

The Importance of Realistic Quantitative Studies of Malware Detection



UC SANTA BARBARA
engineering



Northeastern University



The Malware-Detection Models of Today

- 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

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

Behavioral Detection (in Academia)

- “Static-Aware Malware Detection” - USENIX Security 03
 - 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 Sec 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)

Does Larger Datasets Change the Shape of the Problem?

- Outline of this talk:
 1. Dataset collection
 2. Experiment 1: find the most accurate model
 3. Experiment 2: find uniquely benign behaviors

Datasets

Data Collection: *Malware*

- A large number of malware samples are available from many different online collections: Anubis, Malfease, Open Malware/Offensive Computing, etc.

- Malicious samples extracted from Anubis:
 - 6,000 random samples of active malware
 - From all existing malware categories
 - Botnets
 - Worms
 - Trojans
 - Droppers

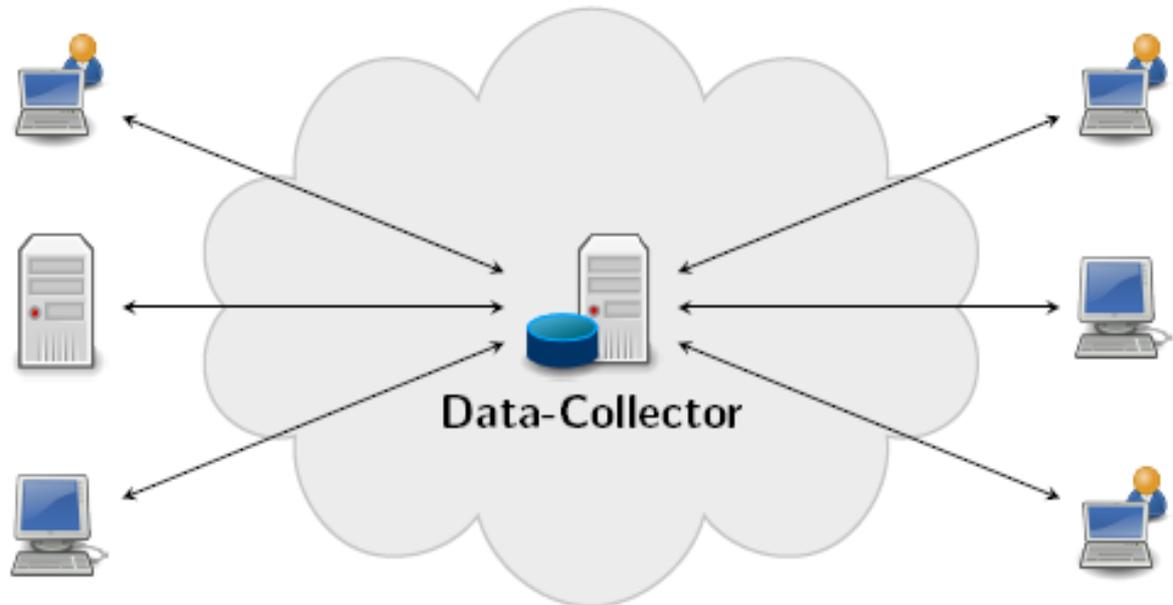
Data Collection: *Benign Programs*

- Challenging problem:
We need information about the normal execution profiles of benign programs.

- Issues:
 - **Privacy**: we need to convince people that their private data are protected.
 - **Diversity**: we need to collect benign data from a different sources: home machines, lab machines, developer machines etc.
 - **Transparency**: the logger should not have a visible performance or safety impact.

Data Collector Infrastructure

- Local, per-host collector intercepts system calls, buffers, and communicates logs to central repository.
- Data collected: $\langle \text{timestamp, program, pid, ppid, system call, args, result} \rangle$



Local Collector

- Kernel-level module collects 79 different system calls in 5 categories:
 - 25 related to files,
 - 23 related to registries,
 - 1 related to networking,
 - 5 related to memory sections.

- User privacy is protected:
 - No actual I/O buffers are logged
 - Resource names are replaced with a random value:
 - Pathnames outside the system path (e.g., C:\Documents and Settings),
 - Registry keys below the user-root registry key (HKLM),
 - IP addresses.



Log collector

Collected Data from Benign Programs

- From 10 real user machines (not under our control) for about a week:

<i>Machine</i>	<i>Usage</i>	<i>Data</i> (GB)	<i>System calls</i> ($\times 10^6$)	<i>Processes</i> ($\times 10^3$)	<i>Applications</i>
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

Normalization Datasets

- We need to eliminate any machine-specific artifacts that may introduce noise.

- 1,200 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'

Datasets

- [malware] 6,000 malware traces from Anubis
(training for malicious behavior)
- [goodware] 180GB of traces collected with our collector
(training for benign behavior and testing for FP)
- [anubis-good] traces from 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)

Experiment 1: What Is the Most Accurate Malware-Detection Model?

