A Quantitative Study of Accuracy in System Call-Based Malware Detection

Davide Canali, Andrea Lanzi, Davide Balzarotti, Christopher Kruegel, Mihai Christodorescu, Engin Kirda
Agenda

- Malware Detection approaches
- Goals and Contributions
- Model Specification
- Evaluation
- Results
- Pitfalls
Malware Detectors

- **Code signatures**
  - Strings or RegExps at the byte level
  - Easy to evade (packing, obfuscation)
  - Still the most widely used in the AV industry

- **Behavioral signatures**
  - Based on high-level, abstract, behavior representations
  - Usually based on system calls
  - Harder to evade
Behavior-based Malware Detectors

- Different models have been considered, but:
  - It's very difficult to understand when, and why, one should be preferred to another
  - They all lack a solid evaluation
    » Tested on very limited datasets
      · Often extracted in controlled environments, from one machine only
      · Tens of malware samples, few benign apps

- Starting to be adopted by the AV industry as well
  - Very few (if any) details available
Goals and Contributions

MAIN GOAL

• Creating a **benchmark** for designing and testing common behavioral malware detectors

CONTRIBUTIONS

• Development of a **systematic testing technique** to evaluate the quality of behavioral-based malware detectors
• Creation of a comprehensive **dataset for validating experiments**
• Evidence that the **empirical evaluation** of a malware detection model is **fundamental**
1. Behavioral Atom

- Represents the fundamental behavioral element that appears in a program syscall trace
  - System call → \textit{NtOpenFile}, \textit{NtClose}, ...
  - Action: high-level operation ("read file", ...) → \textit{ReadFile}, \textit{LoadLibrary}, ...
  - With and without parameters
- Limited to what can be collected \textit{efficiently} at runtime
  - No instruction-level tracking
  - No data-flow / taint information
Model specification - Structures

2. Signature Structure

- Describes how the atoms are combined together
  
  » Sequences (n-grams)
  
  » Tuples (ordered set)
  
  » Bags (unordered set)
  
  » Recursive structures
    (bags of sequences, tuples of ngrams, …)
3. Signature Cardinality
   - Defines how many atoms are included in the structure
     » Bounded by the maximum number of atoms in the sample
     » In practice, limited to the range 2-100
Model specification – Alert threshold

4. Alert Threshold

- How many different signatures must be matched by a program before an alert is raised
- Signatures are matched in no particular order
Too Simple?

- Why only sequences, tuples, and bags?
  - Because it is important to assess the limitations of basic models before new research delves into increasingly more complex models
  - Because they are the basic blocks to compose more complex models

- What about complex structures?
  (previous studies often adopted tree or graph-like structures)
  - Combinations of basic structures (n-grams, bags, and tuples) have the same expressive power of DAG
  - For example, it is possible to use sequences to enumerate all paths in a tree or loop-free graph
Experiment Goals

- Are programs' behaviors better characterized by complex structures, or simple ones?

- How do different parameters affect the models ability to distinguish between benign and malicious behaviors?

- Does moving to more abstract atoms improve detection?

- Which is the best combination of parameters, that:
  - maximizes the detection rate?
  - minimizes the false positives?
Datasets

[malware] – 6,000 malware traces from Anubis (training for malicious behavior)

[goodware] – the 180GB of traces collected with our collector (training for benign behavior and testing for FP)

[anubis-good] – traces of 36 benign apps run in Anubis (filtering Anubis-specific artifacts)

[mal-test] – 1,200 malware traces from a different Anubis machine (used for testing the detection rate)
Datasets

[malware] – From all existing malware categories (botnets, worms, trojans, droppers)

[goodware] – the 180GB of traces collected with our collector (training for benign behavior and testing for FP)

[anubis-good] – traces of 36 benign apps run in Anubis (filtering Anubis-specific artifacts)

[mal-test] – 1,200 malware traces from a different Anubis machine (used for testing the detection rate)
Datasets

[malware] – 6,000 malware traces from Anubis (training for malicious behavior)

[goodware] – the 180GB of traces collected with our collector (training for benign behavior and testing for FP)

