AccessMiner: Using System-Centric Models for Malware Protection

Andrea Lanzi[1] Davide Balzarotti[1] Christopher Kruegel[2]
Mihai Christodorescu[3] Engin Kirda[1]

[1]Institute Eurecom

[2]UC Santa Barbara

[3]IBM T.J. Watson Research

17th ACM Conference on Computer and Communications Security (CCS 2010)

System-call based detector

- Most popular way to characterize the behavior of programs is based on the analysis of the system calls or Win32 API functions.
- Different models have been proposed:
  - Sequences of system calls. (Mukkalama 2004, Kang 2005)
  - System call patterns based on data flow dependencies. (Martignoni 2008, Kolbitsch 2009)
  - System call and argument. (Kirda 2006)

System-Call based real-time detector

Our research motivations

- Most of these detectors follow the **program-centric** approach and they **lack context** that captures how benign programs in general interact with OS.
- The evaluation of the **false positives** for these models are very *poor*:
  - the programs are exercised in a limited fashion.
  - they are often using synthetic inputs.
  - experiments are performed on a single machine.

Our research motivations

Program-centric models fail to capture program behavior at a higher level of abstraction!!!

- The evaluation of the **false positives** for these models are very *poor*:
  - the programs are exercised in a limited fashion.
  - they are often using synthetic inputs.
  - experiments are performed on a single machine.

Our research motivations

Program-centric models fail to capture program behavior at a higher level of abstraction!!!

- The evaluation of the **false positives** for these models are very *poor*:
  - [illegible]

Poor evaluation of False positives!!!

  - experiments are performed on a single machine.

Contribution (1): building a "good" benign data collection

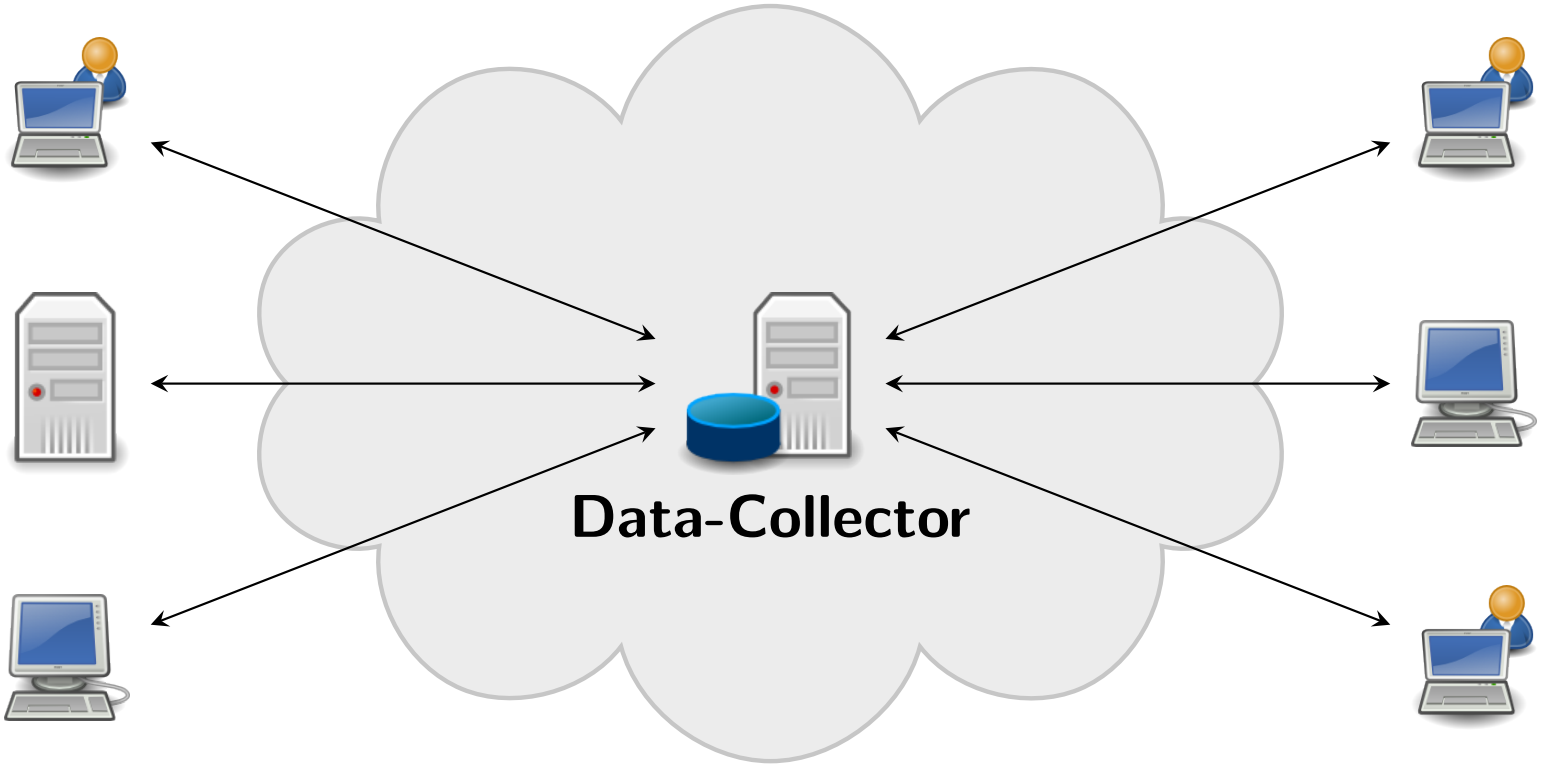
- A large scale of malware data collection is available from different systems collector. (e.g Anubis, Malfease etc.)
- Collecting **a large scale of "real" benign data collection is a challenge**:
  - We need to **convince people** that their private data are protected **(privacy issue).**
  - We need to **collect benign data from a different sources**: home machine, lab machine, developing machine etc.**(data diversity).**
  - The logger should not have any **bad performance impact**. **(logging procedure should be safe).**

Contribution (1): building a "good" benign data collection

We performed a **large scale "real" data collection** of system calls. We collected data for **several weeks and from different real users**. Our dataset contains:

- 1.5 billion of system calls.
- 242 applications.
- 362,000 processes executions.

System data collector



Data Description

`<timestamp, program, pid, ppid, system call, args, result>`

System data collector

- Kernel collector logs 79 different system calls 5 categories:
  - 25 related to files,
  - 23 related to registries,
  - 1 related to networking,
  - 5 related to memory sections.
- Kernel collector protects the user's private data that are obfuscated with a random value:
  - Pathnames that do not belong system-path (e.g.`C:\Documents and Settings`),
  - All registry keys below the user-root registry key (HKLM)
  - All IP addresses.

