You Are What You Do: Hunting Stealthy Malware via Data Provenance Analysis

Qi Wang, Wajih Ul Hassan, Ding Li, Kangkook Jee, Xiao Yu, Kexuan Zou, Junghwan Rhee, Zhengzhang Chen, Wei Cheng, Carl A. Gunter, Haifeng Chen

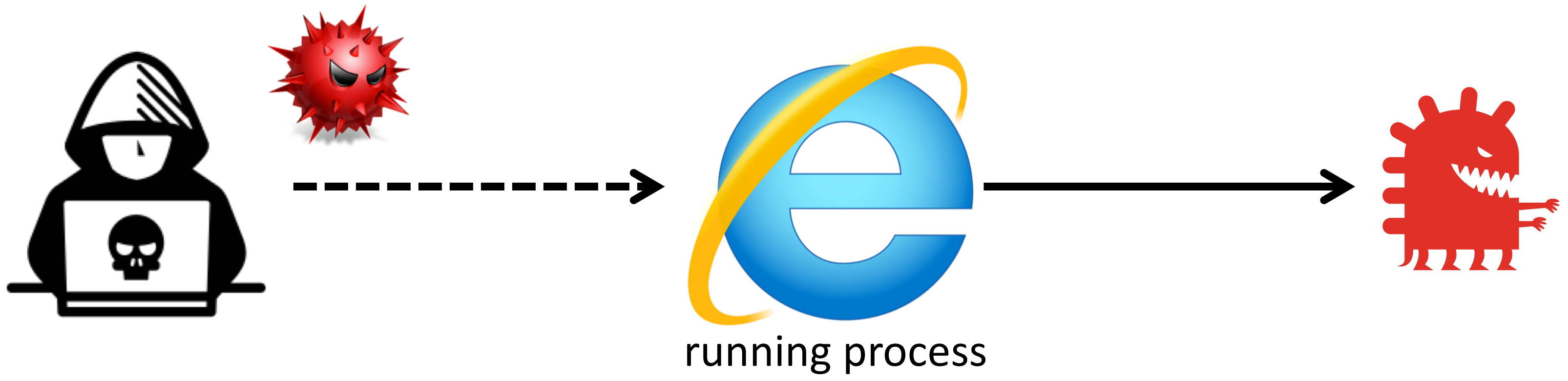
NDSS 2020

Feb 26th, 2020, San Diego

Malware is Becoming Stealthier

- As malware detection has greatly advanced, adversaries are increasingly focusing on new techniques to evade detection.

- One recent line of stealthy attacks achieve their attack goals by impersonating or abusing well-trusted programs (e.g., IE, Java).



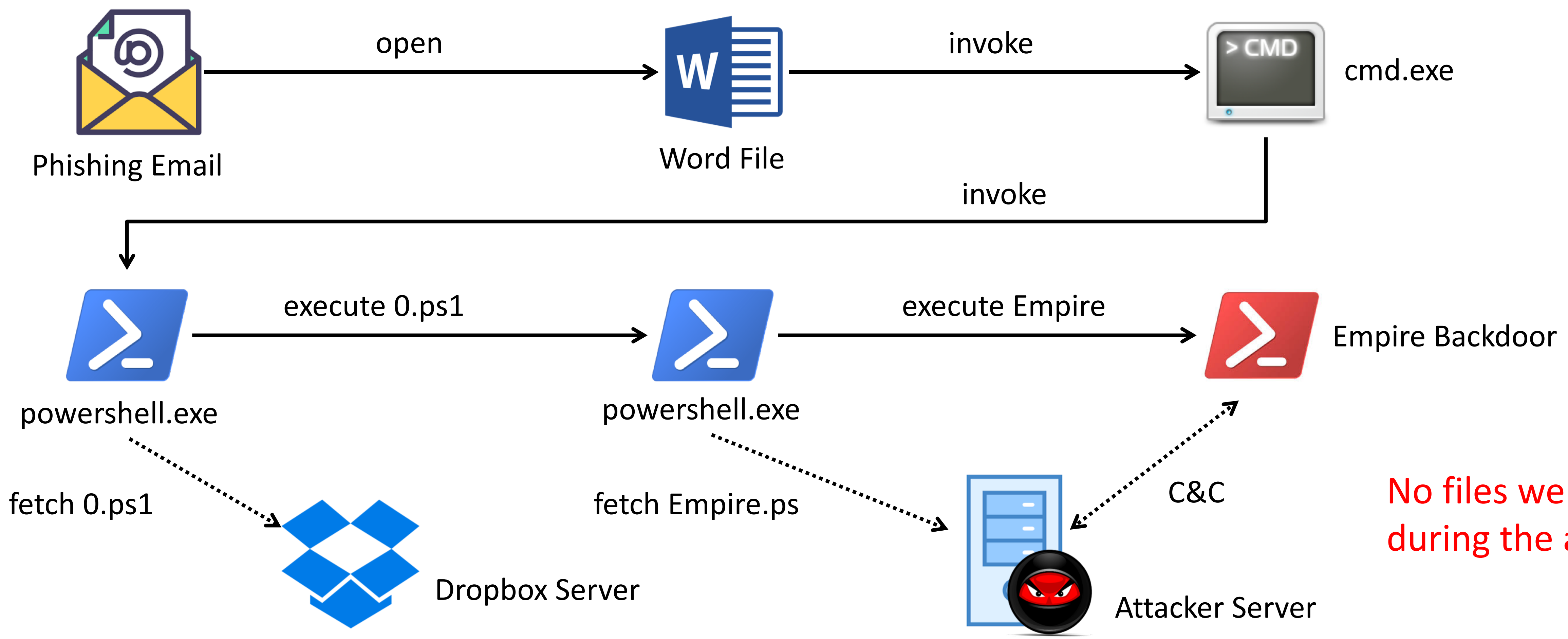
The malicious behavior is blended with benign behaviors of IE.

