DeepMem: Learning Graph Neural Network Models for Fast and Robust Memory Forensic Analysis

Wei Song, Heng Yin, Chang Liu, Dawn Song

University of California

CCS 2018

Summarize the paper

- Problem
  - In memory forensics, it is important to identify kernel data structures from memory dumps
  - Existing approaches have several limitations
    - vulnerable to DKOM attacks, not scalable or efficient, and dependent on domain knowledge of OS
- Contribution
  - Propose a graph-based deep learning approach, DeepMem
    - Devise a graph representation of raw memory
    - Propose a graph neural network model (embedding network and classifier network)
    - Propose a weighted voting mechanism for object detection
- Result
  - DeepMem achieves high precision and recall rate for identifying kernel objects
  - DeepMem is efficient and robust against pool tag manipulation and DKOM process hiding

Memory object detection

- Goal: to search and identify kernel objects in raw memory dumps

- Let $C = \{c_1, c_2, \dots\}$ be the set of kernel data structure types in OS
  - _EPROCESS, _ETHREAD, …
- Given a raw memory dump as input,
  the output is defined as a set of kernel objects $O = \{o_1, o_2, \dots\}$,
  where each object is denoted as a pair $o_i = (addr_i, c_i)$, $c_i \in C$
  - $addr_i$ is the address of the first byte of the object in kernel space
  - c_i is the type of the kernel object

Existing techniques

- Data structure traversal
  - Identify a root object and follow the pointers defined in this object
  - Efficient: we can quickly find more objects by just following pointers
  - Not robust: attackers may modify the pointers to hide important objects, known as Direct Kernel Object Manipulation (DKOM) attacks
- Signature scan
  - Scan the entire memory snapshot for objects that satisfy a unique pattern
  - More robust against DKOM attacks: it does not depend so much on pointers
  - Inefficient and not scalable: it has to search the entire memory snapshot for one kind of objects using one signature
- Both approaches require precise knowledge of data structures and depend on specific versions of the SW or the OS

Goals

- Not rely on the knowledge of OS
- Achieve high efficiency and scalability
- Tolerate content and pointer manipulation of attackers in DKOM attacks

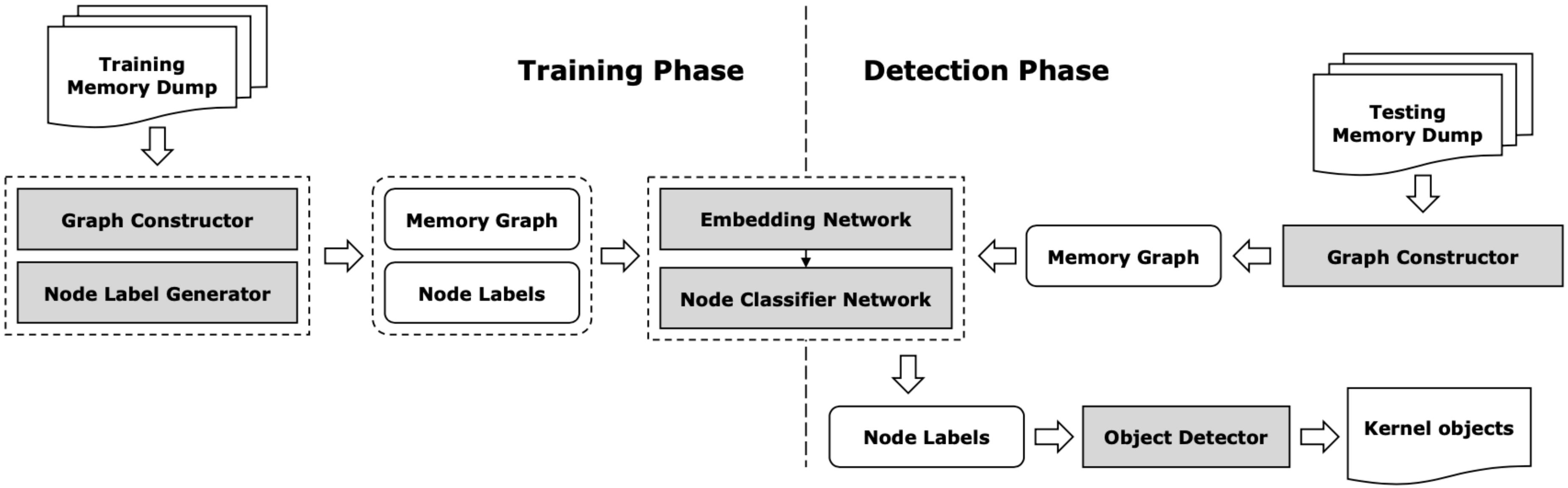
Insight

- Bottleneck of previous approaches is the rule-based search scheme
  - The rules can be hard to construct in the first place
  - The rules cannot easily adapt to an unknown OS and a new version
  - The rules cannot tolerate malicious attackers that attempt to deliberately violate these rules

- New approach should
  - learn the intrinsic features of an object that are stable across OS versions and resilient against malicious modifications
  - be able to detect these objects in a scalable manner