Log collector

System data collector

Machine	*Usage*	*Data* (GB)	*System calls* ($\times 10^6$)	*Processes* ($\times 10^3$)	*Applications*
1	office	18.0	285	55.1	90
2	home	4.5	70	22.4	87
3	home	5.6	89	17.7	46
4	prod.	32.0	491	110.9	41
5	prod.	34.0	514	125.6	42
6	lab.	14.0	7	2.8	73
7	home	1.3	19	3.7	49
8	home	1.2	18	3.0	22
9	dev.	1.6	27	8.5	47
10	dev.	2.3	36	12.9	26
Total		114.5	1,556	362.6	242

Contribution (2): Studying diversity of system calls

- We analyze the **diversity of the system call data** in relationship to a particular model used to capture program behaviors.
- We cast the problem of studying the diversity of our data set as the problem of **understanding whether a model is able to capture the data in a precise fashion**.

n-gram models

- We use **n-grams as the basic technique to models system calls**. The n-gram model has been used as part of many different security solutions.
- n-grams were used to model program activity to **detect software exploits** and to **identify malicious code in network payloads.**

n-gram model example

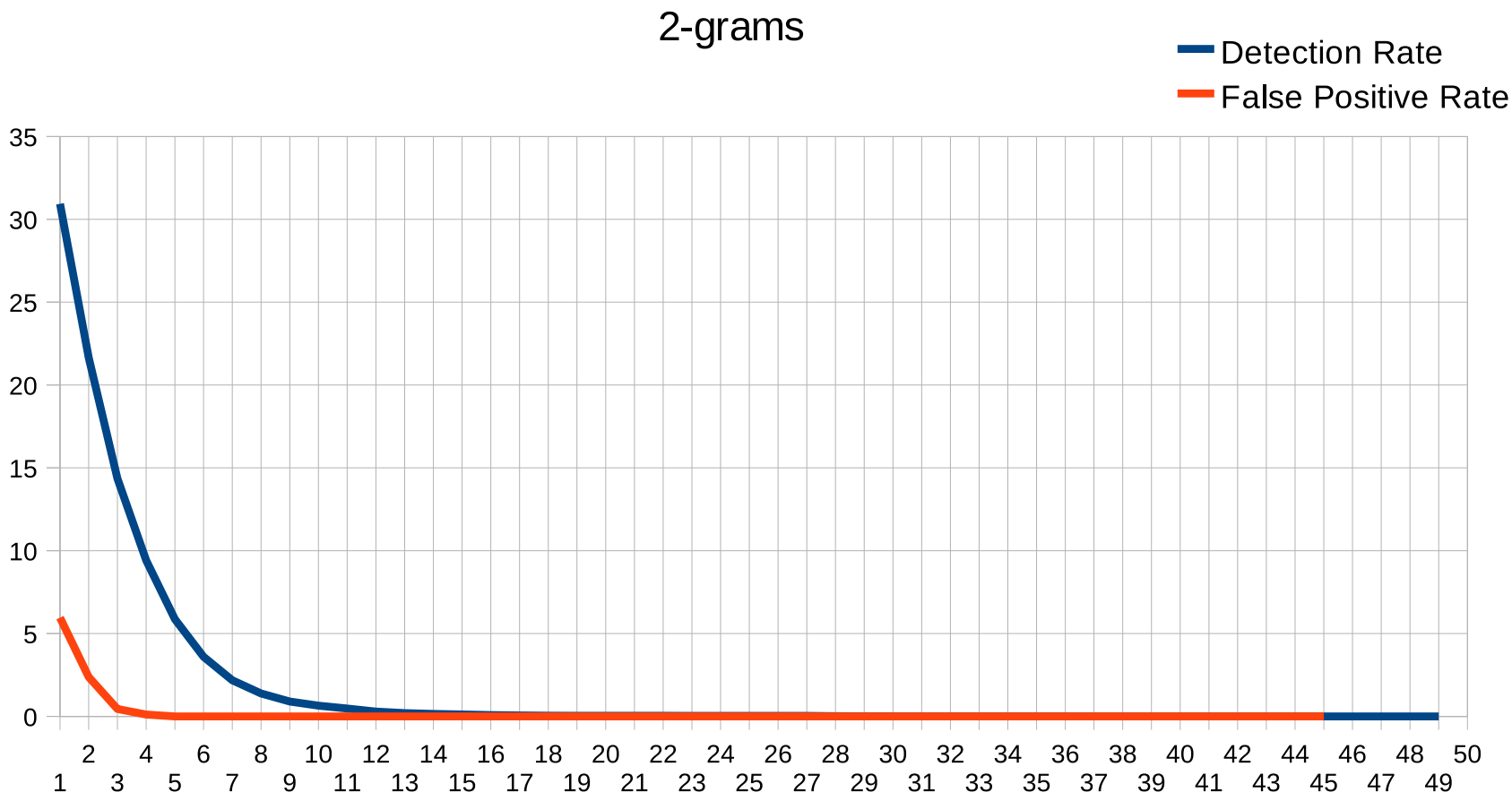
n-gram sequence of system calls invoked by the running program with a sliding window of size n.

- application invokes 5 system call: $< 12, 3, 17, 9, 11 >$
- the 3-gram $\{< 12, 3, 17 >, < 3, 17, 9 >, < 17, 9, 11 >\}$.

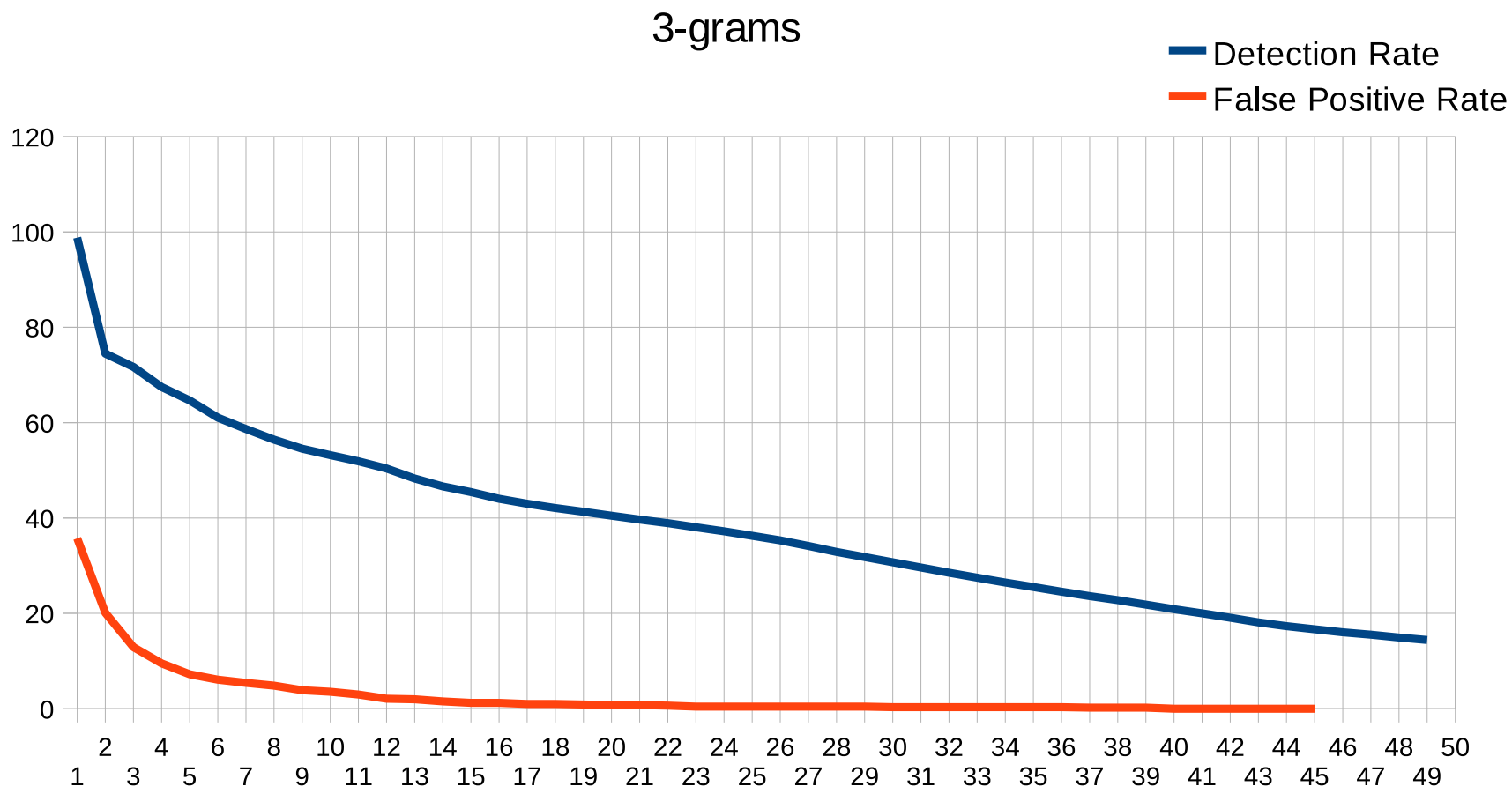
n-gram model

- **Training:** we find all n-grams that appear in some of the malware models but not in any of the models built for the benign programs. For malware model we used 10,000 samples from Anubis system.
- **Detection:** Using the "unique" n-grams we can perform detection. **When benign programs contain more that k instances of n-grams that are considered malicious.**