Goals

- MAIN GOAL:
Identify automatically the most accurate malware-detection model

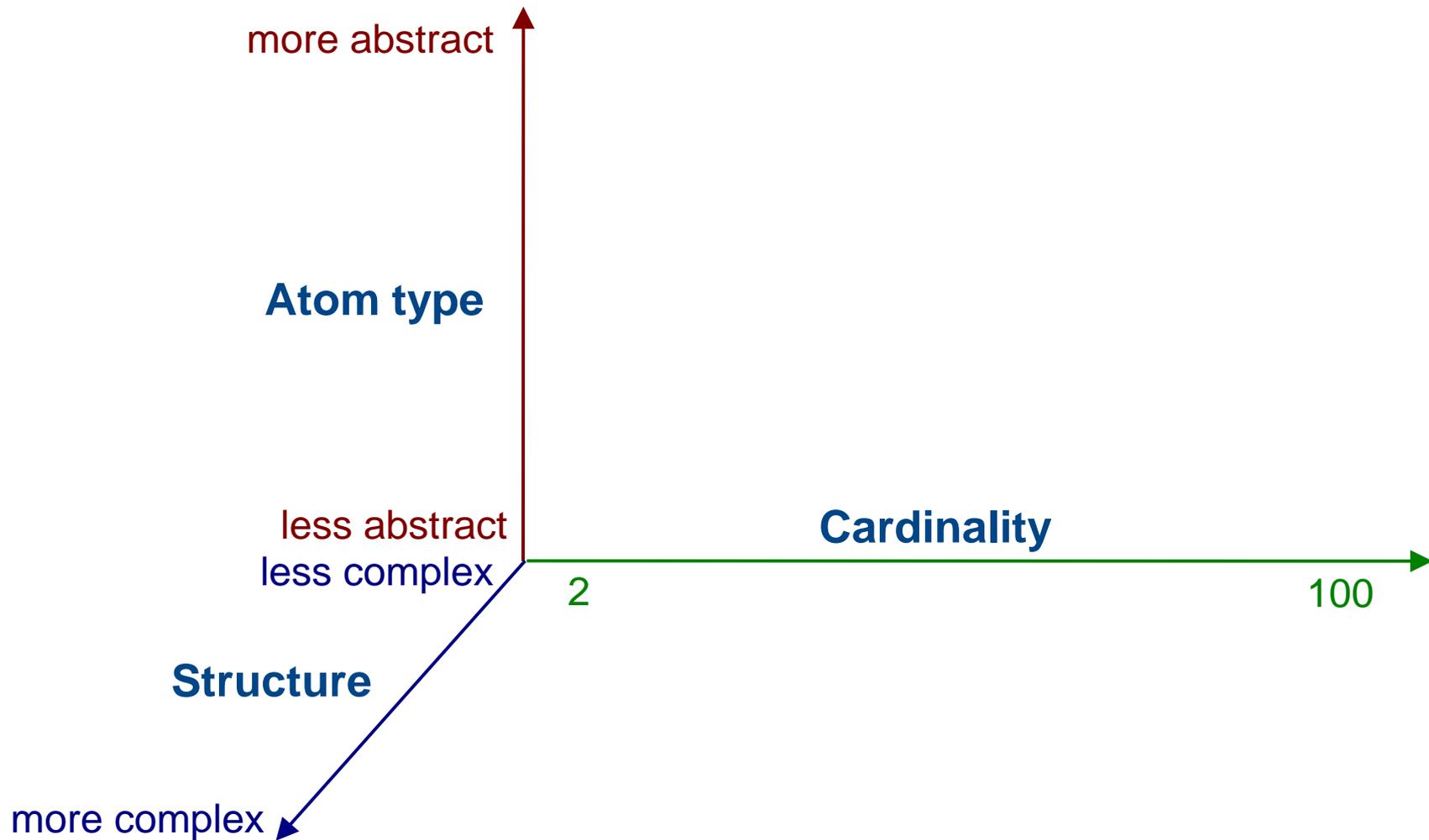
- We need a benchmark for testing behavioral malware detectors:
 - Development of a systematic testing technique to evaluate the quality of behavior-based malware detectors
 - Creation of a comprehensive dataset for validating experiments
 - Evidence that empirical evaluation of malware detection models is a necessary step

- Approach:
Fix a dataset, enumerate detection models, compute accuracy for each model.

Enumerating Malware-Detection Models

- Parameters of interest:
 1. **Atoms** – basic components of the model
 2. **Structure** – relationships between atoms
 3. **Cardinality** – number of atoms in a structure
 4. **Threshold** – number of matched structures needed to trigger an alert

Exploring the Model Space



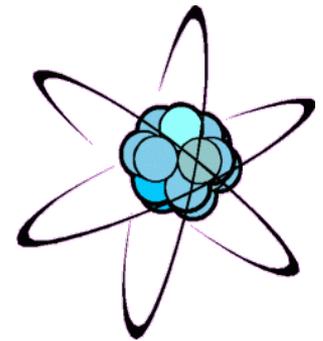
Model Specification: *Atoms*

1. Atoms:

Represent the fundamental behavioral element that appears in a program syscall trace.