[anubis-good] – traces of 36 benign apps run in Anubis (filtering Anubis-specific artifacts)

[mal-test] – 1,200 malware traces from a different Anubis machine (used for testing the detection rate)
Datasets

[malware] – 6,000 malware traces from Anubis (training for malicious behavior)

Kernel module that intercepts syscalls and extracts all the parameters

[goodware] – Collected on 10 real user machines (not under our control) for about a week: 1.56 billions syscalls, 242 unique benign applications, 362,000 process executions

[anubis-good] – traces of 36 benign apps run in Anubis (filtering Anubis-specific artifacts)

[mal-test] – 1,200 malware traces from a different Anubis machine (used for testing the detection rate)
Datasets

[malware] – 6,000 malware traces from Anubis (training for malicious behavior)

[goodware] – the 180GB of traces collected with our collector (training for benign behavior and testing for FP)

[anubis-good] – traces of 36 benign apps run in Anubis (filtering Anubis-specific artifacts)

[mal-test] – 1,200 malware traces from a different Anubis machine (used for testing the detection rate)
Signature Generation

For each model (e.g., “7-bags of syscalls with parameters”):

1. We extract ALL possible combinations from the malware dataset
   - May include pruning (see following slides)
2. We remove the ones that match the anubis-good dataset
3. We create the signatures by removing all the ones that match 9 out of 10 goodware machines
4. We test the false positives of the signature set on the 10th machine, and the detection rate on the malware-test dataset
   - Results are extracted for all possible values of the matching threshold
5. We repeat from step 3 for a total of 10 times (for each excluded machine) and we compute the average between all runs
Signature Generation

- For certain models, extracting all the possible combinations is computationally infeasible
  - e.g., extracting 3-tuples from a sample of 5000 atoms:

\[
\binom{5000}{3} \approx 20.8 \times 10^9 \text{ combinations!}
\]

- **Pruning.** A combination is generated only if:
  - It covers a minimum of 5 malware samples that are not already covered by at least 20,000 other signatures
    » The first threshold prevents overfitting
    » The second threshold prevents the generation of too many signatures for the same sample

- It is a greedy approach... it does not guarantee an optimal result
Exploring the Model Space

Atom type
- more abstract
- less abstract

Structure
- more complex
- less complex

Cardinality
- 2
- 100

4-tuples of actions with parameters
3-bags of 2-tuples of syscalls
50-grams of syscalls
What happens if we move along the axes?

Atom type

more abstract

less abstract

less complex

Structure

more complex

Cardinality 100

2
Key Indicators

Used to compare the models

\[ V_1 \] – point in which the model provides 1% FP rate

\[ V_{90} \] – point in which the model provides 90% detection

\[ V_{MAX} \] – point in which the area under the ROC curve is max
Evaluation

- We explored all the significant points in the model space
  - Some points are not significant, e.g. “n-grams of bags” would not make any sense
  - We stopped increasing the cardinality once we saw the detection rate of the model was always decreasing and $V_{\text{MAX}}$ dropped below 0.2
- 215 different detection models analyzed
- More than 220 million signatures generated
General Results

- **Signature extraction**
  - Extraction times ranged between 20 minutes and 2 days per model (on a 4-core Xeon machine with 16GB of RAM)

