It's a Beautiful Day in the Malware Neighborhood

Matt Maisel

Senior Data Scientist @ Cylance

ACoD, January 2018

Agenda

- Introduction
  - Motivation
  - Problem Statement
- Background
  - Theory and Literature Review
  - Related Malware Similarity Systems
- System
  - Design
  - Implementation
- Discussion
  - Experiments

Motivation

- Search and retrieval of similar malware samples provides context during analysis
  1. Relate context of previously analyzed samples with unknowns
  2. Prioritize outliers for manual analysis and reverse engineering
  3. Process samples in alerts/detections and route to other tools and systems

Problem Statement

- Malware similarity is performed through comparison of raw bytes, static, and dynamic features that distill the semantic characteristics of the samples
- Open source tools mostly rely on indexing signatures, cryptographic hashes, or fuzzy hashes

Please won't you be my neighbor?

- **Nearest Neighbor (NN) Search**: Given a set of n samples $X = x_1, x_2, \ldots, x_{n-1}, x_n \in \mathbf{R}$, return the k nearest neighbors for query sample x_q according to a distance function $d(x_q, x_n)$.
- Approximate variant allows some error threshold ϵ that satisfies: $d(x_q, x) \leq (1 + \epsilon) d(x_q, x_n)$

Problem Statement

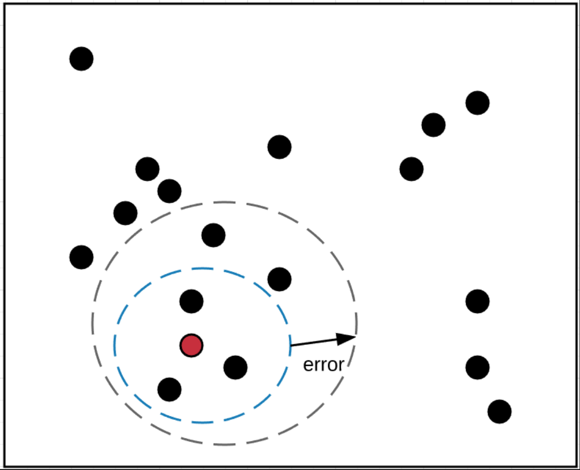


Figure 1: NN query (k=3)

CYLANCE

Toy Feature Representation Example

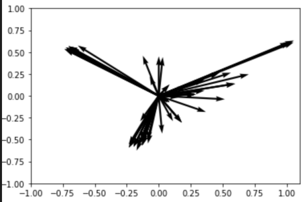
- Feature engineering is key to ensuring a representation captures the true characteristics of malware
- Represents samples in a *n*-dimensional feature space consisting of numerical values
  - x_1 is $\vec{x_1} = \begin{bmatrix} 100 & 3 & 10 & 2 \end{bmatrix}$
  - x_2 is $\vec{x_2} = \begin{bmatrix} 10234 & 4 & 4 & 2 \end{bmatrix}$
  - x_3 is $\vec{x_3} = \begin{bmatrix} 3453 & 6 & 2 & 1 \end{bmatrix}$

Sample	File Size (kB)	# of exported symbols	# of imported symbols	# of imported DLL
x_1	100	3	10	2
x_2	10234	4	4	2
x_3	3453	6	2	1

Table 1: Toy Sample Features

Relating Distance and Similarity

- Cosine similarity: $C(x, y) = \frac{\sum x_i y_i}{\sqrt{\sum x_i^2}\sqrt{\sum y_i^2}}$
- $C(x_1, x_2) = 0.9944$
- $C(x_1, x_3) = 0.9945$



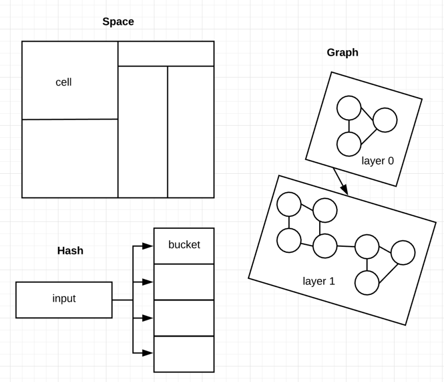
Sample	File Size (kB)	# of export symbols	# of import symbols	# of import DLL
x_1	100	3	10	2
x_2	10234	4	4	2
x_3	3453	6	2	1

Table 2: Toy Sample Features

Theory and Literature Review

Methods

- Space partitioning
- Hashing
- Graph



Space partitioning NN method

- **k-d tree** (Bentley (1975)) offline partitioning of dataset into tree structure on dimensions until each leaf consists of a single sample
- **ball tree** (Omohundro (1989)) divide-and-conquer partitioning procedure that splits root node based on distance from centroid
- **cluster pruning tree** randomly selects $\sqrt{n}$ leaders and partitions followers to nearest neighbor in h levels.