Stealthy Malware/Attacks

- Advanced stealthy techniques are being actively developed.
  - Memory code injection
    - E.g., reflective DLL injection, process hollowing
  - Script-based attacks
    - Embedding payload in documents like MS Word and Excel
  - Vulnerability exploits
    - E.g., CVE- 2019-0541 allows arbitrary code execution in IE

- Various stealthy strategies are being employed.
  - Fileless techniques (i.e., minimizing the usage of regular file systems)
  - Living off the land (i.e., using dual-use tools such as certutil)

A Real-world Stealthy Attack



No files were created during the attack!

Technical reports estimated that stealthy attacks grew by 265% in the first half of 2019, and are 10 times more likely to succeed compared to traditional attacks!

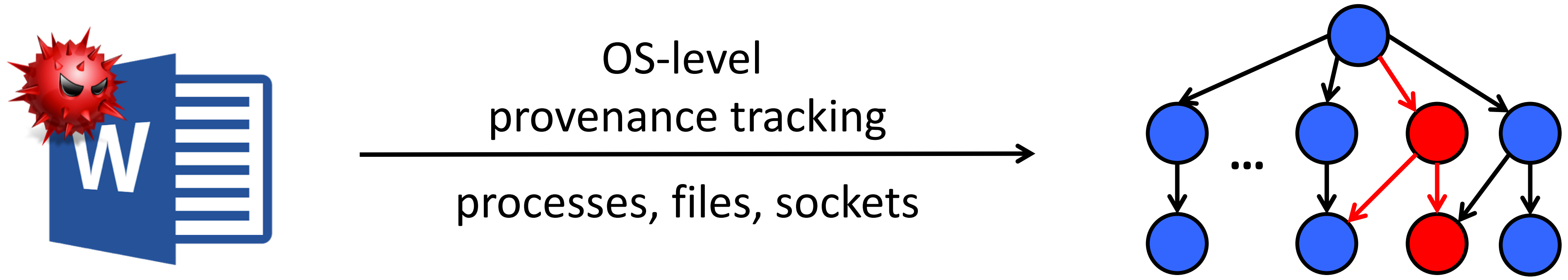
Challenges for Detecting Stealthy Attacks

▪ Taking advantages of well-trusted programs in the system. – Living off the land	$\dashrightarrow$	Could bypass whitelisting.
▪ Residing in the victim process's memory. – Being fileless	$\dashrightarrow$	Signature-based or file-based solutions are ineffective.
▪ There are a variety of stealthy techniques.	$\dashrightarrow$	Solutions target certain techniques do not work for others.

A general and effective approach to detect stealthy attacks is needed!

