# Scalable, Behavior-Based Malware Clustering

U. Bayer, P. Comparetti,
C. Hlauschek, E. Kirda, C. Krügel
NDSS 2009

Presented by
Sal LaMarca

Slides adapted from http://www.iseclab.org/people/pmilani/ndss09-clustering-slides.pdf

## Motivation

- Everyday thousands of new malware sample appear and need to be analyzed
- Systems that automatically analyze samples generate thousands of behavior reports on these new malware, but people are the ones who have to read and use these malware reports to combat malware
- Thus, there is a need to group or cluster malware based on behavior reports that are similar

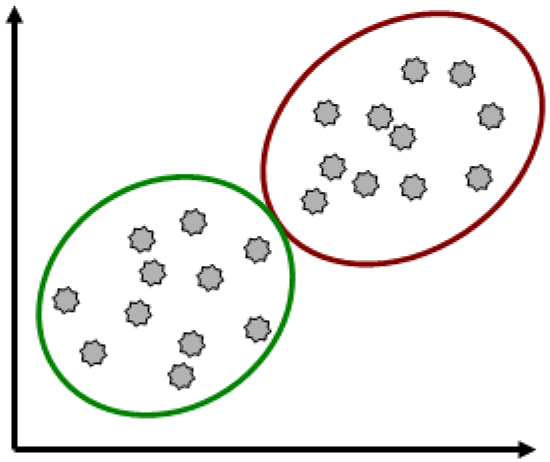
## Why cluster?

- Save time and effort by ignoring reports of malware that have similar behavior
- Prioritize which malware families need to be worked on first
  - We would like to concentrate on creating signatures for malware that targets a wide spread vulnerability and does lots of damage
- Generalize signature generation and patches that work on a family of malware instead of just a few instances of malware

# Clustering vs Classification

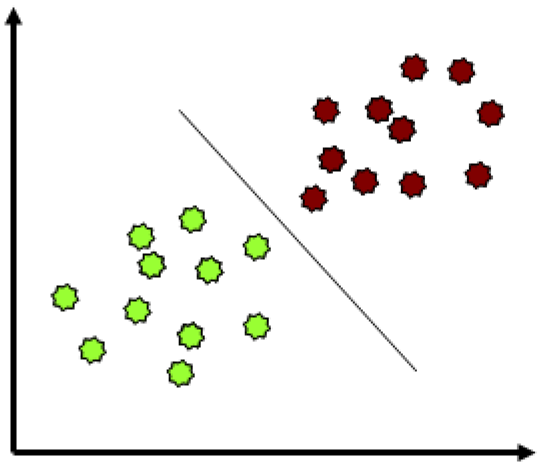
## CLUSTERING

- Data is not labeled
- Group points that are "close" to each other
- Identify structure or patterns in data
- Unsupervised learning



## CLASSIFICATION

- Labeled data points
- Want a "rule" that assigns labels to new points
- Supervised learning



Slide adapted from www.broadinstitute.org/annotation/winter_course_2006/index_files/Classification_and_Clustering.ppt

## Scalable, Behavior Based Malware Clustering

- Scalable
  - The system can process large and larger sets of malware samples  in a reasonable amount of time
- Behavior Based
  - Malware samples are represented by their actions when they execute, and  their actions are described with a behavior profile
- Malware Clustering
  - Find subsets (groups or families) of malware that are similar

