SAFE: Self Attentive Function Embedding for binary similarity

Luca Massarelli
26/04/2019

CIS SAPIENZA
RESEARCH CENTER FOR CYBER INTELLIGENCE AND INFORMATION SECURITY

Outline

- Problem Definition
- Background: Word2Vec
- Background: Recurrent Neural Network
- SAFE: Self Attentive Function Embedding
- YARASAFE

Problem Definition

Consider this simple function:

```
int main() {
    int a,b,c;

    a = 1;
    b = 5;

    c = a + b;

}
```


Problem Definition

Compiled with gcc:

```
local_ch @ rbp-0xc
local_8h @ rbp-0x8
local_4h @ rbp-0x4
3c      55              push rbp
3d      4889e5          mov rbp, rsp
90      c745fc010000.   mov dword [local_4h], 1
97      c745f8050000.   mov dword [local_8h], 5
9e      8b55fc          mov edx, dword [local_4h]
a1      8b45f8          mov eax, dword [local_8h]
a4      01d0            add eax, edx
a6      8945f4          mov dword [local_ch], eax
a9      b800000000      mov eax, 0
ae      5d              pop rbp
af      c3              ret
```

Compiled with clang:

```
local_ch @ rbp-0xc
local_8h @ rbp-0x8
local_4h @ rbp-0x4
90      55              push rbp
91      4889e5          mov rbp, rsp
94      31c0            xor eax, eax
96      c745fc010000.   mov dword [local_4h], 1
9d      c745f8050000.   mov dword [local_8h], 5
a4      8b4dfc          mov ecx, dword [local_4h]
a7      034df8          add ecx, dword [local_8h]
aa      894df4          mov dword [local_ch], ecx
ad      5d              pop rbp
ae      c3              ret
```

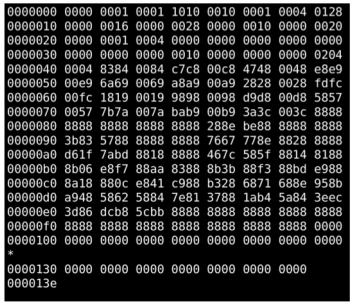
Binary Similarity

How to automatically find if two binary functions have been compiled from the same source code?

- Symbolic execution
- Pattern Matching
- **Machine Learning**

Technological Applications:

- Vulnerability detection in firmware:



Is there any similar function?

Embedded system

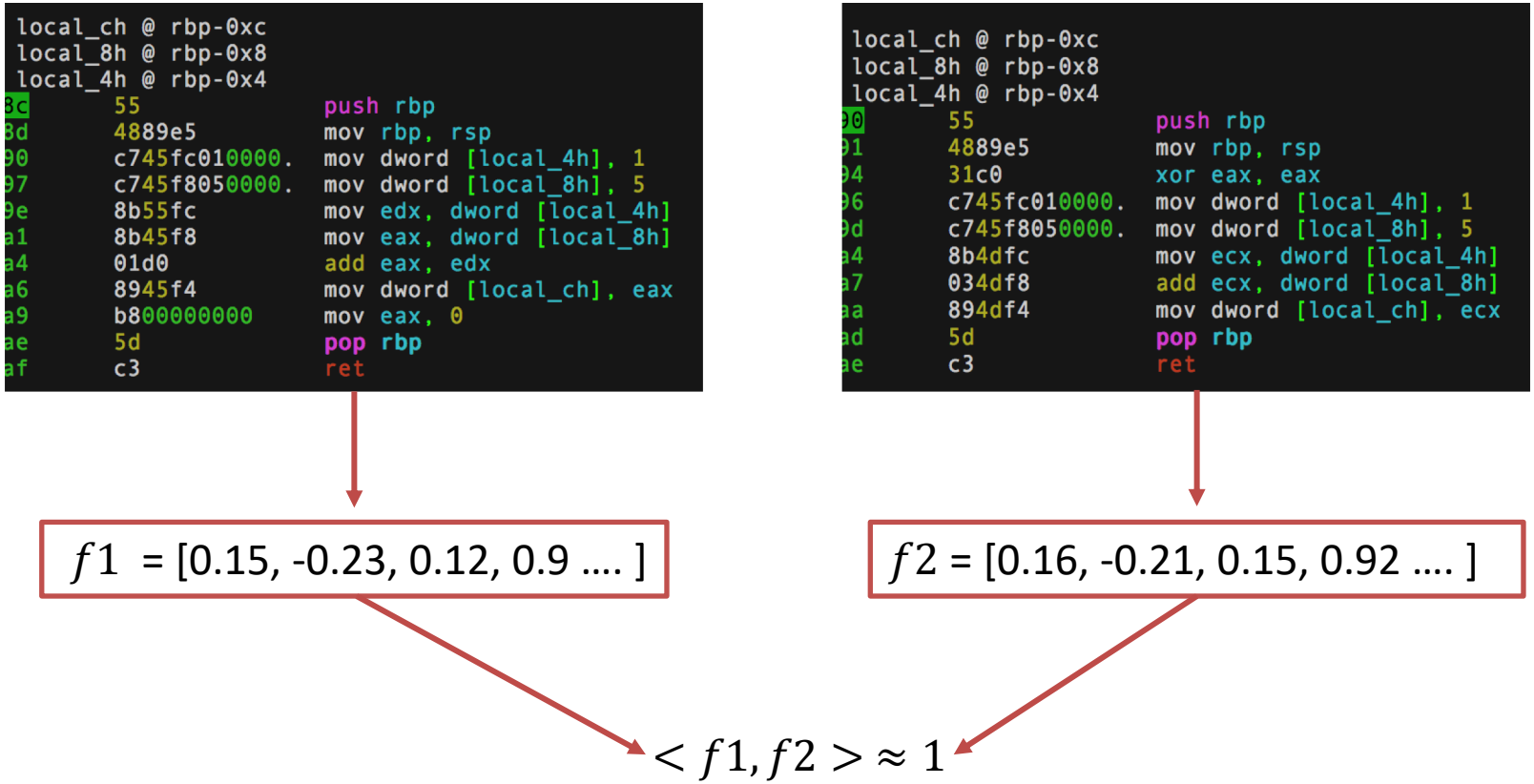
Binary firmware

Binary code of function affected by heartbleed

Technological Applications

- Functions Search Engine;
  - Copyright infringment;
  - Reverse Engineering Aid (Analysis of malware);
- Semantic Classification;

