Binary Code Search

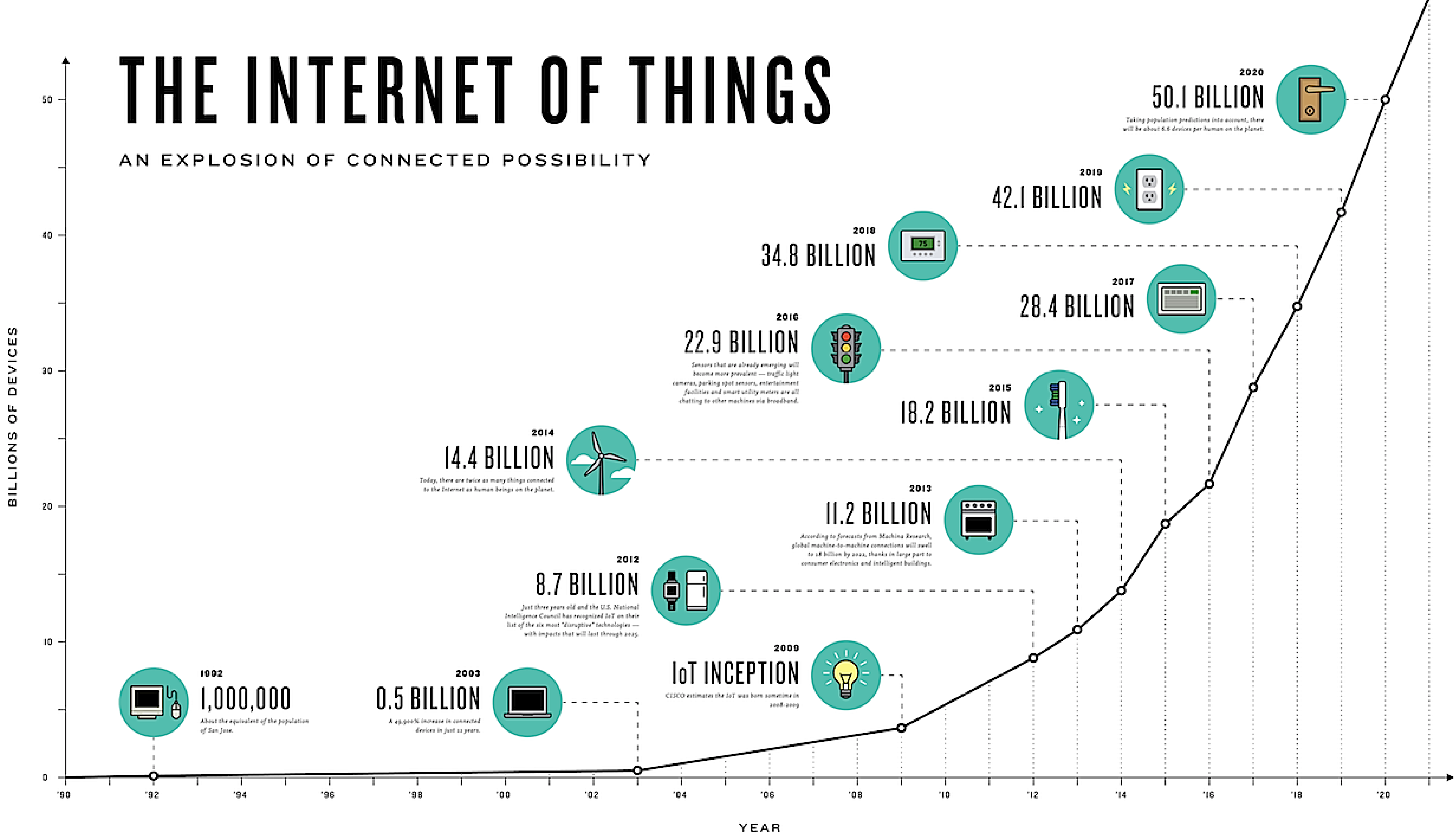
Problem Definition

- Given a piece of binary code (e.g., a binary function)
- Quickly return a set of candidates
  - Semantically equivalent or similar
  - May come from different architectures
  - May be generated by different compilers and options

Applications

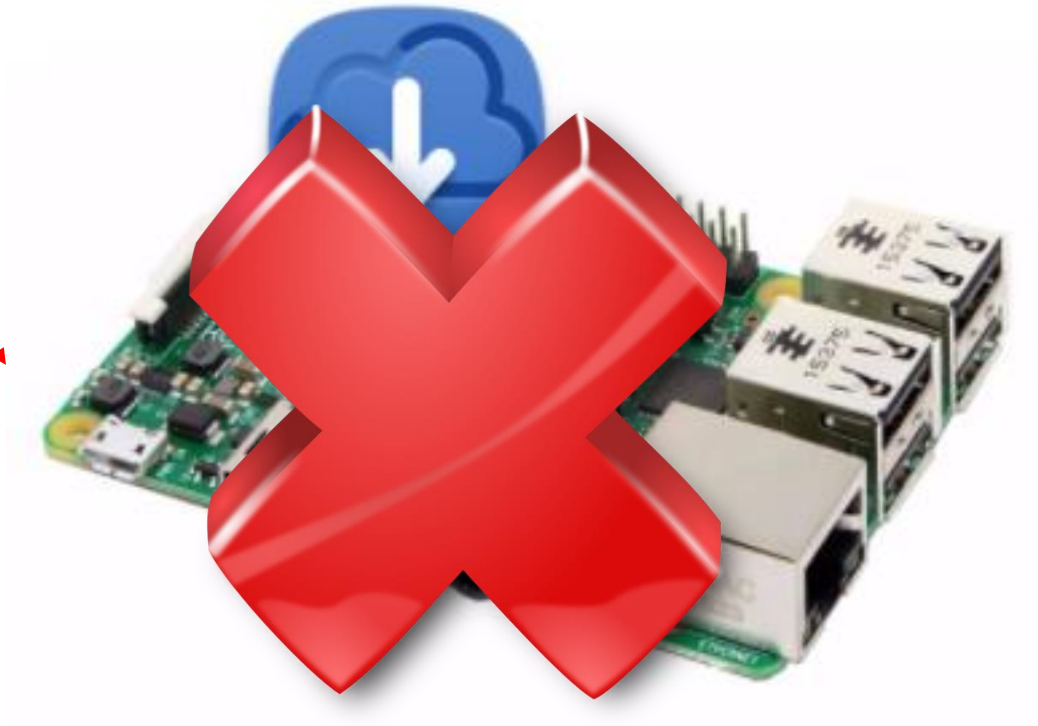
- Plagiarism Detection
- Malware Classification
- Vulnerability Search
  - Emerging topic: vulnerability search in IoT

Internet of Things

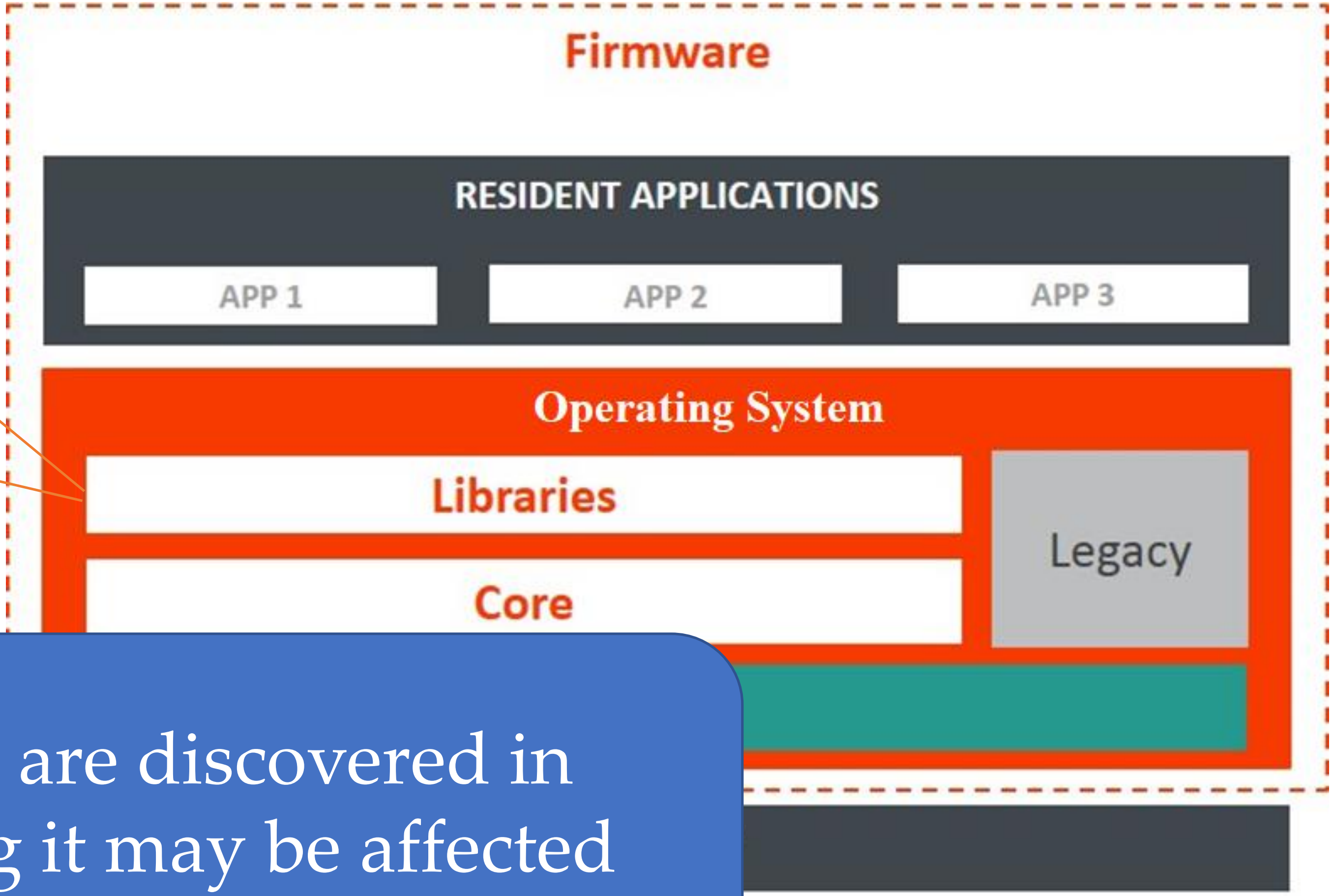

THE INTERNET OF THINGS
AN EXPLOSION OF CONNECTED POSSIBILITY
BILLIONS OF DEVICES
YEAR
0
10
20
30
40
50
'90
'92
'94
'96
'98
'00
'02
'04
'06
'08
'10
'12
'14
'16
'18
'20
1992
1,000,000
About the equivalent of the population of San Jose.
2003
0.5 BILLION
2009
IoT INCEPTION
CISCO estimates the IoT was born sometime in 2008-2009
2012
8.7 BILLION
Just three years old and the U.S. National Intelligence Council has recognized IoT on their list of the six most "disruptive" technologies — with impacts that will last through 2025.
2013
11.2 BILLION
2014
14.4 BILLION
Today, there are twice as many things connected to the Internet as human beings on the planet.
2015
18.2 BILLION
2016
22.9 BILLION
Sensors that are already emerging will become more prevalent — traffic light cameras, parking spot sensors, entertainment facilities and smart utility meters are all chatting to other machines via broadband.
2017
28.4 BILLION
2018
34.8 BILLION
2019
42.1 BILLION
2020
50.1 BILLION