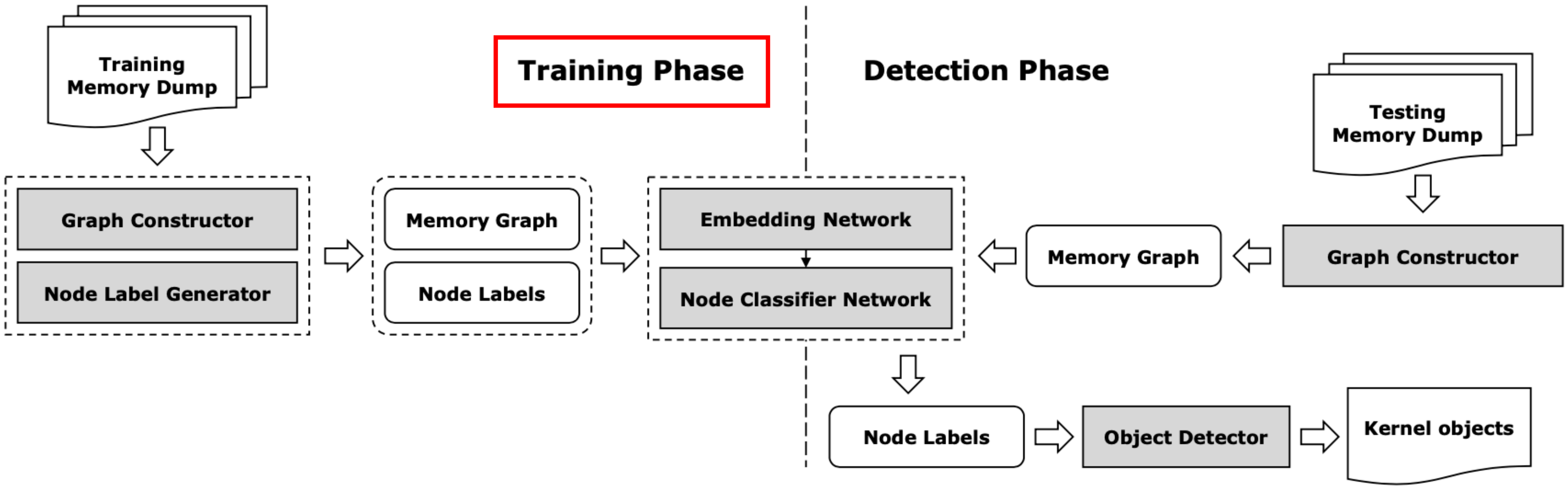
Overview of DeepMem

- DeepMem: a graph-based kernel object detection approach



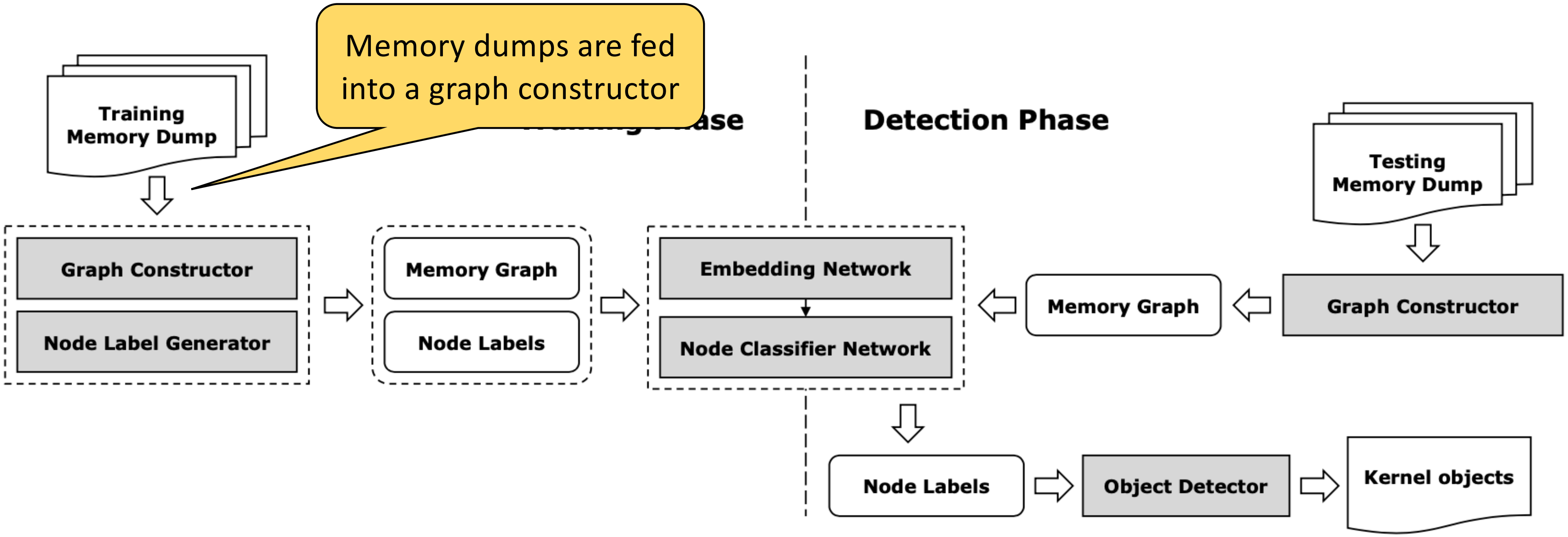
Overview of DeepMem

- DeepMem: a graph-based kernel object detection approach



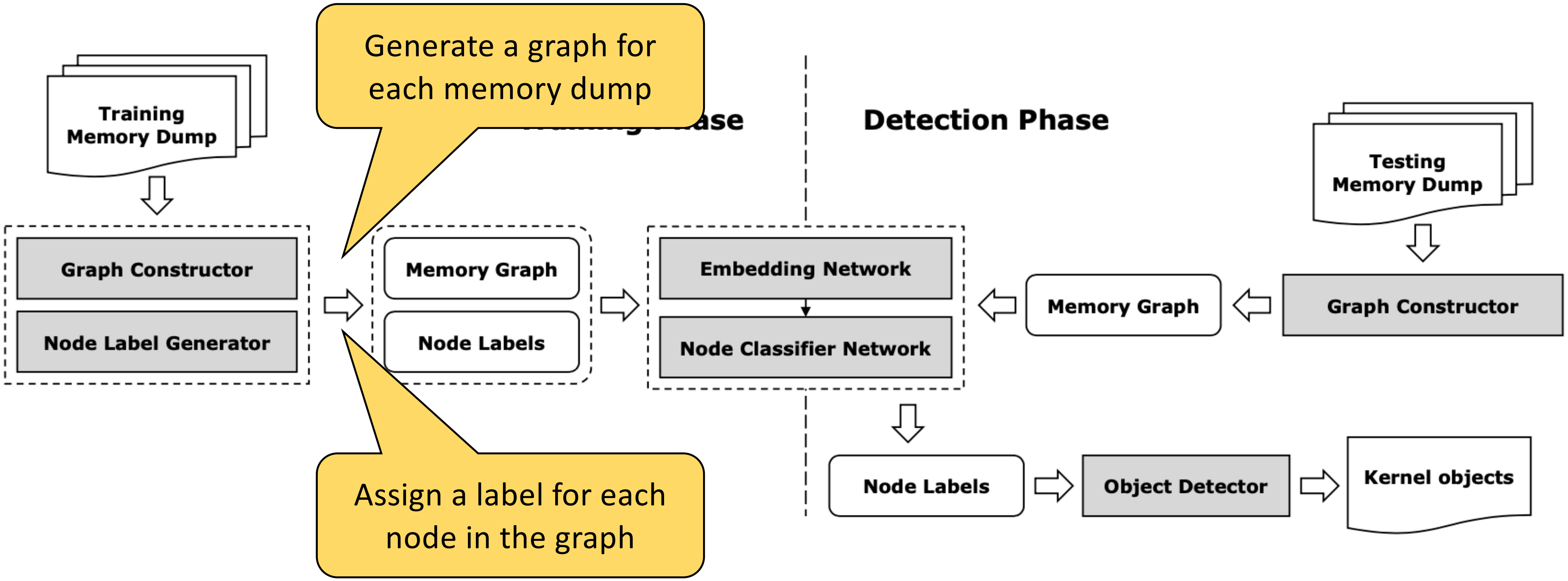
Overview of DeepMem

- DeepMem: a graph-based kernel object detection approach



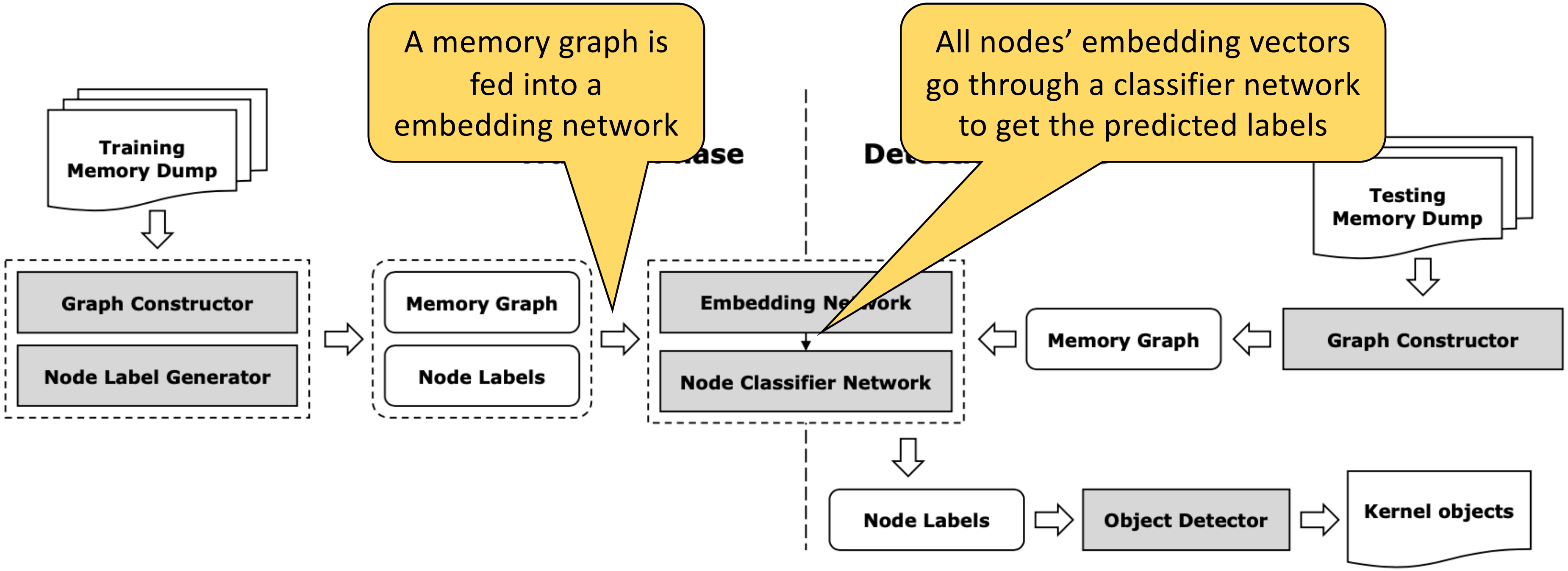
Overview of DeepMem

- DeepMem: a graph-based kernel object detection approach



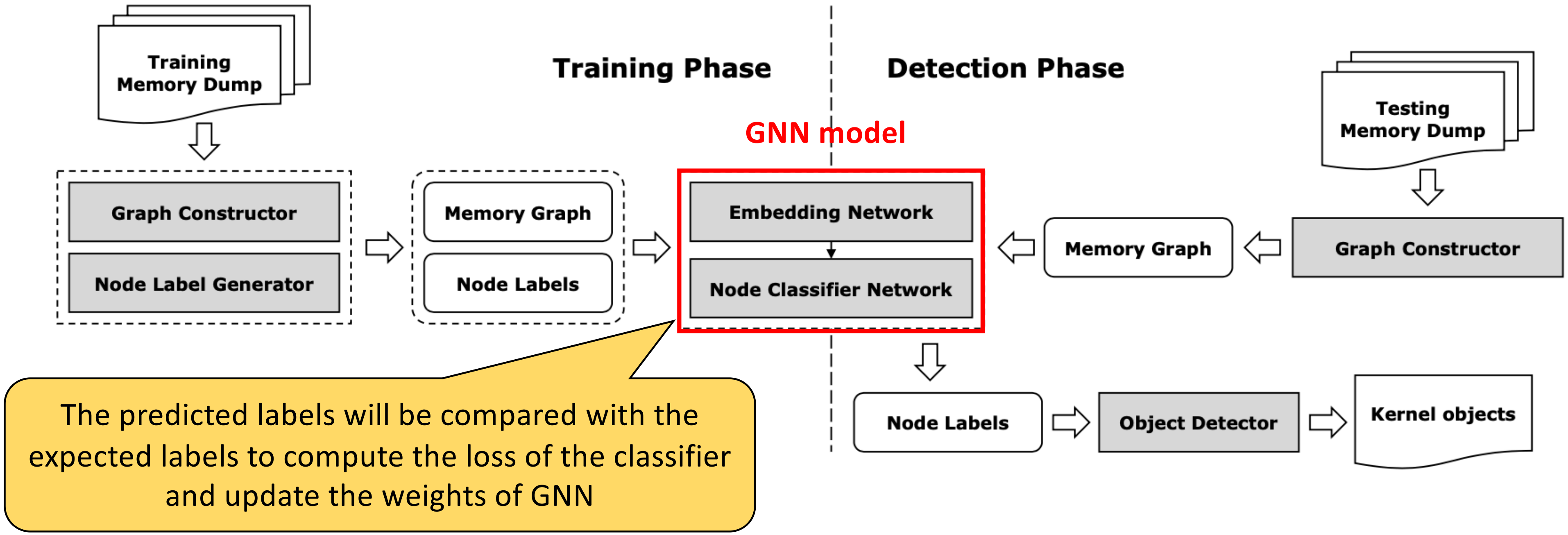
Overview of DeepMem

- DeepMem: a graph-based kernel object detection approach



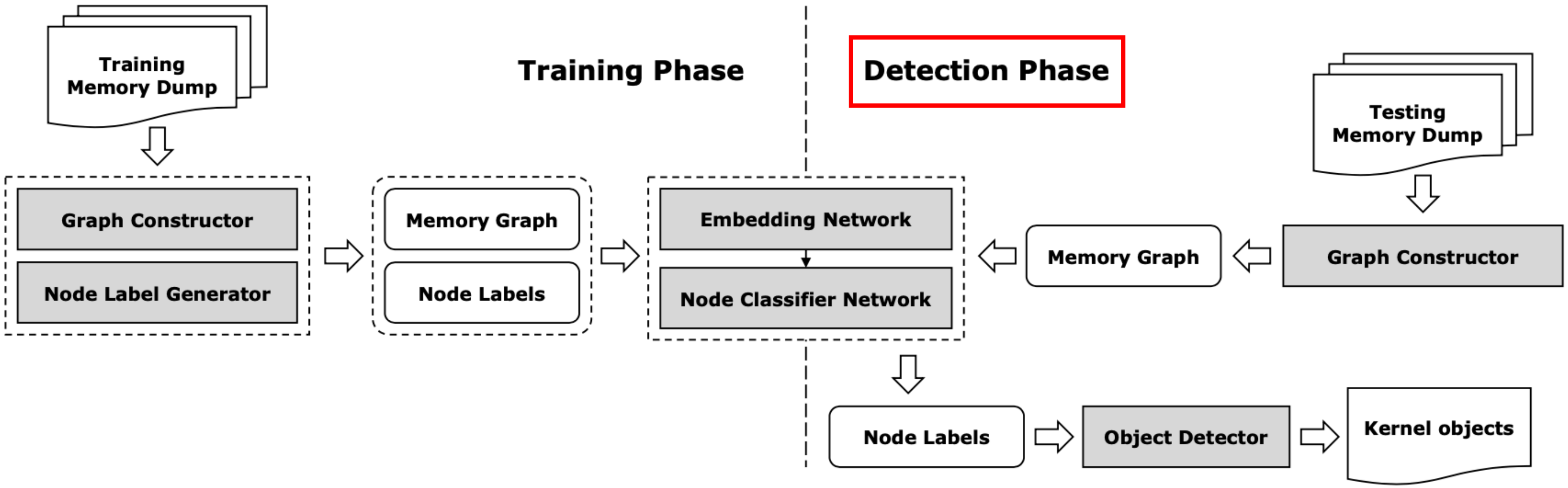
Overview of DeepMem

- DeepMem: a graph-based kernel object detection approach



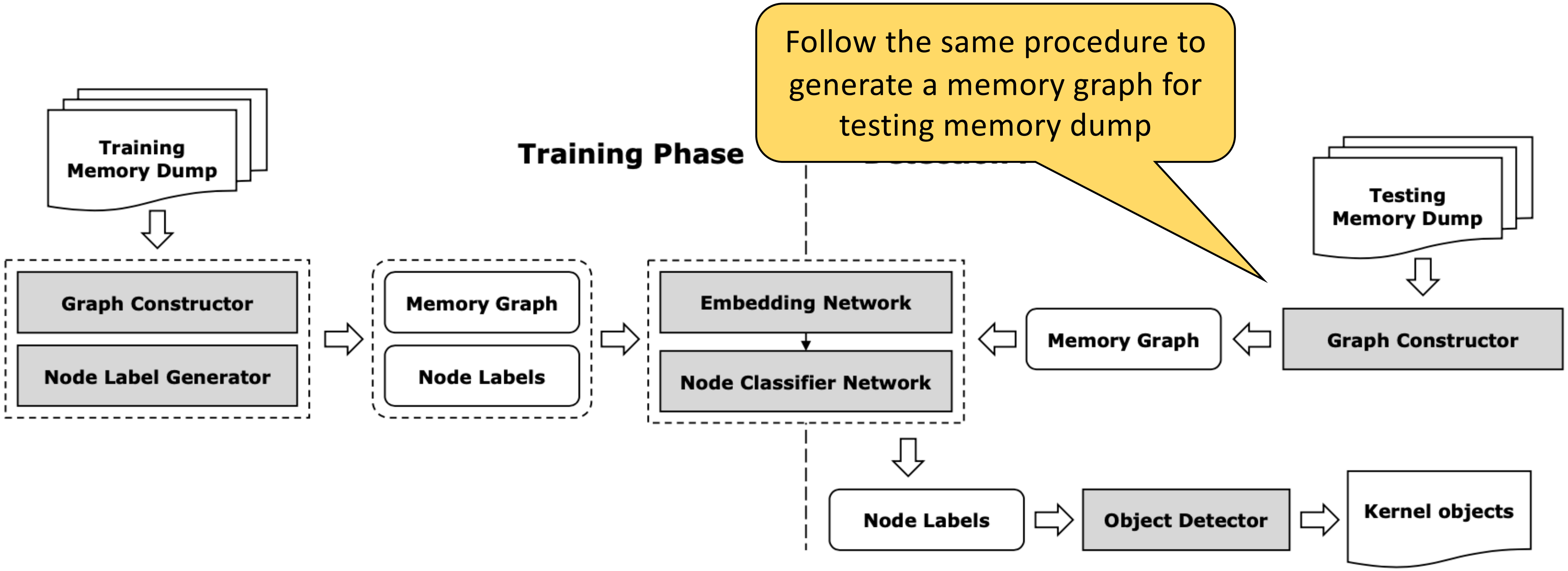
Overview of DeepMem

- DeepMem: a graph-based kernel object detection approach



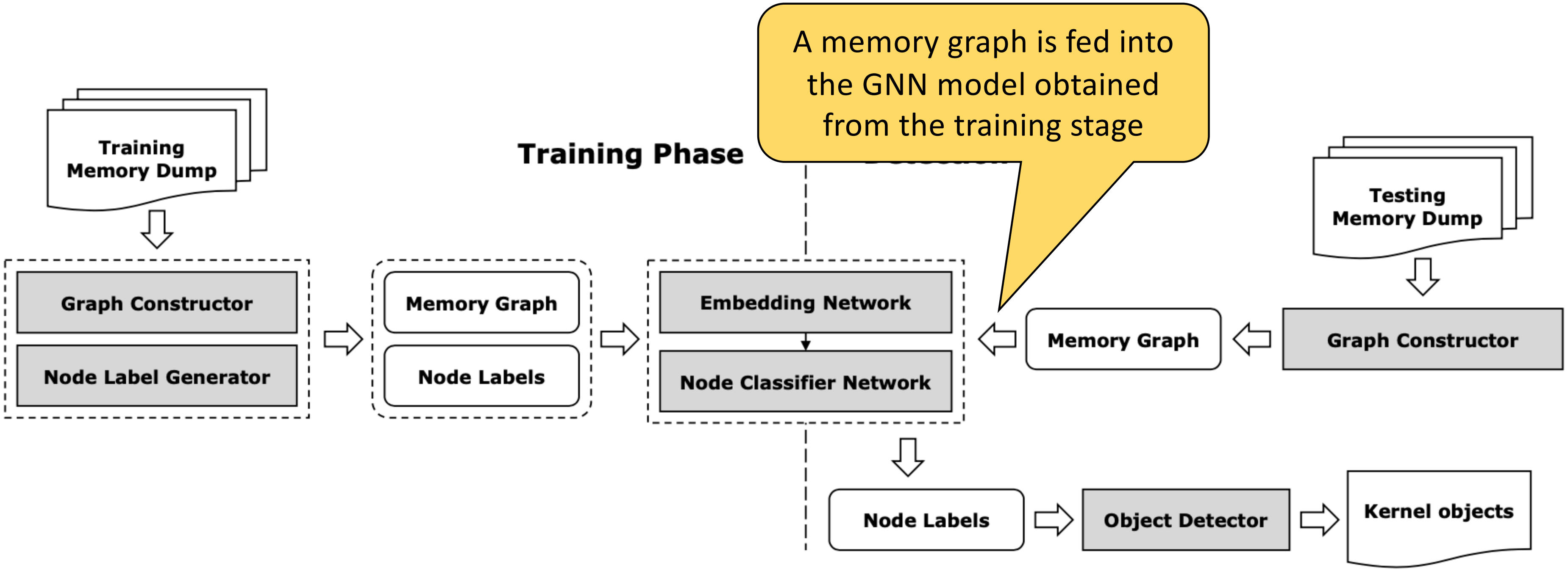
Overview of DeepMem

- DeepMem: a graph-based kernel object detection approach



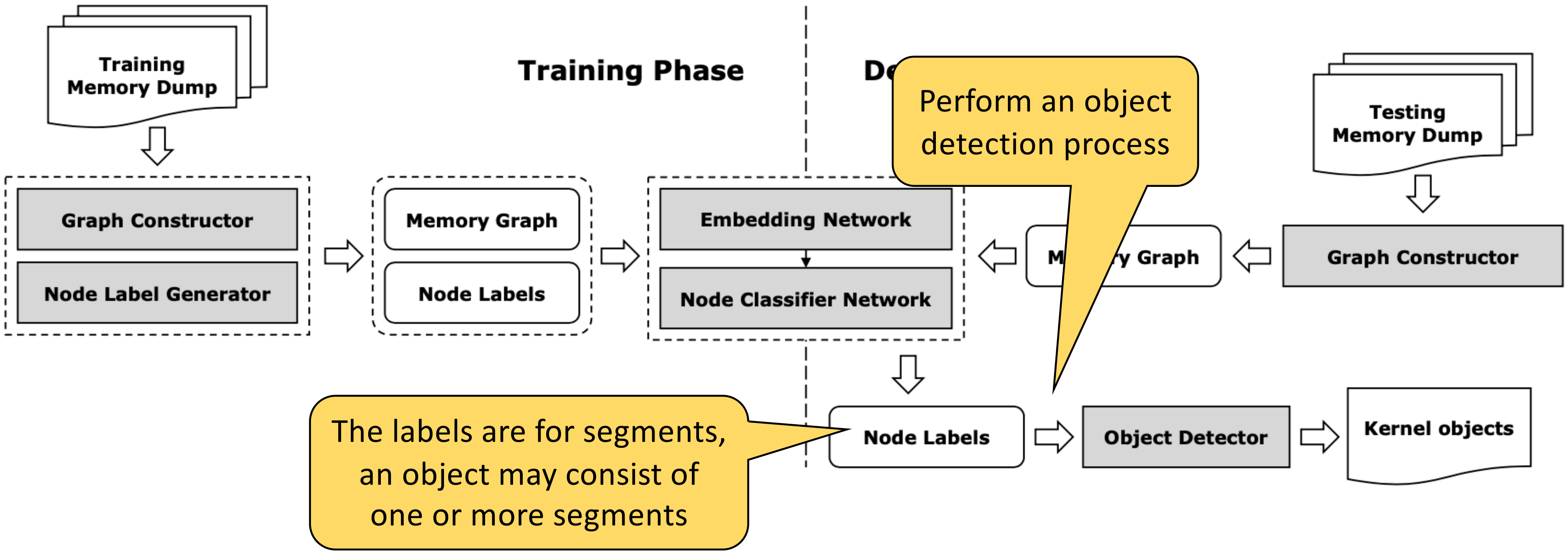
Overview of DeepMem

- DeepMem: a graph-based kernel object detection approach



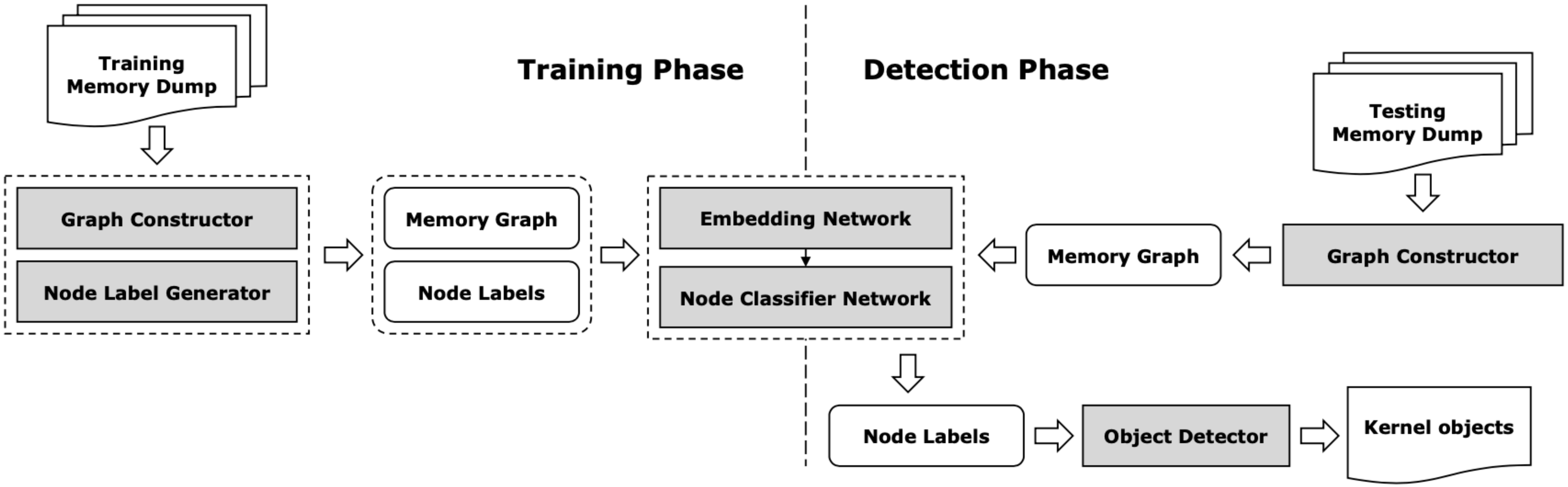
Overview of DeepMem

- DeepMem: a graph-based kernel object detection approach

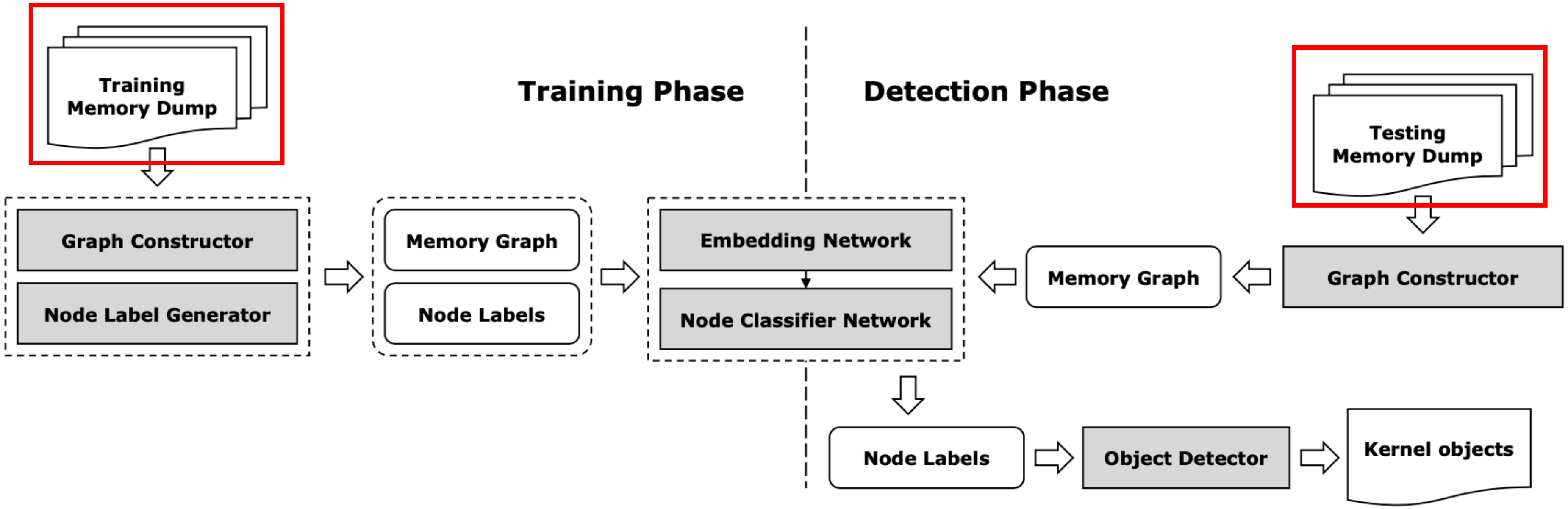


Overview of DeepMem

- DeepMem: a graph-based kernel object detection approach



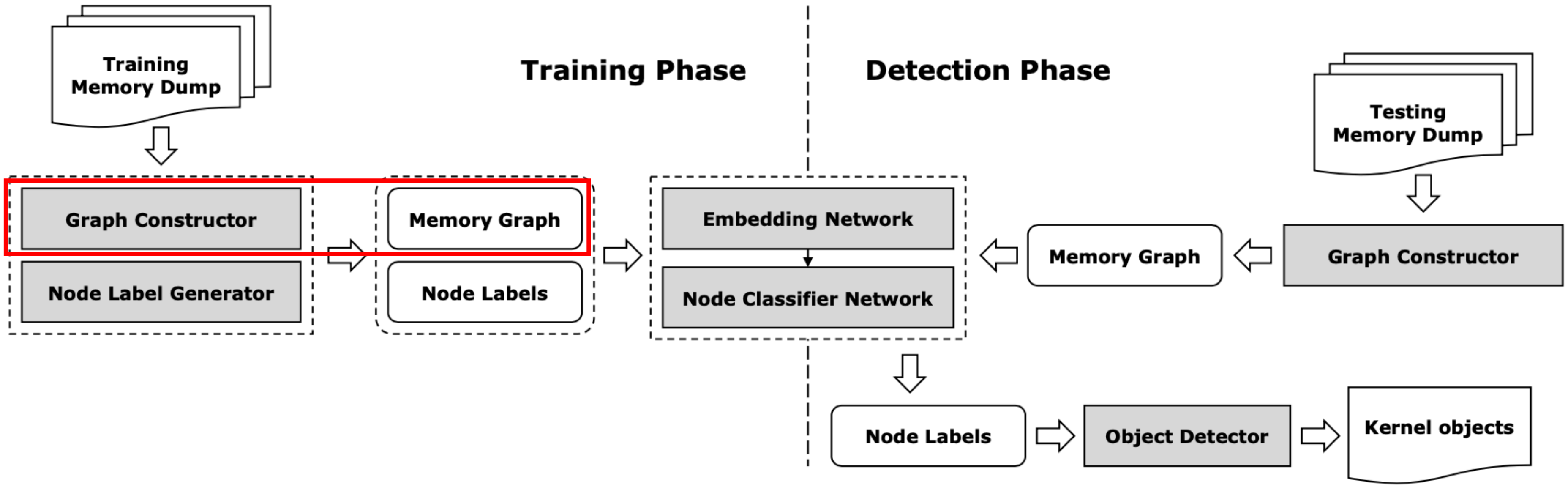
Design of DeepMem



Memory dumps collection

- Install Windows 7 SP1 virtual machine in the VirtualBox
- OS automatically starts 20 to 40 random actions
  - Start programs from a pool of the most popular programs
  - Open websites from a pool of the most popular websites
  - Open random PDF files, office documents, and picture files
- Wait for 2 minutes and dump the memory of the system
- Restart the vm and repeat until collect 400 memory dumps
  - Training dataset: 100 images
  - Validation dataset: 10 images
  - Testing dataset: 290 images

Design of DeepMem