## Overview

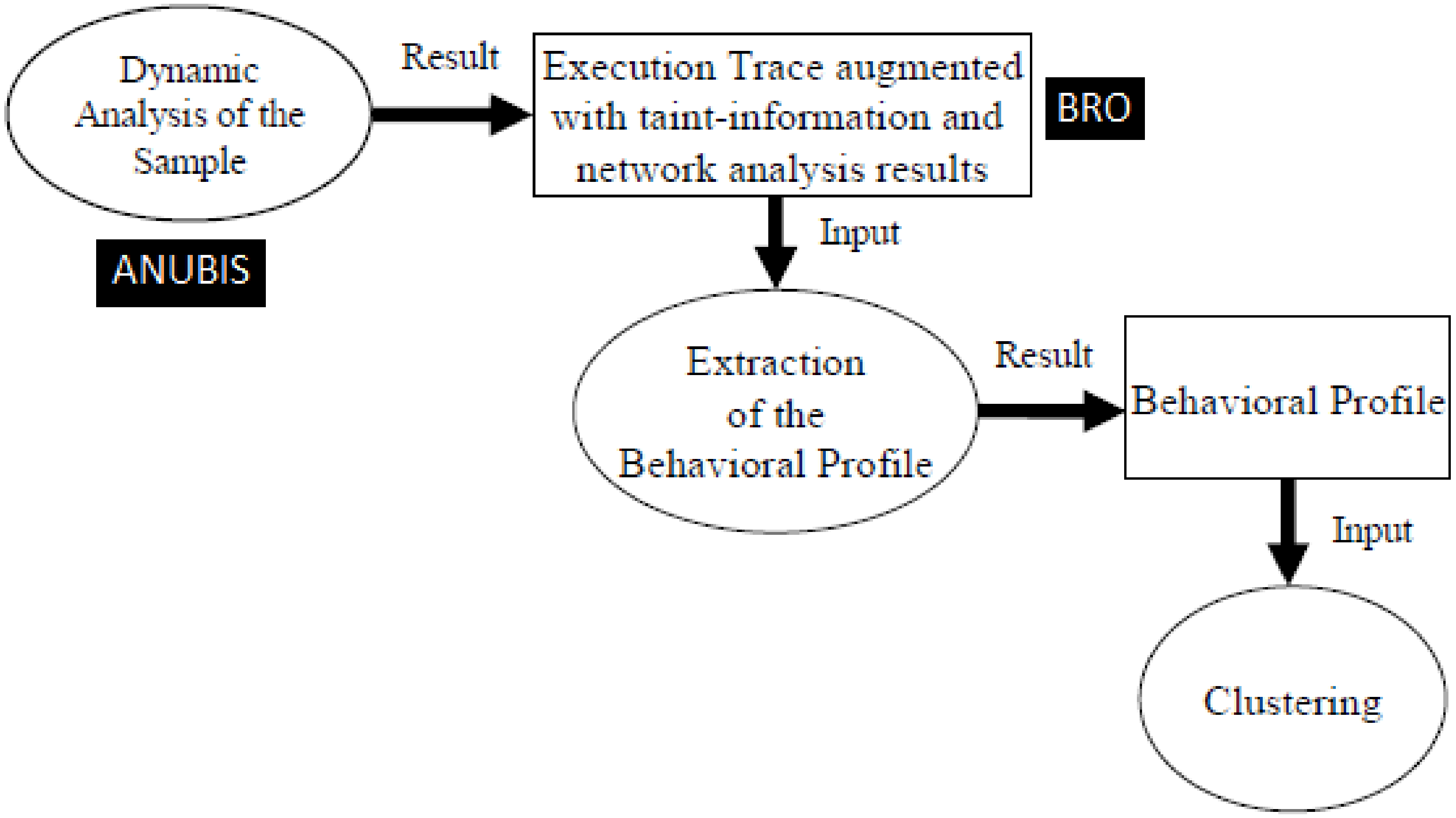

Dynamic Analysis of the Sample
ANUBIS
Result
Execution Trace augmented with taint-information and network analysis results
BRO
Input
Extraction of the Behavioral Profile
Result
Behavioral Profile
Input
Clustering

Image adapted from http://www.iseclab.org/people/pmilani/ndss09-clustering-slides.pdf

## Dynamic Analysis

- Based on the authors' previous work ANUBIS
  - ANUBIS is an OS emulator that generates execution traces for system calls
- Additions to ANUBIS
  - Tainting for locating system call dependencies
    - Their dynamic data tainting is based on previous work from others
  - Control flow dependencies
  - Network analysis from low level socket system calls
    - Uses an existing tool called Bro to identify and parse HTTP, IRC, SMTP, and FTP protocols
- Output
  - Behavior execution trace that contains system call dependencies, control flow dependencies, and network analysis results

## Feature Extraction

- Execution traces creates lots of data, but how much of this data is actual information?
  - We can tackle this problem with feature extraction
- Feature extraction is a type of dimensionality reduction that involves transforming your dataset into relevant feature sets that capture important information.