Function Embedding



CIS Sapienza
Research Center for Cyber Intelligence and Information Security

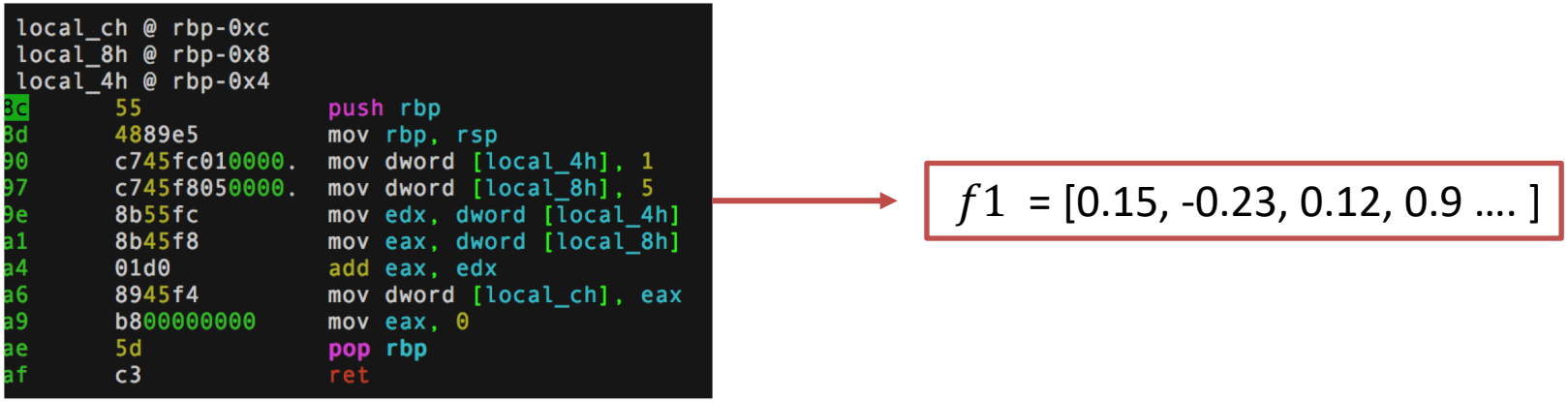
Why embedding?

- Fast computation of similarity and massively parallelizable;
- Privacy preserving;
- Easy to share;

NVIDIA CUDA

Similarity Preserving Functions Embedding

- The binary similarity problem can be reduced to compute similarity – preserving functions embeddings.



How to compute embeddings?

- Graph Clustering[1];
- Symbolic execution of small part of code[2,3];

1. Q. Feng, R. Zhou, C. Xu, Y. Cheng, B. Testa, and H. Yin, "Scalable graph-based bug search for firmware images," in *Proceedings of the 23rd ACM SIGSAC Conference on Computer and Communications Security, (CCS)*. ACM, 2016, pp. 480–491.
2. Y. David and E. Yahav, "Tracelet-based code search in executables," in *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation, (PLDI)*, 2014, pp. 349–360.
3. Y. David, N. Partush, and E. Yahav, "Statistical similarity of binaries," in *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, (PLDI)*, 2016, pp. 266–280.

How to compute embeddings?

- Graph Embedding Neural Networks[1,2];
- Unsupervised representation learning[3]
- Recurrent Neural Networks[4];

1. X. Xu, C. Liu, Q. Feng, H. Yin, L. Song, and D. Song, "Neural network- based graph embedding for cross-platform binary code similarity detec- tion," in *Proceedings of the 24th ACM SIGSAC Conference on Computer and Communications Security, (CCS)*, 2017, pp. 363–376
2. Baldoni, R., Di Luna, G. A., Massarelli, L., Petroni, F., & Querzoni, L. "Unsupervised Features Extraction for Binary Similarity Using Graph Embedding Neural Networks," *arXiv preprint arXiv:1810.09683, 2018.*
3. S. H. Ding, B. C. Fung, and P. Charland, "Asm2vec: Boosting static representation robustness for binary clone search against code obfuscation and compiler optimization," in *(to Appear) Proceedings of 40th Symposium on Security and Privacy, (SP)*, 2019.
4. Massarelli, L., Di Luna, G. A., Petroni, F., Querzoni, L., & Baldoni, R. "SAFE: Self-Attentive Function Embeddings for Binary Similarity," *arXiv preprint arXiv:1811.05296, 2018.*

The word2vec model

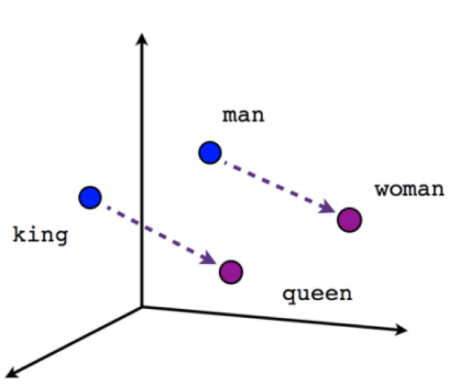
- A word representation model.
- Each word is first associated to a random embedding vector:

$$cat = [0.17, \quad 0.98, \quad -0.13, \quad \dots\ ]$$

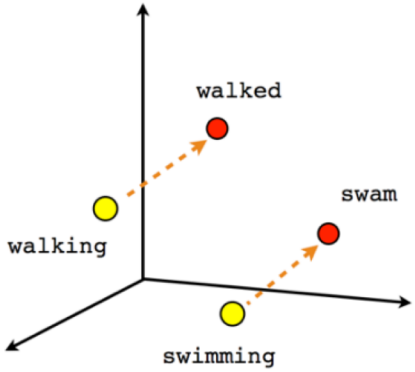
- During the training of the model, vectors are updated such that the network is able to predict the context of each target word.

How powerful is the skip-gram model?