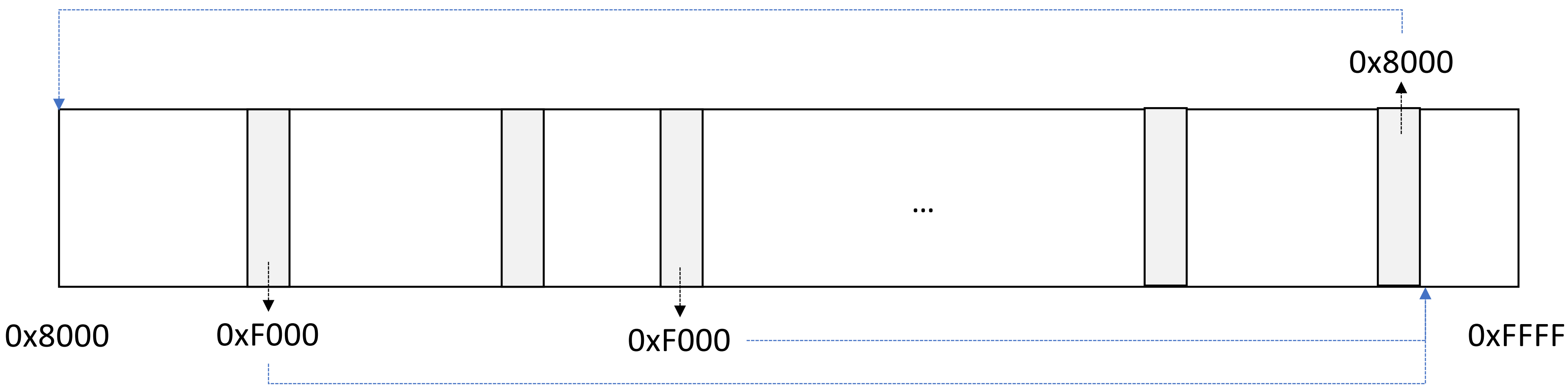


Memory Graph

- Scan all available memory pages in the kernel virtual space of memory dumps

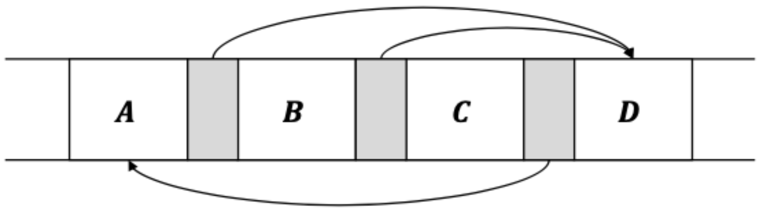
Memory Graph

- Scan all available memory pages in the kernel virtual space of memory dumps
- Locate all the pointers in the pages by finding all fields whose values fall into the range of kernel virtual space

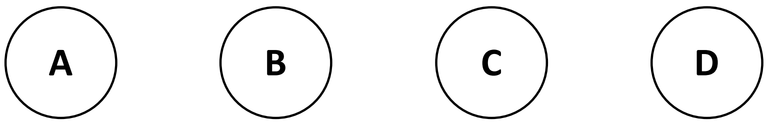


Memory Graph

- A memory graph is a directed graph $G = (N, E_{ln}, E_{rn}, E_{lp}, E_{rp})$,
  - For each segment between two pointers, we create a node
  - N is a node set, each node represents a segment of contiguous memory bytes between two pointer fields

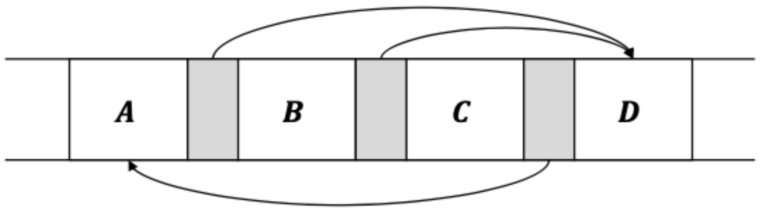


(a) Raw Memory

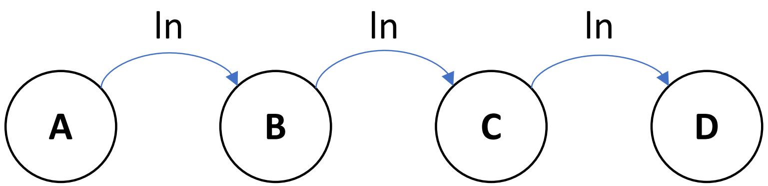


Memory Graph

- A memory graph is a directed graph $G = (N, E_{ln}, E_{rn}, E_{lp}, E_{rp})$,
  - For each node, we find its neighbor nodes and create an edge
  - E_{ln} is an edge set, each edge represents a directed edge from n_i to n_j, and n_i is left neighbor of n_j

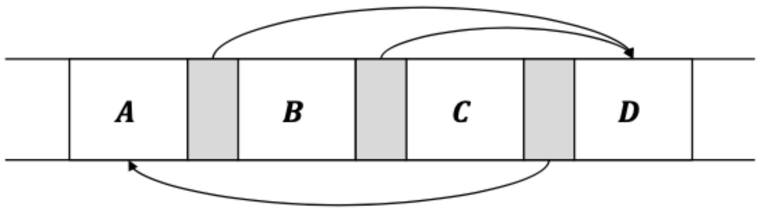


(a) Raw Memory

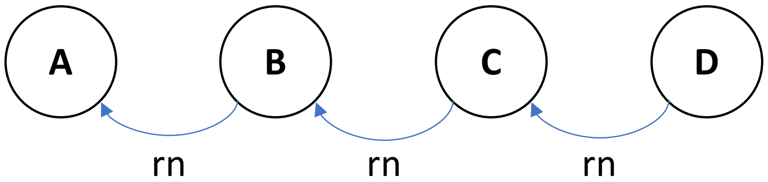


Memory Graph

- A memory graph is a directed graph $G = (N, E_{ln}, E_{rn}, E_{lp}, E_{rp})$,
  - For each node, we find its neighbor nodes and create an edge
  - E_{ln} is an edge set, each edge represents a directed edge from n_i to n_j, and n_i is left neighbor of n_j
  - E_{rn} is an edge set, each edge represents a directed edge from n_i to n_j, and n_i is right neighbor of n_j

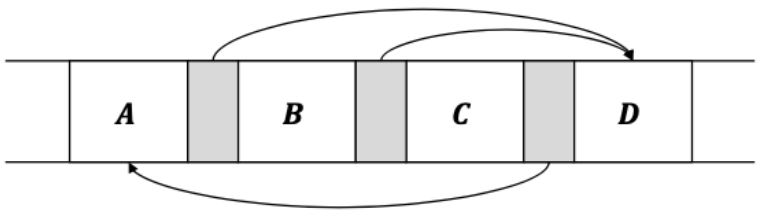


(a) Raw Memory

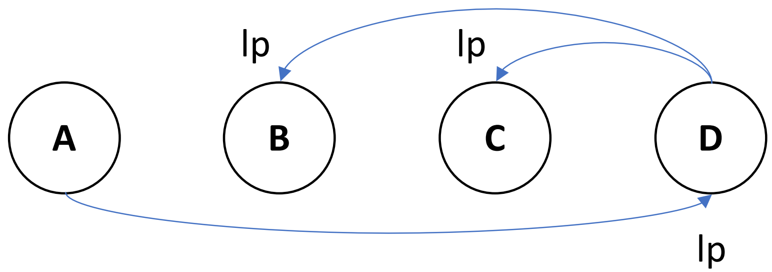


Memory Graph

- A memory graph is a directed graph $G = (N, E_{ln}, E_{rn}, E_{lp}, E_{rp})$,
  - For each pointer, we find its target node and create an edge
  - E_{lp} is an edge set, each edge represents a directed edge from n_i to n_j, and n_i is pointed by a pointer on the left boundary of n_j