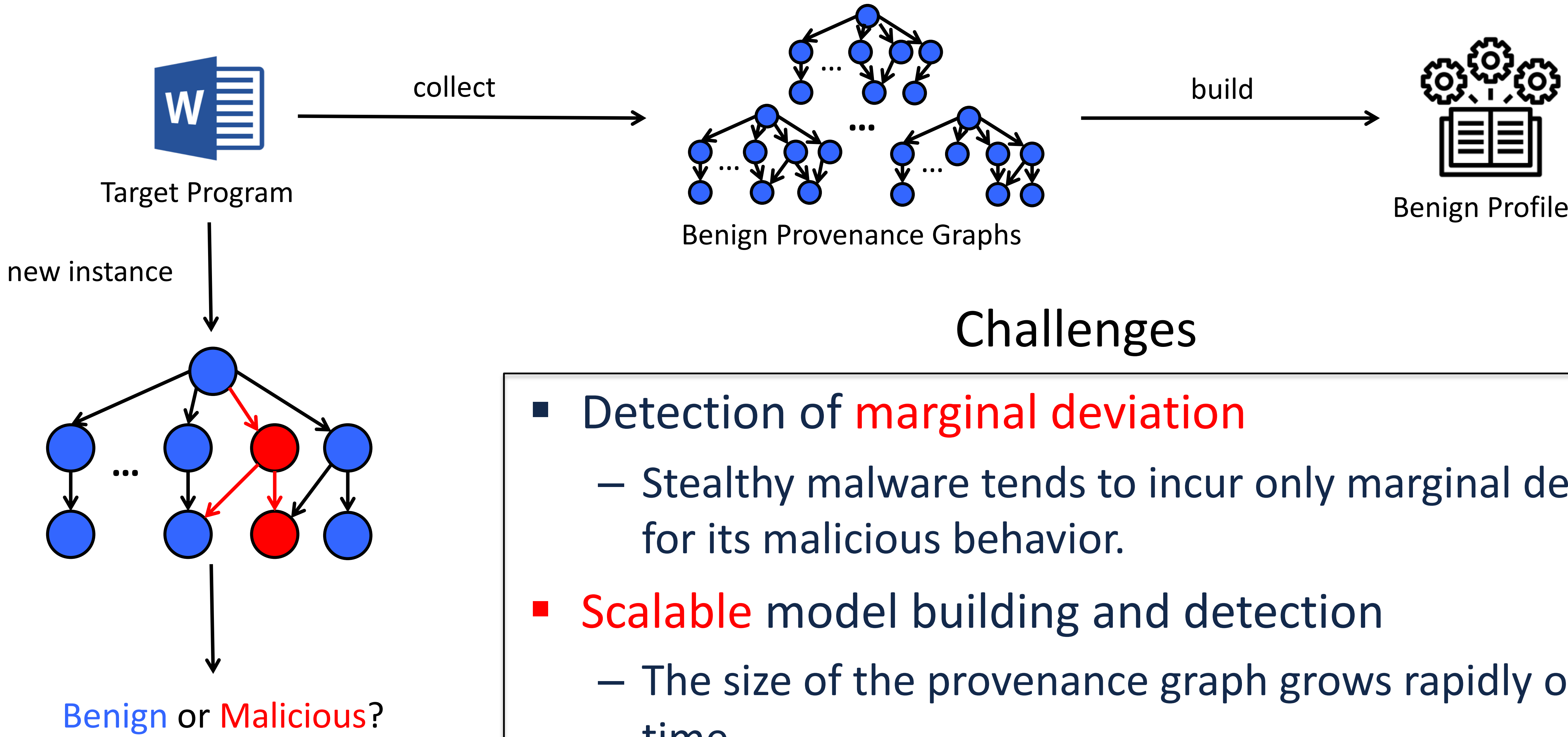
Our Insights

- While a stealthy malware could employ different techniques to impersonate benign processes, its malicious behavior will inevitably interact with the underlying operating systems and leave traces.



- Thus, we could use OS-level provenance analysis to differentiate benign and hijacked (malicious) processes.
  – We consider three types of system entities: processes, files and sockets.