Firmwares ↔ Operating systems to IoT devices

Vulnerability

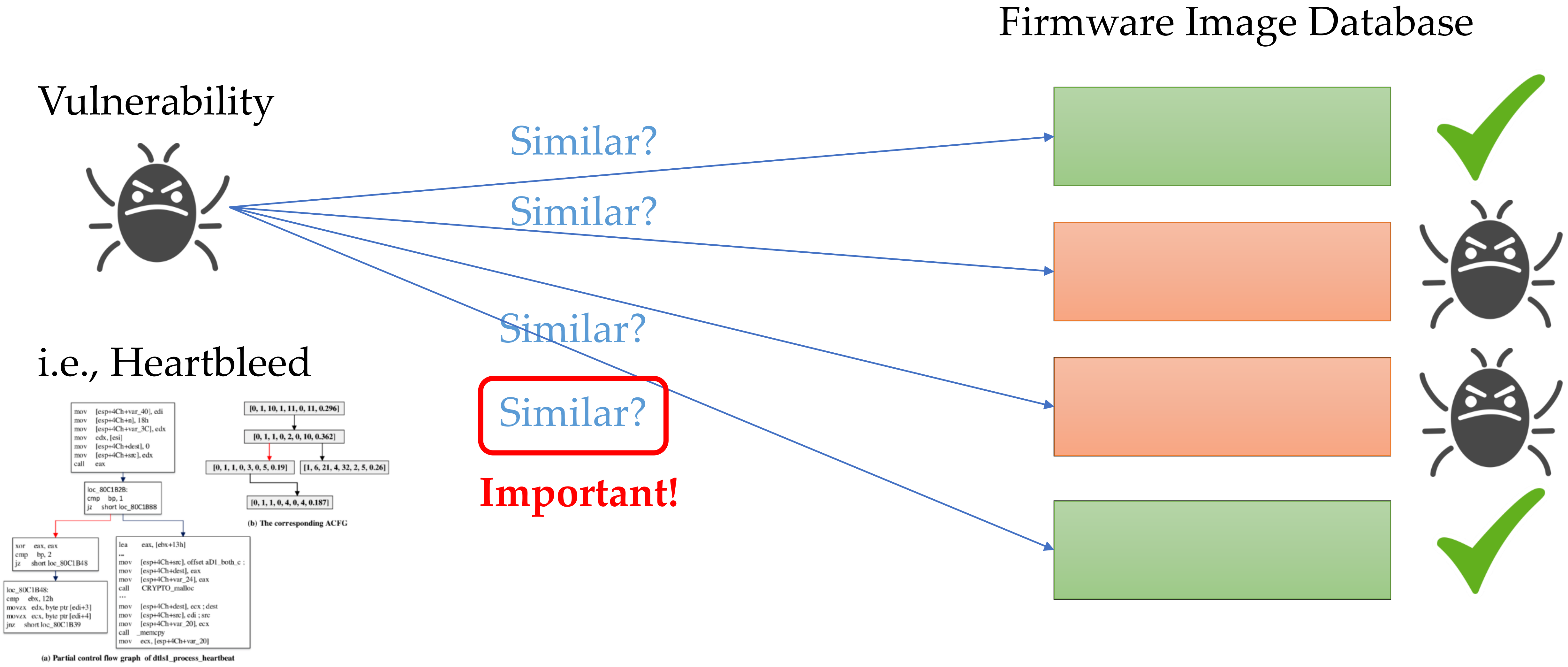


Open source libraries
e.g., OpenSSL



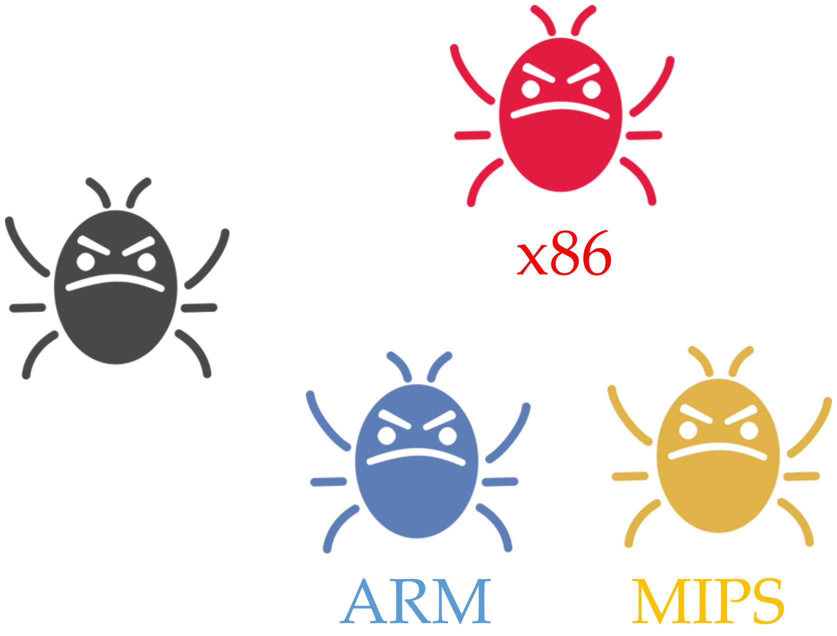
When **new vulnerabilities** are discovered in OpenSSL, all firmware using it may be affected e.g., Heartbleed

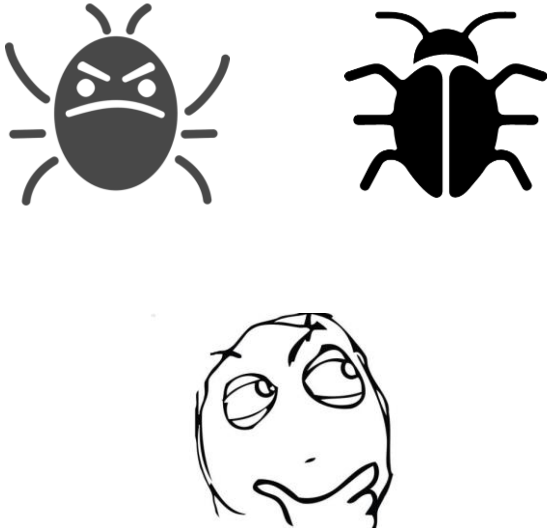
Vulnerability Detection



Challenges for **Binary Code Search**

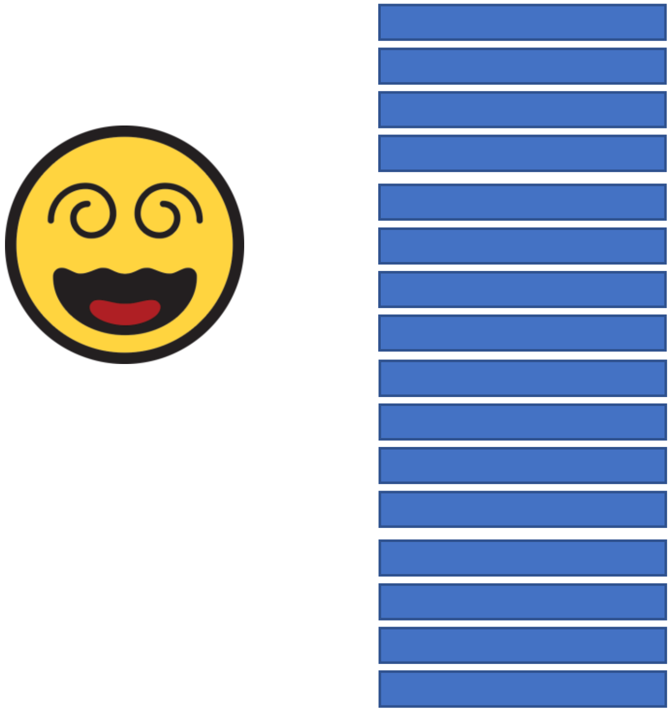
Cross-Platform



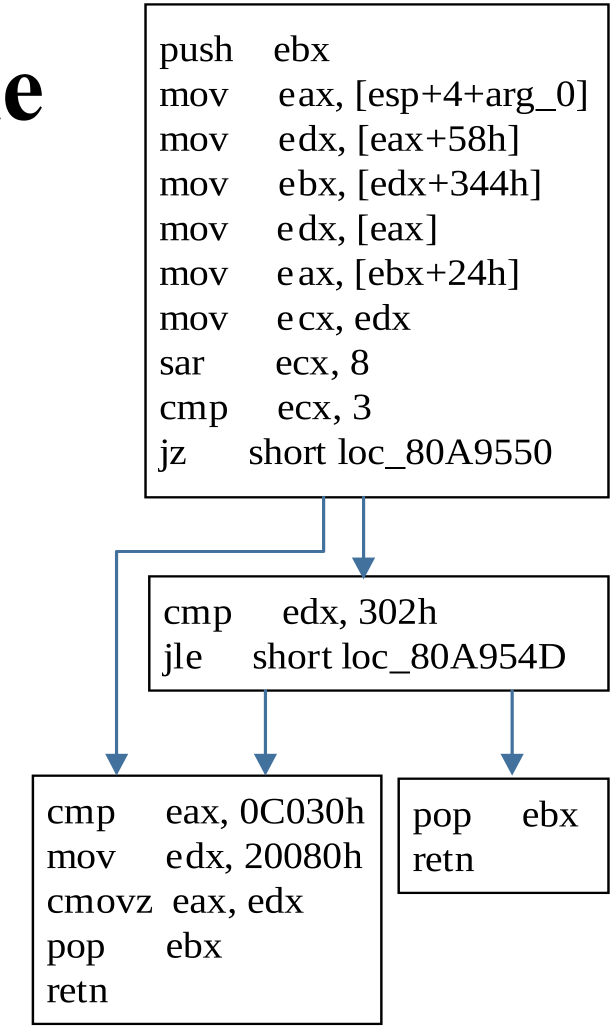


Similar or not similar?
It's a problem!

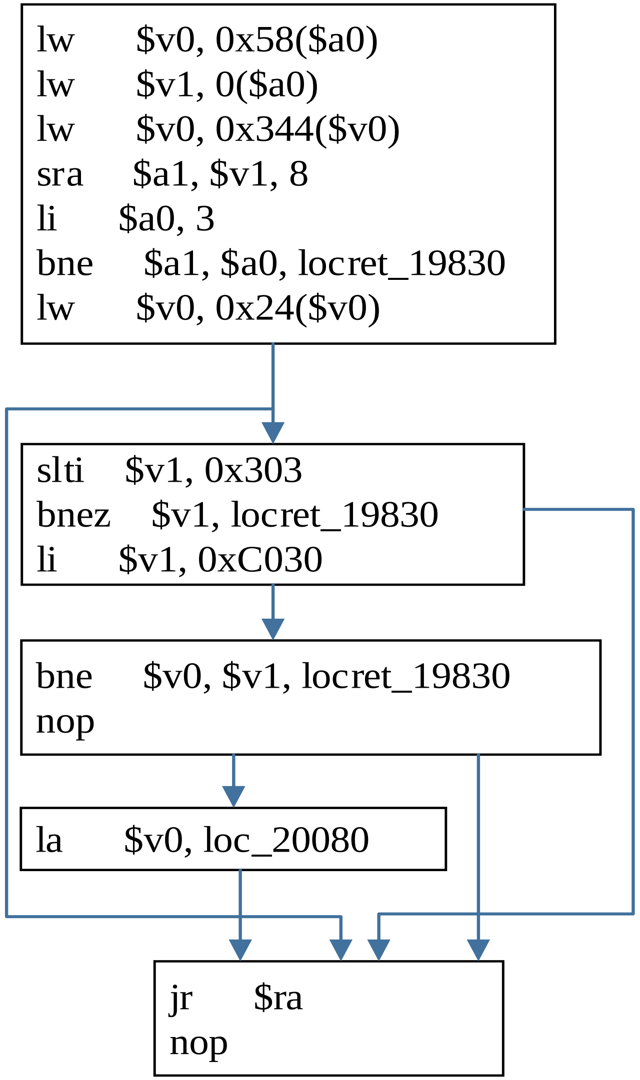
Scalability



An Example



a) x86 assembly



b) MIPS assembly

Existing Binary Code Search Techniques

- **Syntax-based Approach**
  - *Mnemonic code sequence* [**S. M. Tabish et al**. *SIGKDD'09;* **W. M. Khoo et al**. *MSR'13*]
  - *Control flow graph* [**H. Flake. et al.** *DIMVA'04;* **J. Pewny** ***et al. Oakland'15; Eschweiler et al. NDSS'16***]
  - *Call graph* [**X. Hu et al.** *CCS'09*]
- **Semantics-based Approach**
  - *Tracelet* [**Y. David et al**. *PLDI'14*]
  - *Tree expression on basic blocks* [**J. Pewny et al.** *ACSAC'14*]
  - *Symbolic execution* [***D. Gao et al.*** *ICS'08*; **J. Ming, et al** *ISC'12*]

Search for known vulnerabilities

Key challenge: cross-platform code search

- String
  - Backdoors in devices
  - Lack of generality
- “Multi-MH & Multi-k-MH” *[Pewny et al. Oakland’15]*
  - Control-flow graph + I/O pairs
  - Lack of scalability
- “DiscovRe” *[Eschweiler et al. NDSS’16]*
  - Control-flow graph + Statistics features
  - Lack of scalability
  - Lightweight filtering is unreliable

Pair-wise graph matching is expensive!