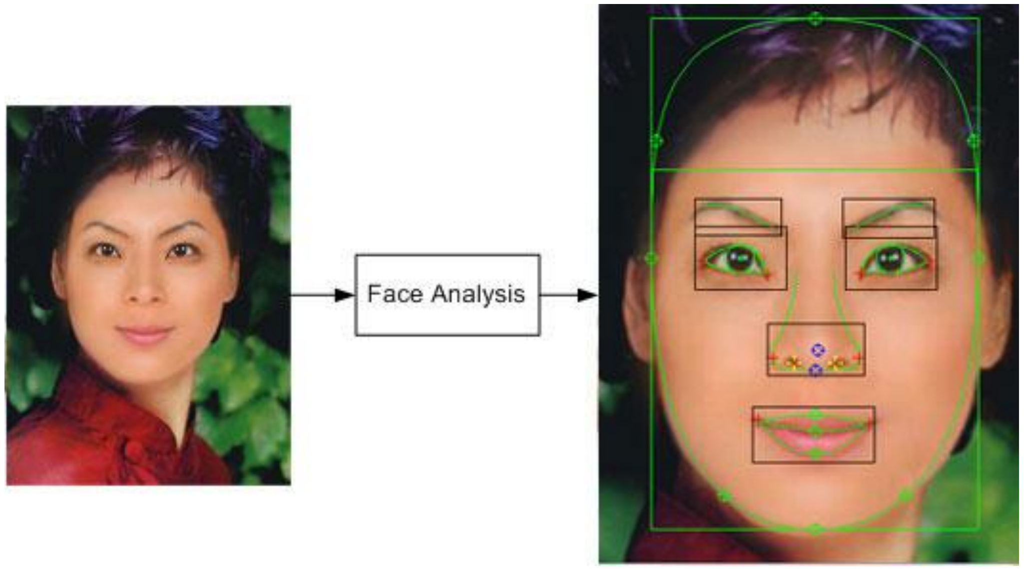


Image from http://www.seestorm.com/technologies/cv/ffe_sdk/

## Behavior profile feature extraction

- Process the behavior execution trace from ANUBIS
- Goal:  generalize the system call traces
  - System calls can vary significantly even between programs that behavior in the same way
  - Remove execution specific artifacts from the trace
- Extract a relevant set of features, the behavior profile, from the execution trace that captures key pieces of information to model the behavior of malware

## The usefulness of general features

- Example:  Different ways to read from a file

```
f = fopen("C:\\test");
read(f, 1);
read(f, 1);
read(f, 1);
```

```
f = fopen("C:\\test");
read(f, 3);
```

- Different system calls with similar semantics
  - i.e. NtCreateProcess and NtCreateProcessEx
- By generalizing, they can easily interleave the trace with unrelated calls

```
f = fopen("C:\\test");
read(f, 1);
readRegValue(..);
read(f, 1);
```

## Main elements of a behavior profile

- OS objects: a resource such as a file that can accessed by a system call
  - Has a name and type
- OS operations: generalizations of a system call
  - Operations on OS Objects
  - Order is not relevant
  - Number of operations on a certain OS Object is not relevant
- Object dependencies: models dependencies (relationships) between objects and their operations
  - Generalizes system call dependencies
  - i.e. copy a file from source to destination
  - Reflects the “true” order of operations
- Control flow dependencies: represents how tainted data is used by the malware

## Behavior profile example

```
src = NtOpenFile("C:\\sample.exe");
// memory map the target file
dst = NtCreateFile("C:\\Windows\\" + GetTempFilename());
dst_section = NtCreateSection(dst);
char *base = NtMapViewOfSection(dst_section);
while(len < length(src)) {
*(base+len)=NtReadFile(src, 1); len++; }
```

```
Op|File|C:\sample.exe
open:1, read:1
Op|File|RANDOM_1
create:1
Op|Section|RANDOM_1
open:1, map:1, mem_write: 1
Dep|File|C:\sample.exe -> Section|RANDOM_1
read - mem_write
```

## Behavior profile example

```
src = NtOpenFile("C:\\sample.exe");
// memory map the target file
dst = NtCreateFile("C:\\Windows\\" + GetTempFilename());
dst_section = NtCreateSection(dst);
char *base = NtMapViewOfSection(dst_section);
while(len < length(src)) {
*(base+len)=NtReadFile(src, 1); len++; }
```

```
Op|File|C:\sample.exe
open:1, read:1
Op|File|RANDOM_1
create:1
Op|Section|RANDOM_1
open:1, map:1, mem_write: 1
Dep|File|C:\sample.exe -> Section|RANDOM_1
read - mem_write
```

