Network Log Anonymization Tool

Overview

This repository contains a comprehensive network log anonymization toolkit designed to process and anonymize various types of network log files including Suricata, Firewall, and Zeek logs. The tool provides multiple anonymization techniques to protect sensitive information while preserving the utility of log data for analysis.

Project Structure

```
Network-Log-anonymisation/
├── config.yaml               # Configuration file for anonymization
settings
├── main.py                   # Main entry point of the application
├── README.md                 # Basic project information
└── anonymizer/               # Core anonymization modules
    ├── __init__.py           # Package initialization
    ├── differential.py       # Differential privacy implementation
    ├── hashing.py            # Hashing utilities
    ├── ip_anonymizer.py      # IP address anonymization
    ├── ipmask.py             # IP address masking techniques
    ├── l-diversity.py        # L-diversity algorithm implementation
    ├── log_parser.py         # Log parsing utilities
    ├── log_reconstructor.py  # Log reconstruction after anonymization
    ├── lowentro.py           # Low entropy anonymization
    ├── masking.py            # General masking techniques
    ├── nonip_diff_priv.py    # Differential privacy for non-IP data
    ├── paper_imple.py        # Implementation of research paper algorithms
    ├── port_anonymizer.py    # Port anonymization
    ├── suricata_parser.py    # Suricata log specific parser
    ├── t_closeness.py        # T-closeness algorithm implementation
    ├── timestamp_anonymizer.py # Timestamp anonymization
    └── urlgeneral.py         # URL generalization techniques
```

Detailed Component Description

Main Components

main.py

This is the entry point of the application. It:

- Parses command line arguments to get the configuration file path
- Loads the YAML configuration
- Orchestrates the log anonymization pipeline:
    1. Parsing logs into a structured format

2. Applying anonymization methods as specified in the config
3. Reconstructing anonymized logs in the original format

config.yaml

This file defines the anonymization settings:

- `log_file`: Path to the input log file
- `log_type`: Type of log (suricata, firewall, zeek)
- `output_log`: Path to save anonymized logs
- `anonymization`: Settings for different field types:
  - `ip`: Methods like "salt", "truncate", "mask"
  - `port`: Methods like "salt", "shuffle", "generalize"
  - `timestamp`: Methods like "round", "degrade"

Anonymizer Modules

log_parser.py

This module parses different types of network logs (Suricata, Firewall, Zeek) into a structured pandas DataFrame format. It:

- Uses regex patterns to extract fields from text logs
- Creates a mapping of original values and their positions in the logs
- Saves structured logs and mappings to temporary CSV files

ip_anonymizer.py

Provides IP address anonymization while preserving subnet structure:

- Uses a salted hash function to create one-to-one mappings for IP octets
- Ensures consistent anonymization across multiple occurrences
- Preserves network structure by anonymizing octets individually

port_anonymizer.py

Anonymizes port numbers:

- Uses salted hashing to create stable one-to-one mappings
- Maps original ports to the non-reserved range (1024-65535)
- Ensures consistency across multiple occurrences of the same port

timestamp_anonymizer.py

Handles timestamp anonymization:

- Implements time rounding to nearest 15-minute intervals
- Reduces the temporal precision while maintaining time sequence

log_reconstructor.py

Reconstructs the anonymized logs in the original format:

- Loads the mapping between original and anonymized values
- Replaces original values in the log files with their anonymized counterparts
- Preserves the original log structure

differential.py

Implements differential privacy techniques:

- Adds calibrated Laplace noise to numerical data
- Provides privacy guarantees through epsilon parameter tuning

ipmask.py

Implements IP address generalization:

- Masks IP addresses according to a specified subnet mask (e.g., /24)
- Creates generalized IP groups to reduce uniqueness

suricata_parser.py

A specialized parser for Suricata log format:

- Extracts fields like timestamp, alerts, classifications, IP addresses, ports
- Creates mapping for field replacement

Advanced Privacy Modules

l-diversity.py

Implements the l-diversity privacy model:

- Ensures sensitive attributes have diverse values within equivalence classes
- Prevents attribute disclosure attacks

t_closeness.py

Implements the t-closeness privacy model:

- Controls the distribution of sensitive attributes within equivalence classes
- Provides stronger privacy guarantees than k-anonymity

lowentro.py

Implements low-entropy anonymization techniques:

- Reduces information content while preserving utility
- Useful for fields with low-entropy requirements

nonip_diff_priv.py

Applies differential privacy techniques to non-IP data:

- Specialized noise addition for categorical and non-network data
- Preserves aggregate statistics while protecting individual records

urlgeneral.py

Provides URL generalization techniques:

- Anonymizes domain parts while preserving structure
- Options for domain generalization and path masking

paper_imple.py

Contains implementations of research paper algorithms:

- Academic privacy-preserving techniques adapted to network log contexts
- Research-based approaches to anonymization

Usage Workflow

1. **Configure Anonymization Settings**:
    - Edit `config.yaml` to specify input log file, type, and desired anonymization methods
2. **Run the Tool**:

```
python main.py --config config.yaml
```

3. **Process Flow**:
    - Logs are parsed into structured format
    - Specified anonymization techniques are applied to respective fields
    - Anonymized values are substituted back into original log format
    - Final anonymized logs are saved to the output path

Anonymization Techniques

IP Address Anonymization

- **Salted Hashing**: Consistent one-way mapping of IP octets
- **Subnet Masking**: Generalizing to subnet level (/24, /16, etc.)
- **Prefix-Preserving**: Maintains network hierarchy while anonymizing

Port Anonymization

- **Salted Hashing**: Maps ports to non-reserved range
- **Generalization**: Groups ports into ranges by service type
- **Randomization**: Randomly maps ports while maintaining consistency

Timestamp Anonymization

- **Rounding**: Reduces precision by rounding to nearest interval
- **Degradation**: Systematically reduces timestamp resolution

Advanced Privacy Models

- **Differential Privacy**: Adds calibrated noise to preserve aggregate statistics
- **L-Diversity**: Ensures diversity of sensitive values within groups
- **T-Closeness**: Controls distribution of sensitive values

Mind Maps

Anonymization Process Flow

```
[Input Logs] → [Log Parser] → [Structured Data] → [Apply Anonymization] →
[Anonymized Data] → [Log Reconstruction] → [Output Logs]
```

Anonymization Techniques Hierarchy

```
Anonymization
├── Network Fields
│   ├── IP Address
│   │   ├── Cryptographic (Hashing)
│   │   ├── Generalization (Subnet Masking)
│   │   └── Perturbation (Adding Noise)
│   └── Ports
│       ├── Cryptographic (Hashing)
│       ├── Generalization (Service Groups)
│       └── Randomization
├── Temporal Fields
│   └── Timestamps
│       ├── Rounding
│       └── Precision Reduction
└── Privacy Models
    ├── Differential Privacy
    ├── K-Anonymity
    ├── L-Diversity
    └── T-Closeness
```

Privacy-Utility Tradeoff

The toolkit allows users to balance privacy protection against data utility:

- Stronger anonymization techniques provide better privacy but reduce utility
- Configuration options allow customization based on specific requirements
- Multiple techniques can be combined for layered protection

Conclusion

This Network Log Anonymization Toolkit provides a comprehensive set of tools for anonymizing network logs while preserving their utility for analysis and research. By offering various anonymization techniques and a flexible configuration system, it enables users to balance privacy requirements with analytical needs.