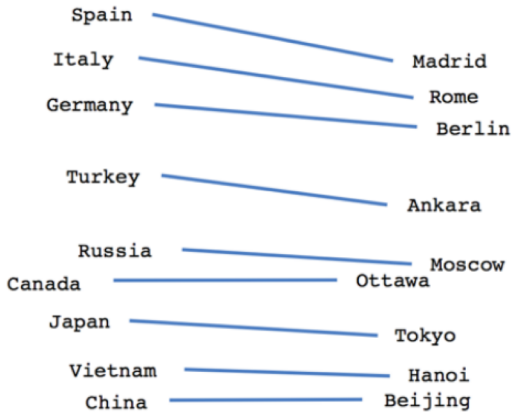
- Despite its simplicity the model is very powerful, as it is able to capture semantic relationship between words:



Male-Female



Verb tense



Country-Capital

Analogies

- The skip-gram model is evaluated using analogies:
  - $queen : king = man : ?$
  - $toronto : canada = rome : ?$
- Analogie can be computed with simple operations between words embedding:
  - $queen - king + man = ?$
  - $toronto - canada + rome = ?$

Word2Vec Results

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Mikolov, Tomas, et al. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Instructions2Vec

- We can apply the word2vec model to embedd assembly instructions: instruction2vec (i2v).
- Instructions need to be normalized before to fed into the i2v model:
  - White spaces are replaced with _
  - Contants greater than 5000 are replaced with «HIMM»
  - Memory locations greater than 5000 are replaced with «MEM»

```
mov rax 0x1b345 → mov_rax_HIMM
```

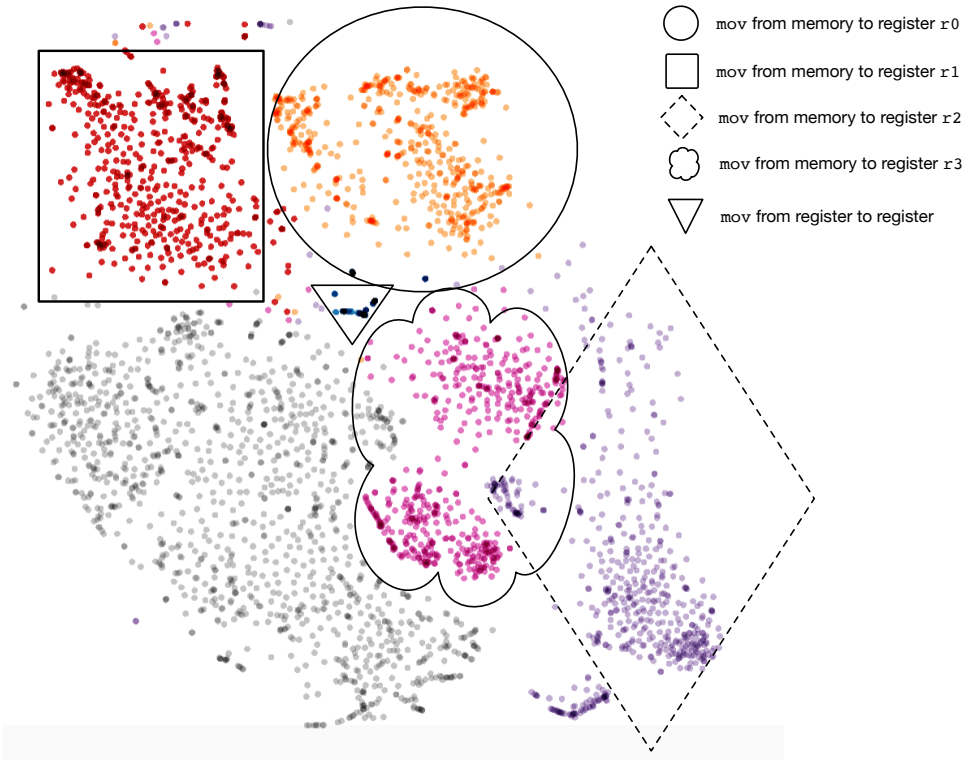

Analogies

- The instruction2vec model is able to answer to analogies:
  - `push_rcx : pop_rcx = push_rbp : ?`
  - `sub_rax_rbx : add_rax_rbx = sub_rax_rcx : ?`

Analogies

- The instruction2vec model is able to answer to analogies:
  - `push_rcx : pop_rcx = push_rbp : pop_rbp`
  - `sub_rax_rbx : add_rax_rbx = sub_rax_rcx : add_rax_rcx`

Instructions Clustering



Recurrent Neural Network (RNN)

- RNN are an obvious choiche when we need to handle sequences of data.

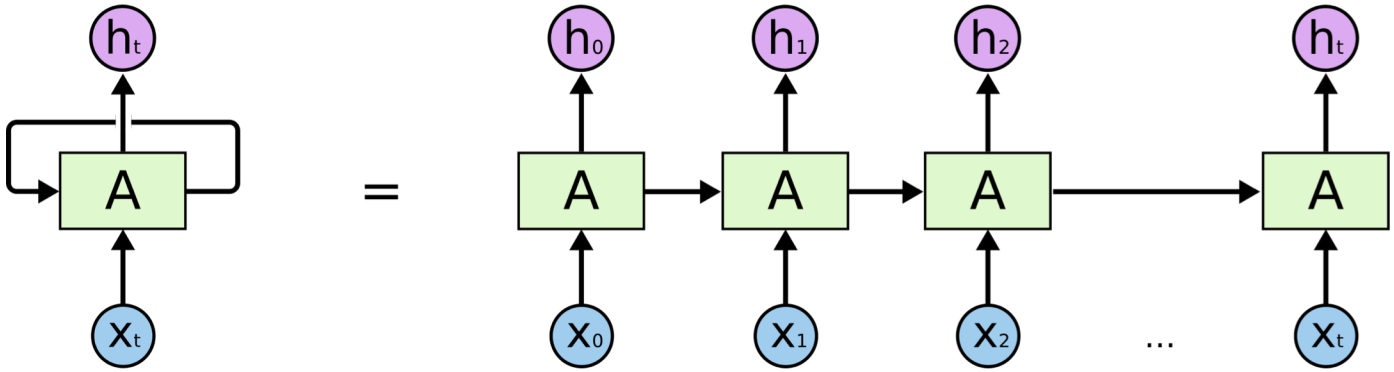


Image from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Network (RNN)

- Each cell takes as input the output of the previous cell and combine it with the input vector at time t.