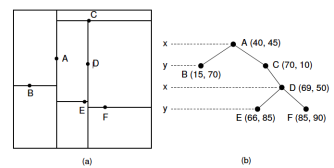


Figure 2: 2-dimensional kd-tree construction[1]

[1] http://homes.sice.indiana.edu/yye/lab/teaching/spring2014-C343/moretrees.php

Hashing NN methods

- Hash function maps an arbitrary input into a finite output
- Non-cryptographic, data-independent, and data-dependent hashing algorithms
- Applied in lookup tables or fast distance approximation
- *Small changes to the input space result in small changes to the output*

Hashing NN methods

Locality Sensitive Hashing (LSH)

Given $H(\cdot)$ containing K hashing functions, LSH maps x_i to a K-bit hash code $\in 0, 1$:

$$H(x_i) = [h_1(x_i), h_2(x_i), ..., h_k(x_i)]$$

Probability that samples x and y are hashed to the same bucket preserves similarity

- Generalized over different distance metrics including Euclidean, Cosine, Jaccard, and others
- MinHash is used to estimate the similarity of sets and is equal to Jaccard similarity

Hashing NN Literature Review

- Kornblum (2006) developed **ssdeep**[2], a **context triggered piecewise hashing (CTPH)** algorithm based on work by Tridgell (2002), edit distance for direct comparison
  - Winter et al. (2013) and Wallace (2015) improve comparison performance by eliminating candidates and indexing partial hashes
- Roussev (2010) introduced **sdhash**[3] which uses statistically improbable features to construct a similarity digest in a sequence of Bloom filters

[2] https://ssdeep-project.github.io/ssdeep/index.html

[3] https://github.com/sdhash/sdhash

Hashing NN Literature Review

- Oliver et al. (2013) introduce **TrendMicro LSH** (TLSH), hamming distance is used to compare digests, slow direct comparison but can improve with index
- LSH Forest by Bawa et al. (2005) designs a self-tuning index with LSH that eliminates data-dependent parameters and improves performance over skewed data

Graph NN methods

- *"neighbor of a neighbor is also likely to be a neighbor"*
- General algorithm approach:
  - Graph is constructed during an offline phase
  - Queries are performed online and identify candidate neighbors via traversal mechanism
  - Iteratively search neighboring nodes until a stopping criteria is reached
- *Early methods converge to local solutions and have expensive graph constructive phases*

Graph NN Literature Review

- Fu et al. (2017) present **Hierarchical Navigable Small World (HNSW)**, an ANN method with a multi-layer graph that greedily identifies candidate samples for comparison
  - Construction inserts elements in random layers and adds edges to neighbors using a heuristic
  - Query is performed by traversing down layers

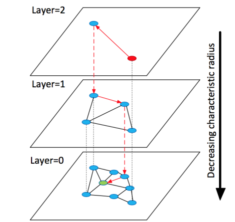


Figure 3: Sketch of query from top to bottom layers

Summary of NN Methods

- Space partitioning
  - k-d tree
  - ball tree
  - **Cluster pruning tree***
- Hashing
  - **ssdeep***
  - sdhash
  - **LSH forest***
  - TLSH
- Graph
  - **HNSW***

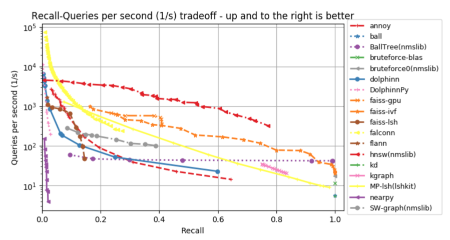


Figure 4: NYTimes @ $k = 100$ (ANN Benchmarks)

Related Malware Similarity Systems

- VirusTotal (2018) offers similarity search based on a structural feature hash
- Wallace (2015) provides an implementation of indexed ssdeep[4] and Abrahamy (2017) extends to use Elasticsearch[5]
- BitShred by Jang et al. (2011) perform pairwise Jaccard similarity in hadoop
- Upchurch and Zhou (2016) use MinHash in the Malware Provenance system which uses a sliding window hash on n-gram features from blocks of disassembled sample

[4] https://github.com/bwall/ssdc

[5] https://github.com/intezer/ssdeep-elastic

Related Malware Similarity Systems

- SARVAM [6] visualizes raw bytes as gray-scale images and compares the distance of computer vision features

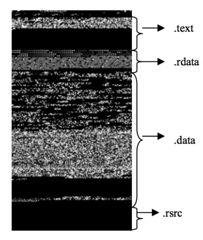


Figure 5: SARVAM image

[6] http://sarvam.ece.ucsb.edu

CYLANCE

Related Malware Similarity Systems

- Rieck et al (2011) released Malheur[7] which uses a sequence representation of behavior extracted from sandbox reports to identify prototypes