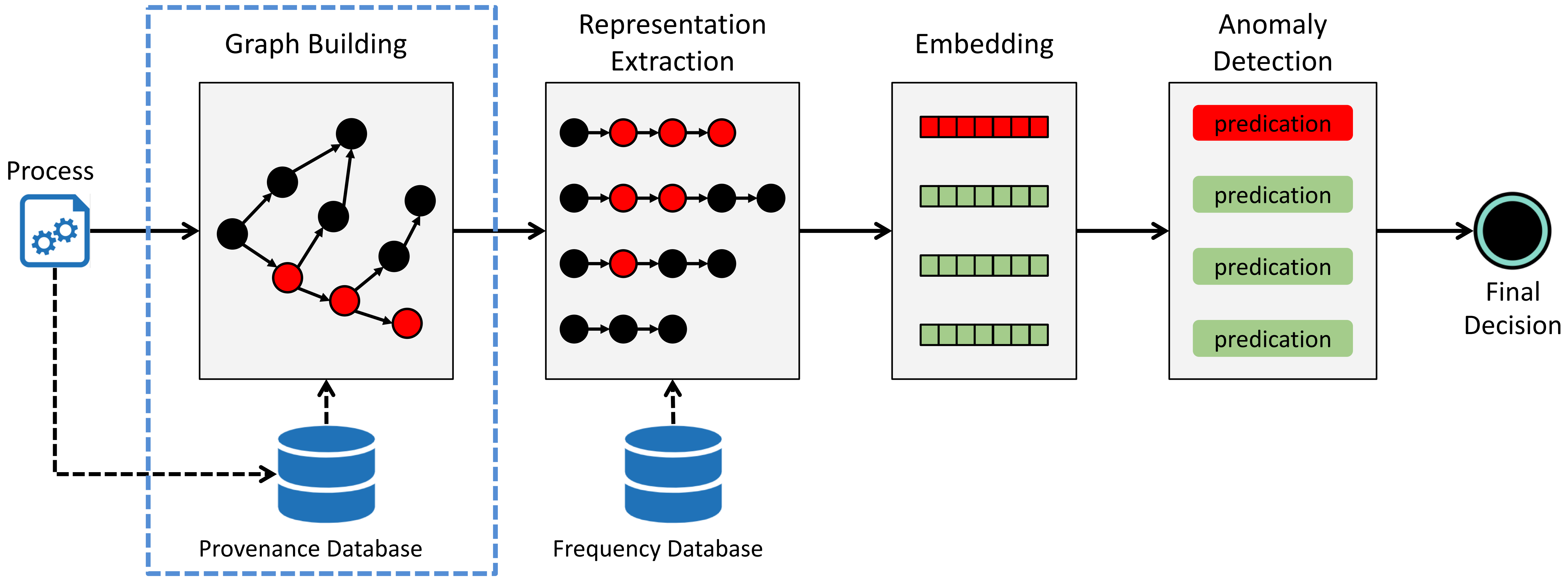
Problem Solved?



Challenges

- Detection of marginal deviation
  - Stealthy malware tends to incur only marginal deviation for its malicious behavior.
- Scalable model building and detection
  - The size of the provenance graph grows rapidly over time.

