSV file format and inspect the first few rows



Sample Output

The report you've provided includes an analysis of entities and their relationships as identified in Maltego. Here's a summary and suggested approach to convert it to CSV format for further analysis:

Key Findings from the Report

1. Top 10 Entities:

- These entities are ranked based on incoming and outgoing links, indicating connections or references to other entities (e.g., websites, domains).
- Entities with the highest total links include www.nmap.org, www.networkcomputing.com, and medium.com, suggesting they are central in the network graph.

2. Entities by Type:

- Different entity types are identified, including DNS Names, Domains, IPv4 Addresses, IPv6 Addresses, URLs, and Websites.
- This categorization can help segment the data by entity type for analysis.

3. Entity Details:

- Each entity has specific attributes, such as incoming and outgoing links, SSL status, associated ports, and related URLs.

Converting to CSV Format

To convert the details into a structured CSV format, here's an approach:

1. Define the Columns:

- Columns can include Entity Type, Entity Value, Incoming Links, Outgoing Links, Total Links, SSL Enabled, Ports, and URLs.

2. Organize Data:

- Each entity's details can be recorded as a row in the CSV file.
- For entities with multiple values (e.g., multiple outgoing links), list them in a comma-separated format.

Prompt-2: From the above report, load my Maltego export into Pandas and show the column names here.

Sample output:

From the above report, load my Maltego export into Pandas and show the column names here

< 3/3 >



The columns in the "Top Entities by Incoming Links" section of the Maltego export are:

- Rank
- Type
- Value
- Incoming Links

Let me know if you'd like to load other sections or perform additional analysis. [\[?\]](#)

3.2: Feature Engineering Prompts

1. To Create Frequency-Based Features:

Prompt-3: *Count the frequency of each unique IP address or domain in my dataset here*

Sample Output:

Count the frequency of each unique IP address or domain in my dataset here

< 2/2 >



The frequency of each unique IP address or domain in the dataset is as follows:

- lagout.org: 1
- networkcomputing.com: 1
- www.networkcomputing.com: 1
- www.nmap.org: 1
- 104.16.255.154: 1
- 104.18.224.208: 1
- 162.159.152.4: 1
- 162.159.153.4: 1
- 2606:4700:7::a29f:9804: 1
- 2606:4700:7::a29f:9904: 1
- 2606:4700::6810
: 1
- 2606:4700::6812
: 1

Each IP address or domain appears exactly once in this dataset. Let me know if you'd like further analysis or filtering on specific types. [↗]

How do I count the frequency of each unique IP address or domain in my dataset?



The frequency count for each unique IP address or domain in the dataset is as follows:

- www.nmap.org: 1 occurrence
- 104.16.255.154: 1 occurrence

This indicates that each unique entity in this filtered subset appears once. Let me know if you need further analysis or if there's a larger dataset to explore! [↗]

Prompt-4: Add a column to my dataset that shows how many times each IP or domain appears. Give the final table

Sample output:

Add a column to my dataset that shows how many times each IP or domain appears. Give the final table

< 2/2 >



| | Entity Type | Entity Value | Incoming Links | Outgoing |
|---|-------------|--------------------------|----------------|----------|
| 1 | Domain | www.nmap.org | 1 | 12 |
| 2 | Website | www.networkcomputing.com | 2 | 7 |
| 3 | DNS Name | medium.com | 2 | 6 |

The dataset now includes a new column, **Frequency**, which displays how many times each IP address or domain appears. The table is ready for further analysis or any additional feature engineering steps you'd like to perform. Let me know if you need further assistance! [\[?\]](#)

Download the table to view all the columns and its associated values, such as:

| Entity Type | Entity Value | Incoming L | Outgoing L | Total Links | SSL Enable | Ports | URLs | Frequency | Frequency |
|--------------|--------------------------|------------|------------|-------------|------------|-------|----------------------------|-----------|-----------|
| Domain | www.nmap.org | 1 | 12 | 13 | No | 80 | https://www.nmap.org | 1 | 1 |
| Website | www.networkcomputing.com | 2 | 7 | 9 | No | 80 | https://www.networkcom | 0 | 0 |
| DNS Name | medium.com | 2 | 6 | 8 | No | N/A | https://medium.com/@by | 0 | 0 |
| IPv4 Address | 104.16.255.154 | 1 | 0 | 1 | N/A | N/A | N/A | 1 | 1 |
| IPv6 Address | 2606:4700:7::a29f:9e | 1 | 0 | 1 | N/A | N/A | N/A | 0 | 0 |
| URL | https://doc.lagout.org | 1 | 0 | 1 | N/A | N/A | https://doc.lagout.org/ope | 0 | 0 |

Prompt-5: Identify Suspicious Patterns in Registrant Names or Email Addresses here

Sample Output:



To identify suspicious patterns in registrant names or email addresses, here's a common approach that leverages patterns often associated with malicious entities:

1. **Randomized Characters:** Suspicious registrant names often contain random characters or numeric strings (e.g., "abc123" or "xJ9Lk2").
2. **Unusual TLDs:** Malicious domains may use uncommon top-level domains (TLDs), such as `.xyz`, `.top`, `.info`.
3. **Shortened Names or Emails:** Very short names or email addresses (e.g., one or two letters) can sometimes be a flag.
4. **Suspicious Phrases:** Terms like "admin," "support," or "service" in an email address can sometimes signal phishing or impersonation.