-> More complex feature representation

-> More accurate

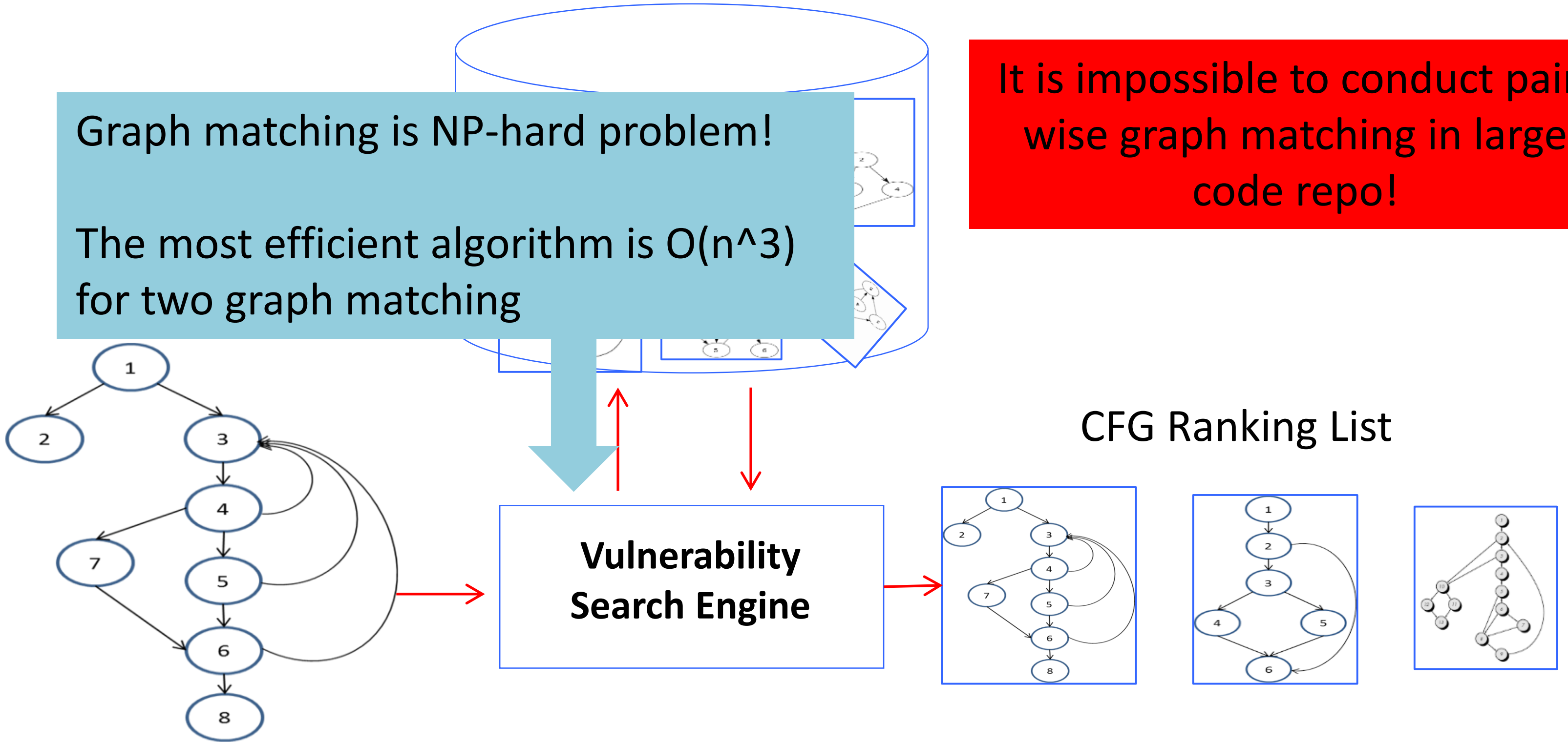
-> Less search efficiency

"Multi-MH & Multi-k-MH" *[Pewny et al. Oakland'15]*
"DiscovRe" *[Eschweiler et al. NDSS'16]*

Graph matching is NP-hard problem!

The most efficient algorithm is O(n^3) for two graph matching

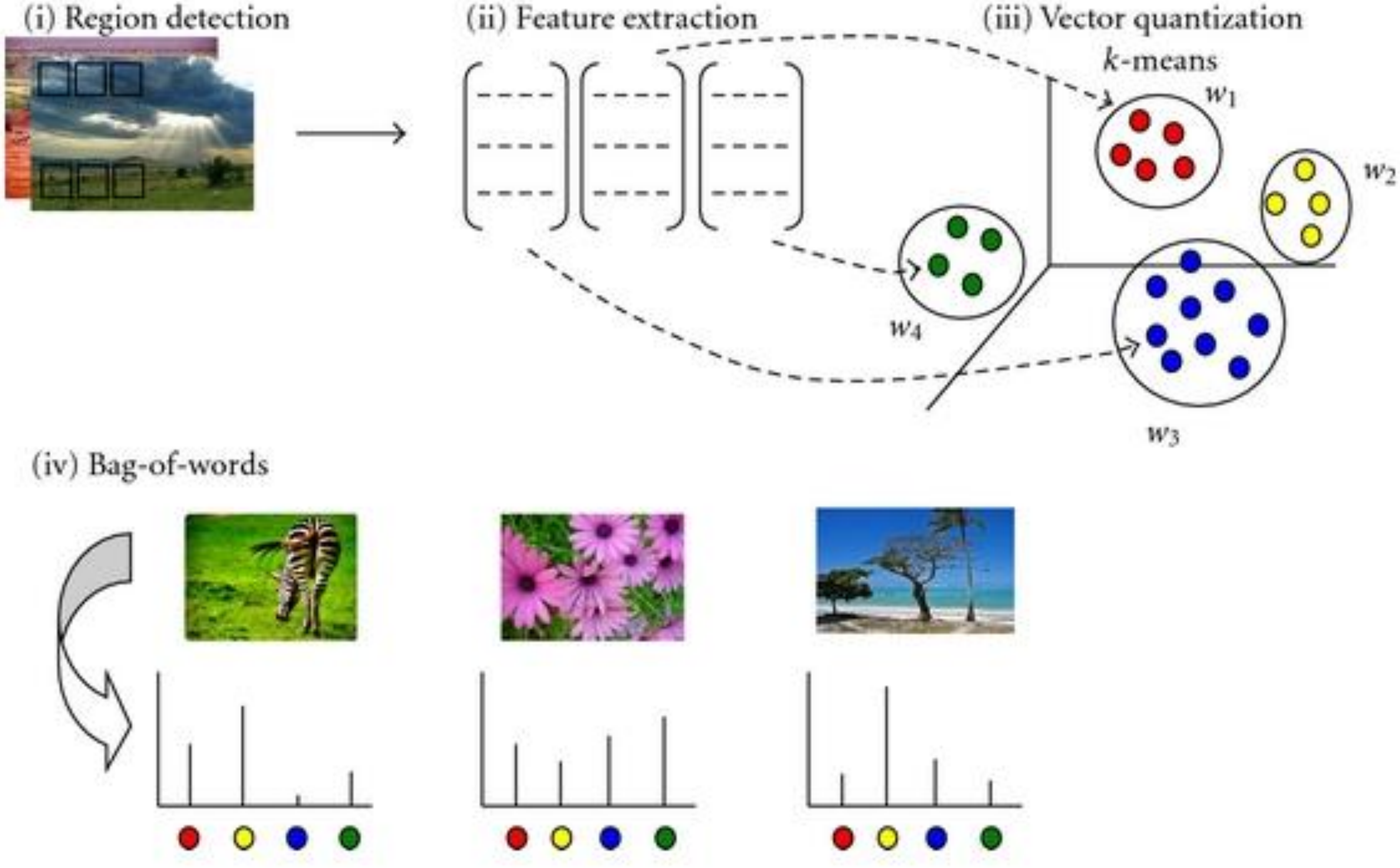
It is impossible to conduct pair-wise graph matching in large code repo!