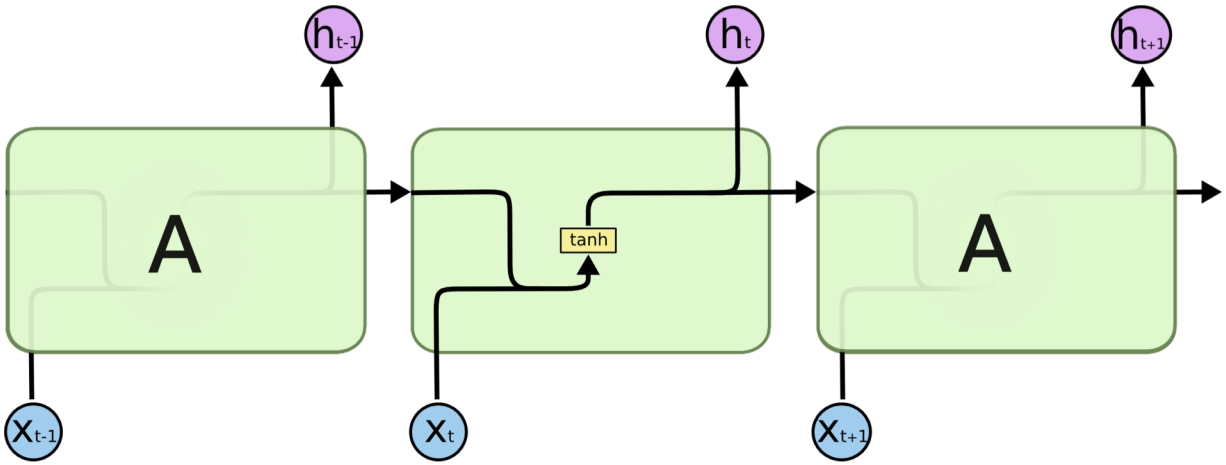
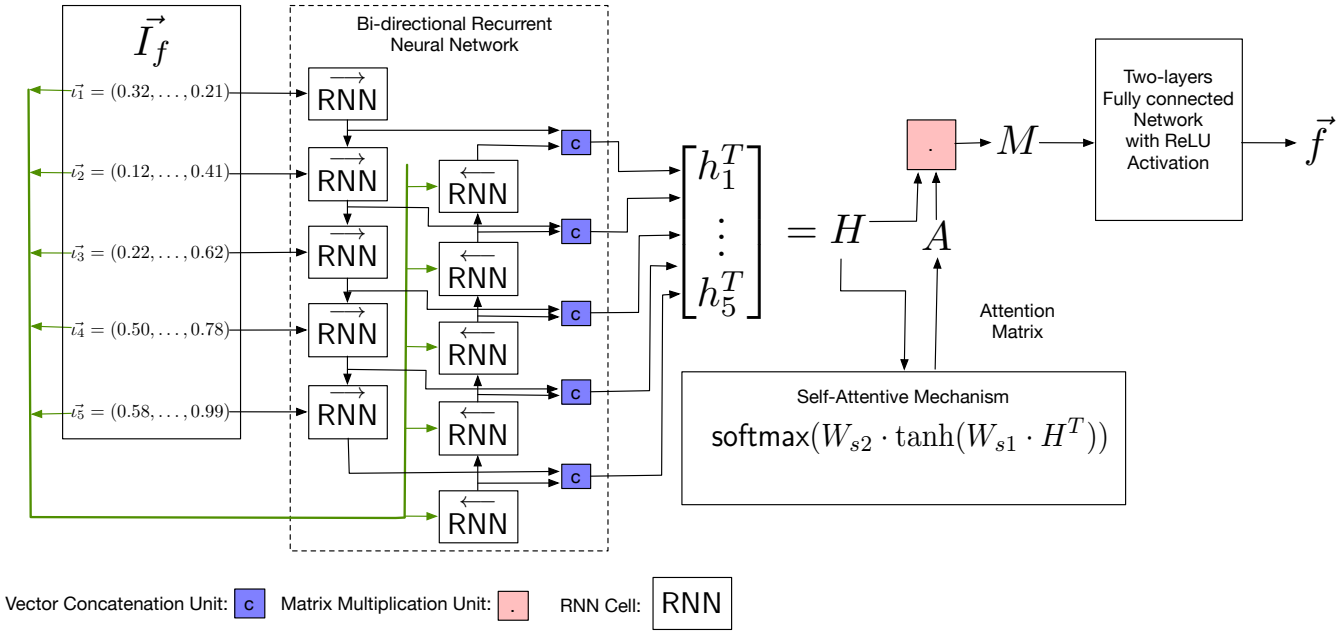


Image from: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Self Attentive RNN

- The output vectors of an RNN can be aggregated together using a self-attentive mechanism:



SAFE: Self Attentive Function Embedding (DIMVA 2019)