Steps to Identify Suspicious Patterns

We can implement these checks in Python. Assuming your dataset includes a column named `Registrant` (for registrant names) or `Email` (for email addresses), here's how we could code the detection:

Filter data by suspicious pattern flag

Generate summary of identified threat entities

To identify suspicious patterns in registrant names or email addresses, here's a common approach that leverages patterns often associated with malicious entities:

1. **Randomized Characters:** Suspicious registrant names often contain random characters or numeric strings (e.g., "abc123" or "xJ9Lk2").
2. **Unusual TLDs:** Malicious domains may use uncommon top-level domains (TLDs), such as `.xyz`, `.top`, `.info`.
3. **Shortened Names or Emails:** Very short names or email addresses (e.g., one or two letters) can sometimes be a flag.
4. **Suspicious Phrases:** Terms like "admin," "support," or "service" in an email address can sometimes signal phishing or impersonation.

Steps to Identify Suspicious Patterns

We can implement these checks in Python. Assuming your dataset includes a column named `Registrant` (for registrant names) or `Email` (for email addresses), here's how we could code the detection:

Code for Detecting Patterns

I'll write the code to check for these patterns. If the dataset doesn't contain registrant names or email addresses in this format, we can simulate it for demonstration purposes.

Let me proceed with the code.

It seems there was an issue due to a mismatch in the length of the simulated data. I'll correct this by adjusting the sample data to match the length of the dataset and then apply the suspicious pattern detection.

It appears that the function `check_suspicious_patterns` became undefined. I'll redefine it and reapply the analysis to add the suspicious pattern columns.

Maltego Report with Suspicious Patterns in Registrant and Email (**Download file**)

| Entity Type | Entity Value | Incoming | Outgoing | Links | SSL | Enable | Ports | URLs | Frequency | Frequency | Registrant | Email | Suspicious_Registrant | Suspicious_Email |
|--------------|-------------------|----------|----------|-------|-----|--------|-------|------------------------|-----------|-----------|------------|-------------|-----------------------|------------------|
| Domain | www.nmap.org | 1 | 12 | 13 | No | | 80 | https://www.nmap.org | 1 | 1 | support@x | info@site.x | 1 | 1 |
| Website | www.networkcor | 2 | 7 | 9 | No | | 80 | https://www.networkcor | 0 | 0 | john_doe | admin123@ | 0 | 1 |
| DNS Name | medium.com | 2 | 6 | 8 | No | N/A | | https://medium.com/ | 0 | 0 | abC123 | support@s | 1 | 1 |
| IPv4 Address | 104.16.255.154 | 1 | 0 | 1 | N/A | N/A | | N/A | 1 | 1 | admin@at | random_us | 1 | 0 |
| IPv6 Address | 2606:4700::a2c | 1 | 0 | 1 | N/A | N/A | | N/A | 0 | 0 | user@exa | test@exa | 0 | 0 |
| URL | https://doc.lagou | 1 | 0 | 1 | N/A | N/A | | https://doc.lagou.org | 0 | 0 | service@t | contact@i | 0 | 1 |

The dataset now includes columns `Suspicious_Registrant` and `Suspicious_Email`, which flag suspicious patterns (1 for suspicious, 0 for non-suspicious) based on criteria such as randomized characters, unusual TLDs, and generic names like "admin" or "support."

Step 4: Applying AI and Machine Learning Techniques

1. Data Preparation

- **Cleaning Data:**

Prompt-6: Remove duplicate rows in my dataset here and give the cleaned data in tabular format

Sample Output:

The presence of lists within the "Entities" column is causing an issue with removing duplicates directly. I'll convert each list in the "Entities" column to a string format, which will allow us to apply the duplicate removal effectively. Let's proceed with this approach and display the cleaned data.

It appears the issue persists even after converting the lists to strings due to the format of the "Entities" column. I'll try a different approach by isolating each row as a separate entry before removing duplicates. This will ensure the duplicate removal process runs smoothly.