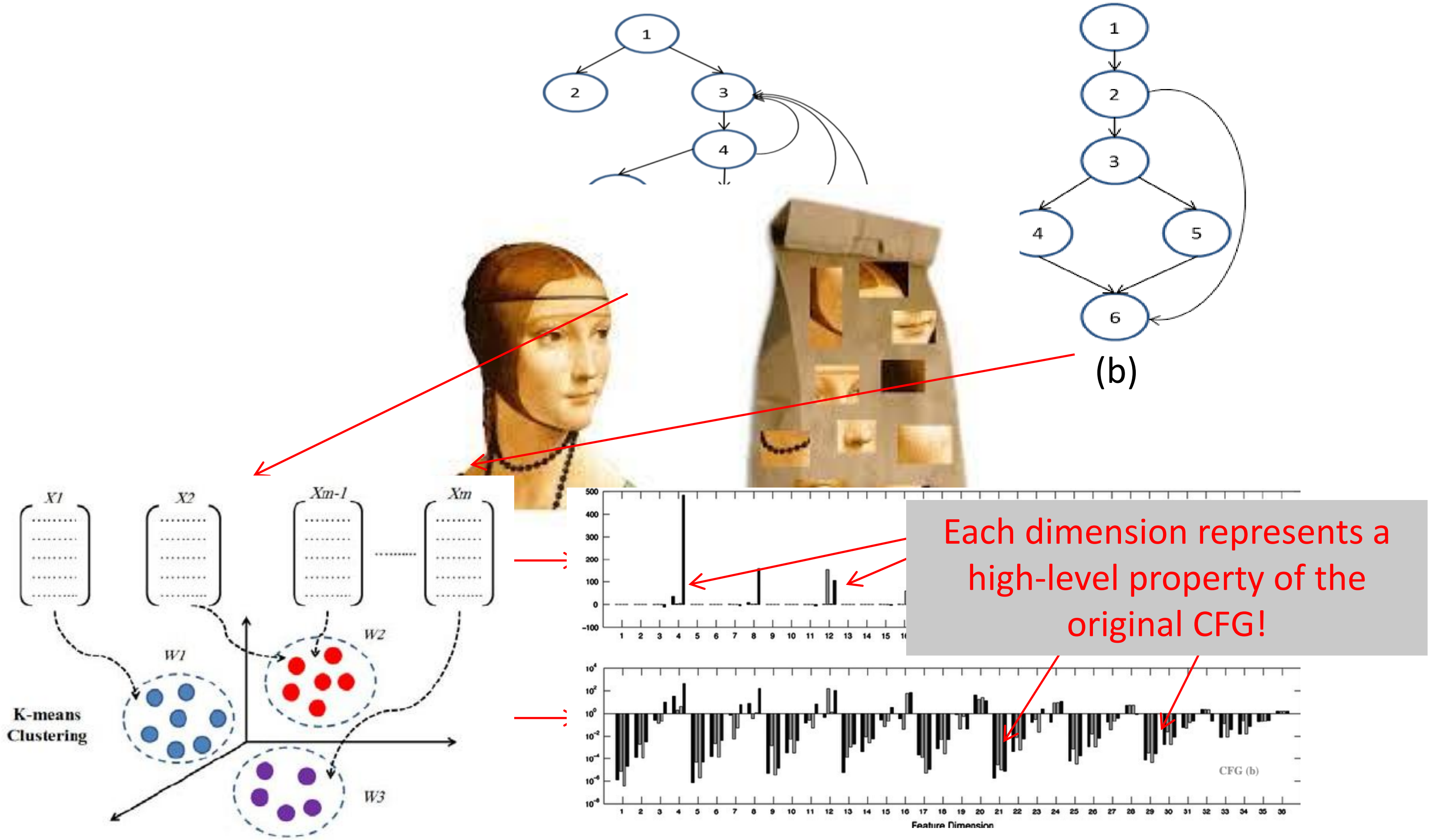
A similar problem

- Image search: tag a similar object in millions of images

We don’t compare images one by one

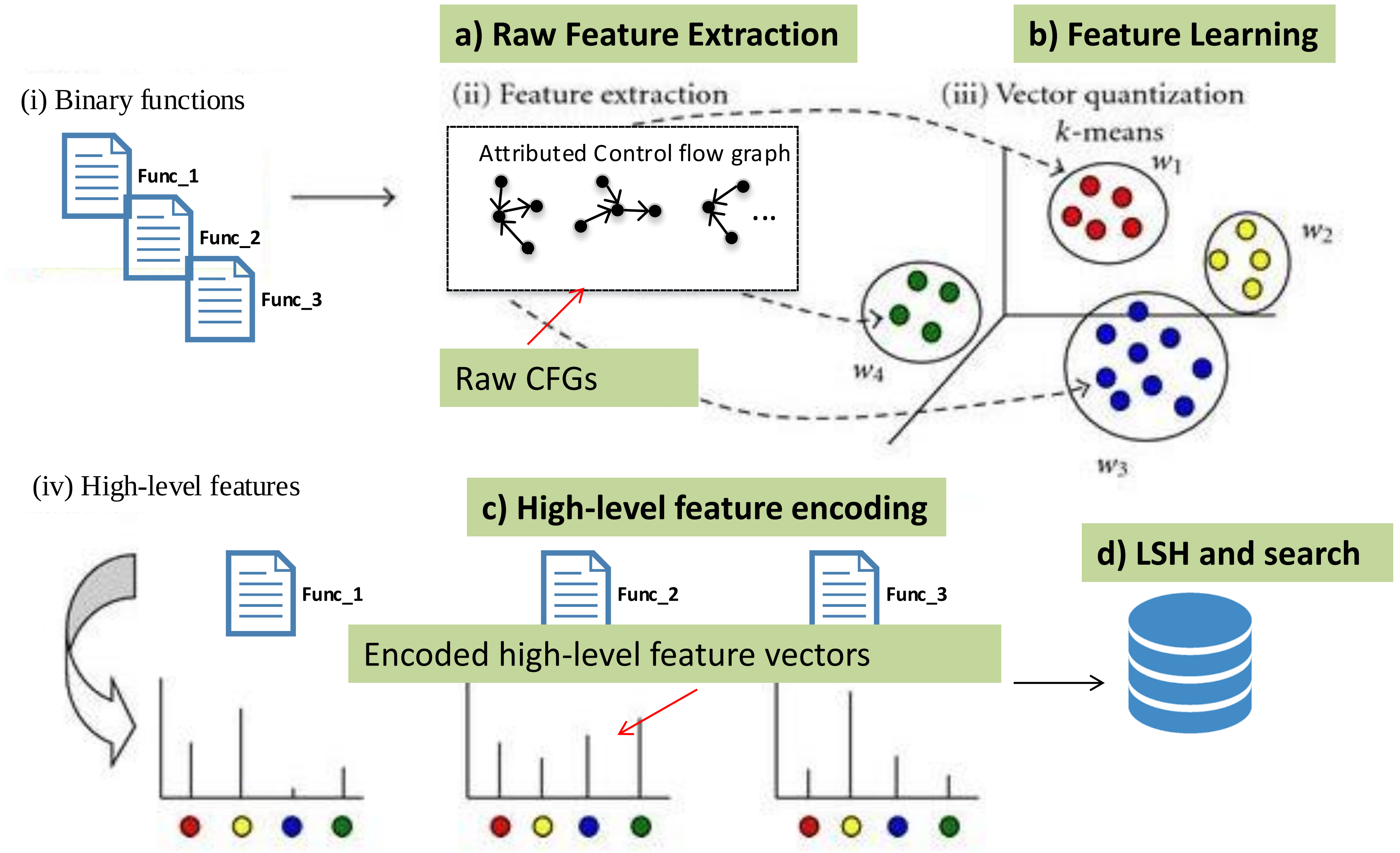


How can we learn high-level feature representations from CFGs?



c. Codebook **d. Feature vector**

Codebook-based approach (Genius, CCS'16)



Raw feature extraction

- Attributed Control Flow Graph

Definition 1. *(Attributed Control Flow Graph) The attributed control flow graph, or ACFG in short, is a directed graph $G = \langle V, E, \phi \rangle$, where V is a set of basic blocks; $E \subseteq V \times V$ is a set of edges representing the connections between these basic blocks, and $\phi : V \to \Sigma$ is the labeling function which maps a basic block in V to a set of attributes in Σ.*

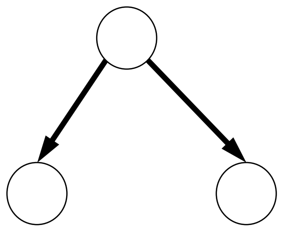
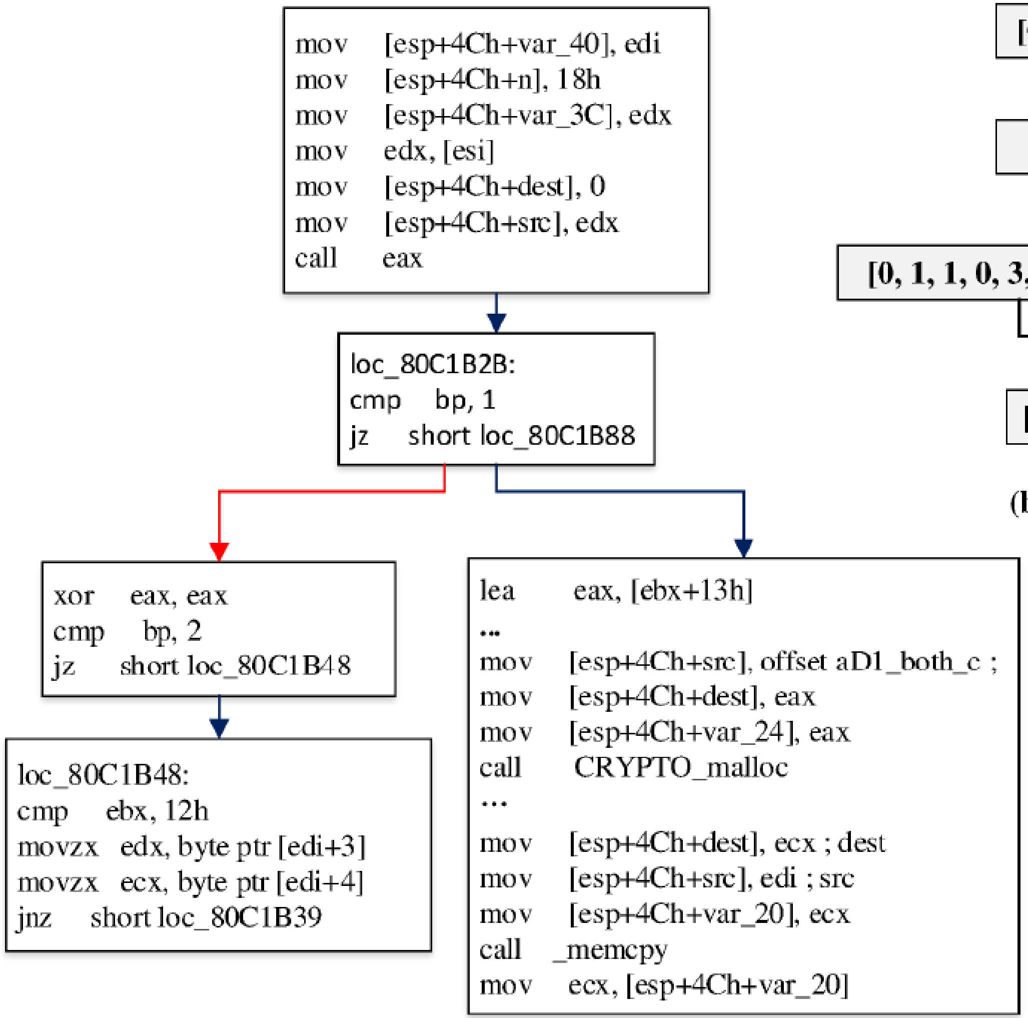


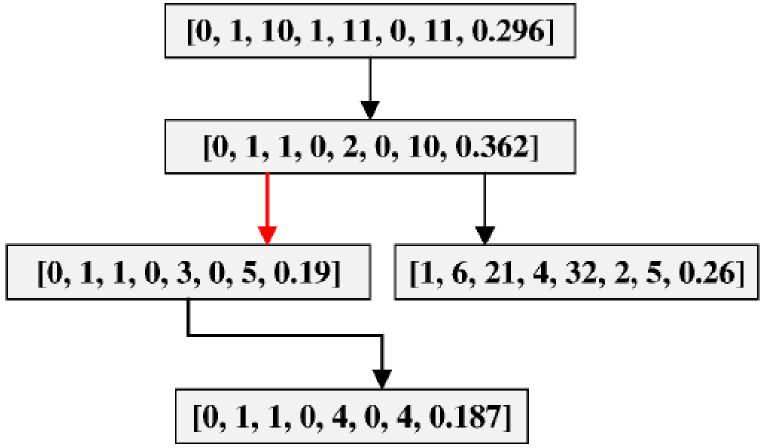
Table 1: Basic-block level features.