- **System call:** NtOpenFile, NtClose, ...
- **Action:** high-level operations (“read file”, ...) → ReadFile, LoadLibrary, ...
- With and without **parameters**

- Limited to what can be collected efficiently at runtime
 - No instruction-level tracking
 - No data-flow / taint information

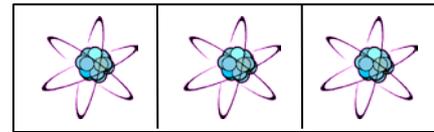


Model Specification: *Structures*

2. Structure:
Describes how the atoms are combined together.

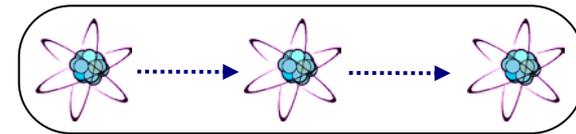
a) **Sequences**

(n-grams)



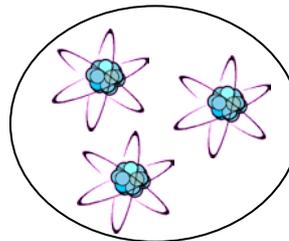
b) **Tuples**

(ordered set)

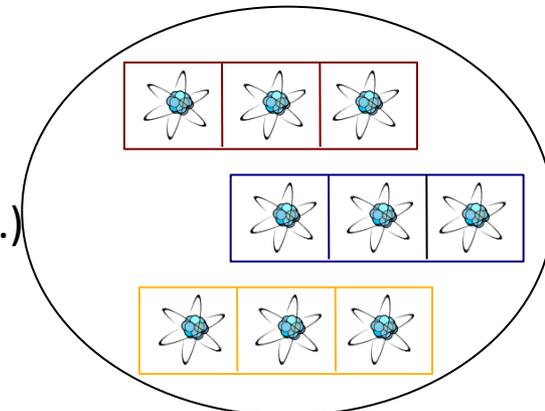


c) **Bags**

(unordered set)



d) **Recursive structures** (bags of n-grams, tuples of n-grams, ...)

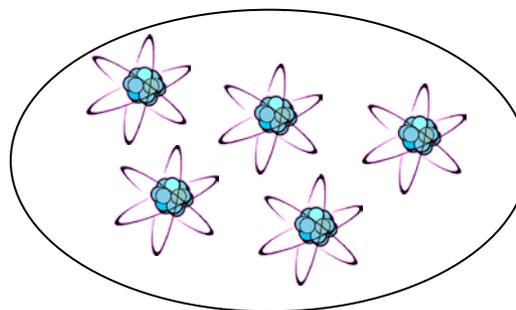
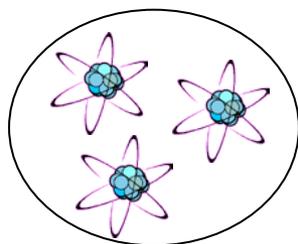


Model Specification: *Cardinality*

3. Cardinality:

Defines how many atoms are included in the structure

- Bounded by the maximum number of atoms in the dataset
- In practice, limited to the range 2-100

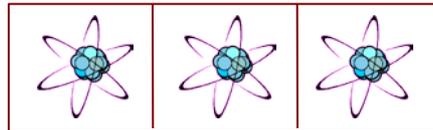
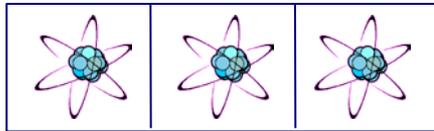