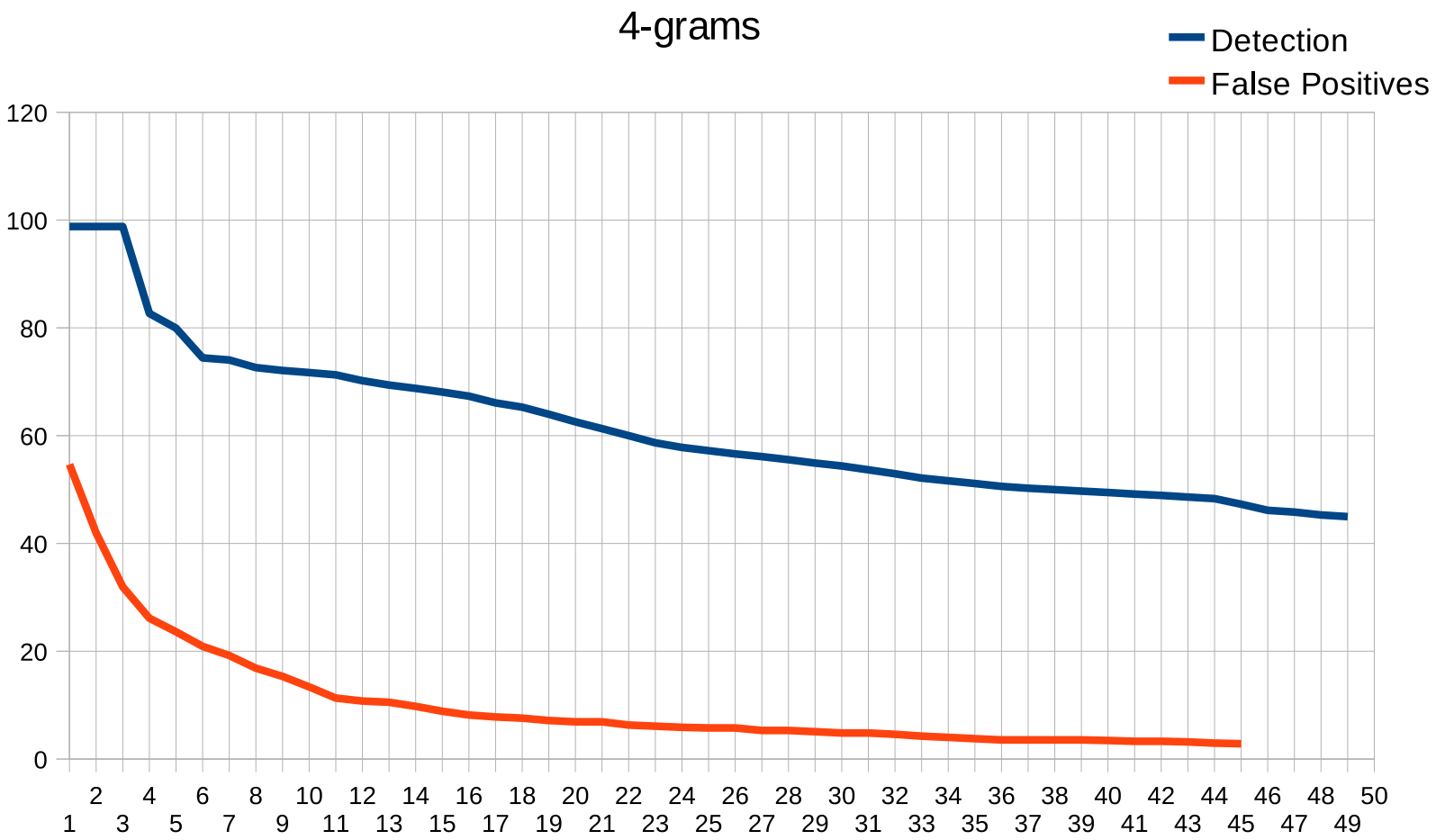
2-gram model detection results



3-gram model detection results



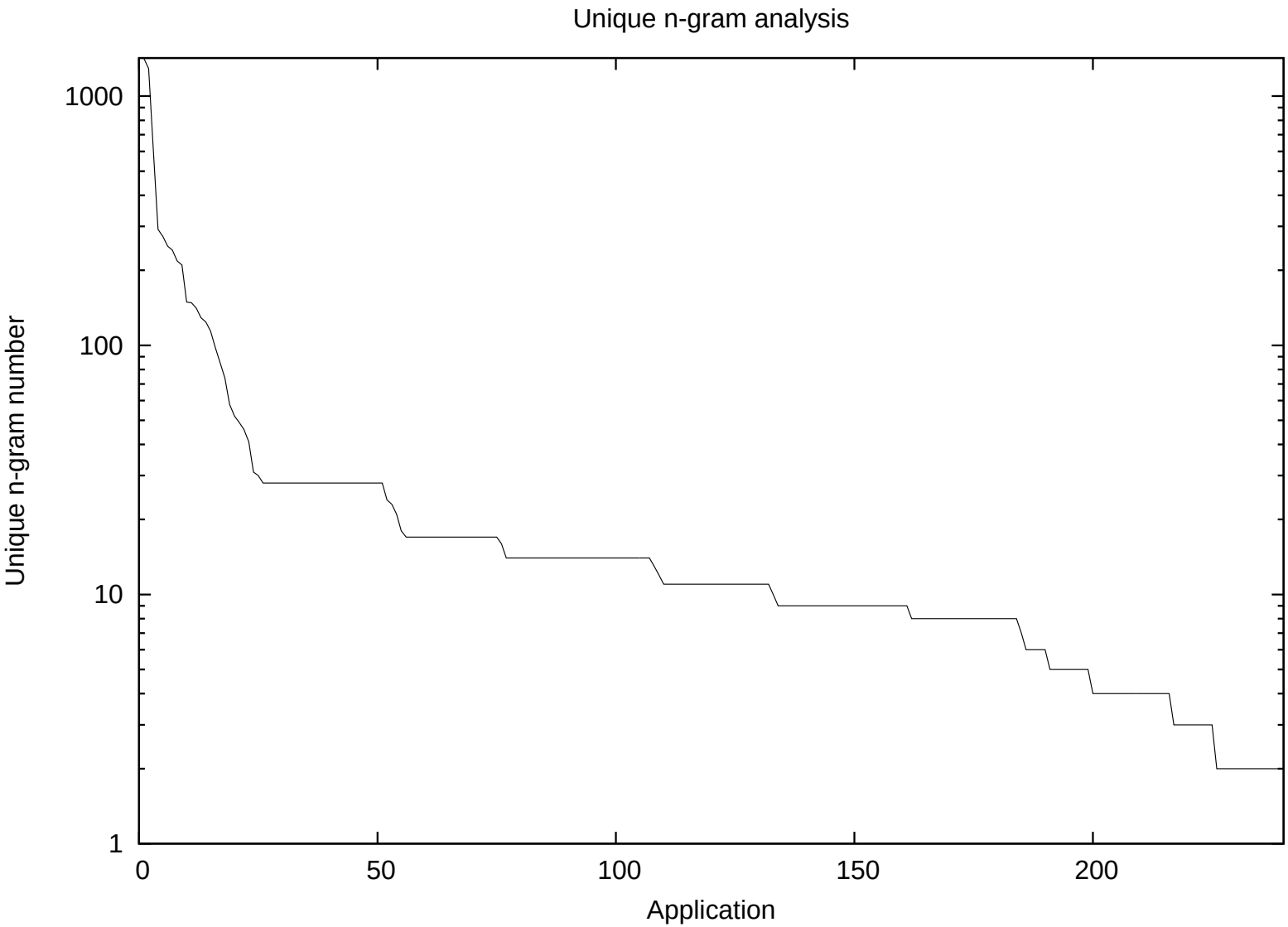
4-gram model detection results



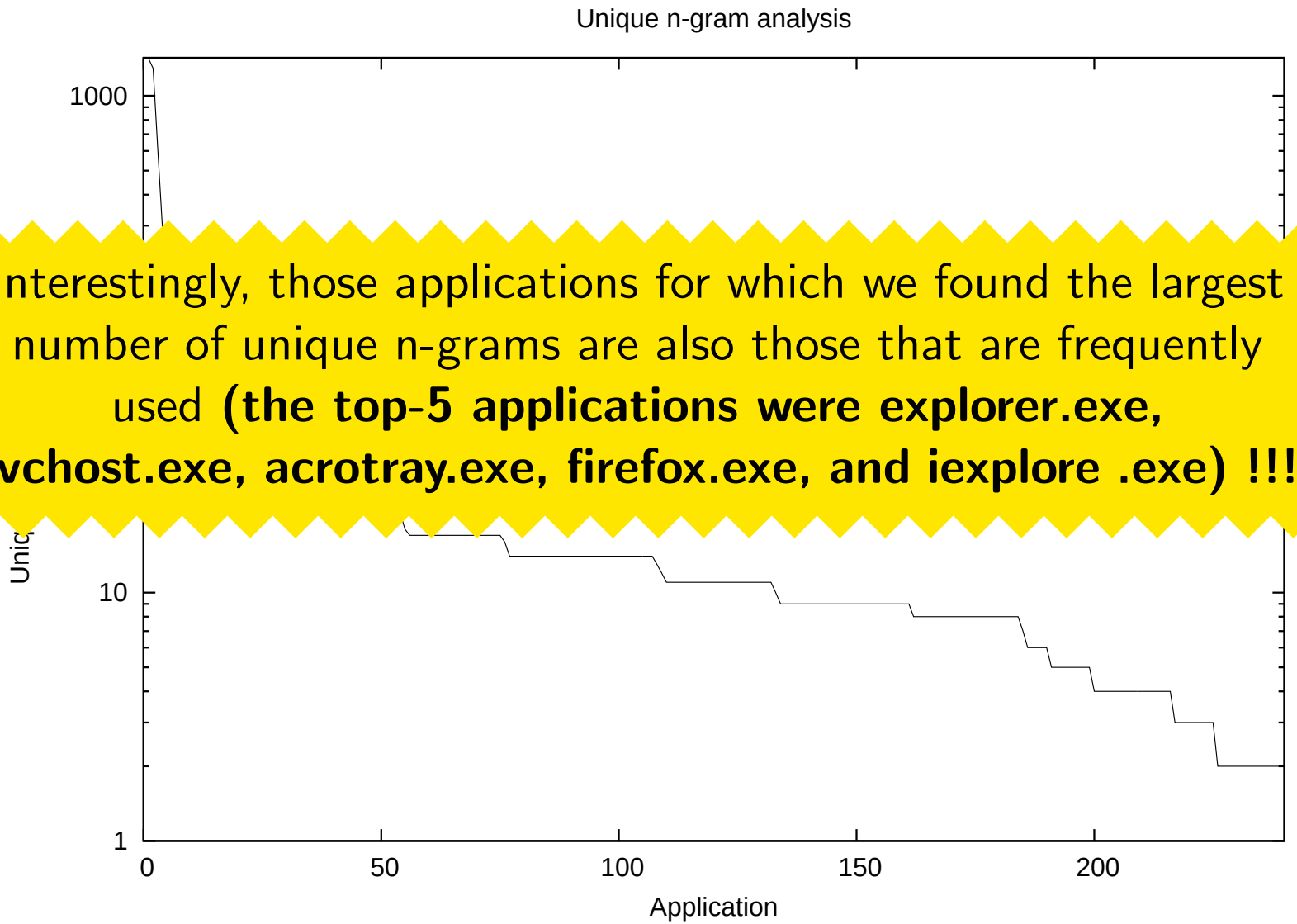
n-gram model detection results

- We examined the number of **unique n-grams that can be found in each of the 242 different applications** that we observe.
- Under the assumption that n-grams are a good model to capture program behavior in general, we would expect that the **number of such unique sequences is low.**

n-gram model detection results



n-gram model detection results



Interestingly, those applications for which we found the largest number of unique n-grams are also those that are frequently used **(the top-5 applications were explorer.exe, svchost.exe, acrotray.exe, firefox.exe, and iexplore .exe) !!!**

Contributions (3): Access activity model

- The intuition is that benign programs in general follow certain ways in which they use the OS resources.
- To capture normal interactions with the filesystem and the Windows registry, we propose **access activity model** specifies a set of labels for OS.
- An access activity model specifies a set of labels for operating system resources (files and registries).

Access activity model

- A label L is a set of access tokens $\{t_0, t_1, \ldots, t_n\}$.
- Each token t is a pair $\langle a, op \rangle$. The first component a represents the application, the second component op represents the type of access).
- The possible values for the operation component of an access token are **read**, **write**, and **execute** for file-system resources (directories), and **read** and **write** for registry sub-keys.

Virtual filesystem

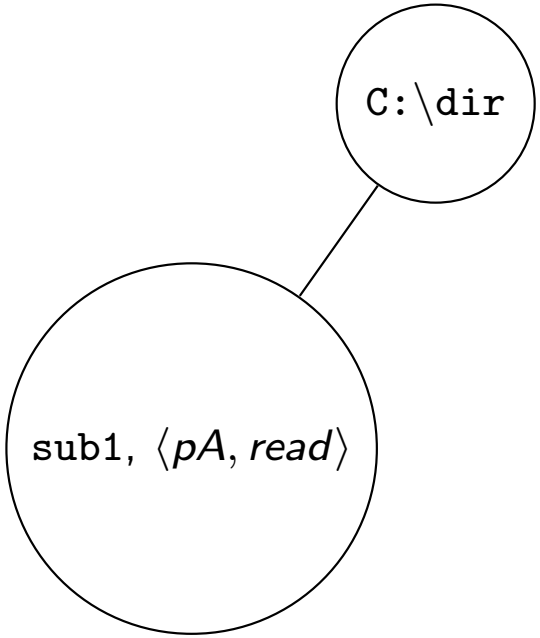
In the first step we build a **unique virtual filesystem** that includes all the file pathnames defined into the benign data logs files. Same filesystem is also build for the registries pathnames.