- **Findings:**
  - All models without parameters perform really **bad** (too generic)
  - Also signatures with **high cardinality** perform quite **bad**
    » But remember that we are looking for “general” signatures that can match multiple samples
  - The best model is “2-bags of 2-tuples of actions, with parameters”:
    **99% detection with 0.4% FP** (variance of 0.00016)
Table 3: Evaluation summary of different types of models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Cardinality Range</th>
<th>$V_{max}$</th>
<th>Best Cardinality</th>
<th>$V_{00}$</th>
<th>$V_{1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n$-grams of syscalls</td>
<td>2–40</td>
<td>0.615</td>
<td>10</td>
<td>31.7%</td>
<td>4.1%</td>
</tr>
<tr>
<td>$n$-grams of syscalls with args</td>
<td>2–40</td>
<td>0.775</td>
<td>3</td>
<td>15.8%</td>
<td>43.3%</td>
</tr>
<tr>
<td>$n$-grams of action</td>
<td>2–75</td>
<td>0.423</td>
<td>15</td>
<td>62.2%</td>
<td>0.4%</td>
</tr>
<tr>
<td>$n$-grams of action with args</td>
<td>2–75</td>
<td>0.737</td>
<td>2</td>
<td>27.1%</td>
<td>45.9%</td>
</tr>
<tr>
<td>bags of syscalls</td>
<td>1–10</td>
<td>0.127</td>
<td>3</td>
<td>–</td>
<td>12.8%</td>
</tr>
<tr>
<td>bags of syscalls with args</td>
<td>1–20</td>
<td>0.736</td>
<td>1</td>
<td>26.4%</td>
<td>43.3%</td>
</tr>
<tr>
<td>bags of actions</td>
<td>1–10</td>
<td>0.004</td>
<td>1</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>bags of actions with args</td>
<td>1–15</td>
<td>0.970</td>
<td>4</td>
<td>0.4%</td>
<td>97.3%</td>
</tr>
<tr>
<td>tuples of syscalls</td>
<td>2–10</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>tuples of syscalls with args</td>
<td>2–10</td>
<td>0.616</td>
<td>2</td>
<td>–</td>
<td>28.0%</td>
</tr>
<tr>
<td>tuples of actions</td>
<td>2–10</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>tuples of actions with args</td>
<td>2–10</td>
<td>0.987</td>
<td>2</td>
<td>0.0%</td>
<td>99.2%</td>
</tr>
<tr>
<td>bags of $n$-grams of syscalls</td>
<td>2–4/2–4</td>
<td>0.500</td>
<td>2/2</td>
<td>–</td>
<td>8.2%</td>
</tr>
<tr>
<td>bags of $n$-grams of syscalls with args</td>
<td>2–4/2–4</td>
<td>0.648</td>
<td>2/4</td>
<td>–</td>
<td>30.2%</td>
</tr>
<tr>
<td>bags of $n$-grams of action</td>
<td>2–4/2–4</td>
<td>0.111</td>
<td>3/4</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>bags of $n$-grams of action with args</td>
<td>2–4/2–4</td>
<td>0.529</td>
<td>2/3</td>
<td>–</td>
<td>22.0%</td>
</tr>
<tr>
<td>bags of tuples of syscalls</td>
<td>2–4/2–4</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>bags of tuples of syscalls with args</td>
<td>2–4/2–4</td>
<td>0.497</td>
<td>2/2</td>
<td>–</td>
<td>33.8%</td>
</tr>
<tr>
<td>bags of tuples of action</td>
<td>2–4/2–4</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>bags of tuples of action with args</td>
<td>2–4/2–4</td>
<td>0.990</td>
<td>2/2</td>
<td>0.42%</td>
<td>–</td>
</tr>
<tr>
<td>tuples of $n$-grams of syscalls</td>
<td>2–4/2–4</td>
<td>0.509</td>
<td>2/2</td>
<td>–</td>
<td>2.9%</td>
</tr>
<tr>
<td>tuples of $n$-grams of syscalls with args</td>
<td>2–4/2–4</td>
<td>0.624</td>
<td>2/3</td>
<td>–</td>
<td>26.5%</td>
</tr>
<tr>
<td>tuples of $n$-grams of action</td>
<td>2–4/2–4</td>
<td>0.142</td>
<td>3/4</td>
<td>–</td>
<td>0.1%</td>
</tr>
<tr>
<td>tuples of $n$-grams of action with args</td>
<td>2–4/2–4</td>
<td>0.536</td>
<td>2/2</td>
<td>–</td>
<td>24.9%</td>
</tr>
<tr>
<td>tuples of bags of syscalls</td>
<td>2–4/2–4</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>tuples of bags of syscalls with arguments</td>
<td>2–4/2–4</td>
<td>0.480</td>
<td>2/2</td>
<td>–</td>
<td>32.4%</td>
</tr>
<tr>
<td>tuples of bags of actions</td>
<td>2–4/2–4</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>tuples of bags of actions with arguments</td>
<td>2–4/2–4</td>
<td>0.873</td>
<td>2/2</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
Impact of Matching Threshold