Type	Feature Name	Weight (α)
Statistical Features	String Constants	10.82
	Numeric Constants	14.47
	No. of Transfer Instructions	6.54
	No. of Calls	66.22
	No. of Instructions	41.37
	No. of Arithmetic Instructions	55.65
Structural Features	No. of offspring	198.67
	Betweeness	30.66

An example of ACFG



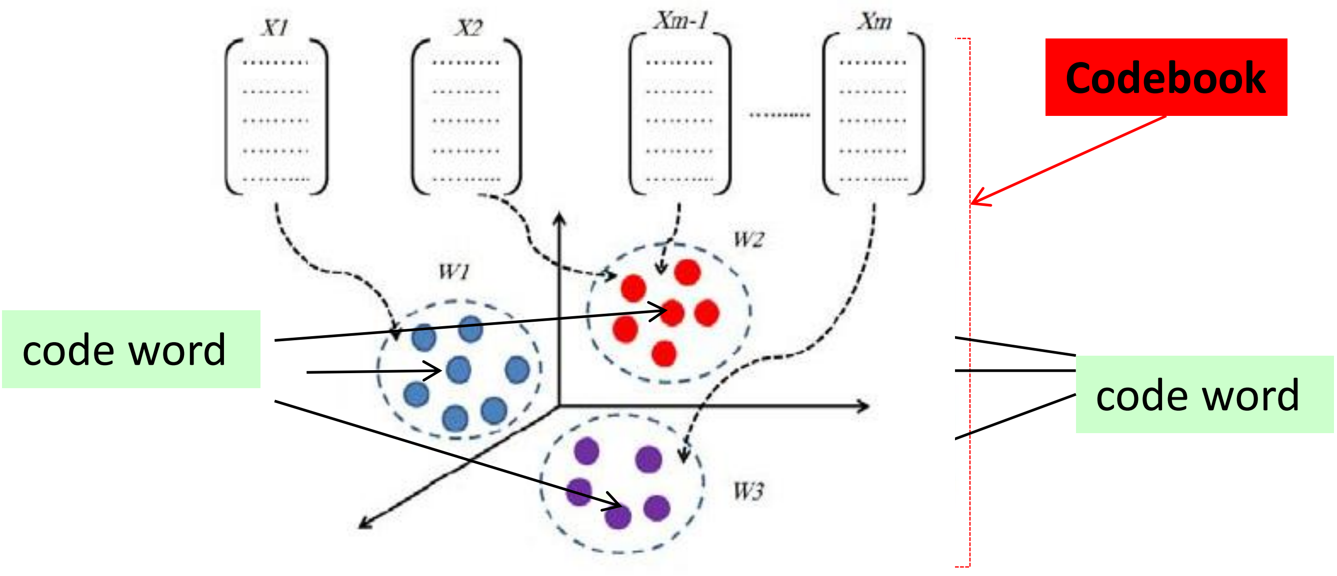
(a) Partial control flow graph of dtls1_process_heartbeat



(b) The corresponding ACFG

Feature learning

Learn a codebook from raw features. Each code word represents one property shared by raw features.



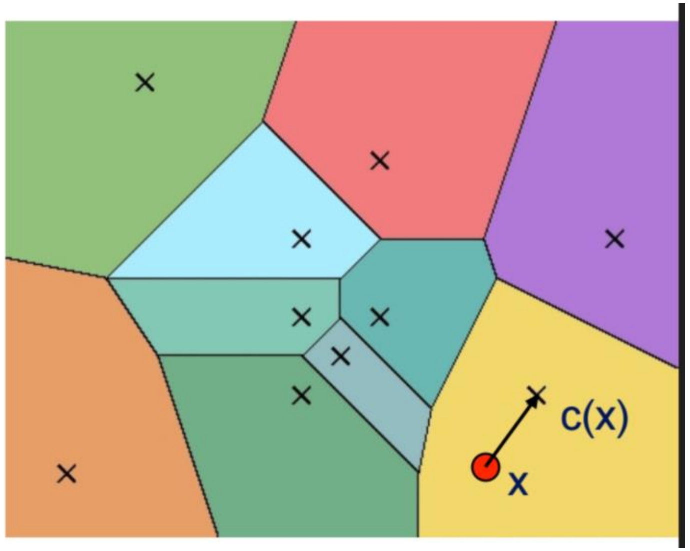
Codebook

Feature learning

- Codebook
  - Each code word is the **centroid** of a cluster of ACFGs
- Clustering on raw features (ACFGs)
  - K-means, hierarchical-k-means, .etc.
- Codebook size
  - Predetermined by # of clusters
  - Bigger Size -> Higher accuracy & Lower Encoding Performance

High-level feature encoding

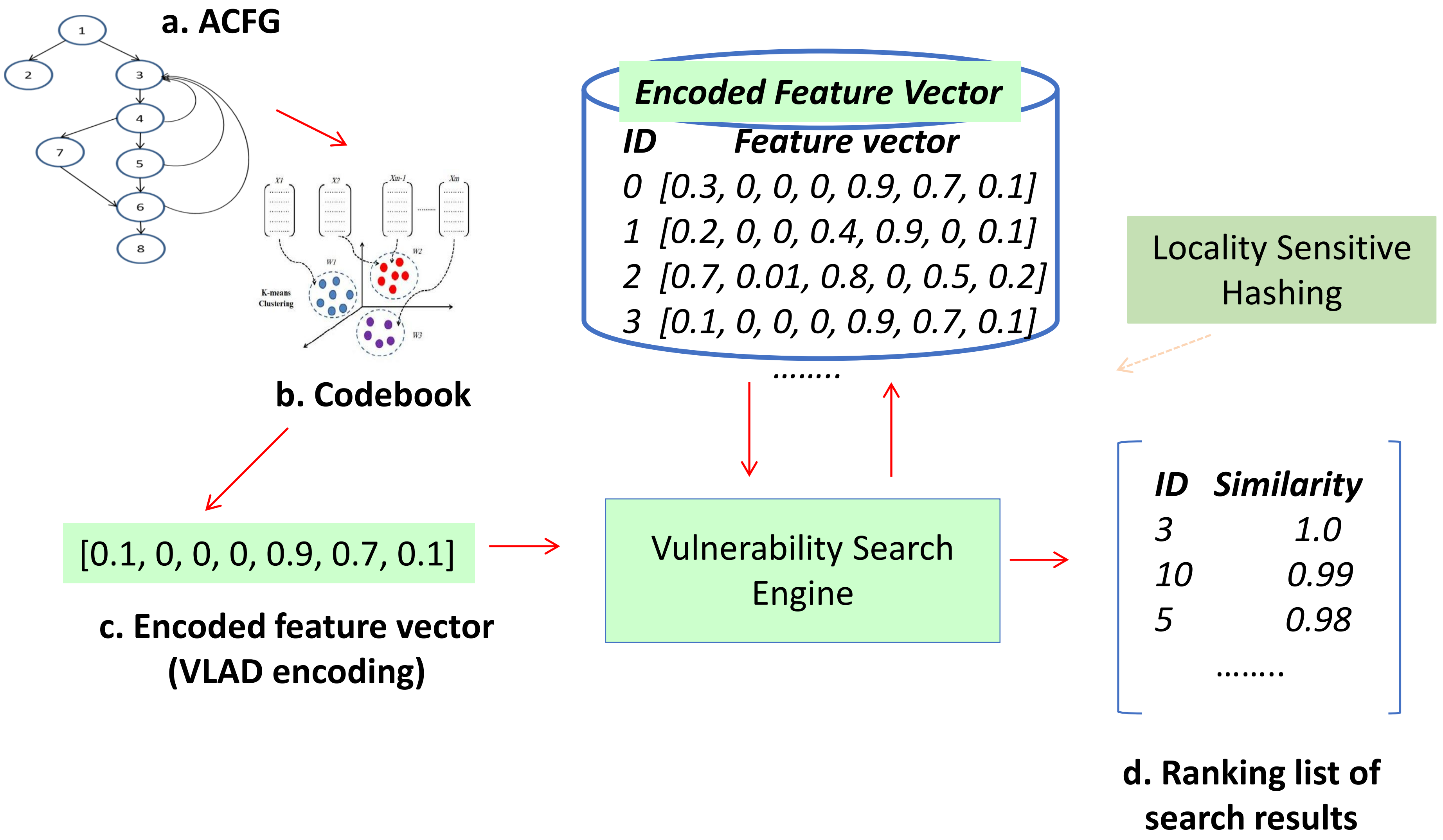
- **VLAD encoding:**
  - Measure the distance between a given ACFG to each centroid
  - To normalize the feature vector, we use graph similarity instead
  - VLAD quantizer is shown below:



$$q(g_i) = \sum_{g_i : NN(g_i) = c_j} [\mathbb{1}(1 = j)\kappa(g_i, c_1), \ldots, \mathbb{1}(n = j)\kappa(g_i, c_n)]^T$$

The similarity score is calculated via graph edit distance

Index and Search



Evaluating Genius

- Dataset Preparation
  - 0.6 billion functions and hundreds of vulnerabilities
- Baseline Preparation
  - Compare with Multi-MH and Multi-k-MH, DiscoveRe, Centroid.
- Performance Evaluation
  - TPR and FPR
  - Search Efficiency
  - Preparation Time
- Case Studies

Genius: Graph Encoding for Bug Search

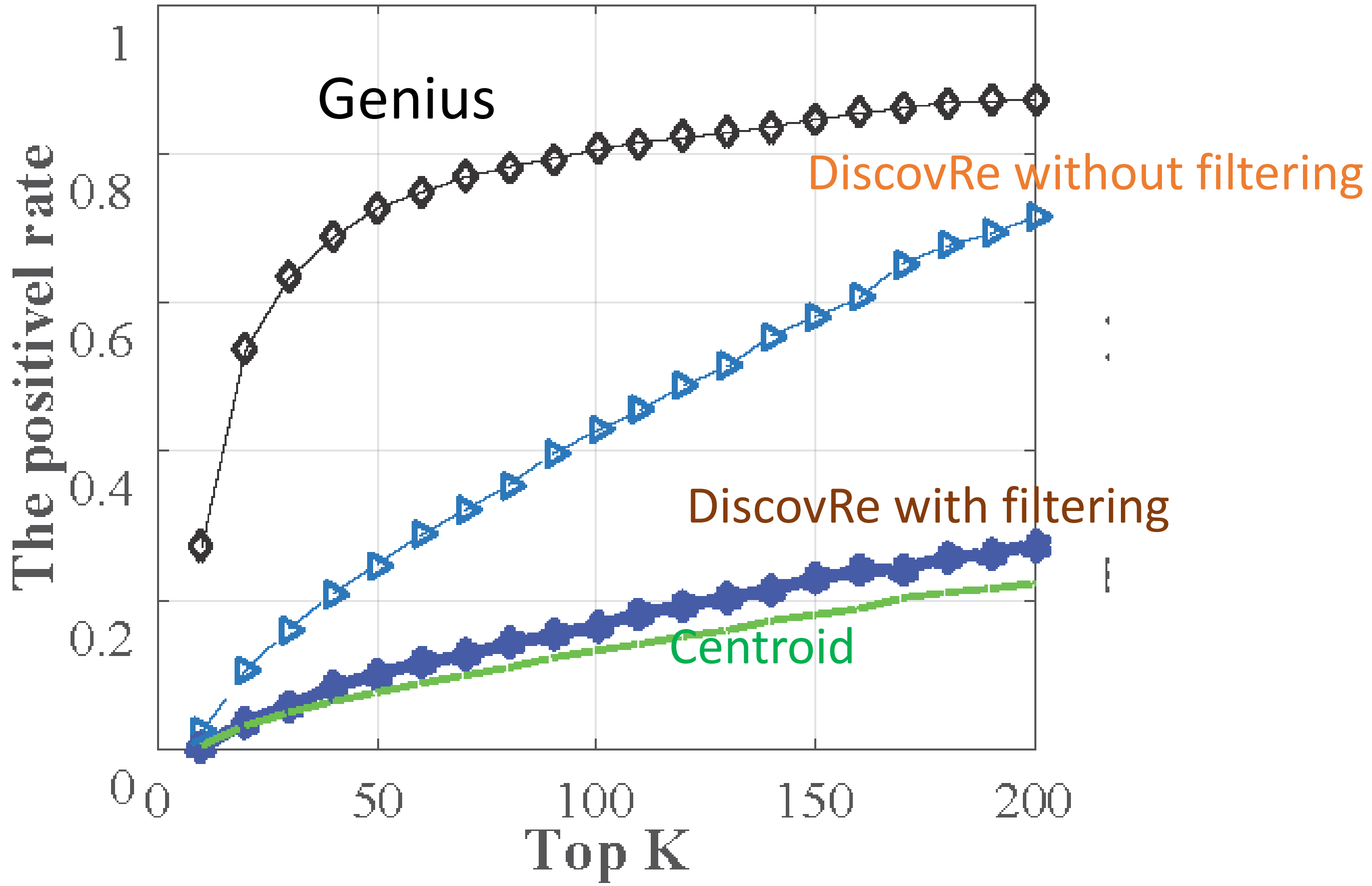
Evaluation: Datasets

- Baseline Dataset
    - BusyBox (v1.21 and v1.20), OpenSSL (v1.0.1f and v1.0.1a) and coreutils (v6.5 and v6.7)
    - x86, ARM, MIPS; all 32 bit
    - 568,134+ functions.
- Firmware Image Dataset
    - 33,045 firmware images
    - 26 different vendors
- Vulnerability Dataset
    - 154 vulnerable functions