Virtual filesystem

`C:\dir\sub1\foo` $\langle pA, read \rangle$

Virtual filesystem

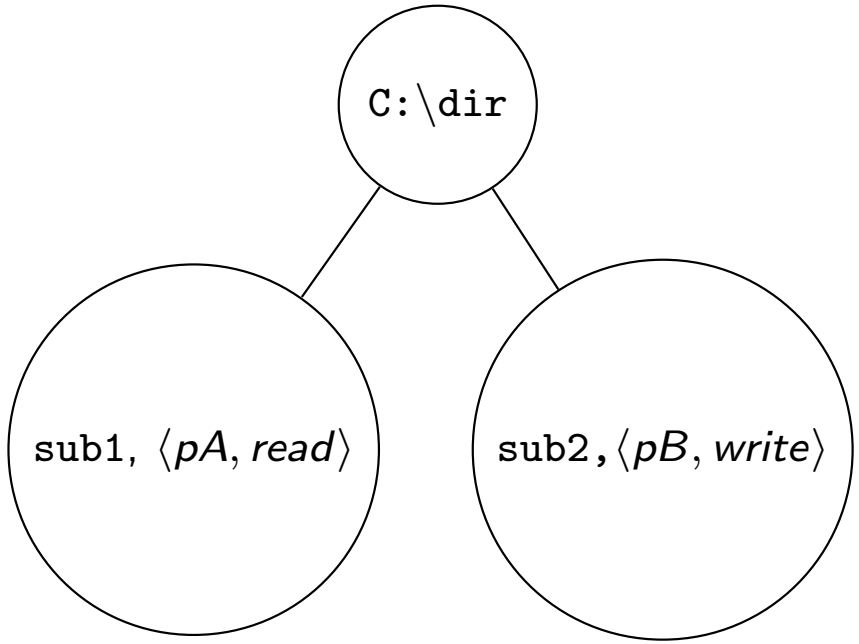
`C:\dir\sub1\foo` $\langle pA, read \rangle$



Virtual filesystem

C:\dir\sub2\bar $\langle pB, write \rangle$

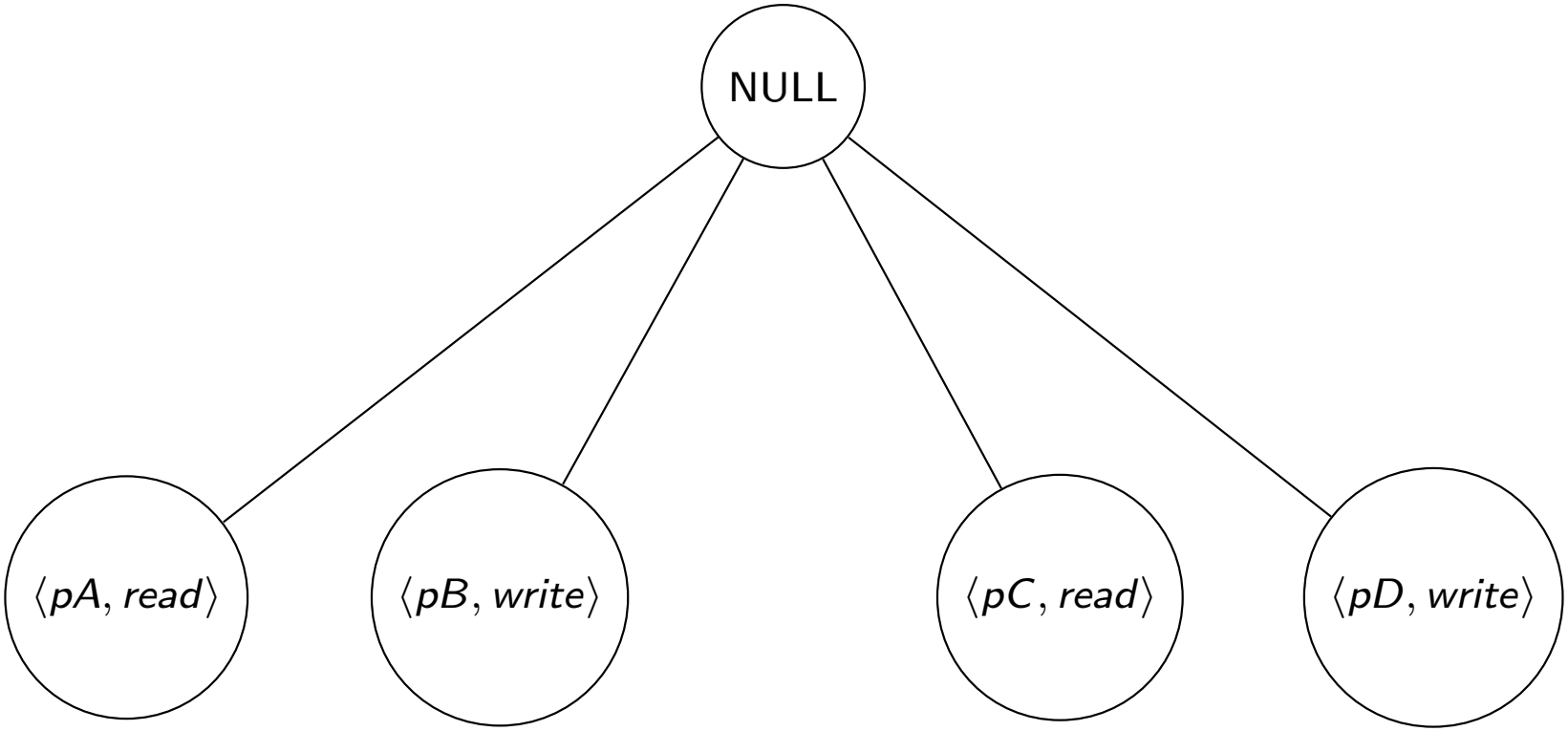
C:\dir\sub1\foo $\langle pA, read \rangle$