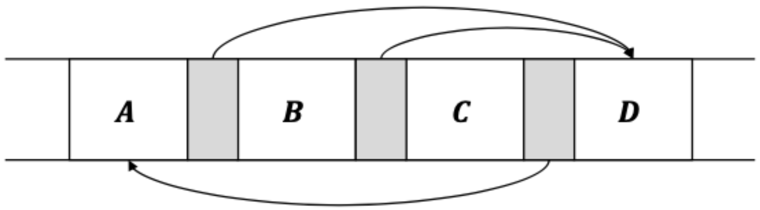


(a) Raw Memory

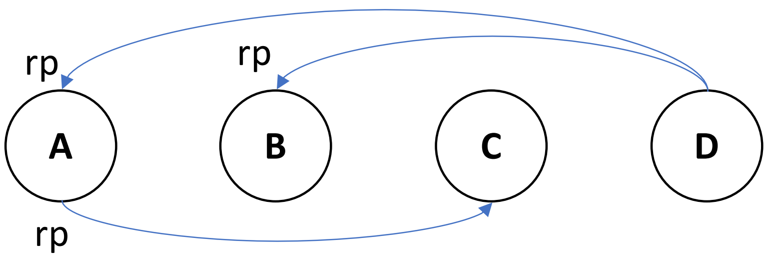


Memory Graph

- A memory graph is a directed graph $G = (N, E_{ln}, E_{rn}, E_{lp}, E_{rp})$,
  - For each pointer, we find its target node and create an edge
  - E_{lp} is an edge set, each edge represents a directed edge from n_i to n_j, and n_i is pointed by a pointer on the left boundary of n_j
  - E_{rp} is an edge set, each edge represents a directed edge from n_i to n_j, and n_i is pointed by a pointer on the right boundary of n_j

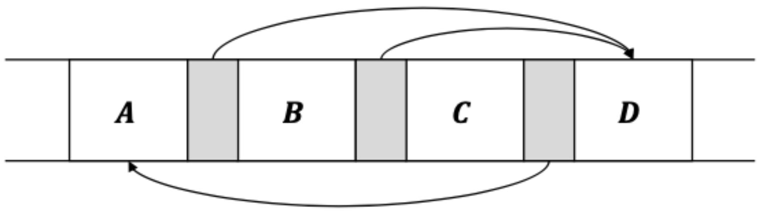


(a) Raw Memory

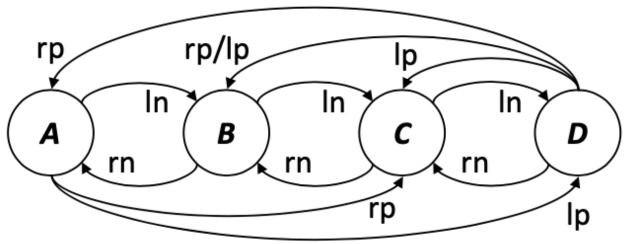


Memory Graph

- A memory graph is a directed graph $G = (N, E_{ln}, E_{rn}, E_{lp}, E_{rp})$,

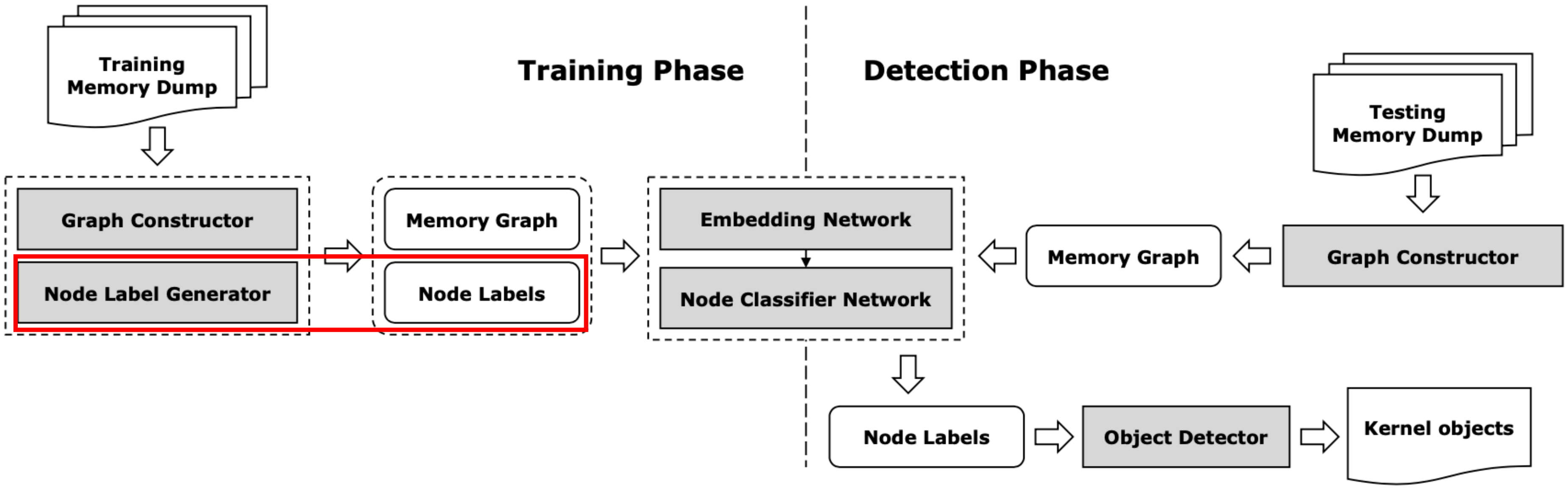


(a) Raw Memory



(b) Memory Graph

Design of DeepMem



Node labeling

- Using Volatility, find out the offset and length information of 6 kernel object types
    - _EPROCESS, _ETHREAD, _DRIVER_OBJECT, _FILE_OBJECT, _LDR_DATA_TABLE_ENTRY, _CM_KEY_BODY
- For each node, determine if it falls into the range of any kernel object
- If so, calculate the offset and length of that node in that kernel object
- Label each node as a 3-tuple of the object type, offset and length

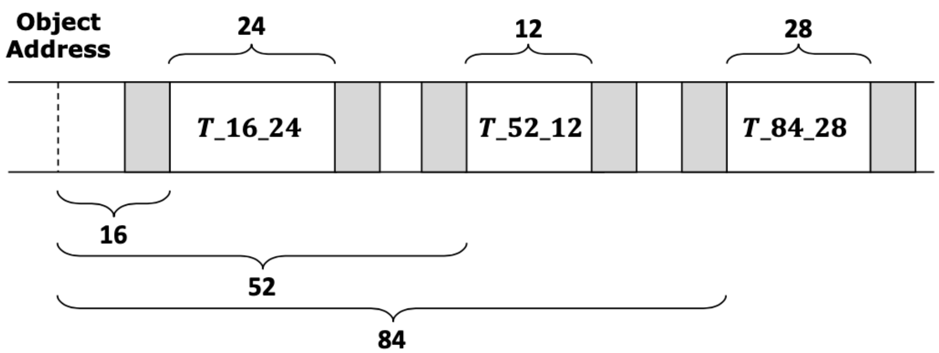


Figure 4: Node Labeling of a _ETHREAD Object

Node labeling

It needs the knowledge of OS

- Using Volatility, find out the offset and length information of 6 kernel object types
  - _EPROCESS, _ETHREAD, _DRIVER_OBJECT, _FILE_OBJECT, _LDR_DATA_TABLE_ENTRY, _CM_KEY_BODY
- For each node, determine if it falls into the range of any kernel object
- If so, calculate the offset and length of that node in that kernel object
- Label each node as a 3-tuple of the object type, offset and length

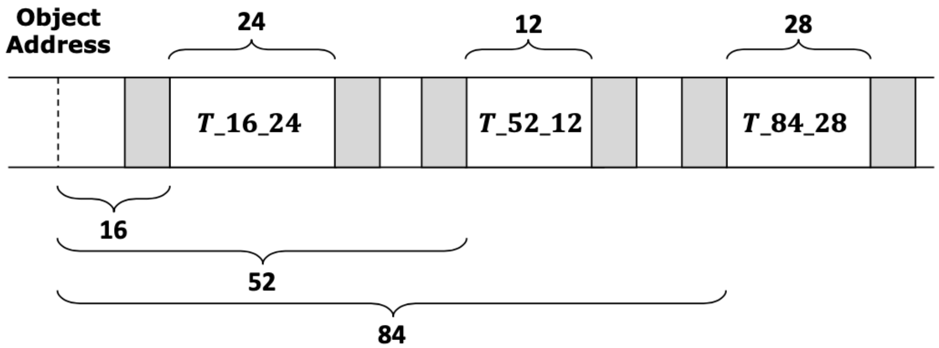


Figure 4: Node Labeling of a _ETHREAD Object

Node labeling

- Using Volatility, find out the offset and length information of 6 kernel object types
    - _EPROCESS, _ETHREAD, _DRIVER_OBJECT, _FILE_OBJECT, _LDR_DATA_TABLE_ENTRY, _CM_KEY_BODY
- Fo … falls into the range of any kernel object
- If … ength of that node in that kernel object
- Label each node as a … le of the … ffset and length

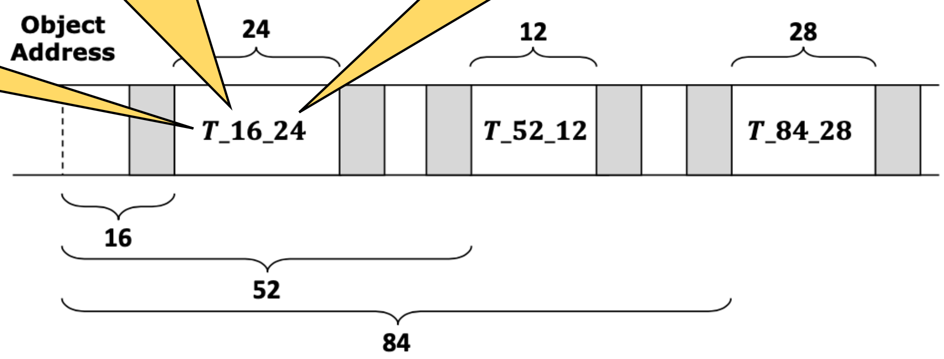


Figure 4: Node Labeling of a _ETHREAD Object

Node labeling

- Select the top 20 most frequent node labels across all kernel objects of type c as key node label set L(c) for type c
- Label the rest nodes in the memory graph as none

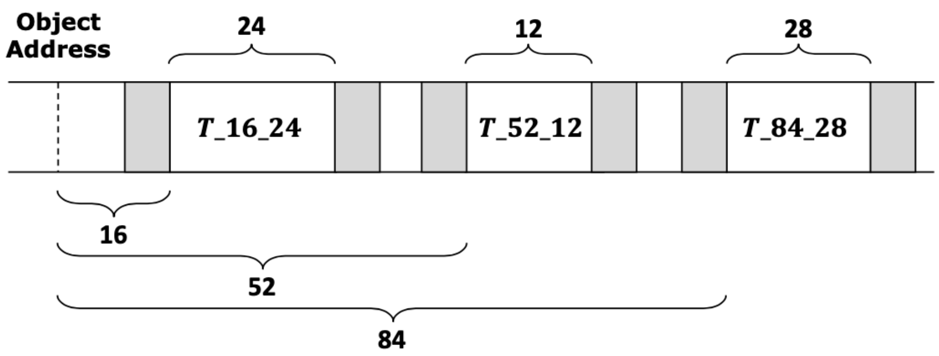
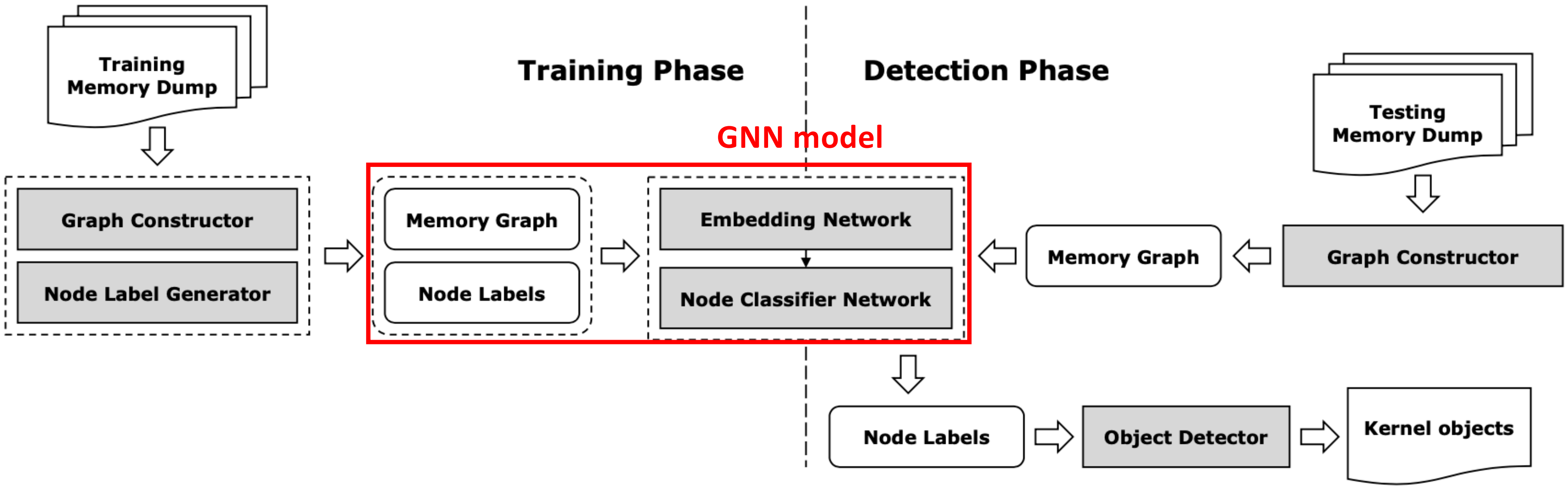


Figure 4: Node Labeling of a _ETHREAD Object

Design of DeepMem



Graph Neural Network (GNN) Model

- Goal 1: extract the node representations -> embedding network
- Goal 2: infer the node labels -> classifier network

- GNN model, $\mathcal{F}$ consists of two jointly-trained subnetworks

$$\mathcal{F} = \psi_{\mathbf{w}_2}(\phi_{\mathbf{w}_1}(\cdot))$$

Embedding network

$$\boldsymbol{\mu}_n = \phi_{w_1}(\boldsymbol{v}_n, \boldsymbol{\mu}_{E_{ln}[n]}, \boldsymbol{\mu}_{E_{rn}[n]}, \boldsymbol{\mu}_{E_{lp}[n]}, \boldsymbol{\mu}_{E_{rp}[n]})$$