Model Specification: *Alert Threshold*

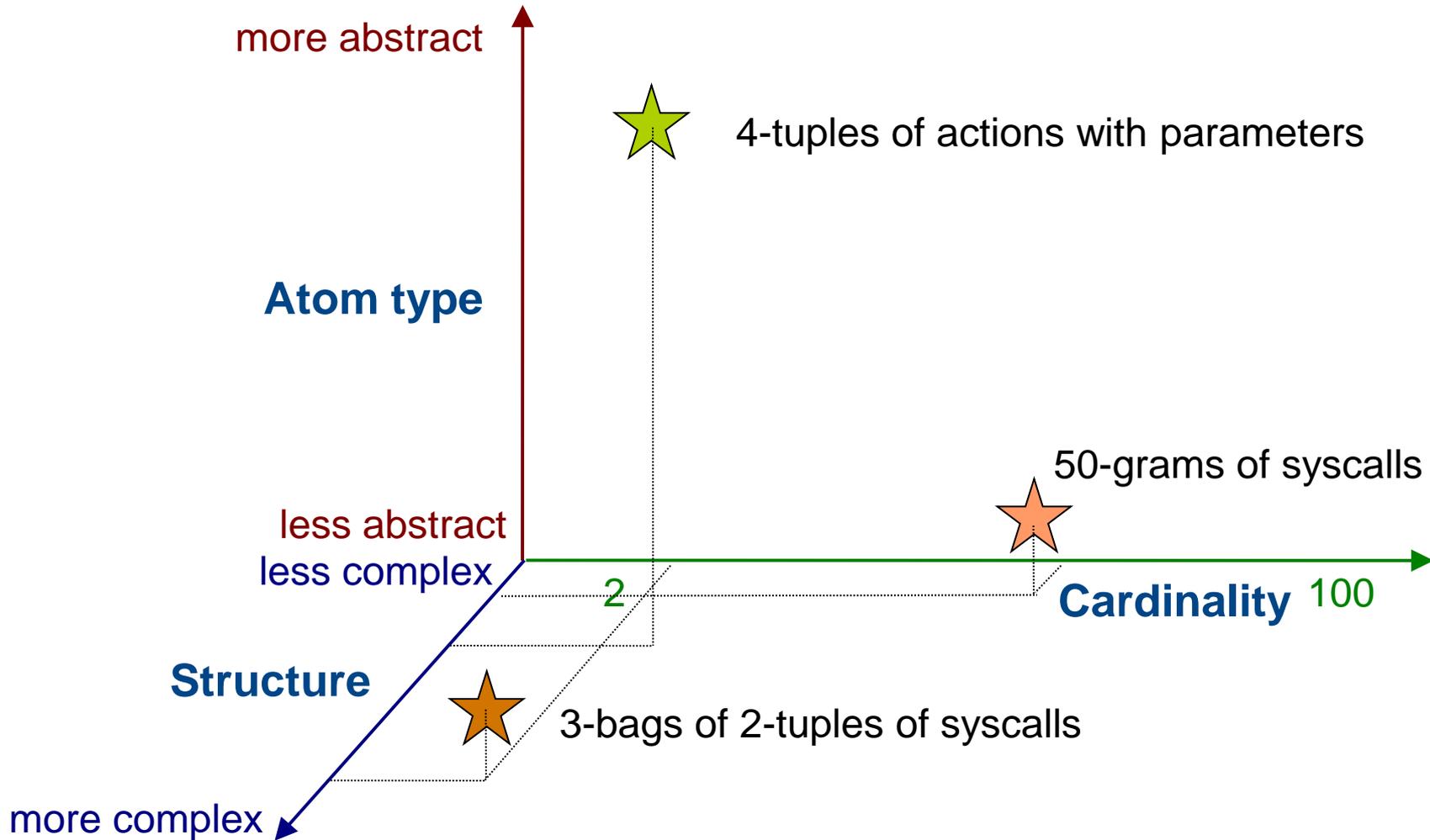
4. Alert Threshold:

How many different signatures must be matched by a program before an alert is raised

- Signatures are matched in no particular order

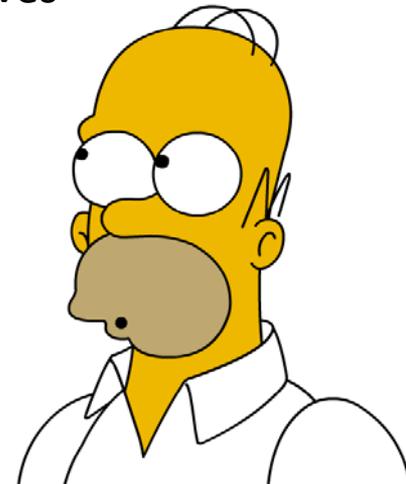


Exploring the Model Space



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
 - Going from 2-grams to 3-grams *should* generate fewer false positives
 - Going from 3-grams to 3-bags *should* generate more false positives



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}

(same for 3-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)

Signatures:

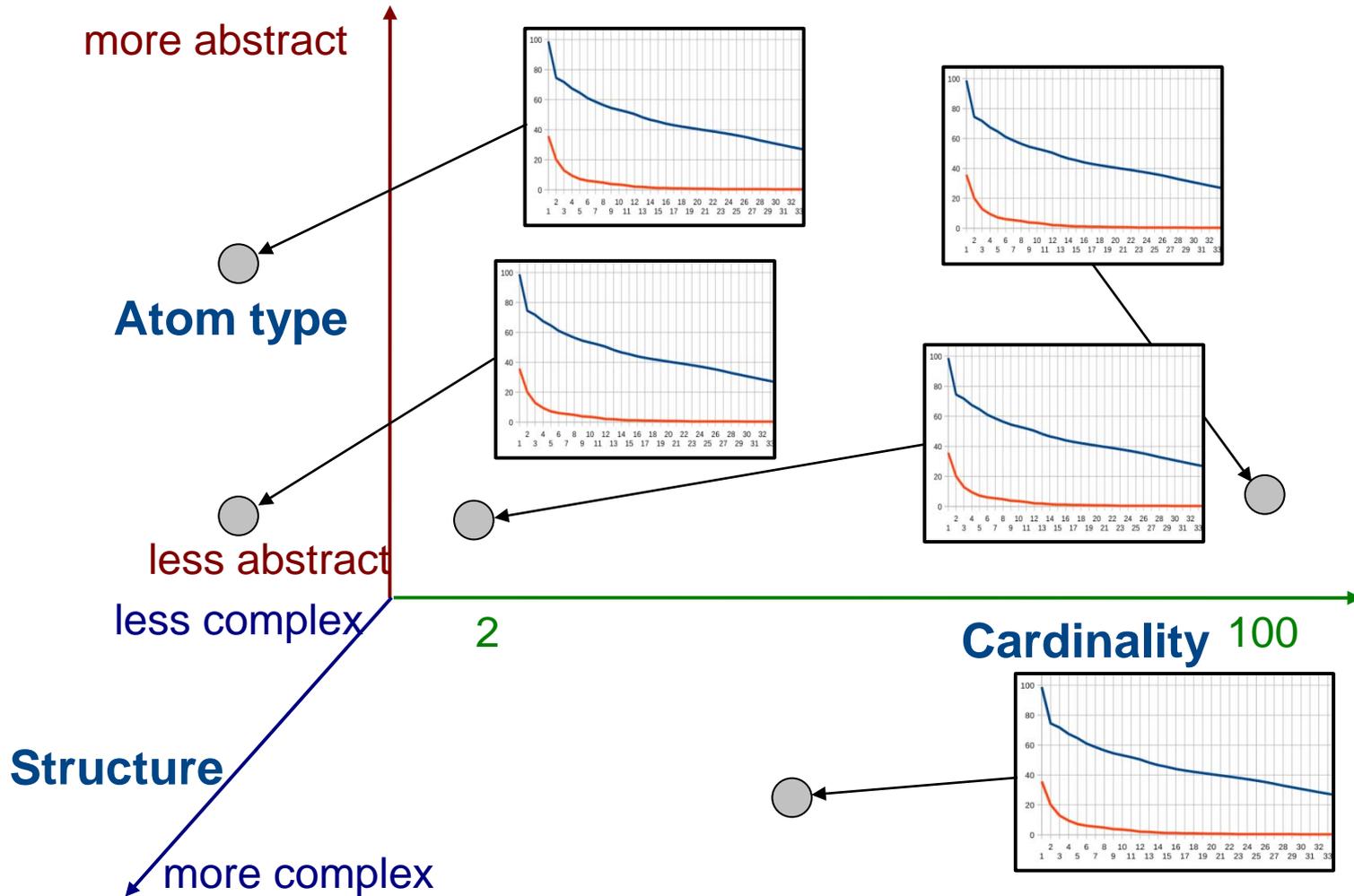
2-grams: [a3,a4] [a4,a5]

3-grams: [a1,a2,a3] [a2,a3,a4] [a3,a4,a5]