Evaluation: Baseline Comparison

- DiscovRe *[Eschweiler et al. NDSS'16*]
  - Re-implemented its core part about graph matching and feature learning
- Multi-MH and Multi-k-MH *[Pewny et al. Oakland'15]*
  - Compared on the same dataset
- Centroid *[Chen et al. USENIX Security'15]*
  - Re-implemented its algorithm
  - A simple encoding that converts a CFG into a number

Evaluation: True Positive Rate



a) Recall rates across different threshold K

Evaluation: Search Efficiency

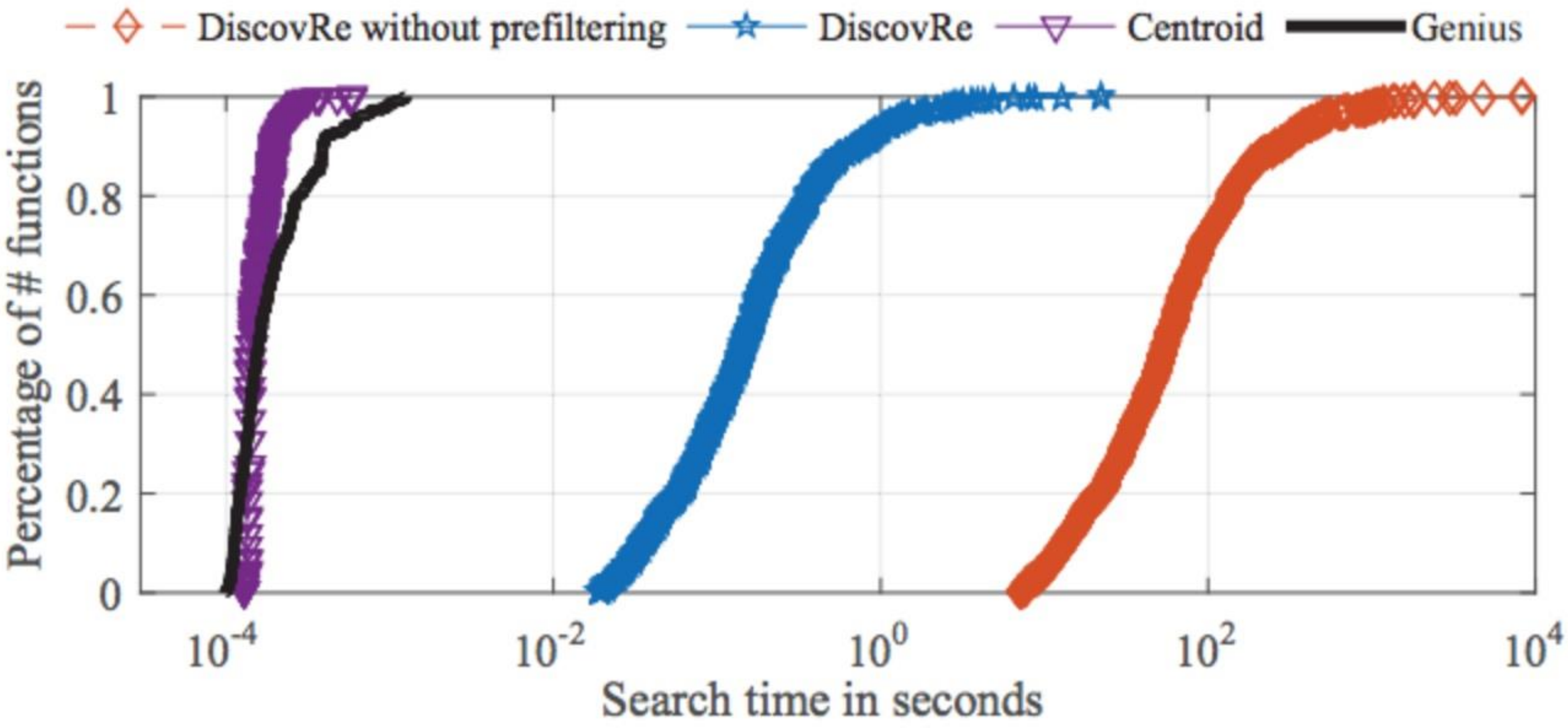


Figure2. The CDFs of search time on Dataset I.

Evaluation: Case Study I

- Search 2 vulnerabilities on 8126 firmware images
  - CVE-2015-1791: top 50 candidates, 14 firmware images potentially affected, 10 confirmed. Two vendors: **D-Link** and **Belkin**.
  - CVE-2014-3508: 24 firmware images potentially vulnerable, 13 confirmed. Vendors are **CenturyLink**, **D-Link** and **Actiontec**.

Evaluation: Case Study II

- Search two latest firmware images for all vulnerabilities
  - D-Link DIR-810 models
  - 154 Vulnerabilities
  - Search time: < 0.1s
  - Check top 100 candidates

Table 4: Case study results for Scenario II

DIR-810L_REVB_FIRMWARE_2.03B02			DIR-810L_REVB_FIRMWARE_2.02.B01		
CVE	**Patched**	**Vulnerability Type**	**CVE**	**Patched**	**Vulnerability Type**
CVE-2016-0703	No	Allows man-in-the-middle attack	CVE-2015-0206	No	Memory consumption
CVE-2015-1790	No	NULL pointer dereference	CVE-2014-0160	Yes	Heartbleed
CVE-2015-1791	Yes	Double free	CVE-2015-0289	No	NULL pointer dereference
CVE-2015-0289	No	NULL pointer dereference	CVE-2016-0797	No	Heap memory corruption
CVE-2014-8275	No	Missing sanitation check	CVE-2016-0798	No	Memory consumption
CVE-2015-0209	No	Use-after-free	CVE-2014-3513	No	Memory consumption
CVE-2015-3195	No	Mishandles errors	CVE-2014-3508	No	Information leakage
#	#	#	CVE-2015-0206	No	Memory consumption
#	#	#	CVE-2014-8275	No	Missing sanitation check

Limitations of Genius

- Encoding is still expensive
  - 1 graph comparison for each word in codebook
- Feature dimension has to be small
  - Confine the search accuracy
- Codebook generation is expensive
  - May take a week to retrain the codebook

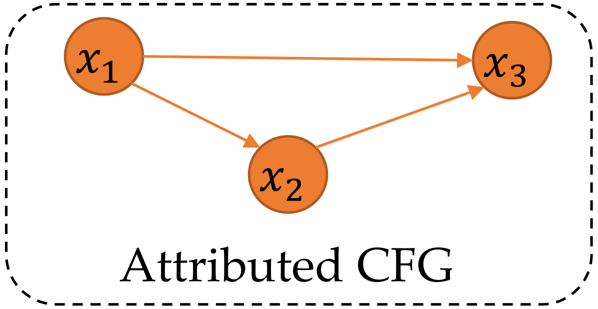
Neural Network-based Graph Embedding for Cross-Platform Binary Code Similarity Detection

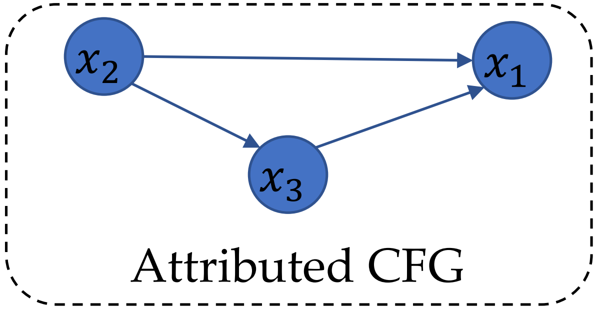
Xiaojun Xu, **Chang Liu**, Qian Feng, Heng Yin, Le Song, Dawn Song

上海交通大學 Shanghai Jiao Tong University
Berkeley University of California
UC Riverside University of California
Georgia Institute of Technology

Two unbeatable advantages of neural network-based similarity detection

Previous approaches on expensive graph-matching based algorithms to detect similarity





Very SLOW!

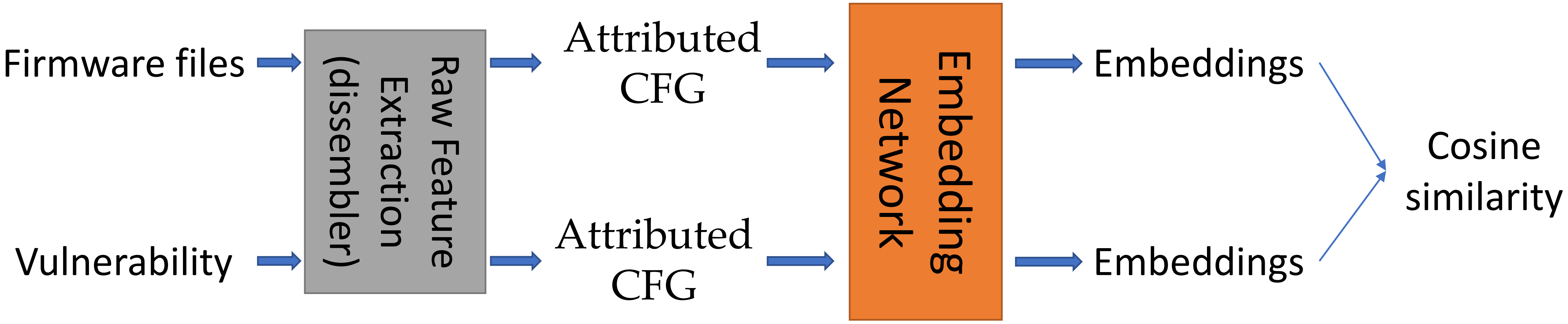
We will show that a neural network-based approach can be much **more efficient!**

Takeaways

Message 1. Our work is one of **the first demonstrations** to show that deep learning techniques can be applied to binary analysis

Message 2. We hope our work can foster **more investigations** on using deep learning approaches for binary analysis

Overall workflow



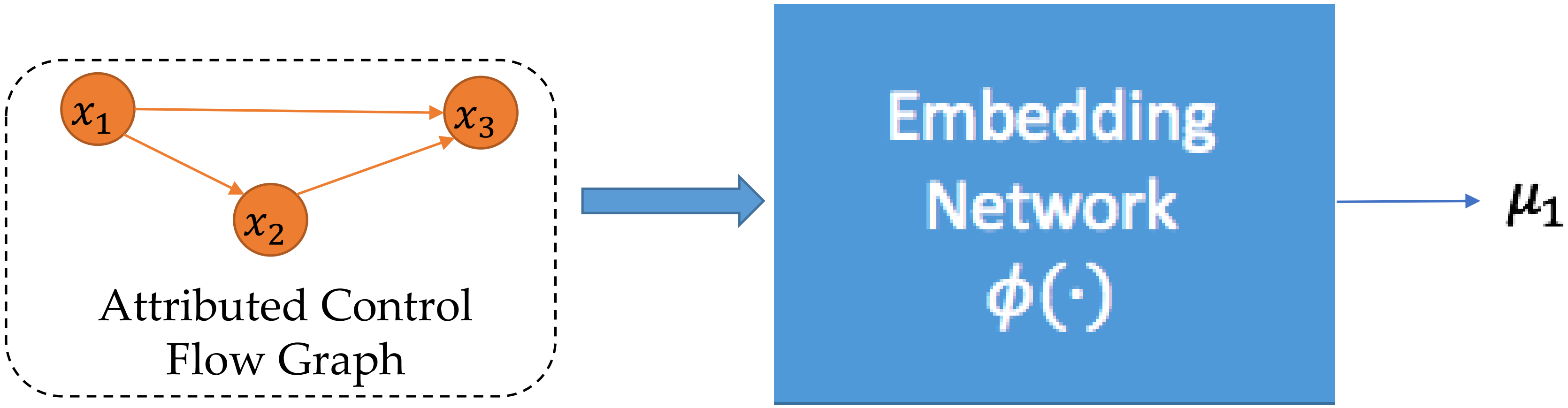
Previous approaches

- Manually designed graph-matching-based algorithms
- Slow
- Effectiveness is limited by graph-matching
- Feng, et al. Scalable Graph-based Bug Search for Firmware Images. *CCS* 2016.