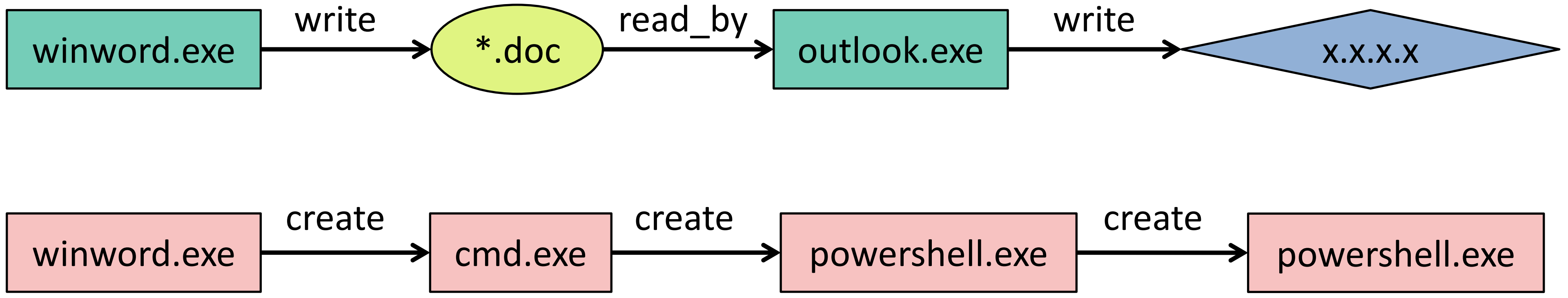
ProvDetector

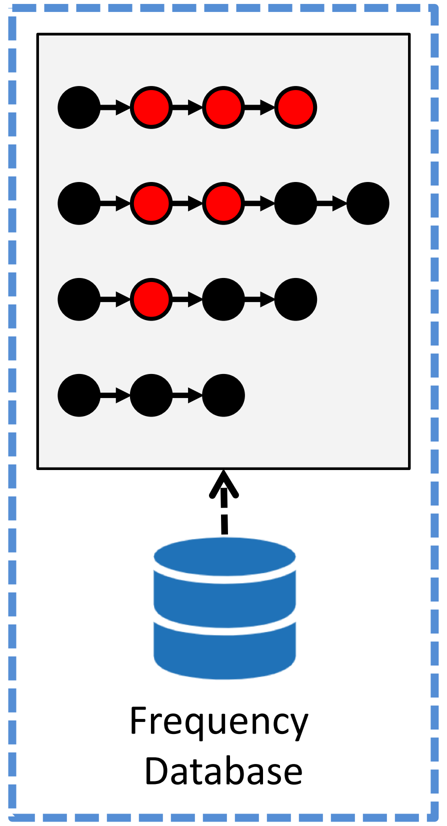


Representation Extraction

- We propose to use causal paths as the features for a provenance graph.
  - The marginal malicious paths are blended with normal paths.

- How to choose the malicious paths?
  - Rare paths are more likely to be malicious.





Rareness-based Path Selection

- We use *regularity score* to define the rareness of a path.
  - For a path $\lambda = \{e_1, e_2, \ldots, e_n\}$re $e = \{src \to dst\}$ularity score is: $R(\lambda) = \prod_{i=1}^{n} R(e_i)$

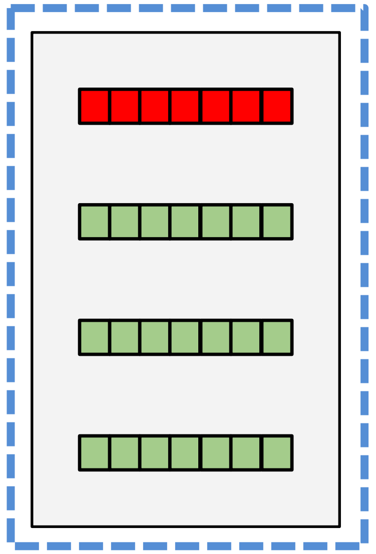
$$R(e) = \underbrace{OUT(src)}_{\text{Out stability}} \underbrace{\frac{|H(e)|}{|H|}}_{\text{Event frequency}} \underbrace{IN(dst)}_{\text{In stability}}$$

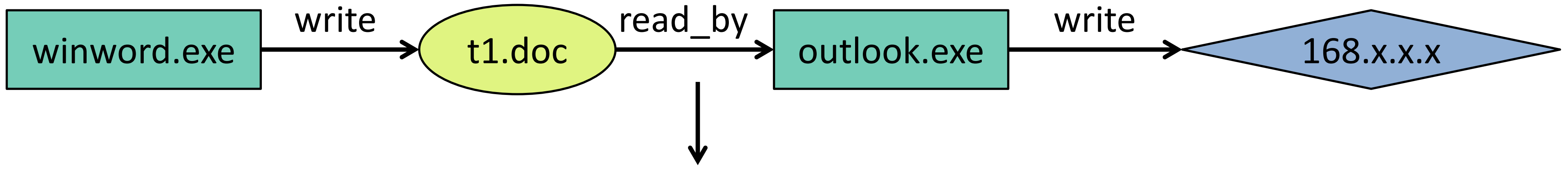
The less frequent and less stable an event is, the less regularity score it has.

Finding paths with the lowest regularity scores from a provenance graph.

Embedding

- How to feed the paths to anomaly detection models?
  - The lengths of causal paths are not fixed.
  - The attributes of nodes are unstructured data (e.g., file names).
- Projecting paths into numerical vector space.
  - We view a causal path as a sentence/document.
    - Each node is treated as a "noun" and each edge is treated as a "verb".
    - Embed the "sentence" into vector using doc2vec.