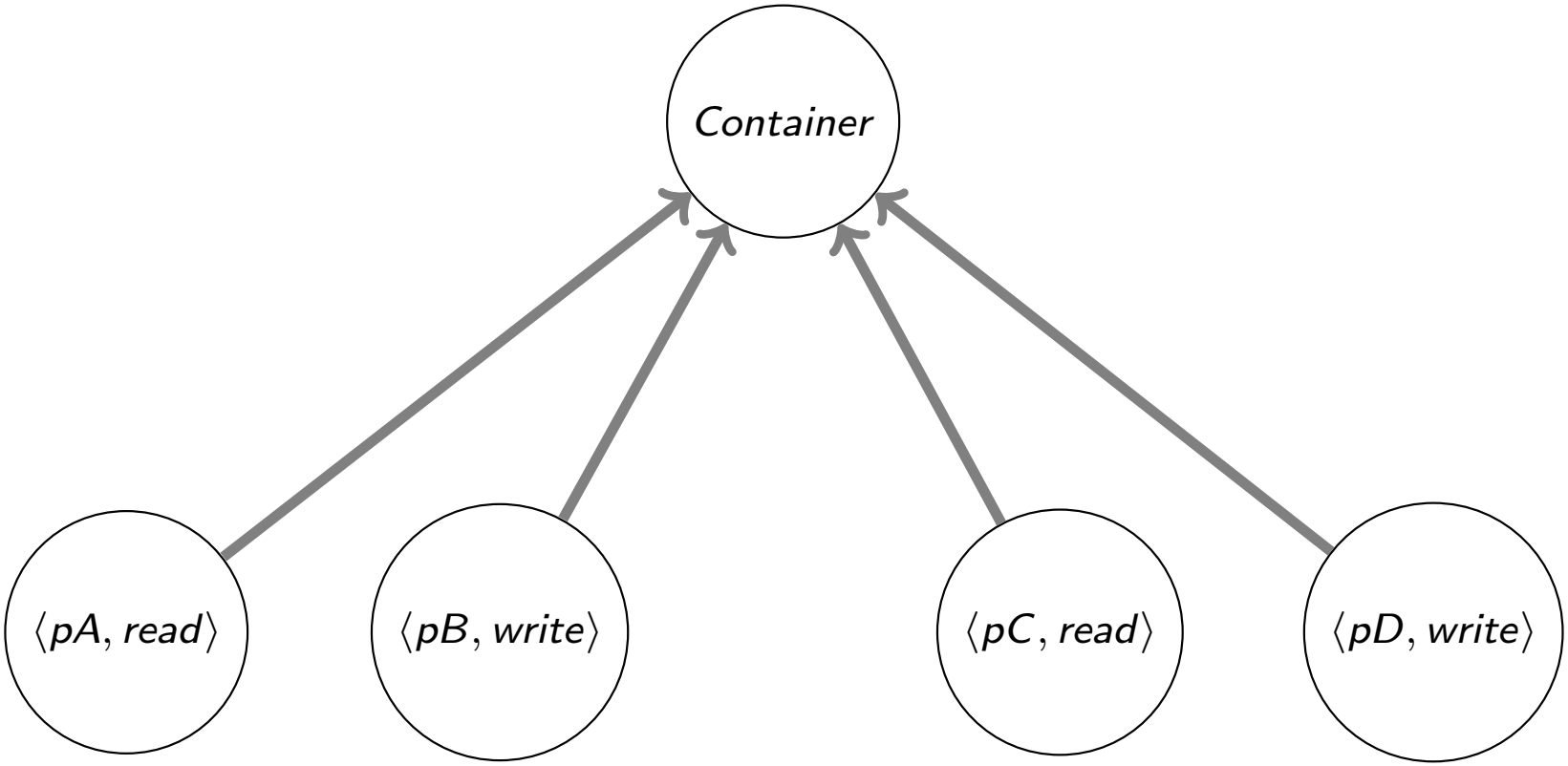
Model generalization: container

A container is typically a directory that holds many "private" folders of different applications. A private folder is a folder that is accessed by a single application only (including all its sub-folders).