Luca Massarelli
Sapienza University Of Rome

Giuseppe Antonio Di Luna
Cini, National Cyber Security Lab

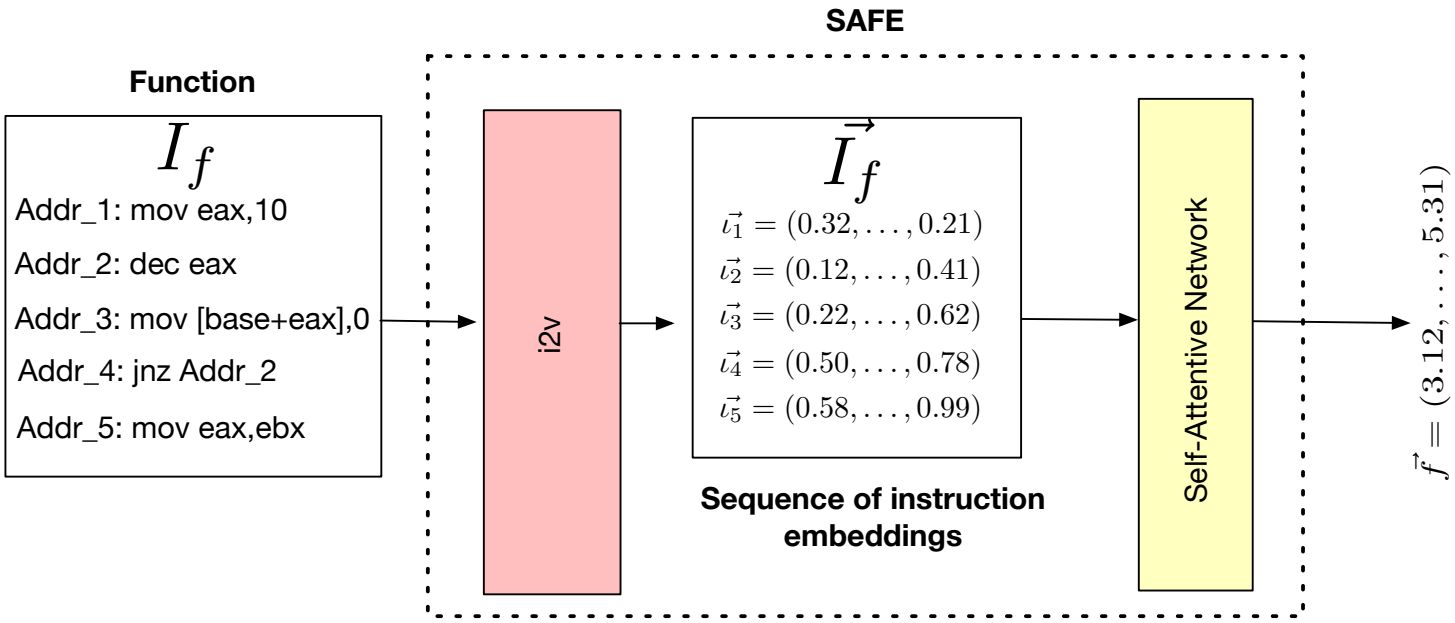
Fabio Petroni
Facebook AI Research

Leonardo Querzoni
Sapienza University Of Rome

Roberto Baldoni
Sapienza University Of Rome

SAFE: Self Attentive Function Embedding

- We use the sequence of instructions contained in binary functions as input for a self attentive network:

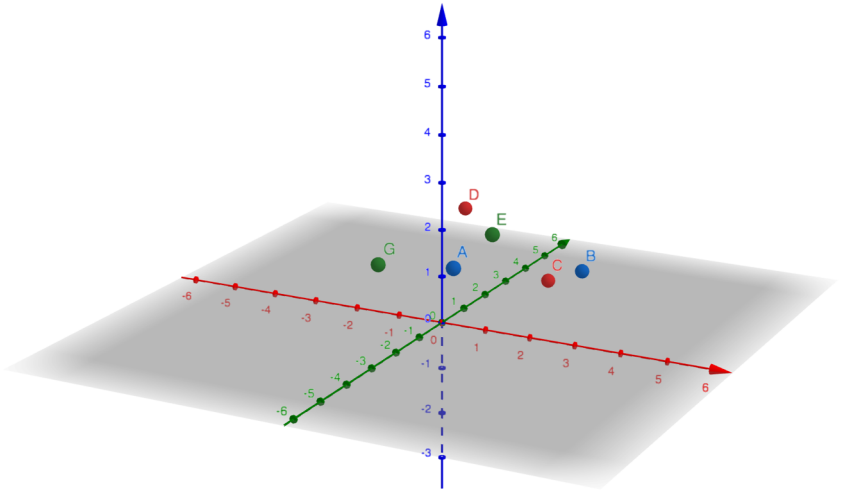


Training the network

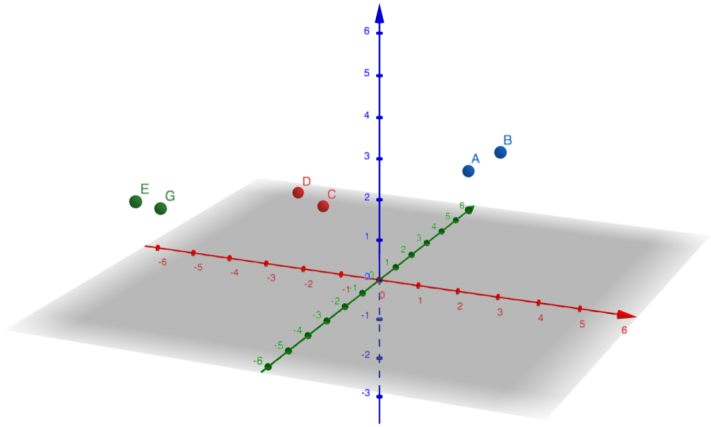
- The weights of the network are initialized randomly.
- If we do not perform the training we will be able already to embedd a binary function…
- … But the embedding will be not similarity-preserving!
- Two similar function will be embedd in two dissimilar vector

Training the network

Before Training



After Training

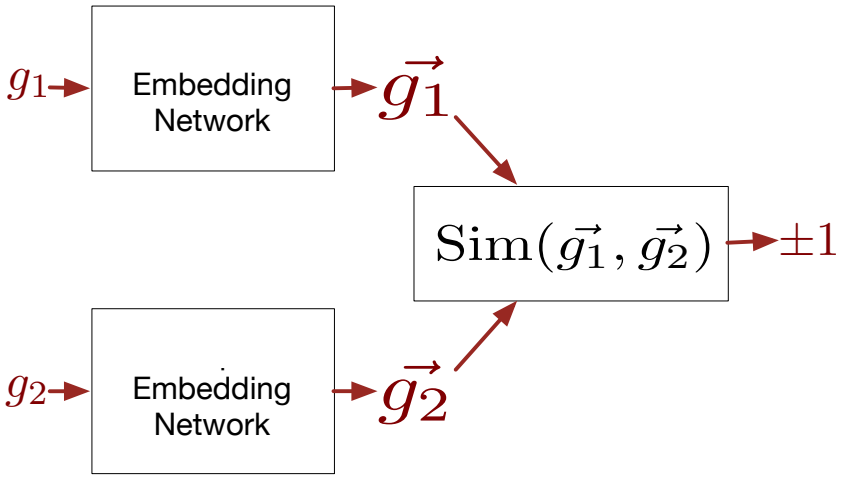


A,B,C,D,E,F are embedding of binary functions.