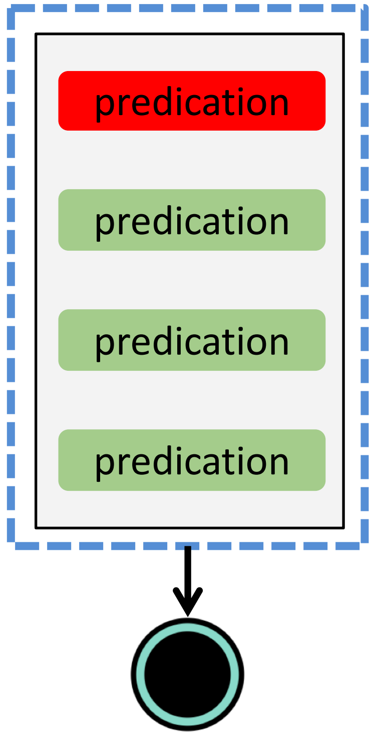
Process:winword.exe write File:t1.doc read by Process:outlook.exe write Socket:168.x.x.x.

In vector space, similar paths are closer while different paths are far away.

Anomaly Detection

- We use a novelty detection model to determine if a path is abnormal.
  - We train the model with only the embeddings of benign paths.
  - It is able to detect unknown attacks or zero-day attacks.

- We then use a threshold-based method to make the final decision.
  - If more than n path vectors are predicted as malicious, we treat the provenance graph as malicious.



Evaluation

- Provenance dataset preparation
  - Malicious dataset
    - We ran about 15,000 malware samples from VirusShare and VirusSign.
  - Benign dataset
    - We deployed ProvDetector in an enterprise with 306 Windows hosts for 3 months.

- We identified 23 target programs in both datasets.
  - Popular applications
    - E.g., IE Browser and Microsoft Word.
  - Preinstalled system tools
    - E.g., Windows Common Line (cmd) and Windows Certificate Services Tool (certutil)

How effective is ProvDetector in detecting stealthy malware?

Detection Accuracy

- We evaluate with the 23 target programs.
  - For each program, we chose 250 benign and 50 malicious processes.
    - 200 benign process were used for training.
    - 50 benign and 50 malicious processes were used for evaluation.
  - For each process, we select the top 20 rarest paths from its provenance graph.

Threshold	Precision	Recall	F1-Score
3	0.957	1.000	0.978
4	0.995	1.000	0.997

Why the Whole Graph is not an effective feature?

Comparison with Graph Embedding

- A graph embedding approach
  - Embedding a provenance graph into a vector using graph2vec.

Approach	Precision	Recall	F1-Score
ProvDetector	0.957	1.000	0.978
graph2vec	0.899	0.452	0.601

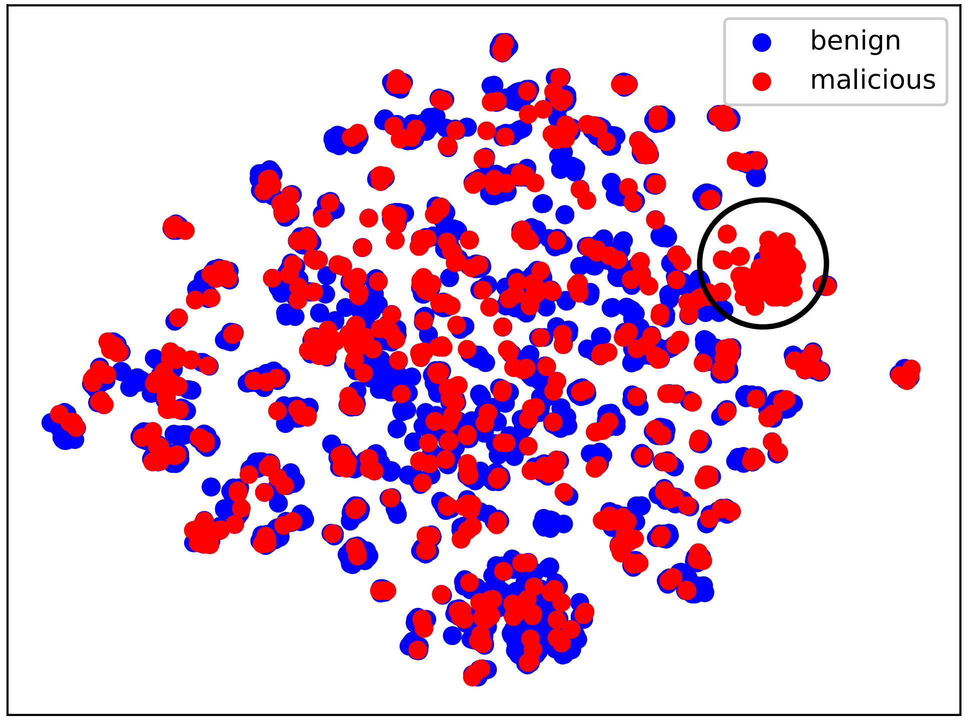
The whole graph is not an effective feature for detecting stealthy attacks!