## Behavior profile example

```
src = NtOpenFile(“C:\\sample.exe”);
// memory map the target file
dst = NtCreateFile(“C:\\Windows\\” + GetTempFilename());
dst_section = NtCreateSection(dst);
char *base = NtMapViewOfSection(dst_section);
while(len < length(src)) {
*(base+len)=NtReadFile(src, 1); len++; }
```

```
Op|File|C:\sample.exe
open:1, read:1
Op|File|RANDOM_1
create:1
Op|Section|RANDOM_1
open:1, map:1, mem_write: 1
Dep|File|C:\sample.exe -> Section|RANDOM_1
read - mem_write
```

## Behavior profile example

```
src = NtOpenFile("C:\\sample.exe");
// memory map the target file
dst = NtCreateFile("C:\\Windows\\" + GetTempFilename());
dst_section = NtCreateSection(dst);
char *base = NtMapViewOfSection(dst_section);
while(len < length(src)) {
*(base+len)=NtReadFile(src, 1); len++; }
```

```
Op|File|C:\sample.exe
open:1, read:1
Op|File|RANDOM_1
create:1
Op|Section|RANDOM_1
open:1, map:1, mem_write: 1
Dep|File|C:\sample.exe -> Section|RANDOM_1
read - mem_write
```

## Behavior profile example

```
src = NtOpenFile(“C:\\sample.exe”);
// memory map the target file
dst = NtCreateFile(“C:\\Windows\\” + GetTempFilename());
dst_section = NtCreateSection(dst);
char *base = NtMapViewOfSection(dst_section);
while(len < length(src)) {
*(base+len)=NtReadFile(src, 1); len++; }
```

```
Op|File|C:\sample.exe
open:1, read:1
Op|File|RANDOM_1
create:1
Op|Section|RANDOM_1
open:1, map:1, mem_write: 1
Dep|File|C:\sample.exe -> Section|RANDOM_1
read - mem_write
```

## Behavior profile example

```
src = NtOpenFile("C:\\sample.exe");
// memory map the target file
dst = NtCreateFile("C:\\Windows\\" + GetTempFilename());
dst_section = NtCreateSection(dst);
char *base = NtMapViewOfSection(dst_section);
while(len < length(src)) {
*(base+len)=NtReadFile(src, 1); len++; }
```

```
Op|File|C:\sample.exe
open:1, read:1
Op|File|RANDOM_1
create:1
Op|Section|RANDOM_1
open:1, map:1, mem_write: 1
Dep|File|C:\sample.exe -> Section|RANDOM_1
read - mem_write
```

# Behavior profile example

```
src = NtOpenFile("C:\\sample.exe");
// memory map the target file
dst = NtCreateFile("C:\\Windows\\" + GetTempFilename());
dst_section = NtCreateSection(dst);
char *base = NtMapViewOfSection(dst_section);
while(len < length(src)) {
*(base+len)=NtReadFile(src, 1); len++; }
```

```
Op|File|C:\sample.exe
open:1, read:1
Op|File|RANDOM_1
create:1
Op|Section|RANDOM_1
open:1, map:1, mem_write: 1
Dep|File|C:\sample.exe -> Section|RANDOM_1
read - mem_write
```

## Scalable LSH clustering

- Many clustering algorithms compute all pair wise distances with takes $O(n^2)$ evaluations
- Even with modern processors and parallelism, $O(n^2)$ algorithms do not scale well for large datasets
- The authors use LSH (locality sensitive hashing), a technique introduced by Indyk and Motwani, to compute an approximate clustering that requires less than $O(n^2)$distance computations
- The idea behind LSH is to hash a set of points in such a way that near or similar points have a much higher collision probability than points that are distant
- They use the Jaccard Index for a measuring similarity:

$$J(a,b) = | a \cap b | / | a \cup b |$$

## Scalable LSH clustering (cont)

- To cluster, they do the following
  1. Transform the behavior profile into a feature set
  2. Use LSH on the feature set to map it to an approximation set of all near pairs of similar features
  3. Remove pairs  of samples that aren’t similar by using the Jaccard Index
  4. Sort the remaining pairs based on similarity to produce an approximate single-linkage, hierarchical clustering based on some threshold value t, where $J(a,b) < t$.