Model generalization: container



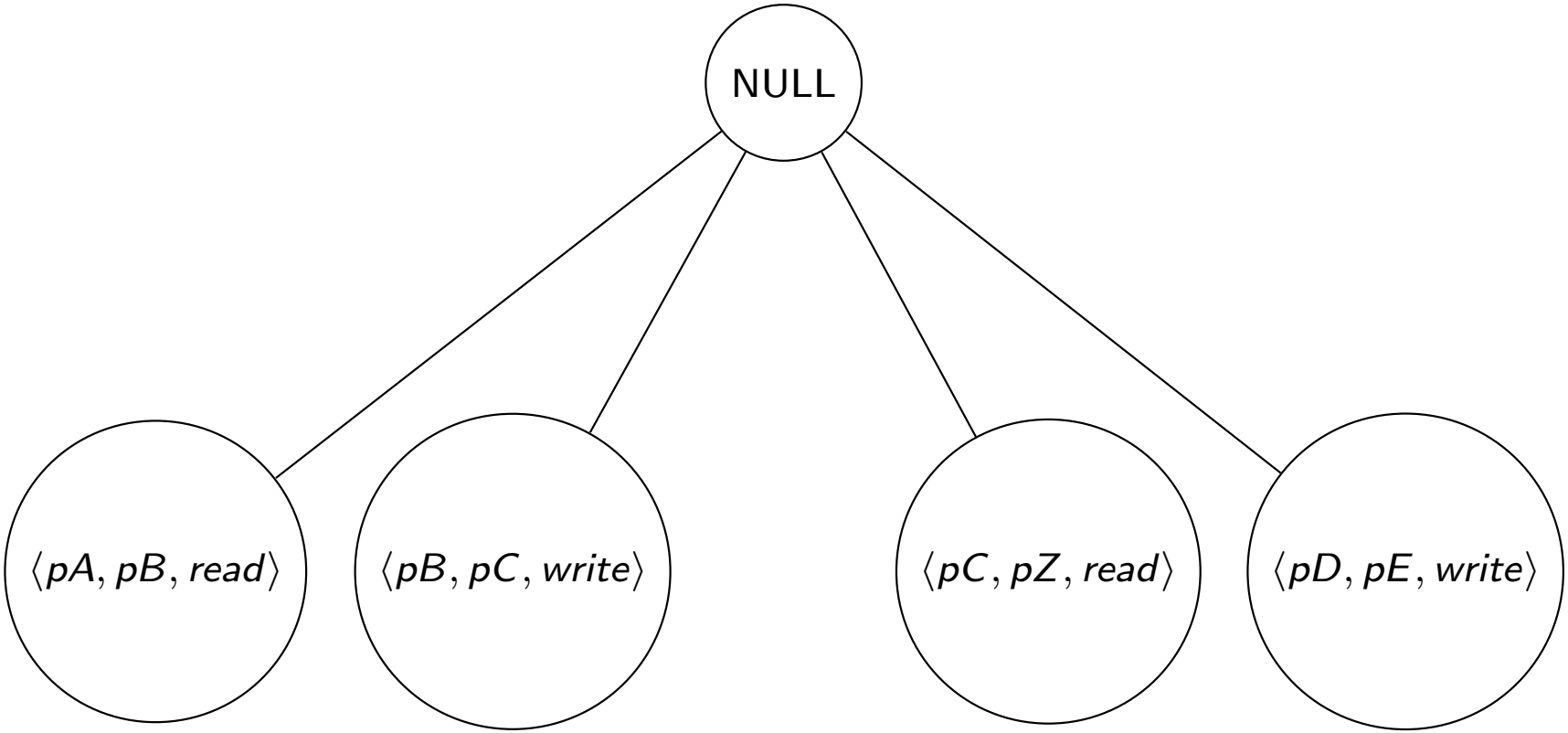
Model generalization: container



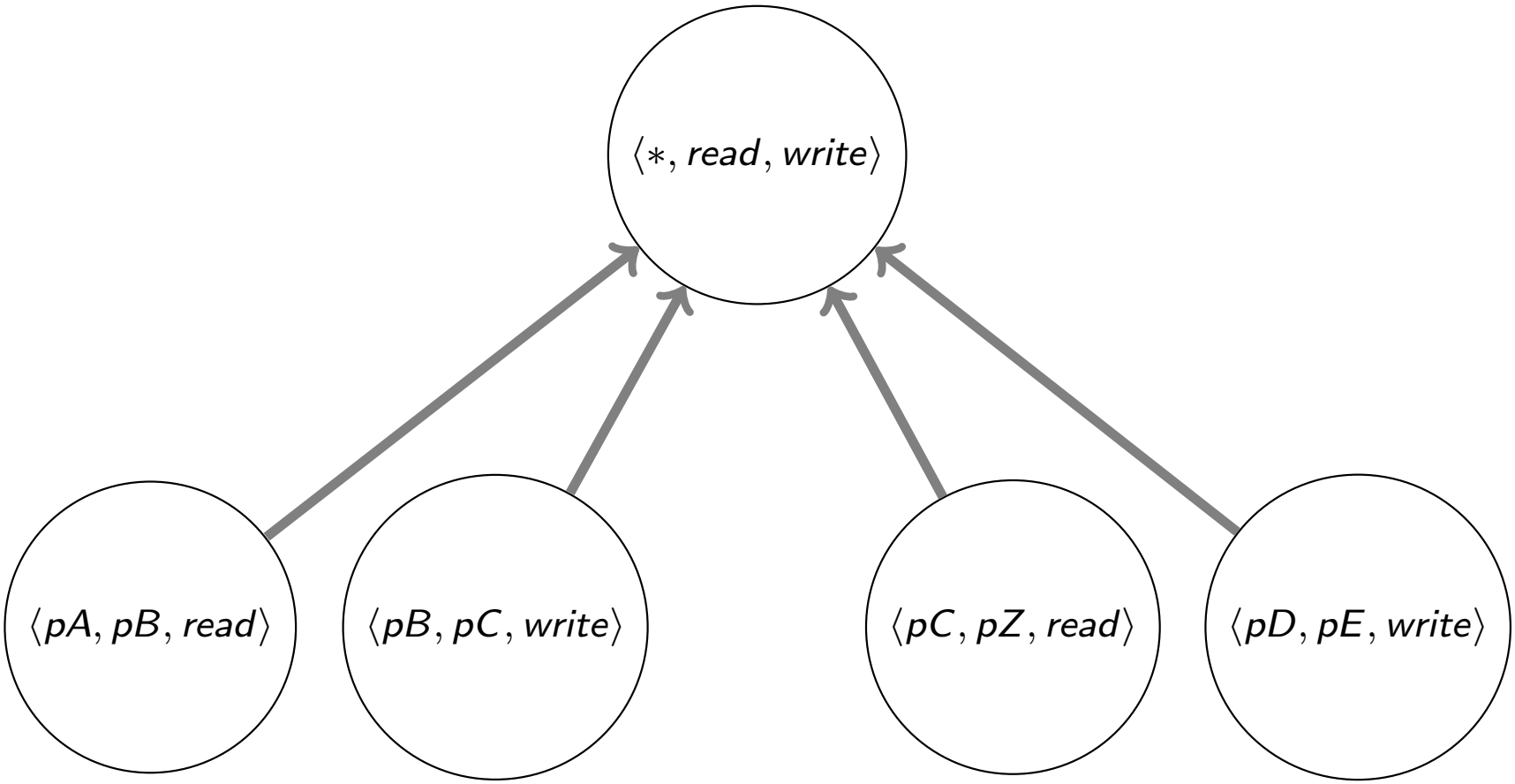
Model generalization: wildcard rule

A wildcard directory is typically a directory that holds many