- μ_n is an embedding vector of node n
- v_n is a d-dimensional vector of node n derived from its actual memory content
  - If the memory segment is longer than d bytes, truncate it and only keep d bytes
    if it is shorter than d bytes, fill the remaining bytes with 0
- For each node n in the memory graph G, the embedding network integrates input vector v_n and the topological information from its neighbors

Embedding network

$$\boldsymbol{\mu}_n = \phi_{w_1}(\boldsymbol{v}_n, \boldsymbol{\mu}_{E_{ln}[n]}, \boldsymbol{\mu}_{E_{rn}[n]}, \boldsymbol{\mu}_{E_{lp}[n]}, \boldsymbol{\mu}_{E_{rp}[n]})$$

$$\downarrow$$

$$\boldsymbol{\mu}_n(t+1) = \phi_{w_1}(\boldsymbol{v}_n, \boldsymbol{\mu}_{E_{ln}[n]}(t), \boldsymbol{\mu}_{E_{rn}[n]}(t), \boldsymbol{\mu}_{E_{lp}[n]}(t), \boldsymbol{\mu}_{E_{rp}[n]}(t))$$

- Implement the embedding vector as a state vector
  - Gradually absorbs information propagated from multiple sources over time
  - Add a time variable into embedding vector computation
- In each iteration, it traverses one layer of neighbor nodes and integrates the neighbors' states
- The more iterations run, the information of farther neighbors are collected

Embedding network

Algorithm 1: Information Propagation Algorithm of Embedding Network ϕ_{w_1}

Input : Memory Graph $G = (N, E_{ln}, E_{rn}, E_{lp}, E_{rp})$, iteration time T

Output : Graph Embedding $\boldsymbol{\mu}_n$ for all $n \in N$

```
Initialize μ_n(0) = 0, for each n ∈ N
for t = 1 to T do
    for n ∈ N do
        β = σ1(Σ_{m∈E_rn[n]} μ_m(t − 1))
        β+ = σ2(Σ_{m∈E_ln[n]} μ_m(t − 1))
        β+ = σ3(Σ_{m∈E_lp[n]} μ_m(t − 1))
        β+ = σ4(Σ_{m∈E_rp[n]} μ_m(t − 1))
        μ_n(t) = tanh(W1 · v_n + β)
    end
end
```

$$\beta = \sigma_1(\textstyle\sum_{m \in E_{rn}[n]} \boldsymbol{\mu}_m(t-1))$$
$$\beta += \sigma_2(\textstyle\sum_{m \in E_{ln}[n]} \boldsymbol{\mu}_m(t-1))$$
$$\beta += \sigma_3(\textstyle\sum_{m \in E_{lp}[n]} \boldsymbol{\mu}_m(t-1))$$
$$\beta += \sigma_4(\textstyle\sum_{m \in E_{rp}[n]} \boldsymbol{\mu}_m(t-1))$$
$$\boldsymbol{\mu}_n(t) = tanh(W_1 \cdot \boldsymbol{v}_n + \beta)$$

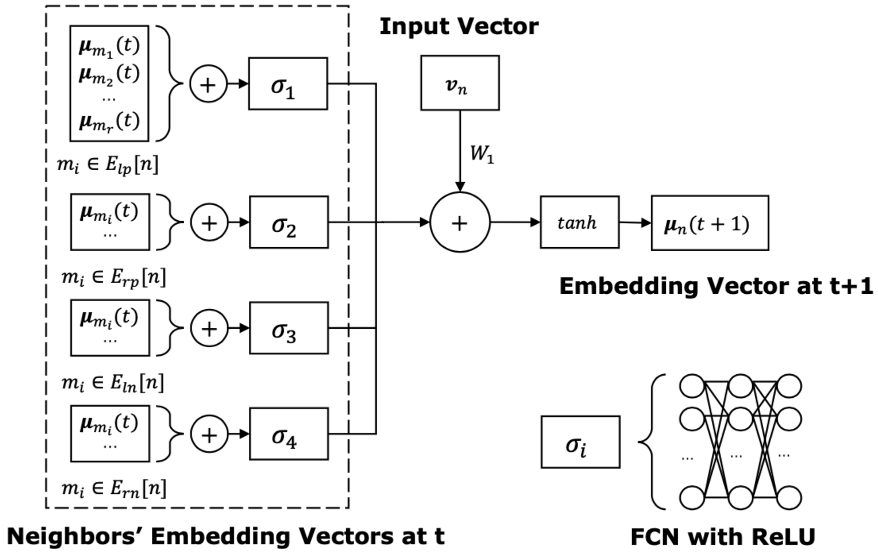


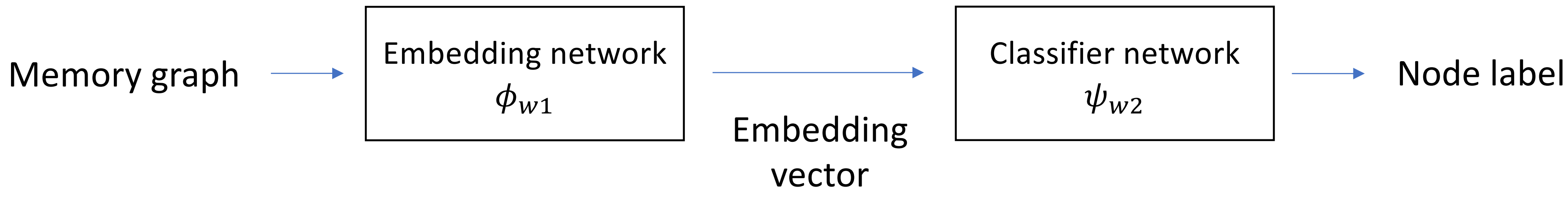
Figure 3: Node embedding computation in each iteration t. Information flows through E_{lp}, E_{rp}, E_{nl}, E_{nr} edges. Embedding vector $\boldsymbol{\mu}_n(t+1)$ gets updated by input vector $\boldsymbol{v}_n$ and its neighbors' embedding vectors at t.

Classifier network

$$\boldsymbol{y_n} = \psi_{\mathbf{w}_2}(\boldsymbol{\mu_n})$$

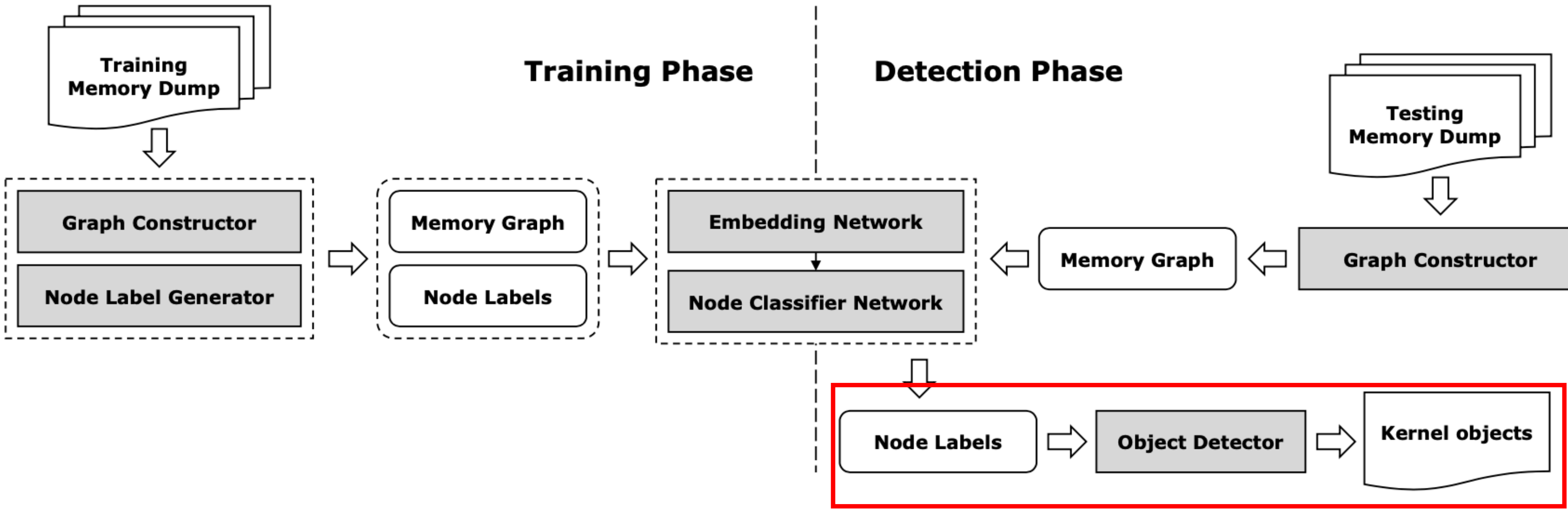
- μ_n is an embedding vector of node n, the output of the embedding network
- y_n is the node label
- Node classifier network is used to map embedding vector to a node label
- Each object type has a classifier
  - a _ETHREAD classifier, a _EPROCESS classifier, etc
- Need to build a multi-class classifier

Graph Neural Network (GNN) Model



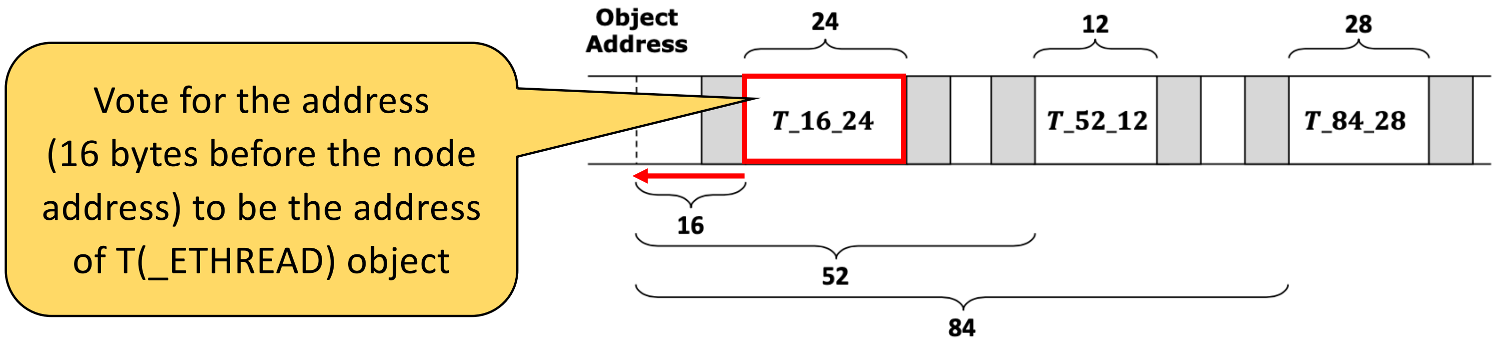
- To train the weights in the GNN model,
  compute the cross-entropy loss between the predicted label and annotated label,
  update weights in the process of minimizing the loss
- The parameters of embedding network w_1(including weights of W_1, σ_1, σ_2, σ_3, σ_4) and parameters of classifier network w_2 are updated and optimized in training

Design of DeepMem



Object detection

- Node label can be considered as a voter that votes for the presence of an object



- Each node in the memory indicates the presence of an object
- Thus with all the node labels, we can generate a set of candidate object addresses S = {s1, s2, ...} and corresponding voters for each address
- Need to determine whether an address s ∈ S is indeed a start address of an object

Object detection

The weights of node label l in predicting objects of type c

- Design a weighted voting mechanism

$$f(s,c) = \begin{cases} 1, & \sum_{l_i \in L(s,c)} \boxed{\frac{\rho(c,l_i)}{\rho(c)}} + \gamma(s,c)) > \delta \\ 0, & otherwise \end{cases}$$

- $L(s,c)$ is the voter set, which is all the node labels of type c that vote for address s
- $\rho(c)$ counts the number of objects of type c in the dataset
- $\rho(c,l)$ counts the number of objects of type c that has node label l in the dataset
- $\gamma(s,c)$ is a function to reward the addresses voted by multiple voters $(|\mathrm{L}(\mathrm{s},\mathrm{c})| - 1)$
- δ is a pre-defined threshold

Object detection

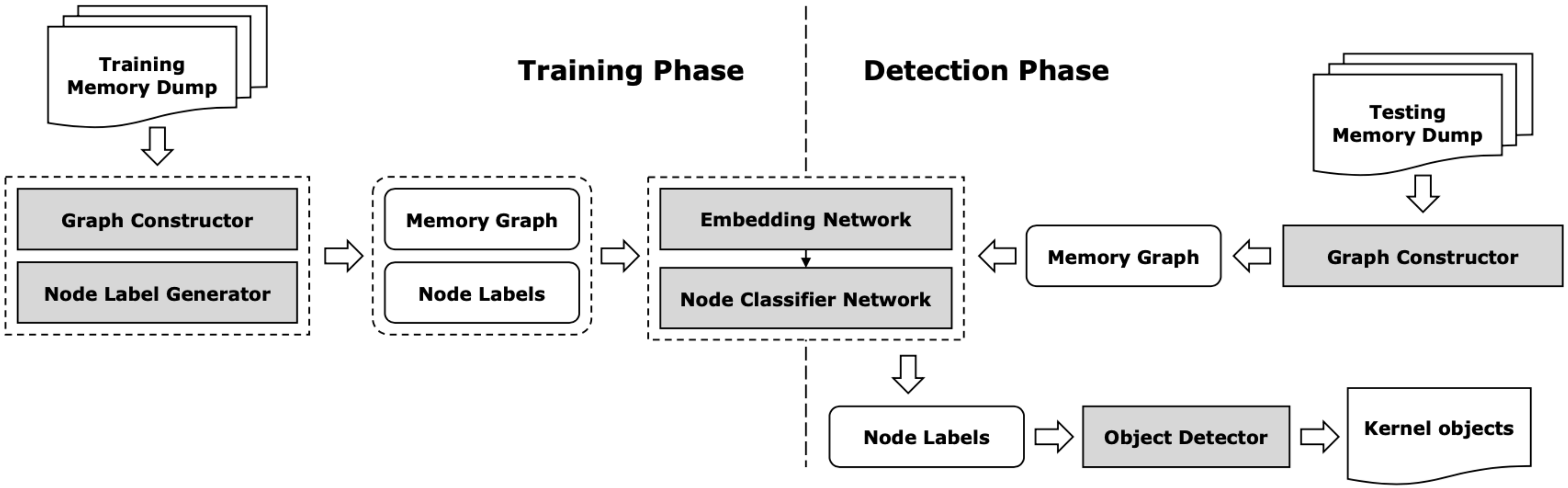
It is possible that
weight sum of two small voters 0.4 + 0.3
< weight value of a single large voter 0.8

- Design a weighted voting mechanism

$$f(s,c) = \begin{cases} 1, & \sum_{l_i \in L(s,c)} \frac{\rho(c,l_i)}{\rho(c)} + \boxed{\gamma(s,c)}) > \delta \\ 0, & otherwise \end{cases}$$

- $L(s,c)$ is the voter set, which is all the node labels of type c that vote for address s
- $\rho(c)$ counts the number of objects of type c in the dataset
- $\rho(c,l)$ counts the number of objects of type c that has node label l in the dataset
- $\gamma(s,c)$ is a function to reward the addresses voted by multiple voters $(|\mathrm{L}(\mathrm{s},\mathrm{c})| - 1)$
- δ is a pre-defined threshold

Design of DeepMem



Default parameters

Parameters	**Value**
Layers of σ	3
Layers of ψ	3
Optimizer	Adam Optimizer
Learning Rate	0.0001
Propagation Iteration T	3
Input Vector Dimension	64
Embedding Vector Dimension	64
keep_prob	0.8

Table 3: Default Parameters of Experiments.