- Both the detection rate and the false positives decrease when the matching threshold is increased
  - The drop is faster for models based on a semantically rich set of atoms (e.g., syscalls with parameters)
Impact of Signature Cardinality

- For low values of the cardinality, adding atoms to the signatures can improve the results
  - Increasing the cardinality above 10 generates signatures that over-fit the malware training dataset, thus decreasing detection (too specific)
  - Recursive structures show similar trends, but drop faster than simple ones
Impact of Atoms and Signature Structure

- Models based on low-level atoms (syscalls)
  - n-grams > bags > tuples
- Models based on high-level atoms (actions)
  - tuples > bags > n-grams

- Recursive structures
  - Tuples and bags provide better results than n-grams
  - Best with high-level atoms (actions) with parameters
Impact on Performances

- Prototype testing on **12 hours of user activity**
  - In python → can be implemented more efficiently

<table>
<thead>
<tr>
<th>Number of signatures</th>
<th>Memory consumption</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ngrams</td>
</tr>
<tr>
<td>250,000</td>
<td>144.7 MB</td>
<td>25 min</td>
</tr>
<tr>
<td>1,000,000</td>
<td>359.6 MB</td>
<td>31 min</td>
</tr>
<tr>
<td>5,000,000</td>
<td>1.0 GB</td>
<td>43 min</td>
</tr>
</tbody>
</table>

- High numbers of signatures lead to high memory consumption
  - The number of signatures is related to the signature cardinality
  - **Signatures of high cardinality may be difficult to employ** in real world deployments
Limits of Analytical Reasoning

- It is very tempting to propose rules, based on intuitions, about the models and their accuracy
- Example:
  - Increasing the cardinality makes the signatures more specific and, therefore, less likely to match on both the goodware and the malware datasets
  - Therefore, a model based on 3-grams should generate less false positives than a model based on 2-grams
  - Similarly a model based on 3-bags generates more false positives than one based on 3-grams
Wrong!

- Extending the property of a signature to the property of the models based on that signature is a very common pitfall
  - Changing a parameter does not only change the matching, but also the number of signatures extracted!
  - Against common sense, making the signatures more specific can, in some cases, increase the FP of the entire model

**Malware**: (a1, a2, a3, a4, a5)
**Goodware**: (a3, a1, a2, a5, a4, a2, a3)
**Signatures**:
  - 2-grams: ?
  - 3-grams: ?
  - k-bags: ?
Wrong!

- Extending the property of a signature to the property of the models based on that signature is a very common pitfall
  - Changing a parameter does not only change the matching, but also the number of signatures extracted!
  - Against common sense, making the signatures more specific can, in some cases, increase the FP of the entire model

Malware: (a1, a2, a3, a4, a5)
Goodware: (a3, a1, a2, a5, a4, a2, a3)
Possible combinations from malware trace:
  2-grams: [a1,a2] [a2,a3] [a3,a4] [a4,a5]
  3-grams: [a1,a2,a3] [a2,a3,a4] [a3,a4,a5]
  2-bags: {a1,a2} {a1,a3} {a1,a4} {a1,a5} {a2,a3} {a2,a4} {a2,a5} {a3,a4} {a3,a5} {a4,a5}
Wrong!

- Extending the property of a signature to the property of the models based on that signature is a very common pitfall
  - Changing a parameter does not only change the matching, but also the number of signatures extracted!
  - Against common sense, making the signatures more specific can, in some cases, increase the FP of the entire model

Malware: (a1, a2, a3, a4, a5)
Goodware: (a3, a1, a2, a5, a4, a2, a3)
Signatures:
- 2-grams: [a1,a2] [a2,a3] [a3,a4] [a4,a5]
- 3-grams: [a1,a2,a3] [a2,a3,a4] [a3,a4,a5]
- 2-bags: {a1,a2} {a1,a3} {a1,a4} {a1,a5} {a2,a3}
  {a2,a4} {a2,a5} {a3,a4} {a3,a5} {a4,a5}
Wrong!