folders that get access from different applications.

Model generalization: wildcard rule



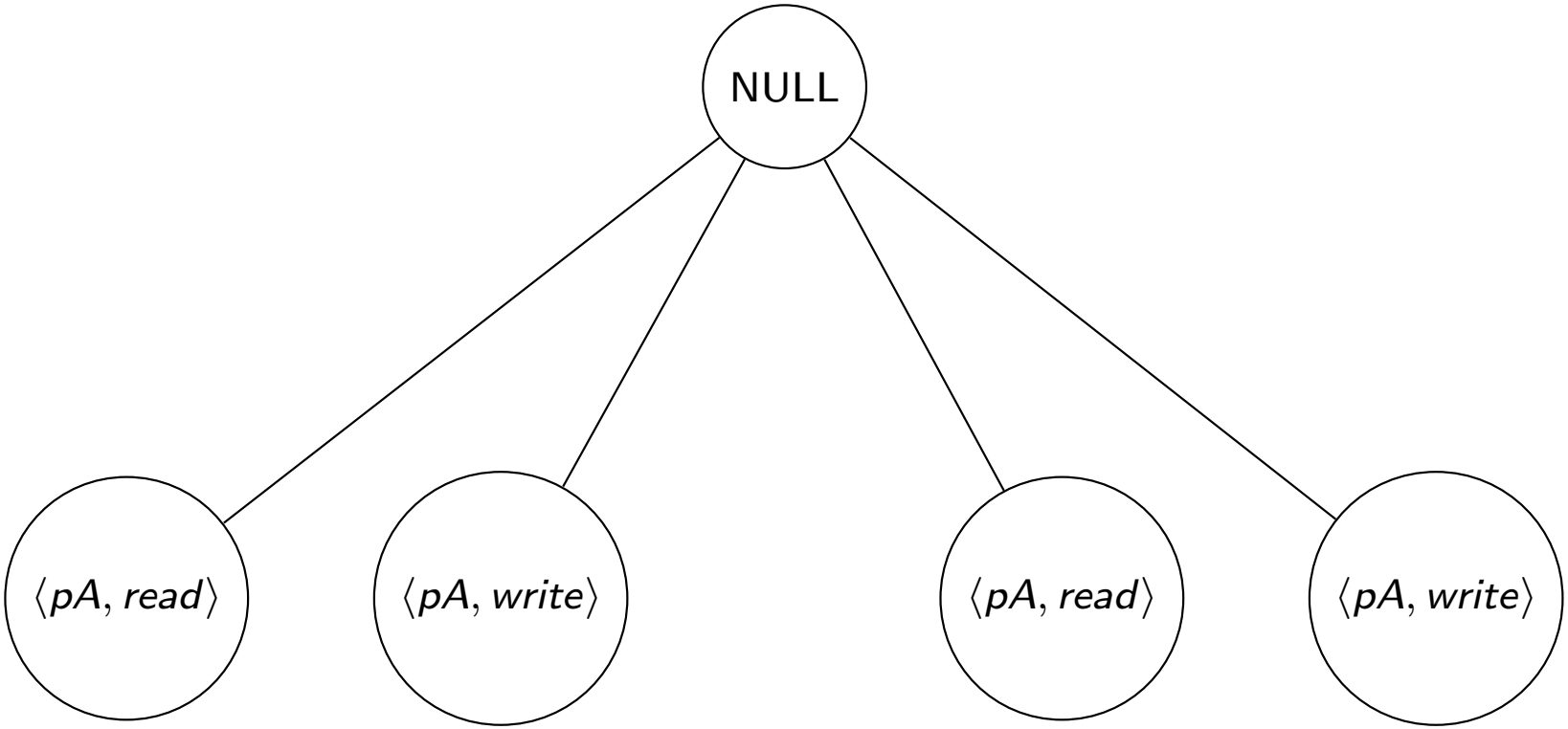
Model generalization: wildcard rule



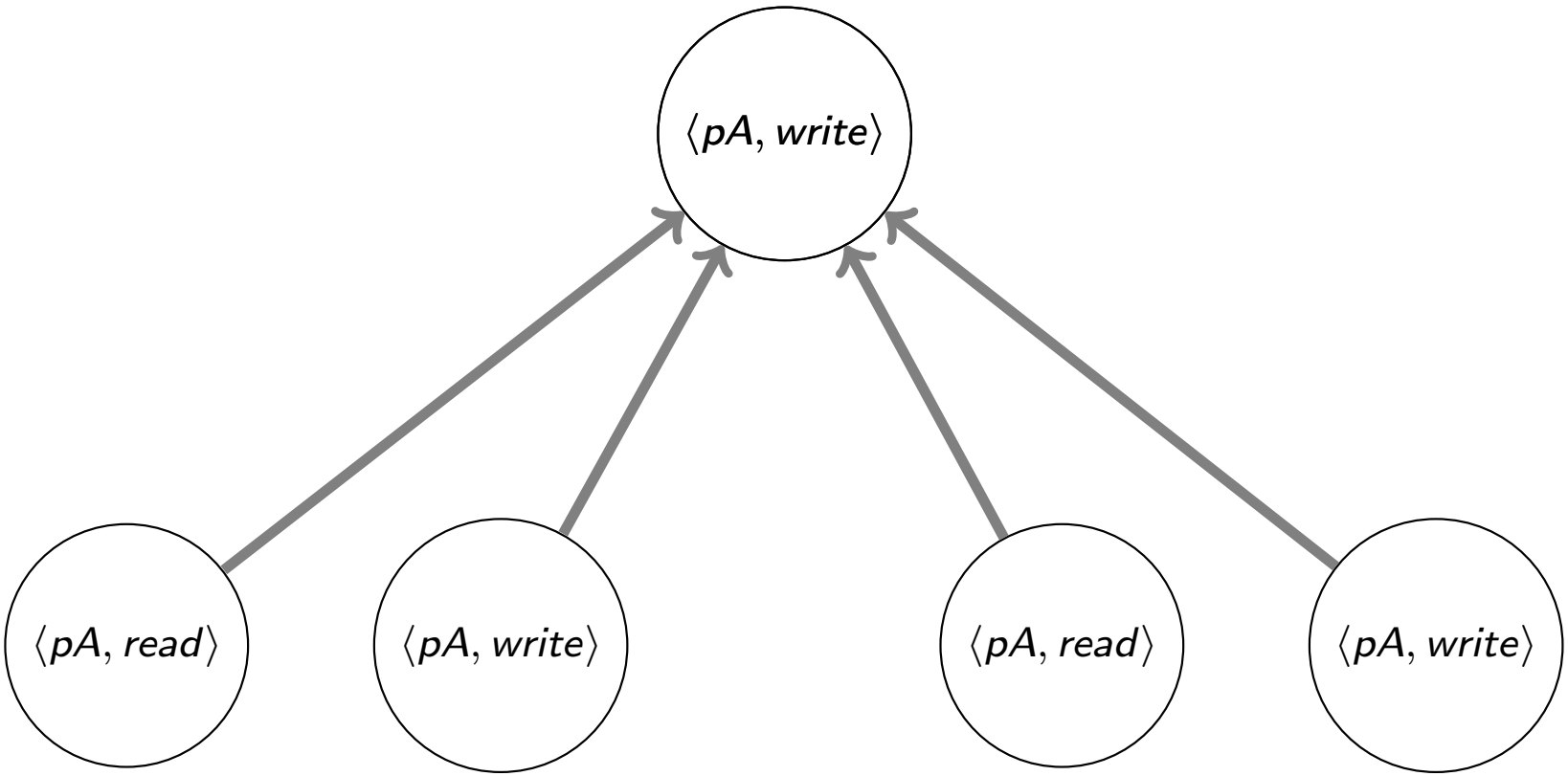
Model generalization: upward propagation