Detection Accuracy

Kernel Object Types	Object Length	Avg. #TP	Avg. #FP	Avg. #FN	Precision%	Recall%	F-Score
_EPROCESS	704	82.834	0.017	0.303	99.979%	99.635%	0.99807
_ETHREAD	696	1211.476	5.514	0.7	99.547%	99.942%	0.99744
_DRIVER_OBJECT	168	108.938	0.255	0.024	99.766%	99.978%	0.99872
_FILE_OBJECT	128	3621.007	67.545	23.045	98.169%	99.368%	0.98765
_LDR_DATA_TABLE_ENTRY	120	139.093	0.0	2.4	100.0%	98.304%	0.99145
_CM_KEY_BODY	44	1979.207	94.621	0.414	95.437%	99.979%	0.97655

Table 2: Object Detection Results on Memory Image Dumps.

The overall recall rate is satisfactory, ranging from 98.304% to 99.979%

Detection Accuracy

Kernel Object Types	Object Length	Avg. #TP	Avg. #FP	Avg. #FN	Precision%	Recall%	F-Score
_EPROCESS	704	82.834	0.017	0.303	99.979%	99.635%	0.99807
_ETHREAD	696	1211.476	5.514	0.7	99.547%	99.942%	0.99744
_DRIVER_OBJECT	168	108.938	0.255	0.024	99.766%	99.978%	0.99872
_FILE_OBJECT	128	3621.007	67.545	23.045	98.169%	99.368%	0.98765
_LDR_DATA_TABLE_ENTRY	120	139.093	0.0	2.4	100.0%	98.304%	0.99145
_CM_KEY_BODY	44	1979.207	94.621	0.414	95.437%	99.979%	0.97655

Table 2: Object Detection Results on Memory Image Dumps.

Most large kernel objects (≥ 120 bytes) have over 98% precision rate

Detection Accuracy

Kernel Object Types	Object Length	Avg. #TP	Avg. #FP	Avg. #FN	Precision%	Recall%	F-Score
_EPROCESS	704	82.834	0.017	0.303	99.979%	99.635%	0.99807
_ETHREAD	696	1211.476	5.514	0.7	99.547%	99.942%	0.99744
_DRIVER_OBJECT	168	108.938	0.255	0.024	99.766%	99.978%	0.99872
_FILE_OBJECT	128	3621.007	67.545	23.045	98.169%	99.368%	0.98765
_LDR_DATA_TABLE_ENTRY	120	139.093	0.0	2.4	100.0%	98.304%	0.99145
_CM_KEY_BODY	44	1979.207	94.621	0.414	95.437%	99.979%	0.97655

Table 2: Object Detection Results on Memory Image Dumps.

Important kernel object types _EPROCESS, _ETHREAD both achieve over 99.6% recall rate, and over 99.5% precision rate

Detection Accuracy

Kernel Object Types	Object Length	Avg. #TP	Avg. #FP	Avg. #FN	Precision%	Recall%	F-Score
_EPROCESS	704	82.834	0.017	0.303	99.979%	99.635%	0.99807
_ETHREAD	696	1211.476	5.514	0.7	99.547%	99.942%	0.99744
_DRIVER_OBJECT	168	108.938	0.255	0.024	99.766%	99.978%	0.99872
_FILE_OBJECT	128	3621.007	67.545	23.045	98.169%	99.368%	0.98765
_LDR_DATA_TABLE_ENTRY	120	139.093	0.0	2.4	100.0%	98.304%	0.99145
_CM_KEY_BODY	44	1979.207	94.621	0.414	95.437%	99.979%	0.97655

Table 2: Object Detection Results on Memory Image Dumps.

We can observe a tendency that larger objects achieve better results

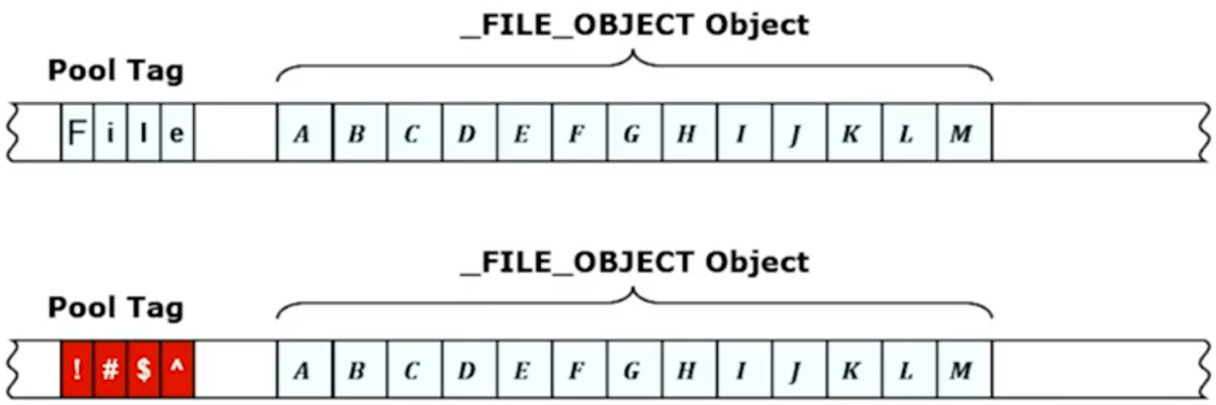
Why? For small objects, there are fewer nodes and pointers inside them then, the chance of obtaining stable key nodes is lower

Robustness

- Pool tag manipulation
  - Evaluate its impact on signature scanning tools and DeepMem
- DKOM process hiding
  - Evaluate if DeepMem is still effective in DKOM process hiding attacks
- Random mutation attack
  - Evaluate whether this approach is resistant to various attack scenarios

Robustness - Pool tag manipulation

- Change the 4 bytes pool tags of each object to random values in memory dump



Robustness - Pool tag manipulation

- Change the 4 bytes pool tags of each object to random values in memory dump
- Using the manipulated dump, test the effectiveness of DeepMem and Volatility plugin
- Randomly select 10 memory dumps as the testing set, and scan _FILE_OBJECT

signature scanning

Method	Avg. #TP	Avg. #FP	Avg. #FN	Precision%	Recall%
filescan	0.3	0.0	3661.8	100%	0.0082%
DeepMem	3627.2	32.9	34.9	99.1%	99.05%

Table 4: Results of _FILE_OBJECT Pool Tag Manipulation

Robustness - Pool tag manipulation

- Change the 4 bytes pool tags of each object to random values in memory dump
- Using the manipulated dump, test the effectiveness of DeepMem and Volatility plugin
- Randomly select 10 memory dumps as the testing set, and scan _FILE_OBJECT

Method	Avg. #TP	Avg. #FP	Avg. #FN	Precision%	Recall%
filescan	0.3	0.0	3661.8	100%	0.0082%
DEEPMEM	3627.2	32.9	34.9	99.1%	99.05%

Table 4: Results of _FILE_OBJECT Pool Tag Manipulation

The filescan plugin cannot correctly report _FILE_OBJECT objects

Why? Need to search for the pool tag of _FILE_OBJECT in the entire memory dump

Robustness - Pool tag manipulation

- Change the 4 bytes pool tags of each object to random values in memory dump
- Using the manipulated dump, test the effectiveness of DeepMem and Volatility plugin
- Randomly select 10 memory dumps as the testing set, and scan _FILE_OBJECT

Method	Avg. #TP	Avg. #FP	Avg. #FN	Precision%	Recall%
filescan	0.3	0.0	3661.8	100%	0.0082%
DeepMem	3627.2	32.9	34.9	99.1%	99.05%

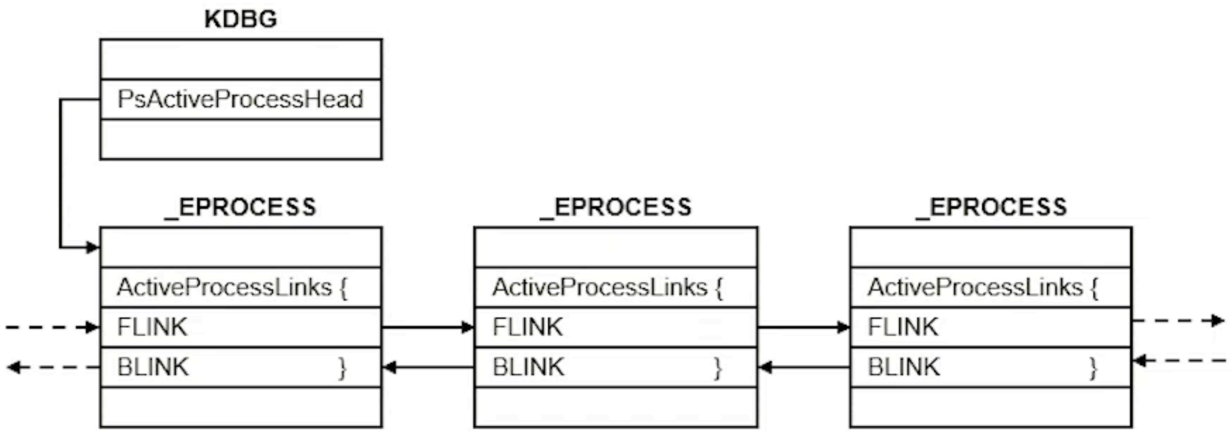
Table 4: Results of _FILE_OBJECT **Pool Tag Manipulation**

DeepMem works normally, achieving a precision of 99.1% and recall of 99.05%

Why? Examine every byte of a memory dump to detect objects,
rather than merely rely on pool tag constraints to locate objects

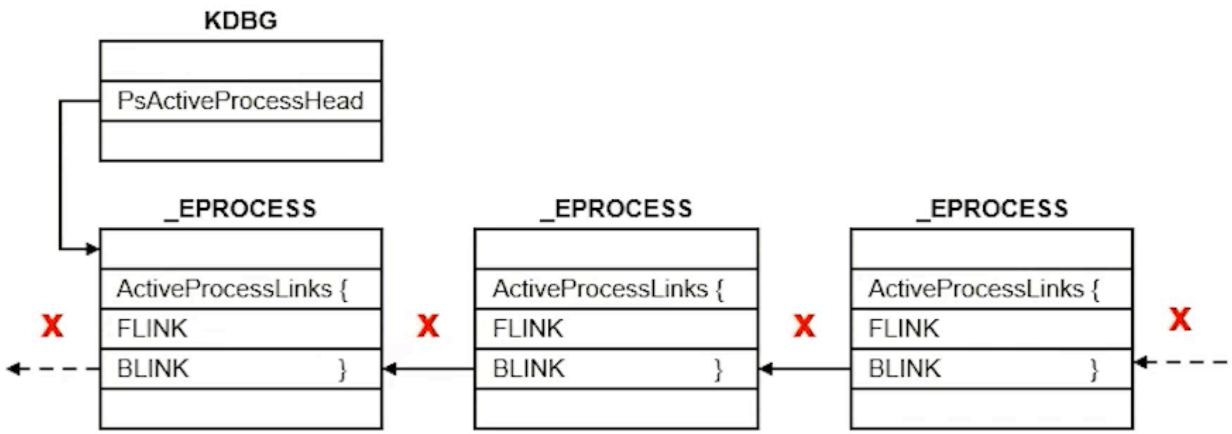
Robustness - DKOM process hiding

- DKOM (Direct Kernel Object Manipulation) attack is to hide a malicious process by unlinking its connections to precedent and antecedent processes in a double linked list



Robustness - DKOM process hiding

- DKOM (Direct Kernel Object Manipulation) attack is to hide a malicious process by unlinking its connections to precedent and antecedent processes in a double linked list



Robustness - DKOM process hiding

- Randomly select 20 memory dumps as a testing set
- Manipulate the value of the forward link field in each _EPROCESS object to random value

list traversal

Method	Avg. #TP	Avg. #FP	Avg. #FN	Precision%	Recall%
pslist	1.05	0.0	85.7	100%	1.21%
DeepMem	86.55	0.0	0.2	100%	99.77%

Table 5: Results of DKOM Process Hiding Attacks

Robustness - DK

They used different # memory dumps (10 for testing pool tag manipulation)

- Randomly select 20 memory dumps as a testing set
- Manipulate the value of the forward link field in each _EPROCESS object to random value

They used different object type (_FILE_OBJECT)

list traversal

Method	Avg. #TP	Avg. #F			
pslist	1.05	0.0	85.7	100%	1.21%
DeepMem	86.55	0.0	0.2	100%	99.77%

Table 5: Results of DKOM Process Hiding Attacks

Robustness - DKOM process hiding

- Randomly select 20 memory dumps as a testing set
- Manipulate the value of the forward link field in each _EPROCESS object to random value

Method	Avg. #TP	Avg. #FP	Avg. #FN	Precision%	Recall%
pslist	1.05	0.0	85.7	100%	1.21%
DeepMem	86.55	0.0	0.2	100%	99.77%

Table 5: Results of DKOM Process Hiding Attacks

The pslist fails to discover most _EPROCESS objects except the first one in each dump

Why? _EPROCESS list is broken by the manipulation,
cannot traverse through the double linked list to find other processes

Robustness - DKOM process hiding

- Randomly select 20 memory dumps as a testing set
- Manipulate the value of the forward link field in each _EPROCESS object to random value

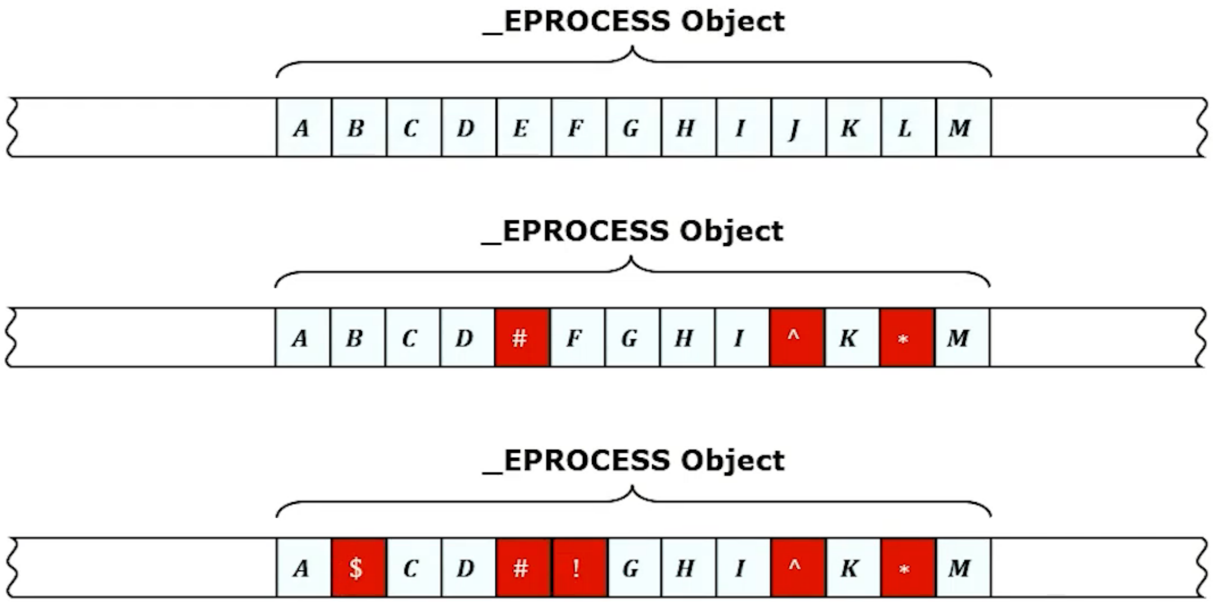
Method	Avg. #TP	Avg. #FP	Avg. #FN	Precision%	Recall%
pslist	1.05	0.0	85.7	100%	1.21%
DeepMem	86.55	0.0	0.2	100%	99.77%