- For large datasets, the dominate computations involved in  LSH clustering  are for computing similarity computations which is $O(ncd)$
  - n is the number of samples, c is the max cluster size for a given threshold t, and d is the average number of features in a sample
  - In practice, $nc << n^2$, so LSCH clustering is less than $O(n^2)$, but in the worst case $c = n$ for identical sample sets, and it is $O(n^2)$

## Reference cluster

- Assessing the quality of the results of a clustering algorithm is inherently difficult
- Thus, the authors created a reference cluster to measure the results of their LSH clustering experiments
- The reference cluster was created by
  - A random sampling of 14,212 malware samples that were submitted to Anubis from Oct. 27th 2007 to Jan. 31st 2008
  - Each malware sample was scanned with 6 different AV programs
  - They selected 2,658 samples out of the 14,212 that were reported to be in the same malware family by the majority of the AV programs
  - For each sample, they examined the corresponding ANUBIS report and manually corrected classification problems
    - Why select samples reported to be in the same malware family by the majority of AV programs, and then manually correct their classification based on the ANUBIS report?
    - Their clustering system uses ANUBIS to create a behavior profile and their reference cluster for evaluating their experiment is also based on ANUBIS and their manual corrections. Could this reference cluster be biased?

# Evaluating LSH clustering

- Experimented using LSH clustering with t = 0.7 on the 2,658 samples and compared it to the reference cluster

- Results
  - Their system produced 87 clusters, while the reference cluster consisted of 84 clusters
  - Precision = 0.984
    - precision measures how well a clustering algorithm distinguishes between samples that are different
  - Recall = 0.930
    - recall measures how well a clustering algorithm recognizes similar samples

Precision and recall with respect to various values of threshold t

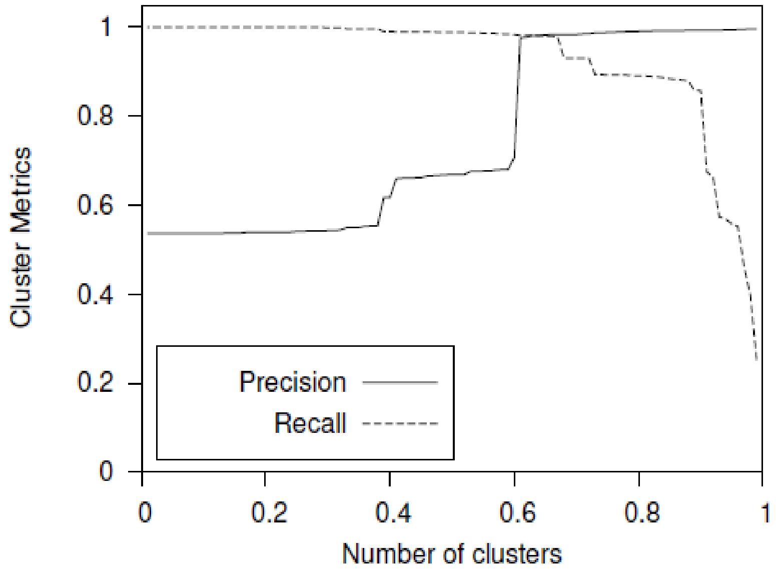
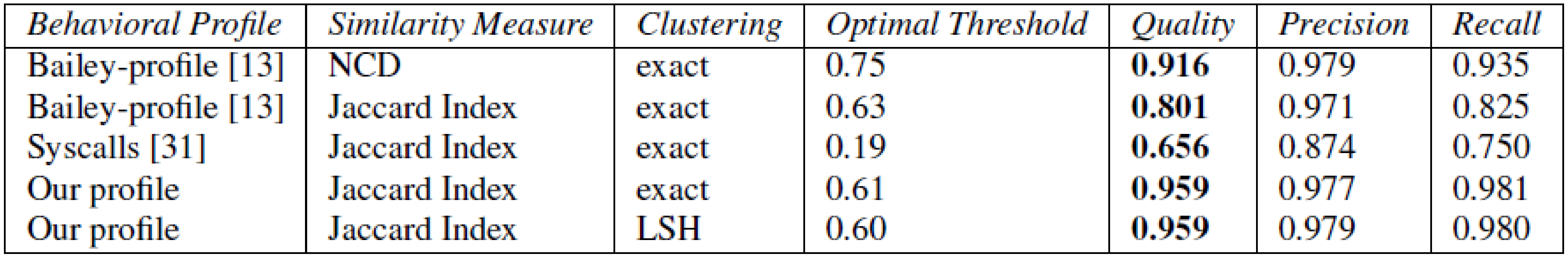


**Figure 3. Precision and recall.**

The paper has a typo in figure 3.
The horizontal axis should be threshold t.