We perform post-order traversal of virtual filesystem and then we apply the **upward propagation**. If we find that all accesses to the sub-folders of one directory were performed by a *single* application `proc`, we add the access token $\langle proc, op \rangle$ to the current label.

Model generalization: upward propagation



Model generalization: upward propagation



Evaluation

- Similar to the analysis for the -n-gram model, we ran ten experiments.We picked one of the machines and we used the other nine to build the model.
- We use the same 10,000 sample of malware that we used for the n-gram analysis.

Malware Detection: Os files resource results

M	*Dr*	*Fp*	*Adr*	*access violations*			*Dw*	*Fd*	
				R	*W*	*E*		*FP*	*Dr*
1	0.656	0.225	0.906	0.000	0.022	0.222	0.864	0.0	0.864
2	0.657	0.173	0.907	0.000	0.011	0.172	0.902	0.0	0.902
3	0.657	0.154	0.907	0.000	0.130	0.043	0.902	0.0	0.902
4	0.657	0.156	0.907	0.024	0.049	0.122	0.902	0.0	0.902
5	0.657	0.143	0.907	0.024	0.024	0.095	0.902	0.0	0.902
6	0.635	0.242	0.877	0.014	0.055	0.242	0.868	0.0	0.868
7	0.657	0.267	0.907	0.020	0.041	0.265	0.901	0.0	0.901
8	0.657	0.045	0.907	0.000	0.045	0.000	0.902	0.0	0.902
9	0.657	0.025	0.907	0.000	0.025	0.000	0.902	0.0	0.902
10	0.657	0.050	0.907	0.000	0.038	0.038	0.902	0.0	0.902
Average	0.655	0.148	0.904	0.008	0.044	0.137	0.895	0.0	0.895

Malware Detection: Os files resource results

M	*Dr*	*Fp*	*Adr*	*access violations*			*Dw*	*Fd*	
				R	*W*	*E*		*FP*	*Dr*
1	0.656	0.225	0.906	0.000	0.022	0.222	0.864	0.0	0.864
2	0.657	0.173	0.907	0.000	0.011	0.172	0.902	0.0	0.902
3	0.657	0.154	0.907	0.000	0.130	0.043	0.902	0.0	0.902
4	0.657	0.156	0.907	0.024	0.049	0.122	0.902	0.0	0.902
5	0.657	0.143	0.907	0.024	0.024	0.095	0.902	0.0	0.902
6	0.635	0.242	0.877	0.014	0.055	0.242	0.868	0.0	0.868
7	0.657	0.267	0.907	0.020	0.041	0.265	0.901	0.0	0.901
8	0.657	0.045	0.907	0.000	0.045	0.000	0.902	0.0	0.902
9	0.657	0.025	0.907	0.000	0.025	0.000	0.902	0.0	0.902
10	0.657	0.050	0.907	0.000	0.038	0.038	0.902	0.0	0.902
Average	0.655	0.148	0.904	0.008	0.044	0.137	0.895	0.0	0.895

Malware Detection: Os files resource results

M	*Dr*	*Fp*	*Adr*	*access violations*			*Dw*	*Fd*	
				R	*W*	*E*		*FP*	*Dr*
1	0.656	0.225	0.906	0.000	0.022	0.222	0.864	0.0	0.864
2	0.657	0.173	0.907	0.000	0.011	0.172	0.902	0.0	0.902
3	0.657	0.154	0.907	0.000	0.130	0.043	0.902	0.0	0.902
4	0.657	0.156	0.907	0.024	0.049	0.122	0.902	0.0	0.902
5	0.657	0.143	0.907	0.024	0.024	0.095	0.902	0.0	0.902
6	0.635	0.242	0.877	0.014	0.055	0.242	0.868	0.0	0.868
7	0.657	0.267	0.907	0.020	0.041	0.265	0.901	0.0	0.901
8	0.657	0.045	0.907	0.000	0.045	0.000	0.902	0.0	0.902
9	0.657	0.025	0.907	0.000	0.025	0.000	0.902	0.0	0.902
10	0.657	0.050	0.907	0.000	0.038	0.038	0.902	0.0	0.902
Average	0.655	0.148	0.904	0.008	0.044	0.137	0.895	0.0	0.895

Malware Detection: Os files resource results

M	*Dr*	*Fp*	*Adr*	*access violations*			*Dw*	*Fd*	
				R	*W*	*E*		*FP*	*Dr*
1	0.656	0.225	0.906	0.000	0.022	0.222	0.864	0.0	0.864
2	0.657	0.173	0.907	0.000	0.011	0.172	0.902	0.0	0.902
3	0.657	0.154	0.907	0.000	0.130	0.043	0.902	0.0	0.902
4	0.657	0.156	0.907	0.024	0.049	0.122	0.902	0.0	0.902
5	0.657	0.143	0.907	0.024	0.024	0.095	0.902	0.0	0.902
6	0.635	0.242	0.877	0.014	0.055	0.242	0.868	0.0	0.868
7	0.657	0.267	0.907	0.020	0.041	0.265	0.901	0.0	0.901
8	0.657	0.045	0.907	0.000	0.045	0.000	0.902	0.0	0.902
9	0.657	0.025	0.907	0.000	0.025	0.000	0.902	0.0	0.902
10	0.657	0.050	0.907	0.000	0.038	0.038	0.902	0.0	0.902
Average	0.655	0.148	0.904	0.008	0.044	0.137	0.895	0.0	0.895

Malware Detection: Os registries resource results

Machine	*Dr*	*Fp*	*WDr*	*WFP*	*Final det. rate*
1	0.567	0.063	0.530	0.063	0.521
2	0.557	0.107	0.540	0.053	0.521
3	0.566	0.179	0.530	0.128	0.062
4	0.557	0.000	0.530	0.000	0.540
5	0.557	0.000	0.530	0.000	0.540
6	0.557	0.015	0.530	0.000	0.540
7	0.597	0.133	0.530	0.000	0.540
8	0.557	0.067	0.530	0.067	0.537
9	0.561	0.100	0.530	0.025	0.521
10	0.557	0.000	0.530	0.000	0.540
Average	0.563	0.066	0.530	0.034	0.486

Table: Detection based on our registry access activity model.

Malware Detection: Conclusion

- We performed a large scale data collection of system calls invoked from a diverse set of benign applications under realistic conditions.
- We analyzed the diversity of the collection of system calls and explored how system-call-based, program centric detectors perform in light of this data.
- We proposed a system-centric approach for malware detection. We demostrated that our model perform characterizes well the operations of benign programs.

Thank you!
Any questions?

Andrea Lanzi
lanzi@eurecom.fr