Table 5: Results of DKOM Process Hiding Attacks

DeepMem can still find 99.77% _EPROCESS objects with 100% precision

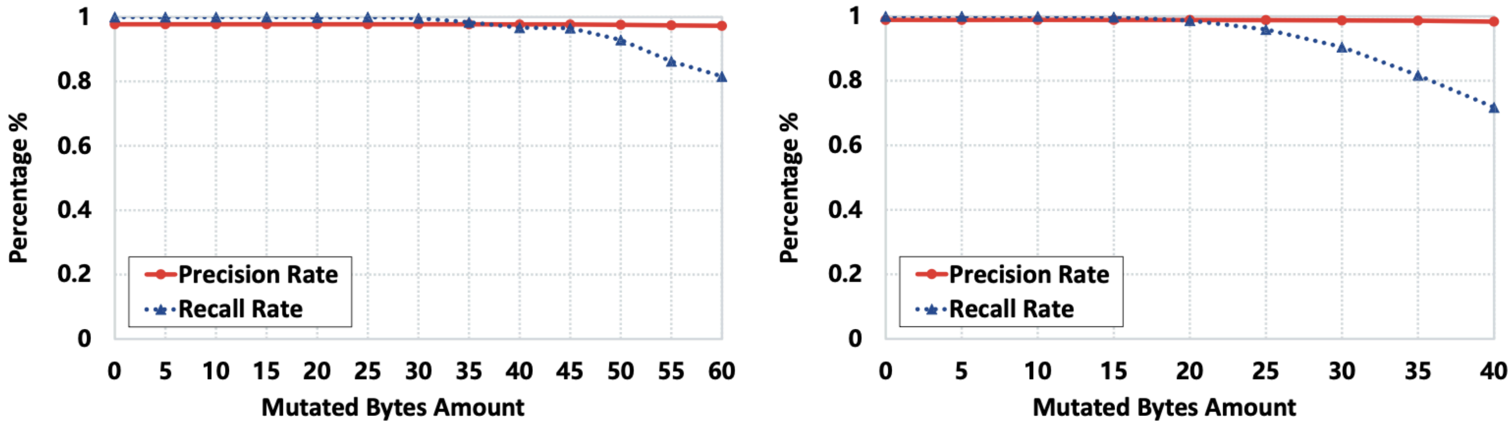
Robustness - Random mutation attack

- Evaluate the detection results by mutating different amount of bytes in objects for _EPROCESS and _ETHREAD objects



Robustness - Random mutation attack

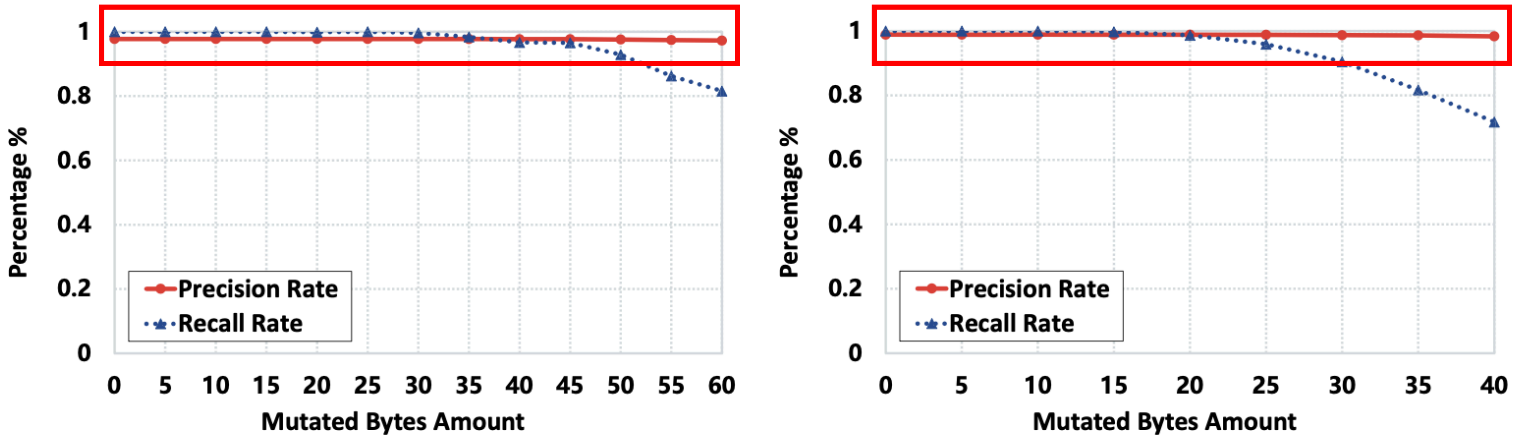
- Evaluate the detection results by mutating different amount of bytes in objects for _EPROCESS and _ETHREAD objects



(a) Random Mutation Attack of _EPROCESS Object (b) Random Mutation Attack of _ETHREAD Object

Robustness - Random mutation attack

- Evaluate the detection results by mutating different amount of bytes in objects for _EPROCESS and _ETHREAD objects



(a) Random Mutation Attack of _EPROCESS Object (b) Random Mutation Attack of _ETHREAD Object

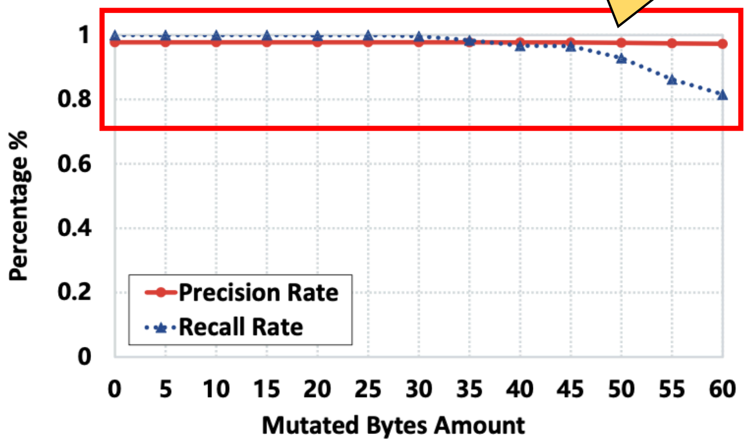
As the number of mutated bytes increases,
the precision rate remains stable at around 97% - 98%

Robustness - Random mutation attack

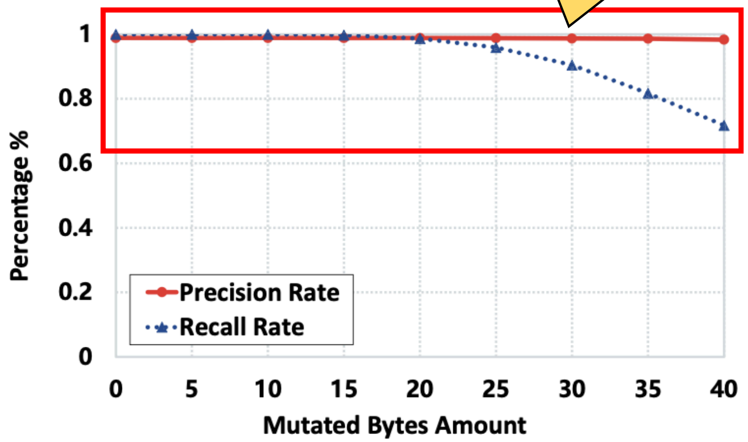
- Evaluate the detecti g different a jects
  for _EPROCESS and _

Robust up to 50 bytes random mutation

Robust up to 30 bytes random mutation



(a) Random Mutation Attack of _EPROCESS Object



(b) Random Mutation Attack of _ETHREAD Object

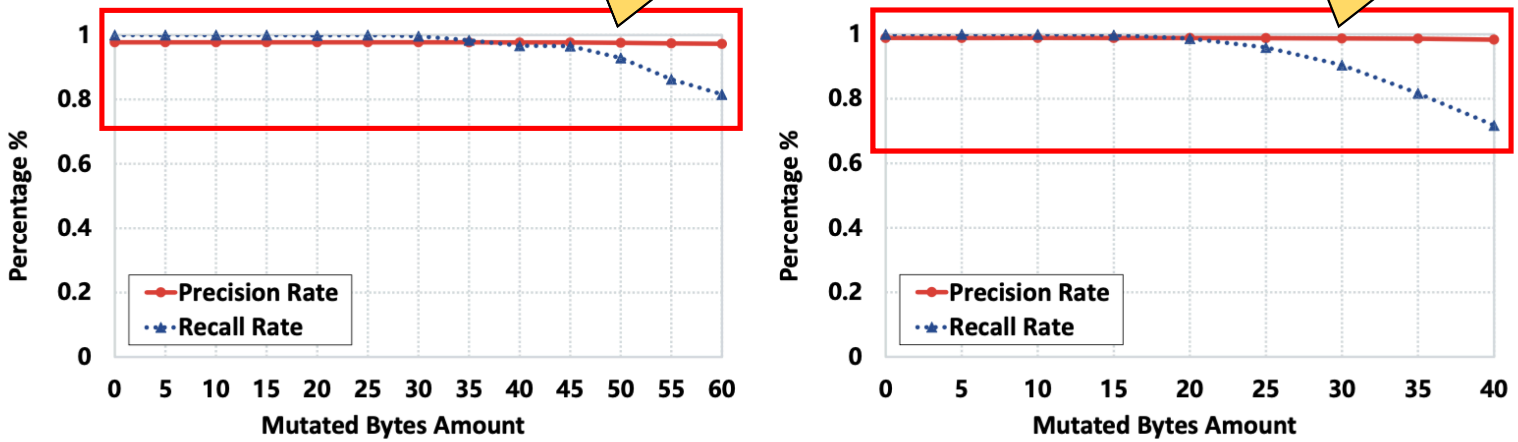
Recall rate curve stays at a high rate at first,
then drops down as the number of mutated bytes further increases

Robustness - Random mutation attack

- Evaluate the detecti … different … jects for _EPROCESS and _E…

Robust up to 50 bytes random mutation

Robust up to 30 bytes random mutation

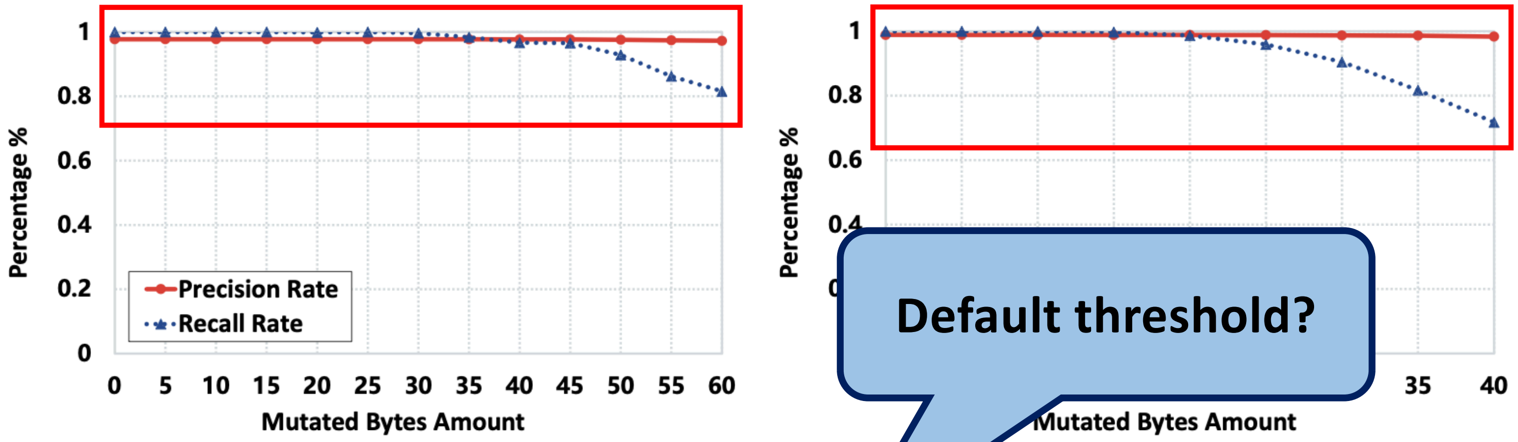


(a) Random Mutation Attack of _EPROCESS Object (b) Random Mutation Attack of _ETHREAD Object

These larger mutations will likely cause system crashes or instability, and therefore might be rarely seen in real-world attacks

Robustness - Random mutation attack

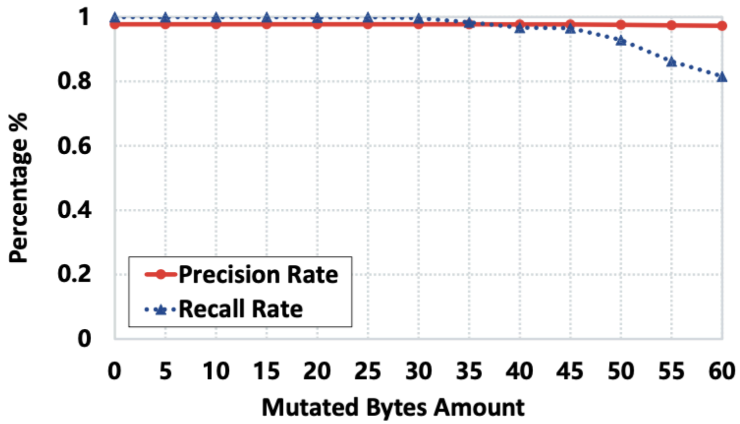
- Evaluate the detection results by mutating different amount of bytes in objects for _EPROCESS and _ETHREAD objects



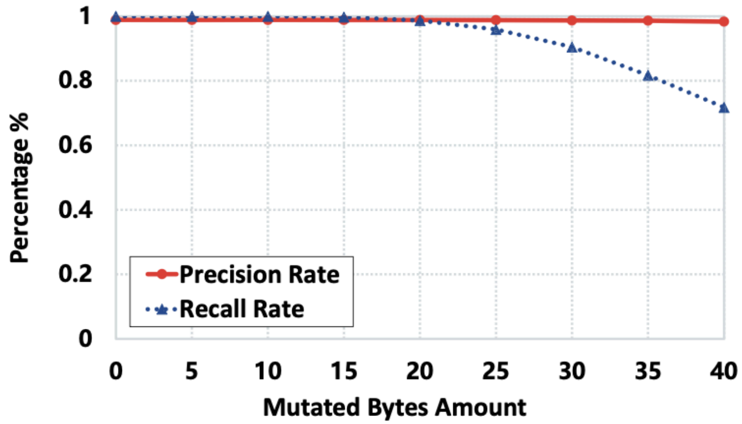
(a) Random Mutation Attack of _EPROCESS Object (b) Ran Mutation Attack of _ETHREAD Object

When set the threshold δ to a low value 1,
the precision rate does not drop significantly

Robustness - Random mutation attack



(a) Random Mutation Attack of _EPROCESS Object



(b) Random Mutation Attack of _ETHREAD Object

- The neural network itself can inherently tolerate small mutations due to the robust features it learns from the training data
- Even when deep model incorrectly predicts the labels of some nodes of an object, the remaining nodes can make cross-validation and collectively conclude the presence of an object

Efficiency

- Evaluate the efficiency of DeepMem by measuring the time allocations in different phases

computing center with GPU

desktop computer without GPU

	Time Measurements	Mean	Std Dev
Training	Training T_t (per object type)	13 Hours	N/A
Detection	Graph Construction T_g (per dump)	79.7 Sec	6.64
	Object Detection T_d (per type)	12.73 Sec	1.24

Table 6: Time Consumption at Different Phases. Training is performed on the computing center. Detection is performed on a desktop computer. The setting is in Section 4.1

Use different machine

Efficiency

- Evaluate the efficiency of DeepMem by measuring the time allocations in different phases

computing center with GPU

desktop computer without GPU

	Time Measurements	Mean	Std Dev
Training	Training T_t (per object type)	13 Hours	N/A
Detection	Graph Construction T_g (per dump)	79.7 Sec	6.64
	Object Detection T_d (per type)	12.73 Sec	1.24

Table 6: Time Consumption at Different Phases. Training is performed on the computing center. Detection is performed o … Section 4.1

Why use desktop computer?