Why Using Paths are More Effective?

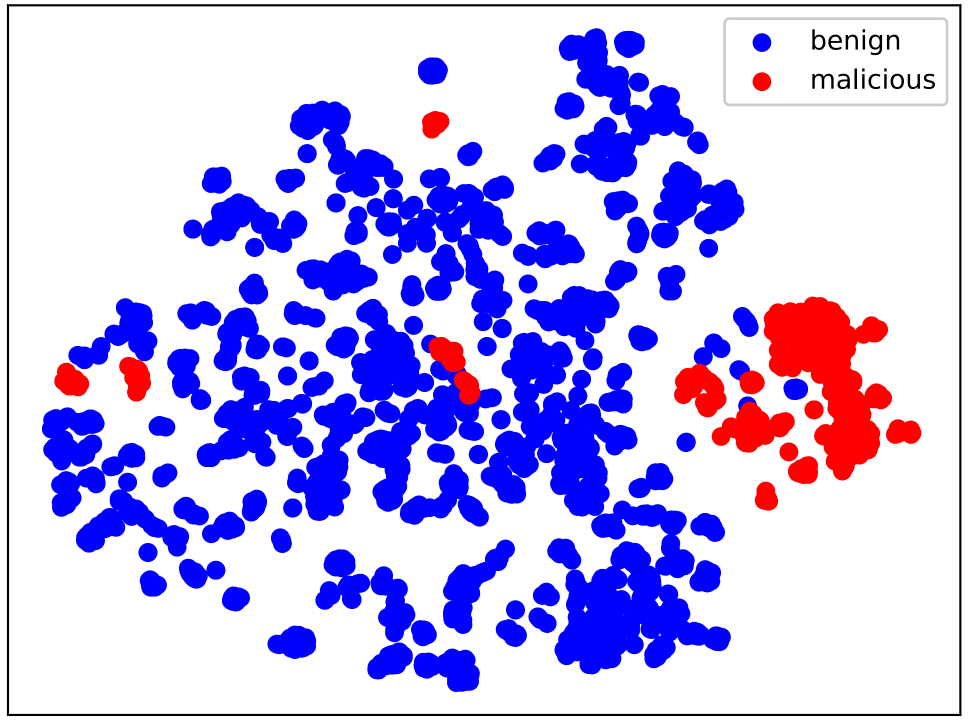
- We use MS word as an example (50 benign and 50 malicious)

We randomly selected 20 paths from benign and malicious graphs.

We selected top 20 rarest paths from benign and malicious graphs.



t-SNE plot of random paths



t-SNE plot of selected paths

Summary

- OS-level data provenance could capture the malicious behavior of stealthy attacks.
- We propose a rareness-based path selection algorithm to identify the potentially malicious part as detection features.
- We present ProvDetector, a provenance-based approach to automatically detect stealthy attacks.
- We demonstrate its effectives through a systematic evaluation in an enterprise environment.

Thanks!