Our approaches:

- Deep graph embedding network
- Design a neural network to extract the features automatically
- Combine Struct2vec and Siamese network

Our approach: structure2vec



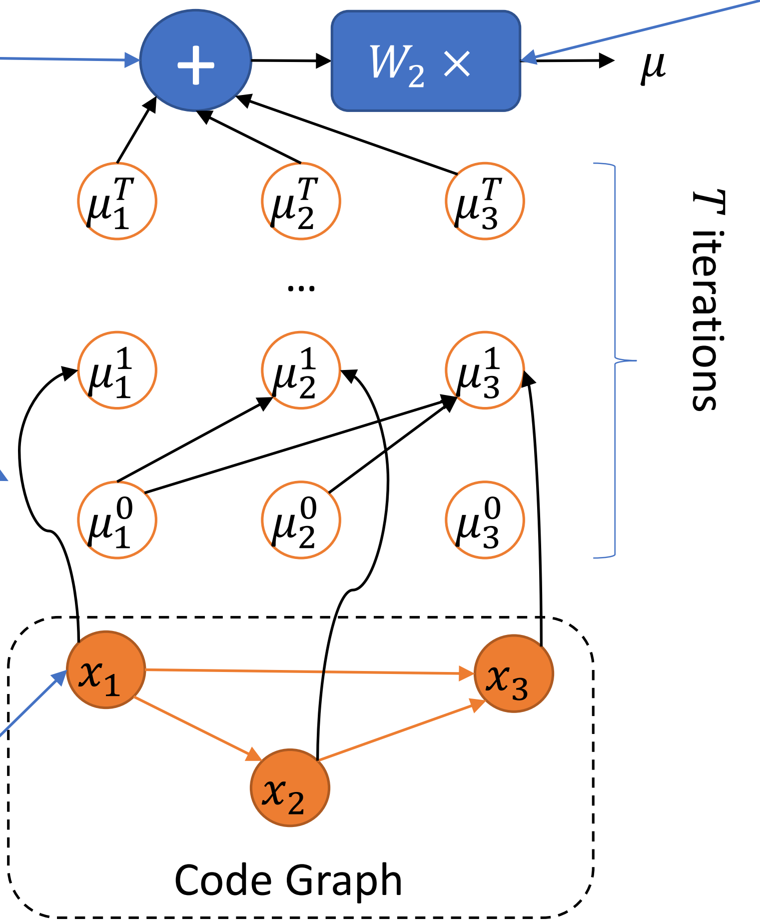
Dai, et al. Discriminative Embeddings of Latent Variable Models for Structured Data. ICML 2016.

Take a closer look at the embedding network

3. After the last iteration, the embeddings on all vertexes are aggregated together

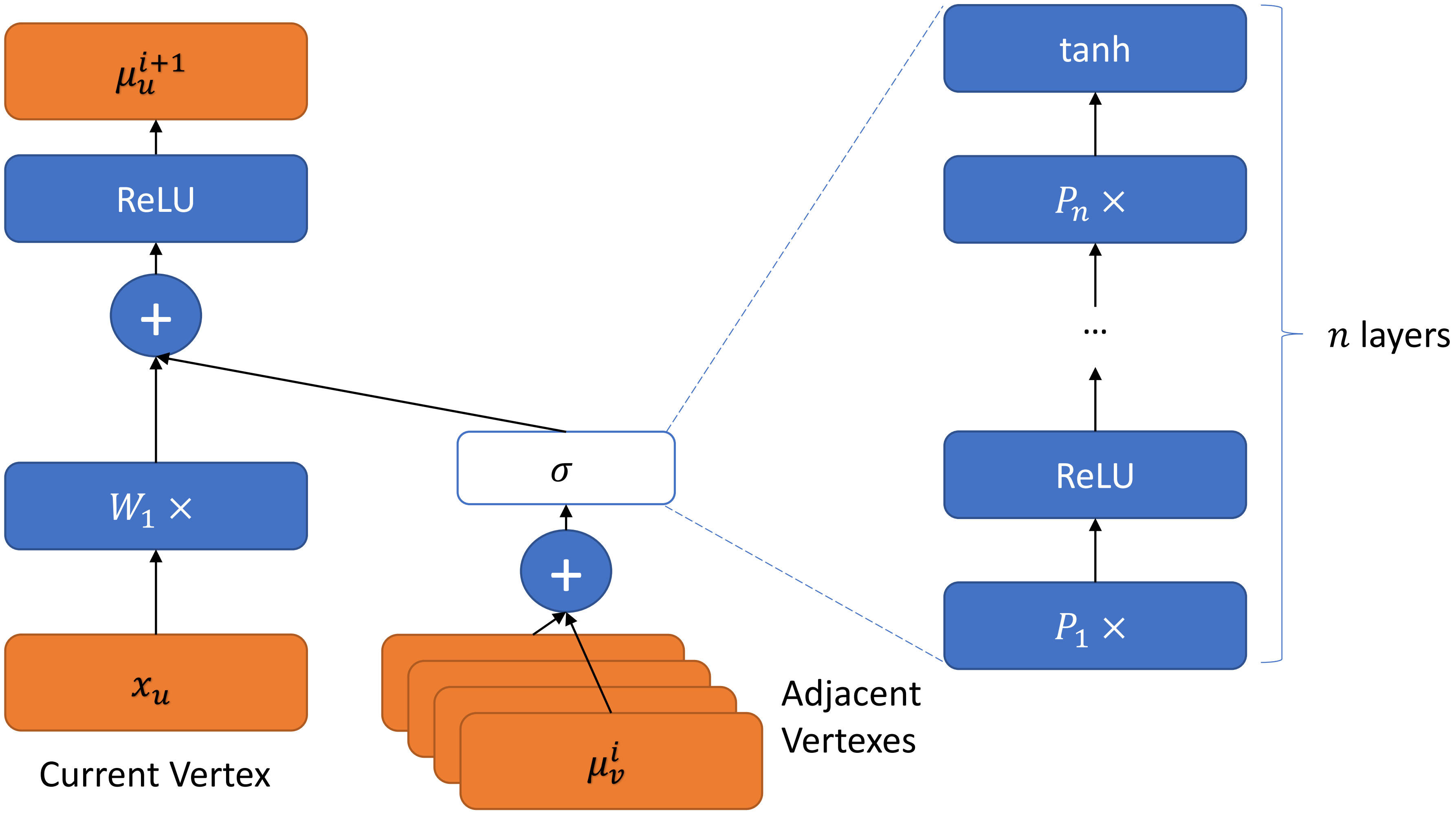
2. In each iteration, the embedding on each vertex is propagated to its neighbors

1. Initially, each vertex has an embedding vector computed from each code block

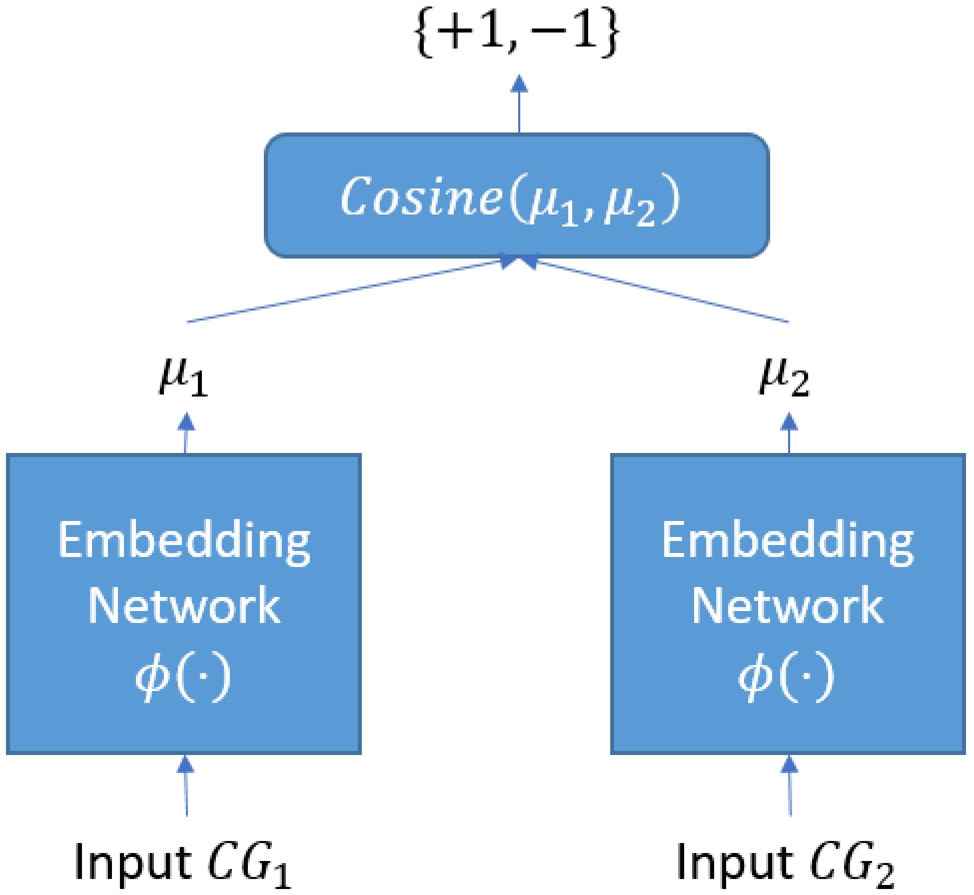


4. An affine transformation is applied in the end to compute the embedding for the graph

Take a closer look at propagation



Training: Siamese



1. Application-independent pretraining
    - Compile given source code into different platforms using different compilers and different optimization-levels
    - A pair of binary functions compiled from the same source code is labeled with +1
    - Otherwise, -1