# Comparative experiment and results

- They compared LSH clustering to Bailey et al. system for clustering and to clustering based on raw system calls
- Bailey et al. clustering system
  - Uses Normalized Compression Distance, and NCD can be thought of as a way to approximate similarity between two samples by representing an overlap between the two samples.
  - NCD is based on complexity theory that similar data when concatenated compresses better than data that is not similar.
- cluster quality = precision * recall

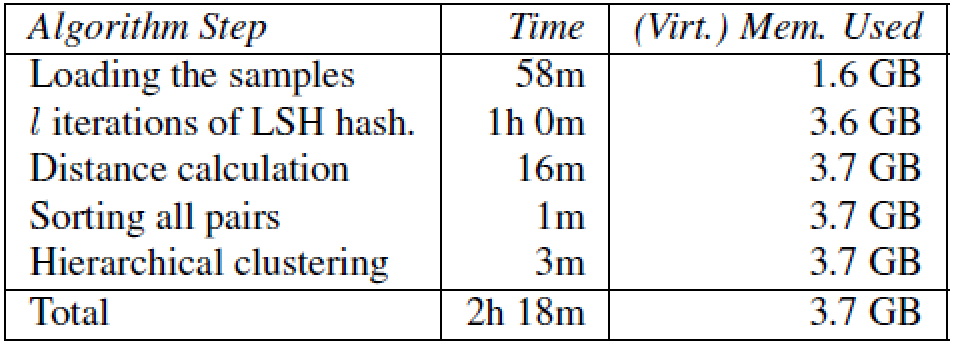


| *Behavioral Profile* | *Similarity Measure* | *Clustering* | *Optimal Threshold* | *Quality* | *Precision* | *Recall* |
|---|---|---|---|---|---|---|
| Bailey-profile [13] | NCD | exact | 0.75 | **0.916** | 0.979 | 0.935 |
| Bailey-profile [13] | Jaccard Index | exact | 0.63 | **0.801** | 0.971 | 0.825 |
| Syscalls [31] | Jaccard Index | exact | 0.19 | **0.656** | 0.874 | 0.750 |
| Our profile | Jaccard Index | exact | 0.61 | **0.959** | 0.977 | 0.981 |
| Our profile | Jaccard Index | LSH | 0.60 | **0.959** | 0.979 | 0.980 |

**Table 2. Comparative evaluation of different clustering methods.**

## Performance experiment and results

- Experimented with 75,692 samples from the complete database of ANUBIS on a XEN VM hosted on a machine with two Quad-Core Xeon (1.86 GHZ) CPUs and 8 GB of RAM with about 7GB of RAM and one physical CPU allocated to the XEN VM.
- Results
  - Their system clustered the set of samples in 2 hours and 18 minutes
  - Their system's memory requirements never exceeded 3.7 GB
  - The authors extrapolated that it would take 6 weeks of running time to cluster based on Bailey et al's system.



| *Algorithm Step* | *Time* | *(Virt.) Mem. Used* |
|---|---|---|
| Loading the samples | 58m | 1.6 GB |
| $l$ iterations of LSH hash. | 1h 0m | 3.6 GB |
| Distance calculation | 16m | 3.7 GB |
| Sorting all pairs | 1m | 3.7 GB |
| Hierarchical clustering | 3m | 3.7 GB |
| Total | 2h 18m | 3.7 GB |

**Table 3. Runtime for 75K samples.**

## Limitations

- Trace Dependence
  - The malware's execution trace will only be accurate if it behaves when being analyzed, and if certain conditions are not met for the malware, it won't behave, and thus it won't be clustered correctly
  - Techniques to explore multiple execution paths could overcome this limitation
- Evasion
  - Craft randomly mutated parts of malware's behavior so that it will be classified into different clusters. Difficult to do.
  - Malware can inject fake data dependencies using a NOP-equivalent operations to taint clean data without modifying its value. And it could hide data dependencies from their system using implicit flows to "clean" tainted data.

## Conclusions

- The paper could be split up into two different papers.
  - One paper for the feature extraction for the behavior profiles, and another for the LSH clustering
- Both the behavior profiles and leveraging LSH to effectively cluster in less than quadratic time are novel contributions in my opinion.
- The experiments were realistic and setup well, they showed the scalability of their system, and they did a good job comparing themselves with another clustering system.

## Questions?