This detection time can be reduced to about 7.7 sec in computing center with GPU

Efficiency

- Evaluate the efficiency of DeepMem by measuring the time allocations in different phases

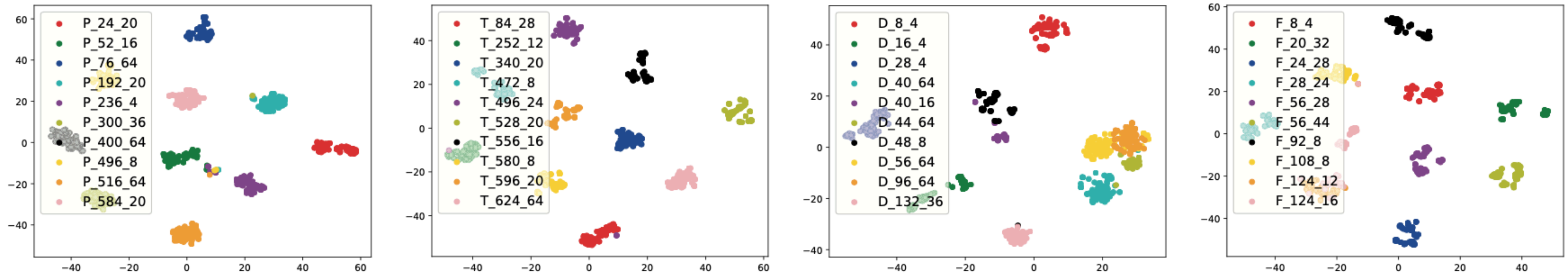
	Time Measurements	Mean	Std Dev
Training	Training T_t (per object type)	13 Hours	N/A
Detection	Graph Construction T_g (per dump)	79.7 Sec	6.64
	Object Detection T_d (per type)	12.73 Sec	1.24

Table 6: Time Consumption at Different Phases. Training is performed on the computing center. Detection is performed on a desktop computer. The setting is in Section 4.1

- It turns a memory dump into a graph structure, which is suitable for fast GPU parallel computation
- Since it converts the memory dump into memory graph, and performs the detection of various object types on this graph, there is no need to scan the raw memory multiple times to match the various set of signatures for different object types

Node embedding

- Collect embedding vectors of different object types at the output layer of the embedding network

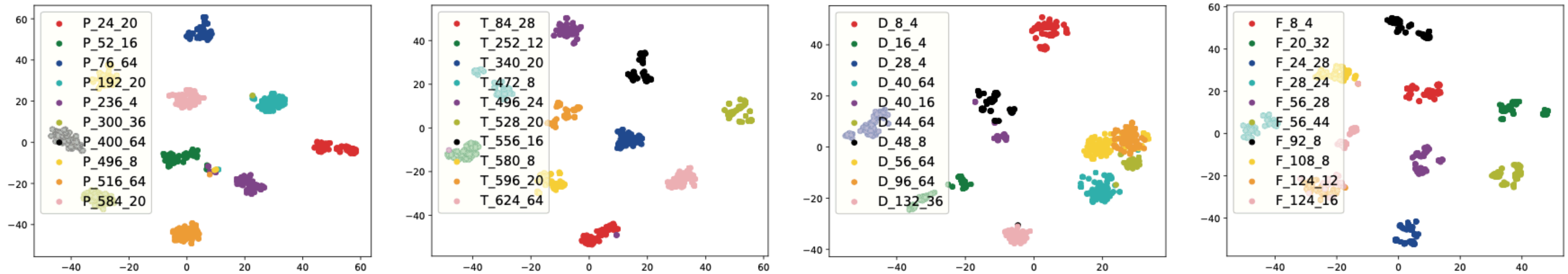


(a) _EPROCESS (b) _ETHREAD (c) _DRIVER_OBJECT (d) _FILE_OBJECT

Points of the same colors locate near each other,
different colors locate far from each other

Node embedding

- Collect embedding vectors of different object types at the output layer of the embedding network

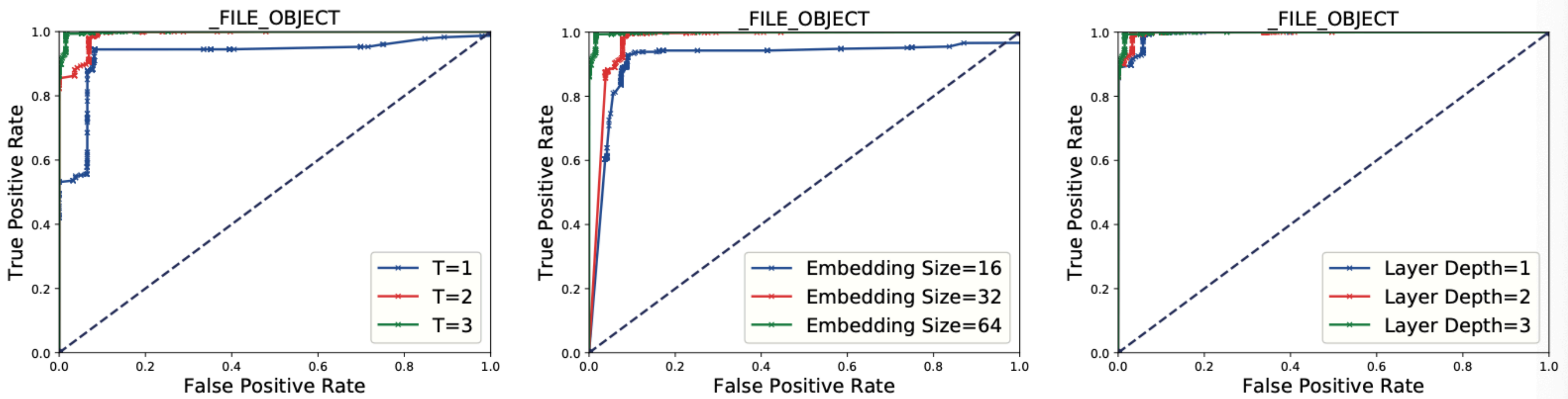


(a) _EPROCESS (b) _ETHREAD (c) _DRIVER_OBJECT (d) _FILE_OBJECT

These embeddings can capture the intrinsic characteristics of nodes

Hyperparameters

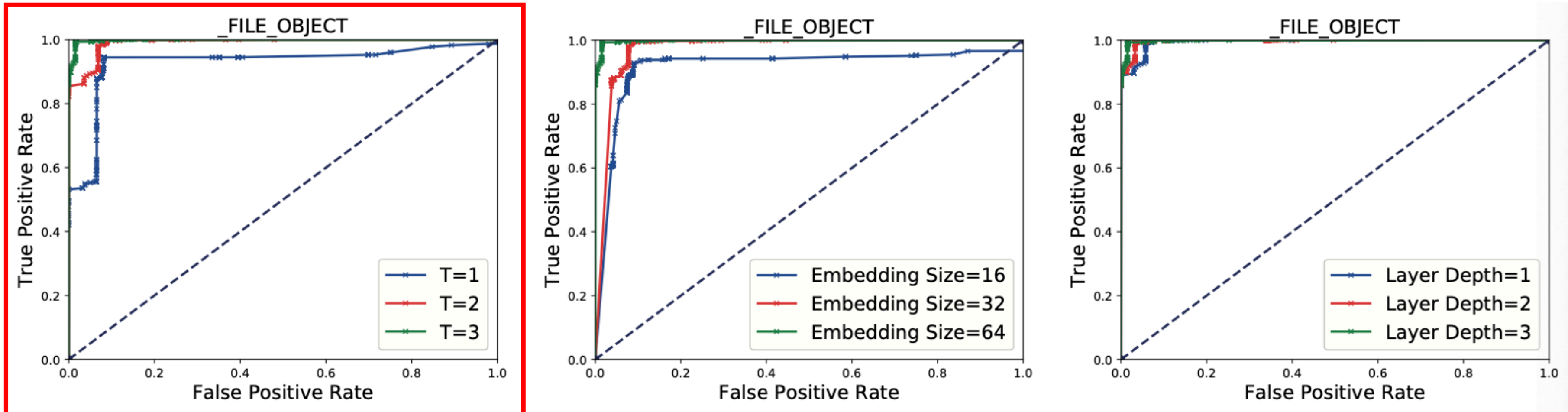
- Evaluate the impact of the different hyperparameters



(a) ROC versus Iterations T (b) ROC versus Embedding Vector Size (c) ROC versus Layer Depth of σ

Hyperparameters

- Performance of the _FILE_OBJECT detector by tuning the iteration parameter T

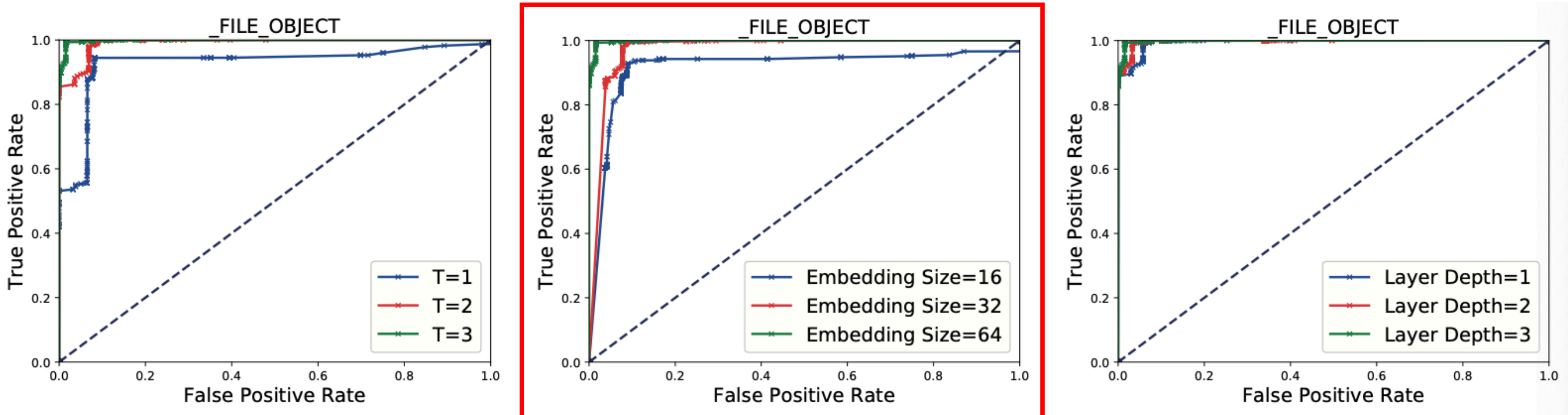


(a) ROC versus Iterations T (b) ROC versus Embedding Vector Size (c) ROC versus Layer Depth of σ

With more information collected through propagation,
the prediction ability of the object detector is further improved

Hyperparameters

- Performance of _FILE_OBJECT detector by tuning embedding vector size

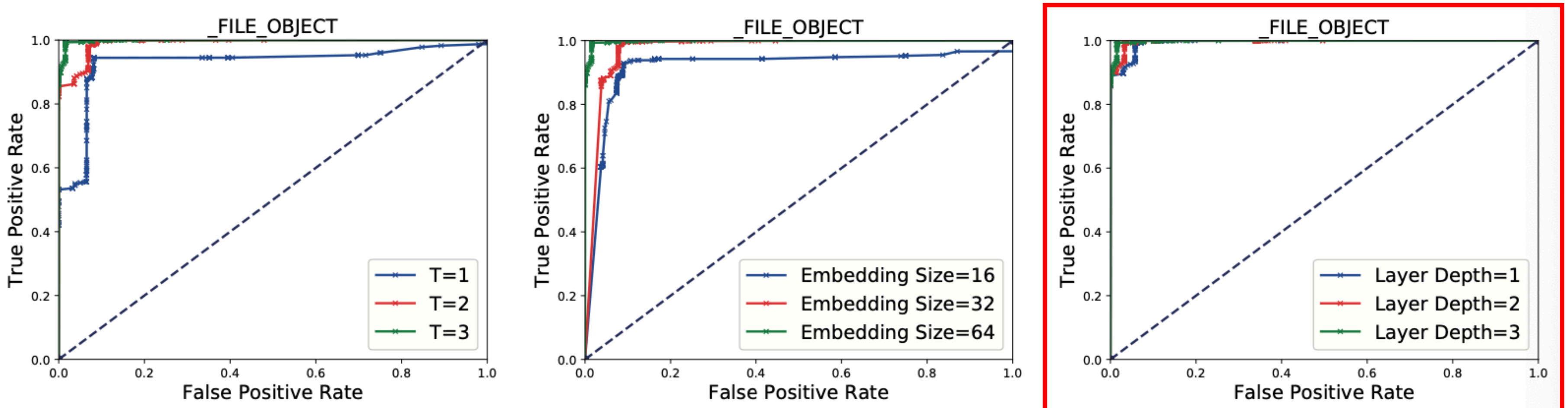


(a) ROC versus Iterations T (b) ROC versus Embedding Vector Size (c) ROC versus Layer Depth of σ

Larger embedding vector size is more expressive
and better approximate the data intrinsic characteristics

Hyperparameters

- Performance of _FILE_OBJECT detector by tuning embedding layers depth



(a) ROC versus Iterations T

(b) ROC versus Embedding Vector Size

(c) ROC versus Layer Depth of σ

The learning ability of deeper neural network is stronger than shallower networks

Discussion

- Pros
  - Not rely on the knowledge of OS source code or kernel data structures
  - Automatically generate features of kernel objects from memory dump
  - Fast and robust to attacks like pool tag manipulation, DKOM process hiding

- Cons
  - DeepMem may not perform well for small objects with few or no pointers
  - Dataset may not be diverse enough
    - Need to use different physical machines, load different drivers, etc.

Q & A