- Extending the property of a signature to the property of the models based on that signature is a very common pitfall
  - Changing a parameter does not only change the matching, but also the number of signatures extracted!
  - Against common sense, making the signatures more specific can, in some cases, increase the FP of the entire model

**Malware**: (a1, a2, a3, a4, a5)
**Goodware**: (a3, a1, a2, a5, a4, a2, a3)
**Signatures**:
  - 2-grams: [a3,a4] [a4,a5]
  - 3-grams: [a1,a2,a3] [a2,a3,a4] [a3,a4,a5]
  - k-bags: none
Conclusions

- The three indicators \( (V_1, V_{90}, V_{\text{max}}) \) don't always provide consistent results
  - The best model depends on the optimization goal

- Empirical testing is fundamental
  - We showed it's easy to fall in common pitfalls when trying to generalize results
  - Future works should be supported by strong evaluation
    - Avoid a-priori rules!
Thank you

For further questions, suggestions, comments:

andrew@iseclab.org
Backup Slides
Behavioral Detection (in Academia)

- “Static-Aware Malware Detection” -
  - Model: templates based on instruction sequences where variables and symbolic constant are used
  - Generation: Manual
  - Dataset: 2 templates tested on 3 malware families
    200k small benign executables (less than 1.5KB each)
  - Assume it is possible to reliably disassemble the programs

- “Mining Specifications of Malicious Behavior” - FSE 07
  - Model: DAG of syscalls (no parameters) generated by comparing benign and malicious programs executions
  - Generation: Automatic
  - Dataset: 16 malware samples, 4 benign applications run for 1 minute each
Behavioral Detection (in Academia)

- “Effective and Efficient Malware Detection at the End Host” - Usenix 09
  - Model: graph of syscalls + program slices to compute the parameter transformations to infer data-flow
  - Generation: Automatic
  - Dataset: 563 malware samples belonging to 6 families, 5 goodware, 1 machine
  - Result: 92% detection on same families, 23% otherwise (5% to 40% overhead)

- “A layered Architecture for Detecting Malicious Behaviors” - RAID 08
  - Model: 3-layer graph (syscalls, similar actions, aggregate/composite effects) for 7 suspicious behaviors (e.g., download and execute, data leak, tcp proxy, ...)
  - Generation: Manual
  - Dataset: 7 malware, 11 goodware
  - Performance: require QEMU + taint analysis + mouse/keyboard tracking
    Up to 34x slowdown
Behavioral-Based Models (AV Companies)

- Very few (if any) details available
- Often mentioned in web-pages and press releases
  - Not much against evasions, but more as a “Signature-less technique to detect unknown malware”
- Adopted (?) by all vendors...
  - Sana Security SafeConnect (2005?)
    - Acquired by AVG in 2009
  - Symantec SONAR (2007)
  - Panda TruePrevent (2007)
  - NovaShield (2008)
Extracting Signatures

```
NtOpenKey("SYSTEM\Cu ... 70B"", 131097)
NtQueryValueKey(1640, "EnableDHCP", 2)
NtQueryValueKey(1640, "DhcpServer", 2)
NtQueryValueKey(1640, "DhcpServer", 2)
NtClose(1640)
NtCreateFile("\\Device\...", 3, 0)
NtClose(1641)
```
Extracting Signatures

NtOpenKey("SYSTEM\Cu ... 70B}", 131097)
NtQueryValueKey("SY...", "EnableDHCP", 2)
NtQueryValueKey("SY...", "DhcpServer", 2)
NtQueryValueKey("SY...", "DhcpServer", 2)
NtClose("SY...")
NtCreateFile("\\Device\...", 3, 0)
NtClose("\\Devi...")