- Function represented by A is similar to function represented by B
- Function represented by C similar to function represented by D
- Function represented by E similar to function represented by G

Training the network

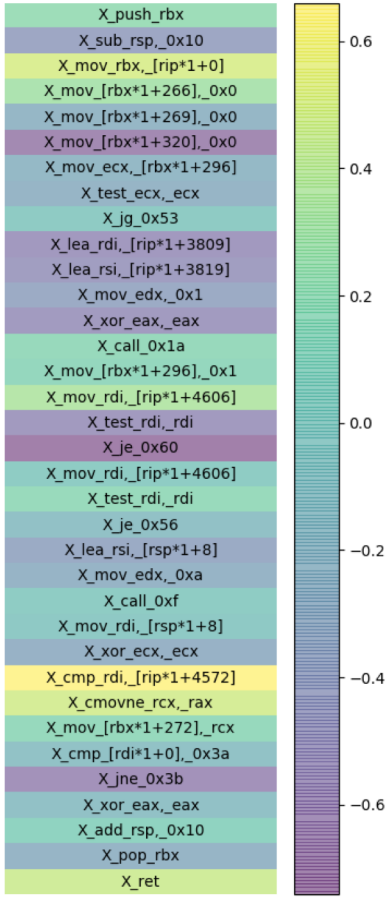
- To train the network we can use a siamese approach:



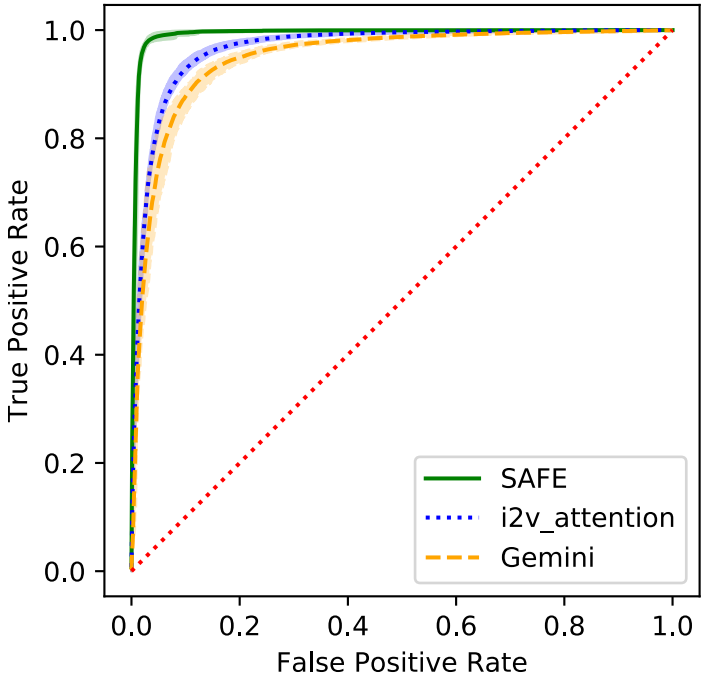
- We choose pairs of (g1,g2) such that we already know if they are similar or not.

Heatmap

- The self attention mechanism associate a different weight to each instructions.
- Some instructions results to be more important for similarity comparison.



Results are impressive!



- AUC:
  - SAFE: 0.99
  - I2v_attention: 0.96
  - Gemini (MFE): 0.95
- We tested SAFE on different task!

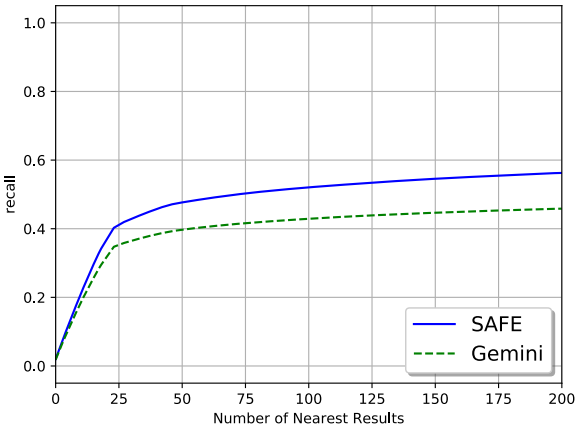
Task 1: Function Search Engine

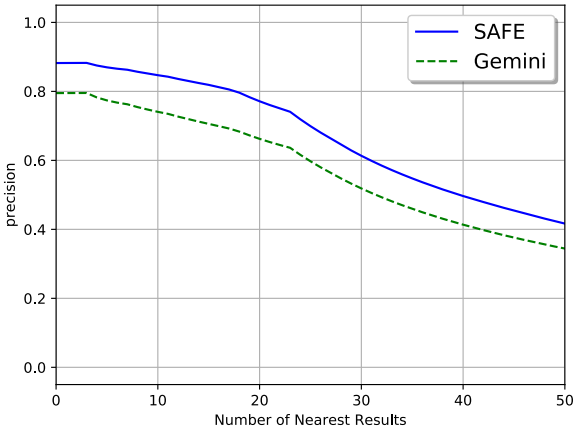
- We have a knowledge base (KB) of functions a query functions, we want to retrieve all functions in the KB similar to the query.
- Our knowledge base is composed by all functions disassembled from postgresql compiled with 12 different compilers and 4 optimizations level (600000 functions).
- For each function in the KB we use SAFE an ordered list of similar functions.

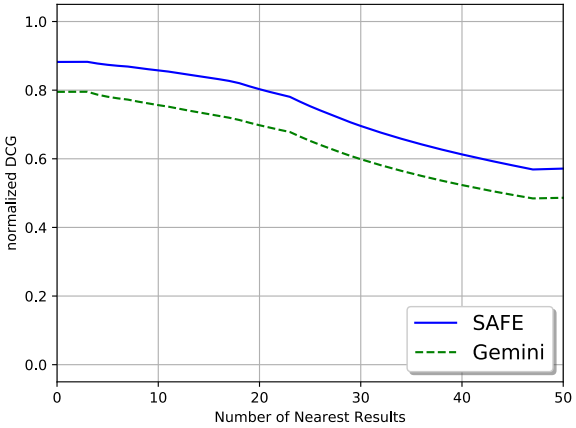
Task 1: Function Search Engine

- k is the number of results we want to retrieve
- The perfomance are evaluated with three measure:
  - **Recall@k**: Ratio of true similar functions retrieved over the minimum between all similar functions in the KB and k.
  - **Precision@k**: Ratio of true similar functions retrieved over k.
  - **Normalised Discounted Cumulative Gain**:
    - [0,1,0,0,1,1]
    - [1,1,1,0,0,0]
    - The second results is better than the first but with the same recall and precision!