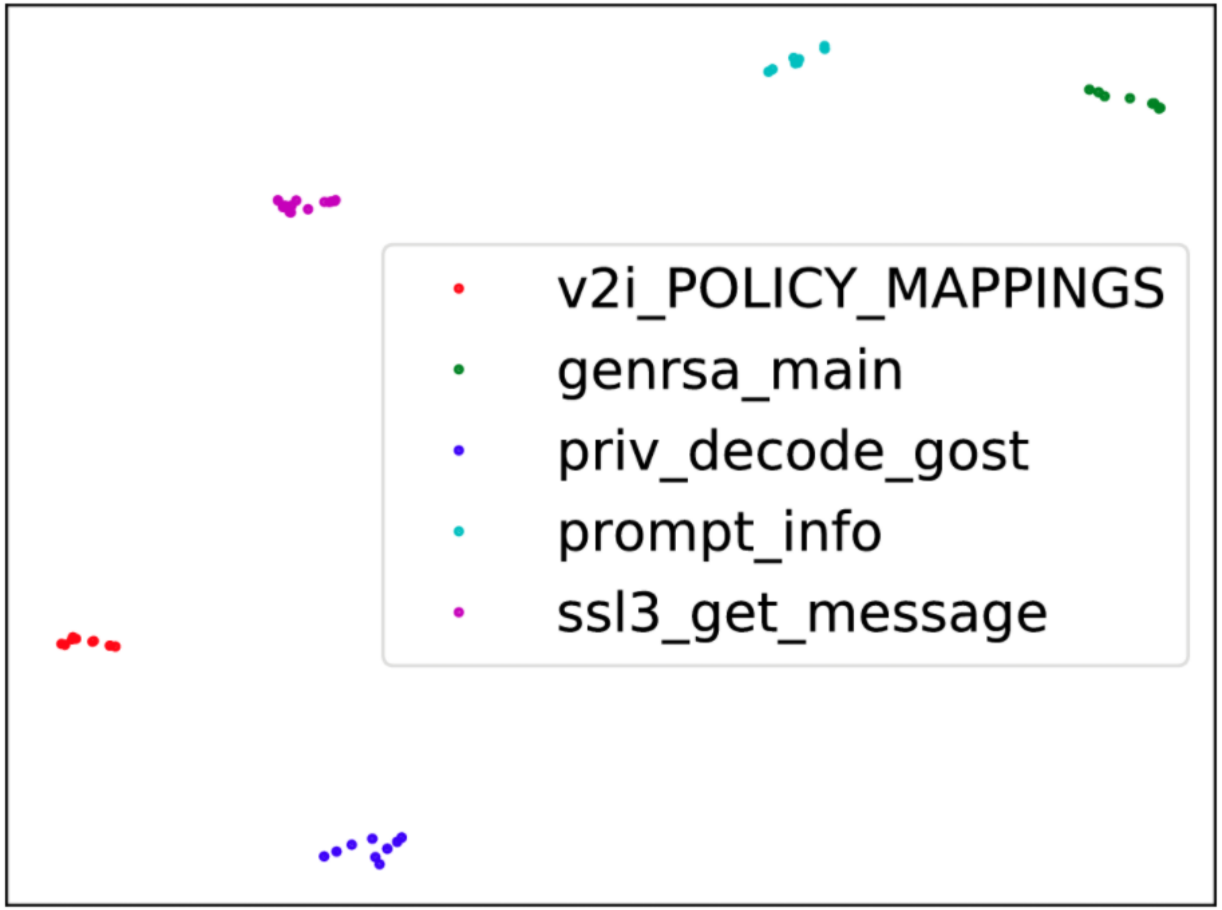
2. Application-dependent retraining
    - Human can label similar and dissimilar pairs of binary functions
    - This additional training data can be used in a retraining process

Training Data Details

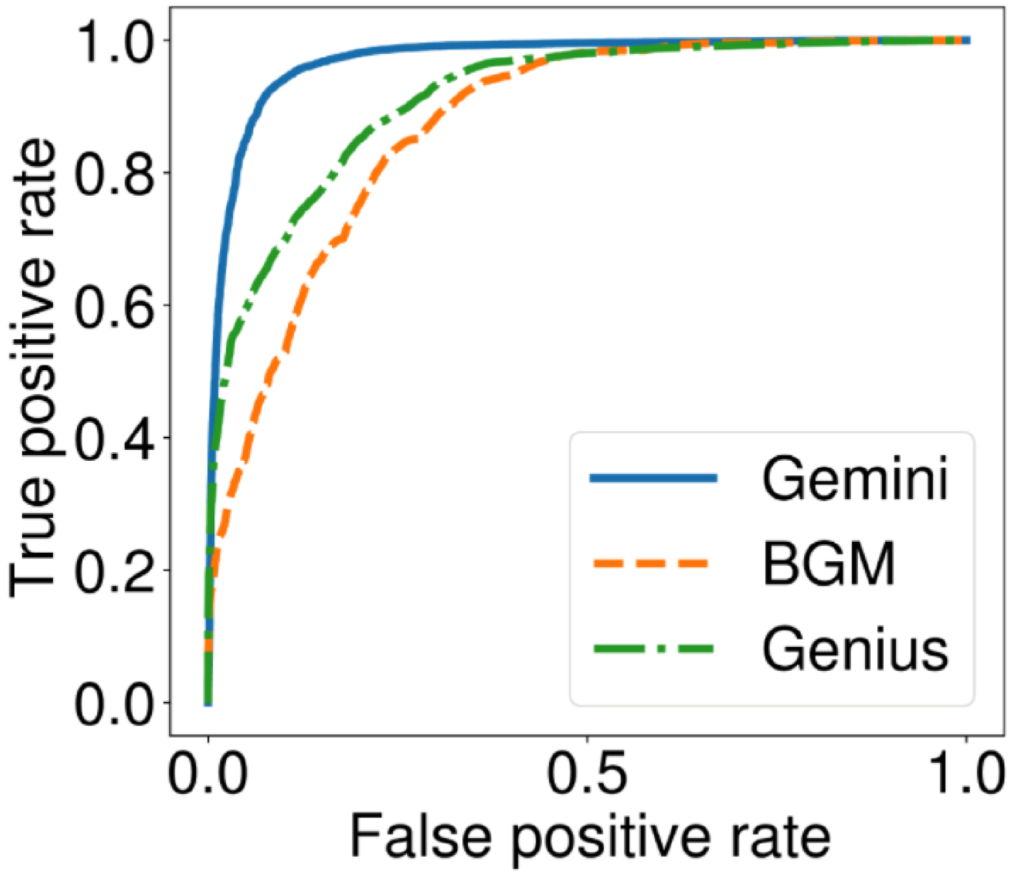
- OpenSSL (version 1.0.1f and 1.0.1u)
  - Compiled using GCC v5.4
  - Emit code to x86, MIPS, ARM
  - Using optimization level O0-O3

	Training	Validation	Testing
x86	30,994	3,868	3,973
MIPS	41,477	5,181	5,209
ARM	30,892	3,805	3,966
Total	103,363	12,854	13,148

Visualizing the embeddings



Accuracy: ROC curve on test data



Serving time (per function processing time)

Previous work: a few secs to a few mins

Now: a few milliseconds

2500 $\times$ to 16000 $\times$ faster!

Training time

Previous work: > 1 week

Now: < 30 mins

Identified Vulnerabilities in Large Scale Dataset

Function Name	Vendor	Firmware	Binary File	Similarity
ssl3_get_new_session_ticket	D-Link	DAP-1562_FIRMWARE_1.00	wpa_supplicant.acfgs	0.962374508
port_check_v6	D-Link	DES-1210-28_REVB_FIRMWARE_3.12.015	in.ftpd.acfgs	0.955408692
sub_42EE7C	TP-Link	TD-W8970B_V1_140624	racoon.acfgs	0.954742193
sub_42EE7C	TP-Link	TD-W8970_V1_130828	racoon.acfgs	0.954742193
prsa_parse_file	TP-Link	Archer_D5_V1_140804	racoon.acfgs	0.949814439
sub_432B8C	TP-Link	TD-W8970B_V1_140624	racoon.acfgs	0.949583828
sub_432B8C	TP-Link	TD-W8970_V1_130828	racoon.acfgs	0.949583828
ssl3_get_new_session_ticket	DD-wrt	dd-wrt.v24-23838_NEWD-2_K3.x_mega-WNR3500v2_VC	openvpn.acfgs	0.94668287
ucSetUsbipServer	TP-Link	WDR4900_V2_130115	httpd.acfgs	0.946312308
ssl3_get_new_session_ticket	Netgear	tomato-Cisco-M10v2-NVRAM32K-1.28.RT-N5x-MIPSR2-110-PL-Mini	libssl.so.1.0.0.acfgs	0.945933044
ssl3_get_new_session_ticket	Tomato_by_Shibby	tomato-K26-1.28.RT-MIPSR1-109-Mini	libssl.so.1.0.0.acfgs	0.945933044
ssl3_get_new_session_ticket	Tomato_by_Shibby	tomato-K26USB-1.28.RT-N5x-MIPSR2-110-VPN	libssl.so.1.0.0.acfgs	0.945932984
ssl3_get_new_session_ticket	Tomato_by_Shibby	tomato-E4200USB-NVRAM60K-1.28.RT-MIPSR2-110-PL-BT	libssl.so.1.0.0.acfgs	0.945932984
ssl3_get_new_session_ticket	Tomato_by_Shibby	tomato-E3000USB-NVRAM60K-1.28.RT-MIPSR2-110-BT-VPN	libssl.so.1.0.0.acfgs	0.945932984
ssl3_get_new_session_ticket	Tomato_by_Shibby	tomato-K26USB-1.28.RT-MIPSR1-109-AIO	libssl.so.1.0.0.acfgs	0.945932984
ssl3_get_new_session_ticket	Tomato_by_Shibby	tomato-Netgear-3500Lv2-K26USB-1.28.RT-N5x--109-AIO	libssl.so.1.0.0.acfgs	0.945932984
ssl3_get_new_session_ticket	Tomato_by_Shibby	tomato-E4200USB-NVRAM60K-1.28.RT-MIPSR2-109-AIO	libssl.so.1.0.0.acfgs	0.945932984
ssl3_get_new_session_ticket	Tomato_by_Shibby	tomato-E1550USB-NVRAM60K-1.28.RT-N5x-MIPSR2-110-Nocat-VPN	libssl.so.1.0.0.acfgs	0.945932984
ssl3_get_new_session_ticket	Tomato_by_Shibby	tomato-K26USB-1.28.RT-N5x-MIPSR2-115-PL-L600N	libssl.so.1.0.0.acfgs	0.945932984
ssl3_get_new_session_ticket	Tomato_by_Shibby	tomato-E1550USB-NVRAM60K-1.28.RT-N5x-MIPSR2-110-BT-VPN	libssl.so.1.0.0.acfgs	0.945932984
ssl3_get_new_session_ticket	Tomato_by_Shibby	tomato-E3000USB-NVRAM60K-1.28.RT-MIPSR2-108-PL-VPN	libssl.so.1.0.0.acfgs	0.945932984
ssl3_get_new_session_ticket	Tomato_by_Shibby	tomato-E1550USB-NVRAM60K-1.28.RT-N5x-MIPSR2-110-Mega-VPN	libssl.so.1.0.0.acfgs	0.945932984
ssl3_get_new_session_ticket	Tomato_by_Shibby	tomato-E1200v2-NVRAM64K-1.28.RT-N5x-MIPSR2-108-PL-Max	libssl.so.1.0.0.acfgs	0.945932984
ssl3_get_new_session_ticket	Tomato_by_Shibby	tomato-K26USB-1.28.RT-MIPSR1-109-Mega-VPN	libssl.so.1.0.0.acfgs	0.945932984
ssl3_get_new_session_ticket	Tomato_by_Shibby	tomato-E3000USB-NVRAM60K-1.28.RT-MIPSR2-109-Big-VPN	libssl.so.1.0.0.acfgs	0.945932984
ssl3_get_new_session_ticket	Tomato_by_Shibby	tomato-E4200USB-NVRAM60K-1.28.RT-MIPSR2-108-PL-Nocat-VPN	libssl.so.1.0.0.acfgs	0.945932984
ssl3_get_new_session_ticket	Tomato_by_Shibby	tomato-Netgear-3500Lv2-K26USB-1.28.RT-N5x--110-ND-AIO	libssl.so.1.0.0.acfgs	0.945932984
ssl3_get_new_session_ticket	Tomato_by_Shibby	tomato-E4200USB-NVRAM60K-1.28.RT-MIPSR2-109-Nocat-VPN	libssl.so.1.0.0.acfgs	0.945932984

Among top 50: **42** out of **50** are confirmed vulnerabilities

Previous work: **10**/**50**

Takeaways

Message 3. Deep learning approaches can be not only **more effective**, but also **more efficient** in learning embedding representations for binary **programs**.

Message 4. **Program analysis** can be a **novel application domain** of deep learning techniques toward a more secure world.