Normalization
Extracting Signatures

NtOpenKey("SYSTEM\Cu ...
70B} ", 131097)
NtQueryValueKey("SYS...
EnableDHCP", 2)
NtQueryValueKey("SYS...
DhcpServer", 2)
NtQueryValueKey("SYS...
DhcpServer", 2)
NtClose("SYS...
)  
NtCreateFile("\Device\...
", 3, 0)
NtClose("\Device...
)  

S1: NtOpenKey("SYSTEM\Cu ...
", 131097)

S2: NtQueryValueKey, NtQueryValueKey, NtQueryValueKey

S3: ReadKeyValue("SYSTEM\Cu ...
\EnableDHCP")
Goodware Dataset

- Kernel module to intercept **syscalls** and extract **all the parameters**
  - 79 different system calls in 5 categories (filesystem, networking, registry, memory)
- Collected on 10 real user machines (not under our control) for about a week
  - 1.56 billions syscalls
  - 242 unique benign applications
  - 362,000 process executions
  - 180 GB of execution traces
Data Collection (goodware)

- We run our module on **10 real user machines** (not under our control) for about **a week**
  - 4 Home/Laptop machines
  - 1 Office
  - 1 Lab
  - 2 production
  - 2 development

- **Collected data:**
  - 1.56 billions syscalls
  - 242 unique benign applications
  - 362,000 process executions
  - 180 GB of execution traces
Data Collection (malware)

- Malicious samples extracted from Anubis
  - 6000 random samples of active malware
  - From all existing malware categories
    » Botnets
    » Worms
    » Trojans
    » Droppers
    » ...

Normalization datasets

- 1200 additional samples from Anubis
  - Extracted from a different machine than the ones used in production
  - Still from multiple malware families
  - Named 'malware-test'

- 36 execution traces of benign applications
  - Executed under Anubis
  - Named 'anubis-good'

- Purpose of these two datasets:
  - Eliminating any machine-specific artifacts that may introduce noise in our evaluation results
Impact of Pruning Techniques

● Our pruning approach is greedy
  – The extracted signatures depend on the order of the samples in the training set

● We picked one model, and built signatures with 3 different orderings of the training samples, with any cardinality
  – Different orderings sensibly affect the number of extracted signatures
  – The 3 key indicators are only marginally affected
    » Fluctuations of 3% max.
  – The trends between different models were not affected
Impact on Performances

- Prototype testing on **12 hours of user activity**
  - In python → can be implemented more efficiently

<table>
<thead>
<tr>
<th>Number of signatures</th>
<th>Memory consumption</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>ngrams</td>
</tr>
<tr>
<td>250,000</td>
<td>144.7 MB</td>
<td>25 min</td>
</tr>
<tr>
<td>1,000,000</td>
<td>359.6 MB</td>
<td>31 min</td>
</tr>
<tr>
<td>5,000,000</td>
<td>1.0 GB</td>
<td>43 min</td>
</tr>
</tbody>
</table>

- High numbers of signatures lead to high memory consumption
  - The number of signatures is related to the signature cardinality
  - **Signatures of high cardinality may be difficult to employ** in real world deployments
Number of Signatures

- Extracting and matching signatures that contain a large number of elements is extremely time consuming

- n-grams
  - Signature numbers keep growing linearly with cardinality
  - Those that actually contribute to detection decrease for cardinalities higher than 10 (overfitting)

- Bags
  - Very high number of signatures (because too general)

- Sequences
  - Similar to n-grams, but more matching signatures
Insights on Signatures

- Most of the **FPs** are generated by signatures related to **registry operations**
  - Top ten registry keys associated to autostart locations were more often a cause of false positives than detection