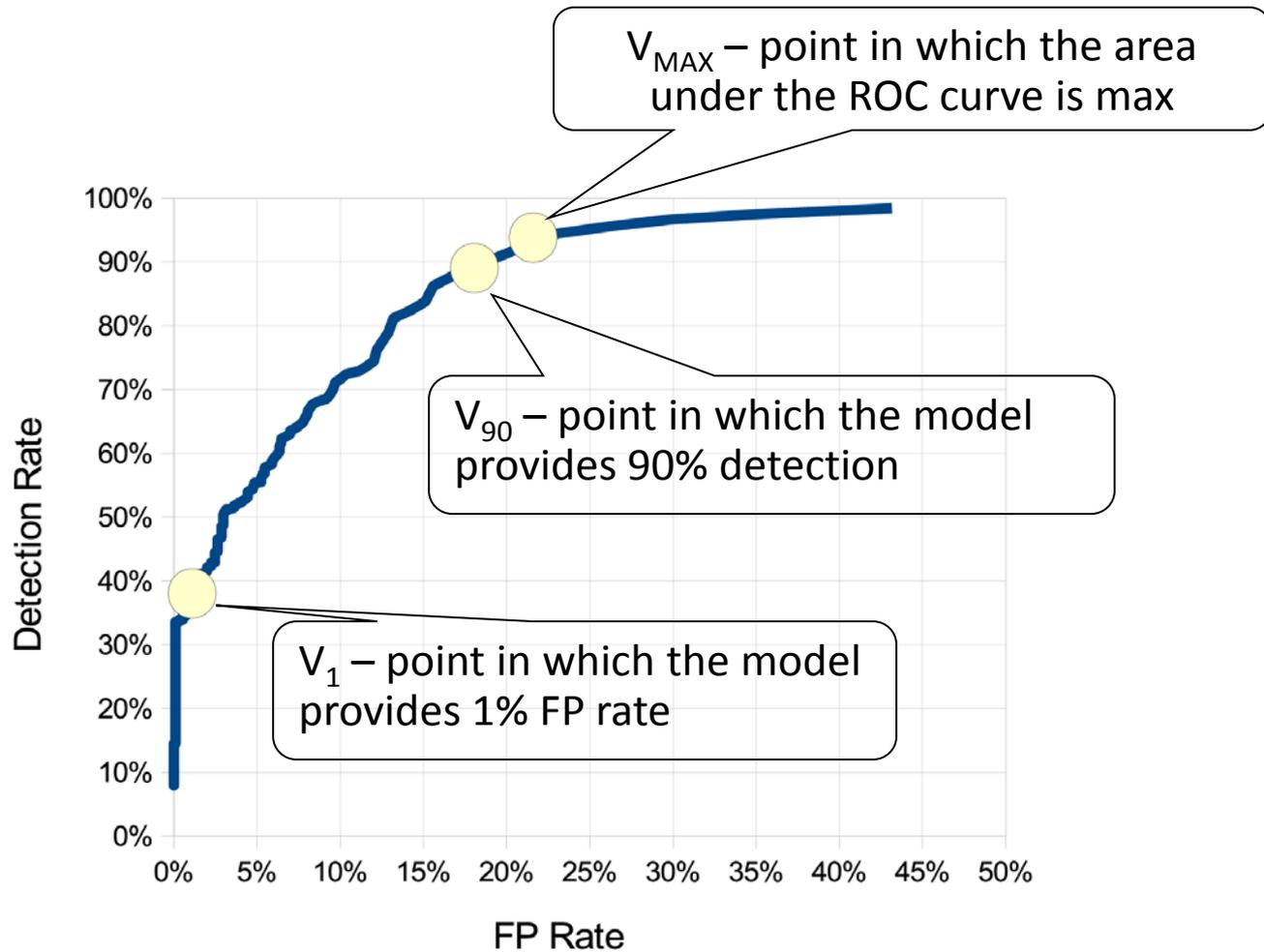
k-bags: none



What Happens If We Move Along the Axes?



Key Indicators to Compare Models



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 VMAX dropped below 0.2
- **215 different classes of detection models** analyzed
- More than **220 million models** generated

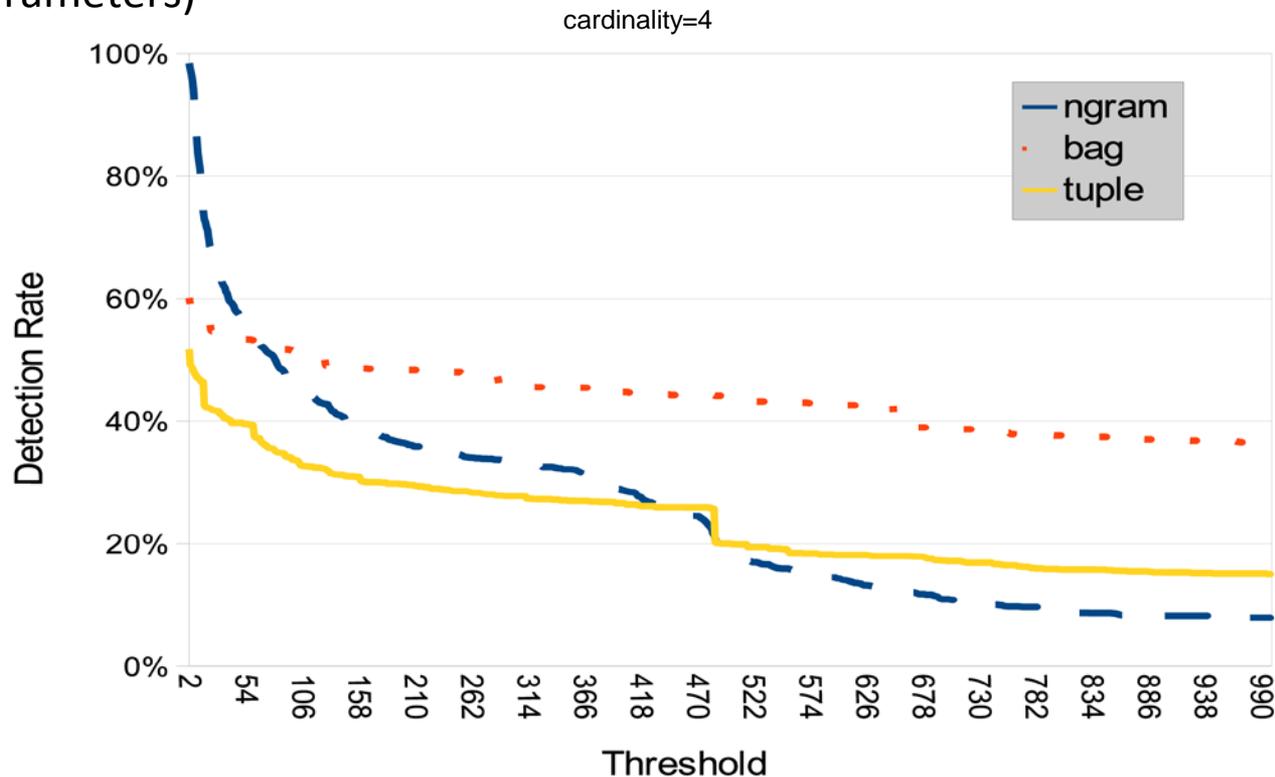
General Results

- 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)