Task 1: Function Search Engine







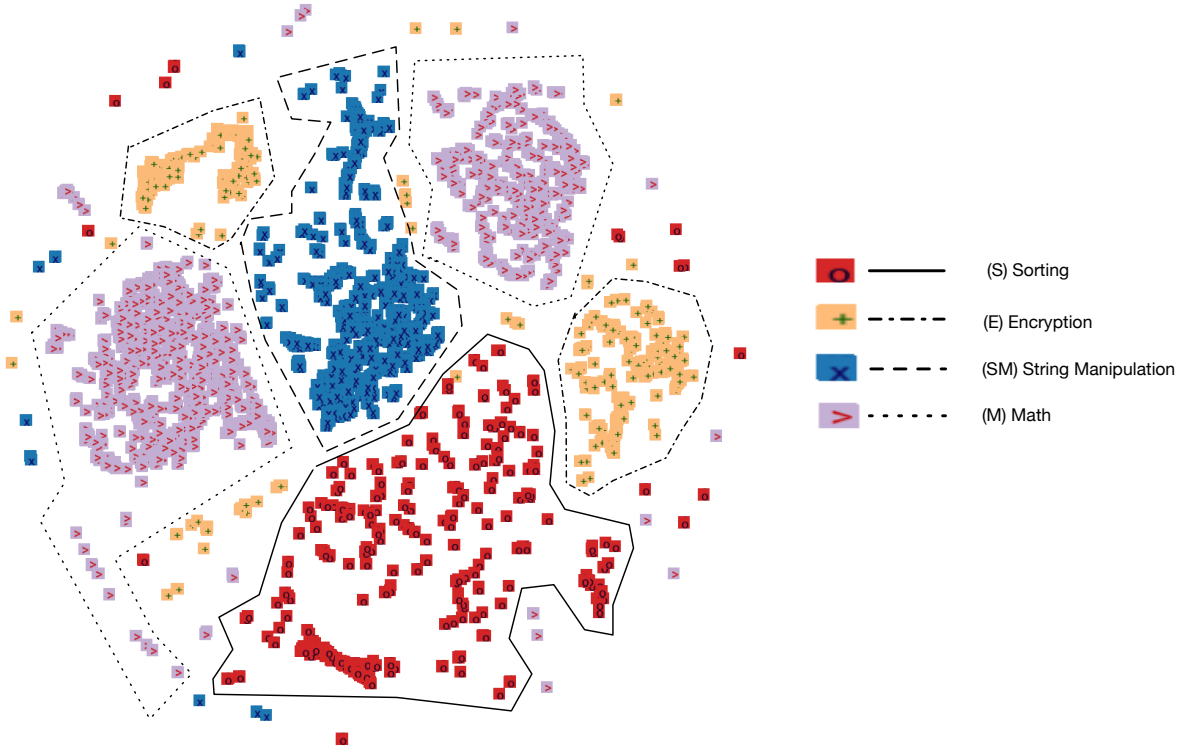
Task 3: Semantic Classification

- We create a dataset containing functions belonging to 4 different semantic classes:
  - Sorting
  - String Manipulation
  - Encryption
  - Math

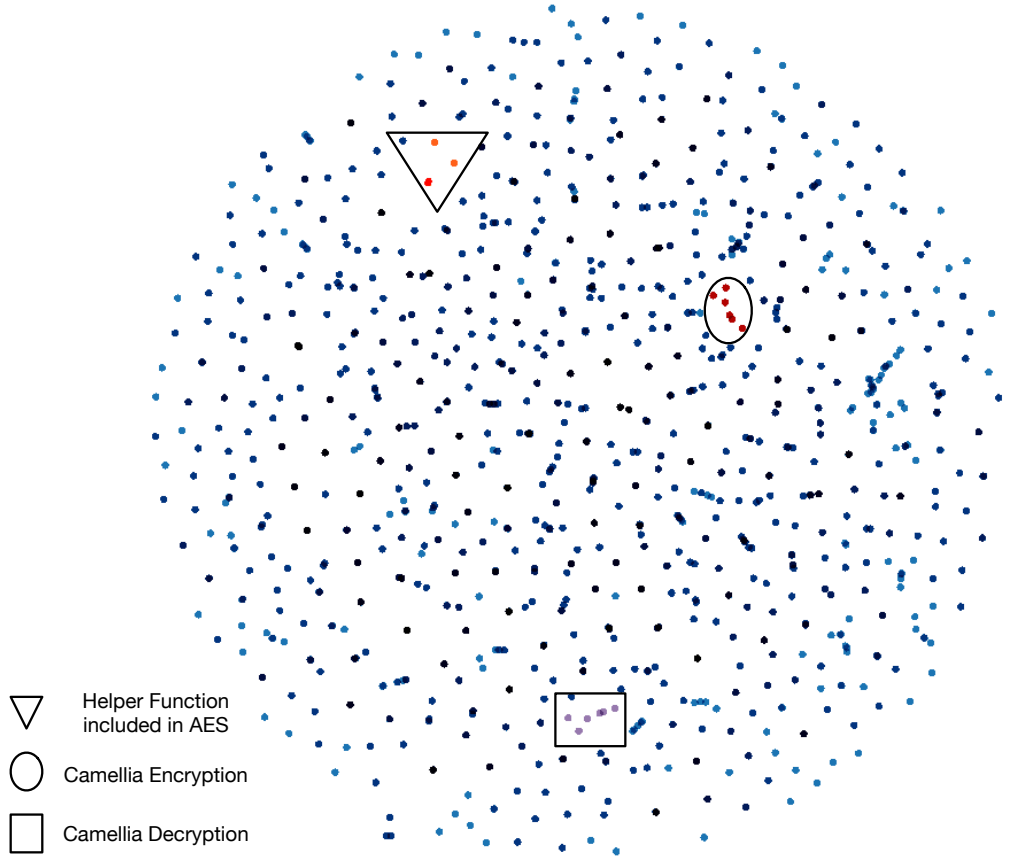
- We create their embeddings using SAFE and then we build an SVM classification model to infer the sematic of each function.