- The “best” signatures often contained the **LoadLibrary action**

- **Tuples** perform better than bags not because of their ordering, but because they **can model repetitions**
Table 3: Evaluation summary of different types of models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Cardinality Range</th>
<th>$V_{max}$</th>
<th>Best Cardinality</th>
<th>$V_{00}$</th>
<th>$V_{1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>n-grams of syscalls</td>
<td>2–40</td>
<td>0.615</td>
<td>10</td>
<td>31.7%</td>
<td>4.1%</td>
</tr>
<tr>
<td>n-grams of syscalls with args</td>
<td>2–40</td>
<td>0.775</td>
<td>3</td>
<td>15.8%</td>
<td>43.3%</td>
</tr>
<tr>
<td>n-grams of action</td>
<td>2–75</td>
<td>0.423</td>
<td>15</td>
<td>62.2%</td>
<td>0.4%</td>
</tr>
<tr>
<td>n-grams of action with args</td>
<td>2–75</td>
<td>0.737</td>
<td>2</td>
<td>27.1%</td>
<td>45.9%</td>
</tr>
<tr>
<td>bags of syscalls</td>
<td>1–10</td>
<td>0.127</td>
<td>3</td>
<td>–</td>
<td>12.8%</td>
</tr>
<tr>
<td>bags of syscalls with args</td>
<td>1–20</td>
<td>0.736</td>
<td>1</td>
<td>26.4%</td>
<td>43.3%</td>
</tr>
<tr>
<td>bags of actions</td>
<td>1–10</td>
<td>0.004</td>
<td>1</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>bags of actions with args</td>
<td>1–15</td>
<td>0.970</td>
<td>4</td>
<td>0.4%</td>
<td>97.3%</td>
</tr>
<tr>
<td>tuples of syscalls</td>
<td>2–10</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>tuples of syscalls with args</td>
<td>2–10</td>
<td>0.616</td>
<td>2</td>
<td>–</td>
<td>28.0%</td>
</tr>
<tr>
<td>tuples of actions</td>
<td>2–10</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>tuples of actions with args</td>
<td>2–10</td>
<td>0.987</td>
<td>2</td>
<td>0.0%</td>
<td>99.2%</td>
</tr>
<tr>
<td>bags of n-grams of syscalls</td>
<td>2–4/2–4</td>
<td>0.500</td>
<td>2/2</td>
<td>–</td>
<td>8.2%</td>
</tr>
<tr>
<td>bags of n-grams of syscalls with args</td>
<td>2–4/2–4</td>
<td>0.648</td>
<td>2/4</td>
<td>–</td>
<td>30.2%</td>
</tr>
<tr>
<td>bags of n-grams of action</td>
<td>2–4/2–4</td>
<td>0.111</td>
<td>3/4</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>bags of n-grams of action with args</td>
<td>2–4/2–4</td>
<td>0.529</td>
<td>2/3</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>bags of tuples of syscalls</td>
<td>2–4/2–4</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>bags of tuples of syscalls with args</td>
<td>2–4/2–4</td>
<td>0.497</td>
<td>2/2</td>
<td>–</td>
<td>33.8%</td>
</tr>
<tr>
<td>bags of tuples of action</td>
<td>2–4/2–4</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>bags of tuples of action with args</td>
<td>2–4/2–4</td>
<td>0.990</td>
<td>2/2</td>
<td>0.42%</td>
<td>–</td>
</tr>
<tr>
<td>tuples of n-grams of syscalls</td>
<td>2–4/2–4</td>
<td>0.509</td>
<td>2/2</td>
<td>–</td>
<td>2.9%</td>
</tr>
<tr>
<td>tuples of n-grams of syscalls with args</td>
<td>2–4/2–4</td>
<td>0.624</td>
<td>2/3</td>
<td>–</td>
<td>26.5%</td>
</tr>
<tr>
<td>tuples of n-grams of action</td>
<td>2–4/2–4</td>
<td>0.142</td>
<td>3/4</td>
<td>–</td>
<td>0.1%</td>
</tr>
<tr>
<td>tuples of n-grams of action with args</td>
<td>2–4/2–4</td>
<td>0.536</td>
<td>2/2</td>
<td>–</td>
<td>24.9%</td>
</tr>
<tr>
<td>tuples of bags of syscalls</td>
<td>2–4/2–4</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>tuples of bags of syscalls with arguments</td>
<td>2–4/2–4</td>
<td>0.480</td>
<td>2/2</td>
<td>–</td>
<td>32.4%</td>
</tr>
<tr>
<td>tuples of bags of actions</td>
<td>2–4/2–4</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>tuples of bags of actions with arguments</td>
<td>2–4/2–4</td>
<td>0.873</td>
<td>2/2</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>