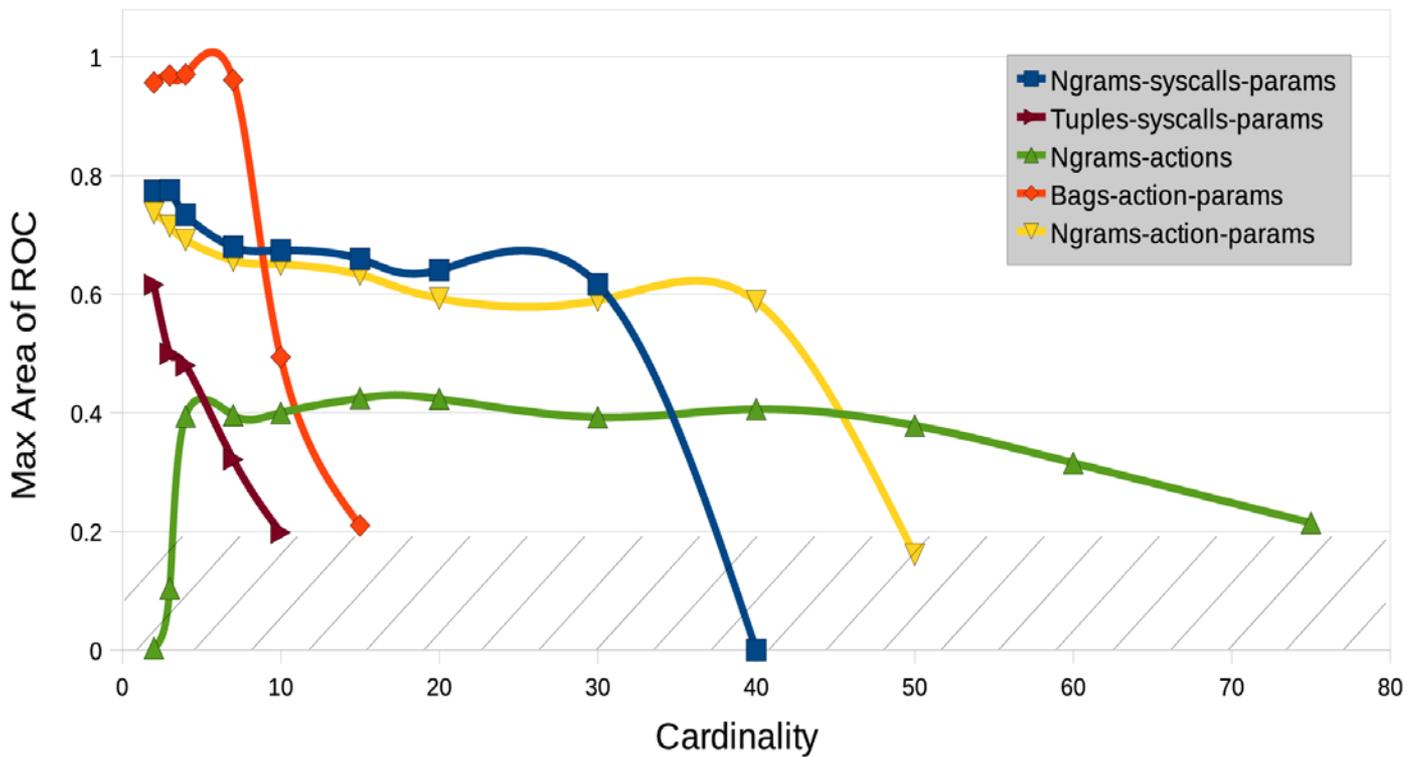
Impact of Matching Threshold

- 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

Conclusions

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

- Empirical testing is crucial
 - 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!

Experiment 2: Are Benign Programs Behaviorally Similar?

Common Benign Behaviors

- The intuition is that benign programs in general follow certain ways in which they use the OS resources.
- To capture normal interactions with the file system and the Windows registry, we propose an **access activity model**.