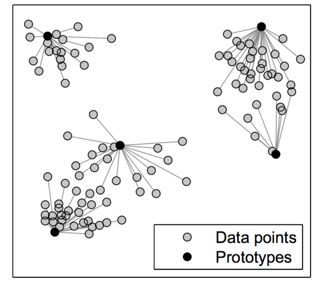


Figure 6: Prototype Selection

[7] http://www.mlsec.org/malheur/

System Design

1. **Extract** and store sample metadata and raw feature data
2. **Transform** data into feature vectors with per filetype pipeline and dimensionality reduction
3. **Fit** indexes for NN methods on feature matrices or raw file bytes
4. **Query** index with an input sample and return k-nearest neighbors along with relevant contextual features

rogers v0.0.1

- Command line application developed in Python 3
- Multiprocessing pool for feature extraction
- Protocol Buffer messages for raw feature data
- Sqlite for local database storage
- Neighborhood graph visualization in Jupyter Notebook[8] using Plotly[9]

```
(env) mmaisel:rogers/ (developx) $ rogers -h
usage: rogers [-h] [-v] [--print] [--conf CONF] {feature,db,index} ...

Malware similarity tool

positional arguments:
  {feature,db,index}  commands

optional arguments:
  -h, --help          show this help message and exit
  -v                  Set debug logging
  --print             Print results instead of table output
  --conf CONF         Path to rogers configuration file
```

[8] https://jupyter.org/

[9] https://plot.ly/

System Implementation

NN Indexes

1. Indexed ssdeep
2. Cluster Pruning tree[10]
3. LSH Forest[11]
4. HNSW[12]

Feature Engineering and Vectorization

- Static feature extraction for PE using header and Yara Rules Project[13]
- Offline vectorization pipeline based on Latent Semantic Indexing (LSI)

---

[10] https://github.com/facebookresearch/pysparnn

[11] http://scikit-learn.org/0.18/modules/generated/sklearn.neighbors.LSHForest.html#sklearn.neighbors.LSHForest

[12] https://github.com/searchivarius/nmslib

[13] https://github.com/Yara-Rules/rules

Experiments

Name	Samples (n)	Classes (k)
variant2015	83	8
vtcluster-jan2018	27000	15

Table 3: Datasets

$$Precision@k = \frac{\text{relevant } \cap \text{ retrieved}}{k}$$

Name	Index size (MB)	P@5	P@10	P@50	P@100
Indexed ssdeep	79	0.1093	0.0546	0.0109	0.0054
Pruning tree	409	0.8209	0.8209	0.8209	0.7898
LSH Forest	104	**0.9953**	**0.9976**	**0.9639**	**0.9390**
HNSW	**51**	0.7511	0.7488	0.7604	0.7619
Brute Force	96	0.9976	0.9976	0.9644	0.9391

Table 4: NN Method Comparison

Remarks and Future Work

- **Feature Engineering**
  - Add support for more file type vectorizers beyond PE
  - Extract multiple modalities, e.g. dynamic[14]
  - Feature selection and learning representations[15]
- **Experiments**
  - Parameter Optimization
  - Additional Benchmarks
- **Use Cases**
  - Indexing of benign samples?
  - Partial Fit

[14] http://www.hexacorn.com/blog/2017/12/31/happy-new-year-2018-get-yourself-logs-from-250k-sandboxed-samples/

[15] https://www.blackhat.com/docs/us-16/materials/us-16-Berlin-An-AI-Approach-To-Malware-Similarity-Analysis-Mapping-The-Malware-Genome-With-A-Deep-Neural-Network.pdf

Questions?

- https://github.com/cylance/rogers
- Pull request are welcome!
- mmaisel@cylance[.]com

Appendix A - Feature Engineering

Modality	Variable Type	Examples
Raw Bytes	Continuous	entropy of byte ngrams, similarity hash digest (e.g. ssdeep, tlsh)
Static	Continuous	file size, PE image size, code size, # of sections, compile timestamp
Static	Categorical	import symbols, import dlls, exported symbols, opcodes
Dynamic	Categorical	system API calls, spawned processes, network activity
Dynamic	Continuous	# of registry operations, # of file system operations, # of network operations
Contextual	Categorical	AV and Yara detection names, observed hostnames, file path names , user account

Table 5: Examples of Feature by Modality and Type

Appendix B - Protocol Buffers

```
message Feature {
    // type of variable
    message Variable {
        enum Type {
            CATEGORICAL = 0; // values in a specific category
            CONTINUOUS  = 1; // infinite number of values
            DISCRETE    = 2; // limited to certain number of values
            ORDINAL     = 3; // ordered variable, expects int value,
        }
    }
    // type of feature space
    message Modality {
        enum Type {
            BYTES       = 0; // raw byte features
            STATIC      = 1; // structural / static features
            DYNAMIC     = 2; // behavioral / dynamic features
            CONTEXTUAL  = 3; // contextual features
        }
    }

    Variable.Type type = 1;
    Modality.Type mode = 2;
    Value value        = 3;
}
```

Figure 7: Protocol buffer message definition for Feature