- We obtain 0.95 accuracy over cross-validation!

Task 3: Semantic Classification



Functions Clustering



Task 3: Applications

- One possible applications of task 3 is to find encryption functions inside malware.
- We try this on two different ransomware sample:
  - Teslacrypt:
    - 3372c1edab46837f1e973164fa2d726c5c5e17bcb888828ccd7c4dfcc234a370
  - Vipasana:
    - 0442cfabb3212644c4b894a7e4a7e84c00fd23489cc4f96490f9988e6074b6ab

TeslaCrypt

- SAFE identify 7 functions as vulnerables:
  - 0x41e900,
  - 0x420ec0,
  - 0x4210a0,
  - 0x4212c0,
  - 0x421665,
  - 0x421900,
  - 0x4219c0

If you want to play…

- https://github.com/gadiluna/SAFE

YARA

- YARA is a pattern matching engine used all over the world for malware hunting[1].

```
rule silent_banker : banker
{
    meta:
        description = "This is just an example"
        thread_level = 3
        in_the_wild = true

    strings:
        $a = {6A 40 68 00 30 00 00 6A 14 8D 91}
        $b = {8D 4D B0 2B C1 83 C0 27 99 6A 4E 59 F7 F9}
        $c = "UVODFRYSIHLNWPEJXQZAKCBGMT"

    condition:
        $a or $b or $c
}
```

1) https://virustotal.github.io/yara/

YARASAFE

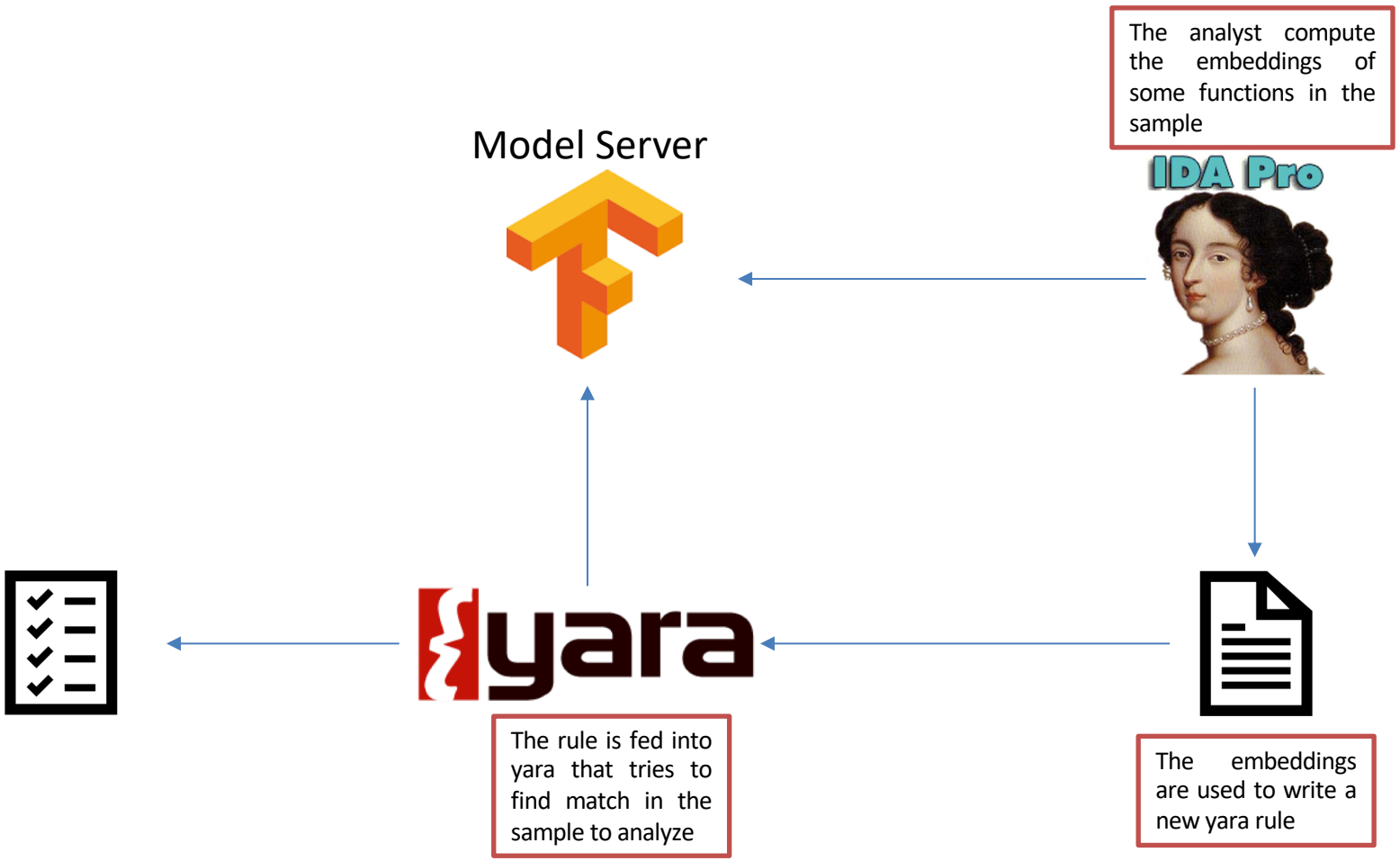
- Writing yara is a tedious process, you have to take into consideration that the code could slightly change with recompilation…
- Why not use SAFE embeddings as a yara signature?

YARASAFE

- Three main parts:
  - IDA Pro Plugin
  - YARASAFE Module
  - Tensorflow Server

https://github.com/lucamassarelli/yarasafe

YARASAFE


The analyst compute the embeddings of some functions in the sample
Model Server
IDA Pro
yara
The rule is fed into yara that tries to find match in the sample to analyze
The embeddings are used to write a new yara rule

The yarasafe YARA module

- The yara module uses as disassembler radare2.
- We embedded python interpreter into yara in order to easily use different disassembler like IDA or GHIDRA!

DEMO TIME