Access Activity Model

- Each resource that appears in a collected trace receives an access label L , which is a set of access tokens:

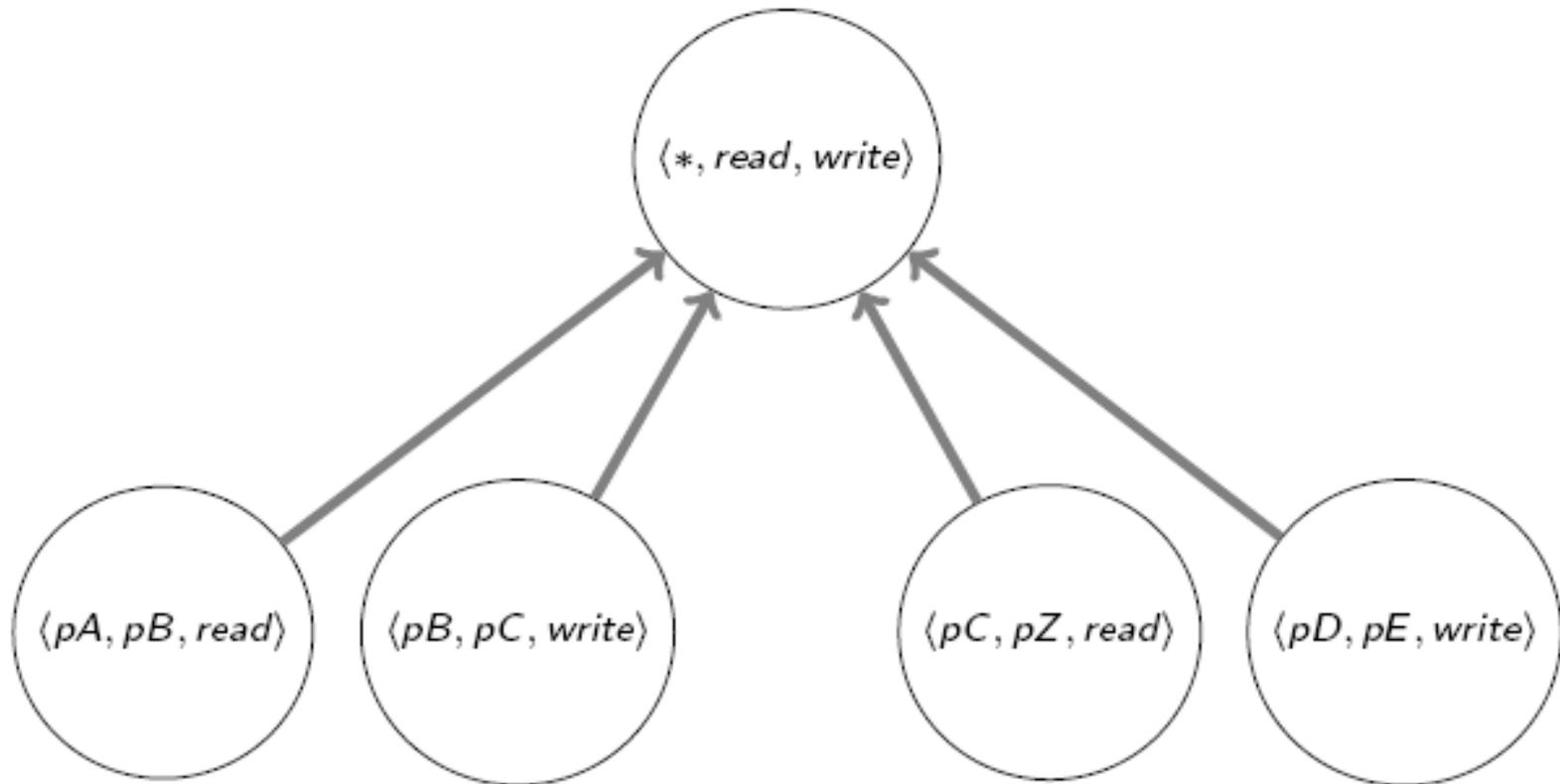
$$L = \{ t_1, t_2, \dots, t_n \}$$

Each token t_i is a application-operation pair:

$$t_i = \langle app, op \rangle$$

- Operations:
 - read, write, and execute for file-system directories, and
 - read and write for registry keys.
- Resources are organized into a virtual resource hierarchy (e.g., a filesystem tree):
 - Generalization rules propagate labels to intermediate directories
 - Container directories, temporary directories automatically identified

A Simple Example Access Activity Model



Malware Detection Using the Access Activity Model

- Build model from traces of benign programs:
 - Need many programs, executed in diverse environments, by different users
- Then check the execution of any suspicious program against this access activity model:
 - Each node in the virtual resource hierarchy is a rule specifying which programs have what kind of access to that resource
- Any violations are indicative of malware.
- On our test set: **89% detection rate with 0% false positives.**

Conclusions

- It is **crucial to have diverse and large datasets** for both benign programs and malware.
- Large datasets produce **qualitatively different** detection results over small datasets.
 - The **accuracy of many classes of detection models is non-linear**.
- Analysis of large traces of benign programs indicates that **common benign behavior exist** (in how the OS resources are used).

References:

- AccessMiner: Using System-centric Models for Malware Protection. A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda. CCS 2010.
- A Quantitative Study of Accuracy in System Call-Based Malware Detection. D. Canali, A. Lanzi, D. Balzarotti, C. Kruegel, M. Christodorescu, and E. Kirda. ISSTA 2012.

Questions?

The Importance of Realistic Quantitative Studies of Malware Detection



UC SANTA BARBARA
engineering



Northeastern University