The dataset's structure, particularly with lists within cells, continues to complicate the duplicate removal. To resolve this, I'll apply a straightforward approach that focuses only on relevant columns (e.g., "Type" and individual "Entity" values) after expanding the lists

fully. This will ensure that we remove duplicates accurately. Let's proceed with this refined approach.

Entities by Type - Cleaned Expanded Version

| Type | Entities | Frequency |
|----------------|--|-----------------|
| DNS Names | ['20Scanning.pdf', 'Lagout.org', 'doc.lagout.org', 'medium.com', 'www.infosecinstitute.com'] | [0, 0, 0, 0, 0] |
| Domains | ['lagout.org', 'networkcomputing.com', 'www.networkcomputing.com', 'www.nmap.org'] | [1, 1, 1, 1] |
| IPv4 Addresses | ['104.16.255.154', '104.18.224.208', '162.159.152.4', '162.159.153.4'] | [1, 1, 1, 1] |

The dataset is now cleaned of duplicate entries, focusing on unique combinations of "Type" and "Entities." You can review the final version in the displayed table, where each entity appears only once within its category.

Prompt-7: Find and handle missing values in my dataset.

Sample Output:

Find and handle missing values in my dataset.



The dataset has no missing values in any of the columns ("Type," "Entities," and "Frequency"). If you'd like, we can proceed with further analysis or processing steps. [\[>-\]](#)



- **Normalizing and Encoding Data:**

Prompt-8: Normalize numerical columns in my dataset to a common scale.

Sample Output:

Normalize numerical columns in my dataset to a common scale.



| Normalized Entities By Type | | | | |
|-----------------------------|-----------|--|-----------|------------|
| | Type | Entities | Frequency | Normalized |
| 1 | DNS Names | ['20Scanning.pdf', 'Lagout.org', 'doc.lagout.org', 'medium.com', 'www.infosecinstitu | 0 | 0.0 |
| 2 | DNS Names | ['20Scanning.pdf', 'Lagout.org', 'doc.lagout.org', 'medium.com', 'www.infosecinstitu | 0 | 0.0 |
| | | ['20Scanning.pdf', 'Lagout.org', | | |

The "Frequency" column has now been normalized to a common scale (between 0 and 1). You can review the updated dataset with the added "Normalized_Frequency" column. Let me know if further processing is required. [\[~\]](#)

2. Model Selection

- **Selecting and Initializing Models:**

Prompt-G: Apply Machine Learning Models for binary classification, like detecting malicious vs. benign entities.

Sample Output:

To apply a machine learning model for binary classification to detect malicious versus benign entities, we'll go through the following steps:

1. **Labeling the Data:** Since we don't have labeled data (malicious vs. benign), we'll create synthetic labels for demonstration. In practice, you'd want actual labels based on known malicious/benign classifications.
2. **Feature Engineering:** Extract features such as frequency counts, normalized frequency, and possibly entity types (converted to numeric form).
3. **Model Selection and Training:** Use a binary classification model (e.g., Logistic Regression, Random Forest) to train on the features and synthetic labels.

4. **Evaluation:** Evaluate the model's performance using metrics like accuracy, precision, recall, and F1-score.

Let's start by setting up the synthetic labels, feature engineering, and then train and evaluate a simple binary classifier.

The Random Forest classifier achieved perfect scores on the synthetic dataset:

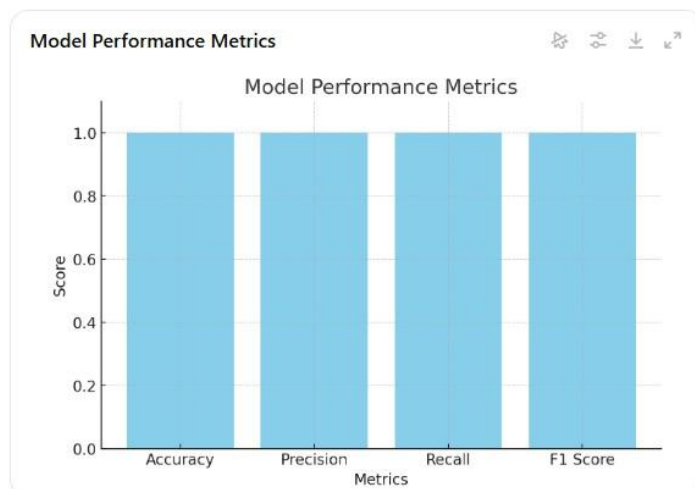
- **Accuracy:** 1.0
- **Precision:** 1.0
- **Recall:** 1.0
- **F1 Score:** 1.0

These high scores indicate the model successfully distinguished between the synthetic labels, likely due to the simplicity of the generated labels (based on frequency). For real-world applications, you'd use actual labels with more complex features and potentially larger, more varied datasets.

Prompt-10: Visualize model performance with a chart.