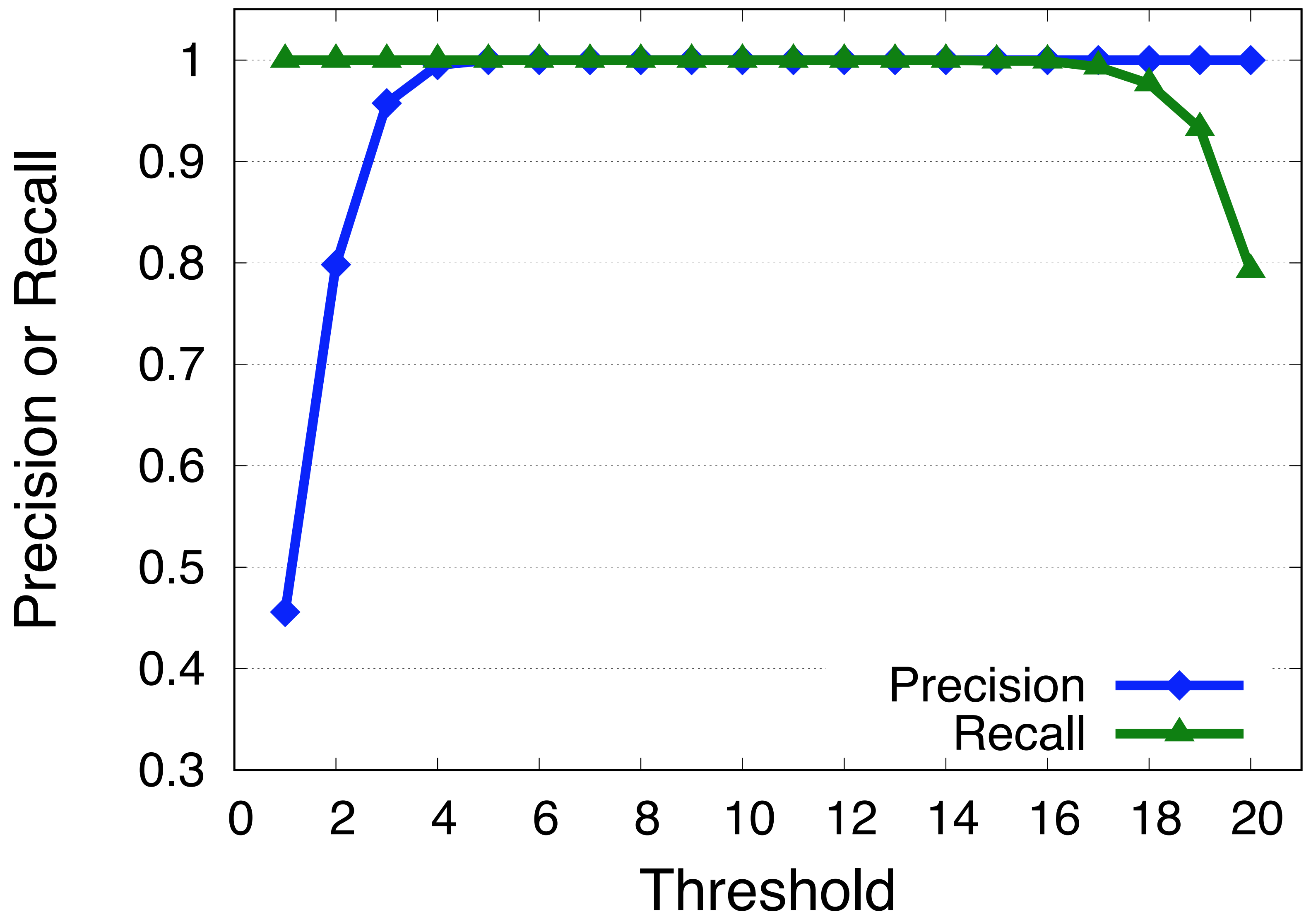
Q&A

Backup Slides

Implementation

- We implement ProvDetector for both Windows and Linux.
  - Provenance tracking is implemented with Windows ETW framework and the Linux Audit framework.
  - The provenance graph builder and the representation extractor are implemented using about 15K lines of Java code.
  - Embedding and anomaly detection are implemented in Python.
- Provenance Data Preprocessing
  - Path Abstraction
    - We remove user specific details from process entities and file entities.
    - E.g., *:/USERS/*/DESKTOP/PAPER.DOC
  - Socket Connection Abstraction
    - We remove the source part of an outgoing connection and the destination part of an incoming connection.

Graph-level Detection Accuracy



Runtime Performance

- Training overhead
  - One-time effort

- Detection overhead
  - Building provenance graphs and path selection. (7s)
  - Embedding the selected paths. (20ms)
  - Prediction overhead of the anomaly detection model. (1.2ms)

For an enterprise which has 100 hosts and there are 30 programs to monitor, it will take 5.7 hours per day to check all the created instances in the enterprise.

Mimicry Attacks

- Editing distance between causal paths
  - We define the editing distance between two causal paths as the minimum number of actions needed to convert one path to another.
    - Add, modify and delete
  - On average, the editing distance between malicious paths and benign paths is about five.

- Since a causal path embeds the contextual causality among different system entities (e.g., processes), it is much harder to evade ProvDetector than the approaches that focus only on the behavior of one process.

Anti-analysis Malware

- A lot of today's malware has anti-analysis capabilities.
  - E.g., anti-VM or anti-debug
- 289 (26%) of the malware samples in our evaluation are identified as anti-VM by VirusTotal.
- 238 (20.7%) of them are identified to be anti-debug by VirusTotal.
- Unlike virtualization based solutions, ProvDetector is designed to run on bare metal machines and does not require isolated environments.