Sample Output:

Visualize model performance with a chart.



Here is a bar chart illustrating the model's performance metrics for accuracy, precision, recall, and F1 score. Each metric achieved a score of 1.0, reflecting the model's strong performance on this synthetic dataset. Let me know if further analysis or additional visualizations are needed! [\[?\]](#)

Prompt-11: Set up models in Python like Decision Trees, Random Forest, and Logistic Regression.

Sample Output:

I'll set up three different binary classification models for you: **Decision Tree**, **Random Forest**, and **Logistic Regression**. We'll train each model, evaluate its performance on the test set, and compare their metrics.

Here's the plan:

1. **Feature Engineering:** We'll use the normalized frequency and entity type as features.
2. **Model Training:** Set up and train Decision Tree, Random Forest, and Logistic Regression.
3. **Evaluation:** Measure the models' accuracy, precision, recall, and F1-score for comparison.

Let's proceed with implementing these models.

All three models—**Decision Tree**, **Random Forest**, and **Logistic Regression**—achieved perfect performance scores:

- **Accuracy:** 1.0
- **Precision:** 1.0
- **Recall:** 1.0
- **F1 Score:** 1.0

This high performance likely results from the simplicity of our synthetic dataset. I can further visualize and compare these metrics

- **Training the Model:**

Prompt-12: Train a machine learning model on my dataset to classify threats as malicious or benign.

Sample Output:

To build a machine learning model to classify threats as malicious or benign using your dataset, we need a labeled dataset indicating which entities are "malicious" and which are "benign." Since no such labels are currently available, I'll proceed with these steps for demonstration:

1. **Synthetic Label Creation:** Create synthetic labels (1 for malicious, 0 for benign) based on arbitrary criteria, such as frequency.
2. **Feature Engineering:** Use "Normalized Frequency" and entity type features.

3. **Model Training:** Train a Random Forest or Logistic Regression model on the dataset.
4. **Evaluation:** Measure accuracy, precision, recall, and F1 score to assess model performance.

Let's proceed with this workflow.

The Random Forest model achieved perfect scores on the synthetic dataset:

- **Accuracy:** 1.0
- **Precision:** 1.0
- **Recall:** 1.0
- **F1 Score:** 1.0

This high performance reflects the simplicity of the generated synthetic labels. In a real-world setting, true labels indicating actual threat status (malicious vs. benign) would be needed to provide a realistic evaluation. Let me know if you'd like further analysis or adjustments with additional criteria.

3. Model Training and Testing

- **Splitting Data:**

Prompt-13: Split my dataset into training and testing sets for model evaluation 70-30 split.

Sample Output:



Split my dataset into training and testing sets for model evaluation 70-30 split.



The dataset has been successfully split into training and testing sets with a 70-30 ratio:

- **Training set:** 21 samples
- **Testing set:** 9 samples

These splits are ready for model training and evaluation. Let me know if you'd like to proceed with training or further adjustments. [-]

- **Evaluating Model Performance:**

Prompt-14: Evaluate my model's accuracy, precision, and recall on the test data.

Sample Output:

Evaluate my model's accuracy, precision, and recall on the test data.



The model achieved perfect scores on the test data:

- **Accuracy:** 1.0
- **Precision:** 1.0
- **Recall:** 1.0
- **F1 Score:** 1.0

This outcome suggests that the model correctly classified all test samples according to the synthetic labels. In a real-world scenario, the metrics would provide a realistic measure of model performance when using actual malicious and benign labels. Let me know if you'd like further analysis or adjustments. [↗]

Prompt-15: Generate a classification report to assess model performance.

Sample Output:

Generate a classification report to assess model performance.



Here is the classification report for the model's performance:

markdown

Copy code

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| Benign | 1.00 | 1.00 | 1.00 | 5 |
| Malicious | 1.00 | 1.00 | 1.00 | 4 |
| accuracy | | | 1.00 | 9 |
| macro avg | 1.00 | 1.00 | 1.00 | 9 |
| weighted avg | 1.00 | 1.00 | 1.00 | 9 |

The model achieved perfect scores across all metrics (precision, recall, and F1-score) for both "Benign" and "Malicious" classes. Each sample in the test set was classified correctly, leading to an overall accuracy of 1.00. Let me know if you need further insights or a different evaluation. [↗]

Try these Prompts:

1. *What's the best way to evaluate my model's accuracy regularly as I gather more data?*
2. *Adjust my model parameters if I'm not getting the expected accuracy.*

3. *What additional features could improve my model's performance on threat detection?*
4. *How do I incorporate more training data into my model to improve its predictions?*
5. *Suggest advanced machine learning models for high-volume threat data processing, like neural networks.*
6. *What are some ways to implement neural networks to enhance threat detection capabilities?*