DeLink: Source File Information Recovery in Binaries

ISSTA ’24, September 16–20, 2024, Vienna, Austria

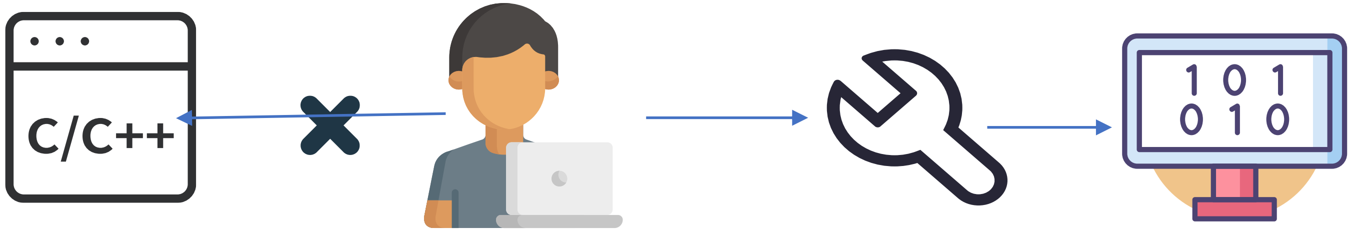
Zhe Lang,Shichao Lv,Zhanwei Song
Zhiqiang Shi,Limin Sun
Institute of Information Engineering, CAS
Beijing, China School of Cyber Security,
UCAS Beijing, China

Zhengzi Xu
Nanyang Technological University
Singapore, Singapore Imperial Global
Singapore Singapore, Singapore

Xiaohui Chen
China Mobile Research Institute Beijing,
China chenxiaohuiyjy@chinamobile.com

Introduction

• **Program Comprehension**



Program Comprehension in Binary Analysis include:

• **Instruction-level analysis**

time-consuming and inadequate

• **Function-level analysis** (Focus of previous work, Debin(ccs 18), NFRE(issta 21),SymLM(ccs 22),Exact(ACSAC 20))

• **High-level analysis** (previous work： BCD(ccs 18))

C1: Lack of Explicit File Boundaries

C2: Difficulty in Feature Selection for File Name Prediction

Background

Terminologies

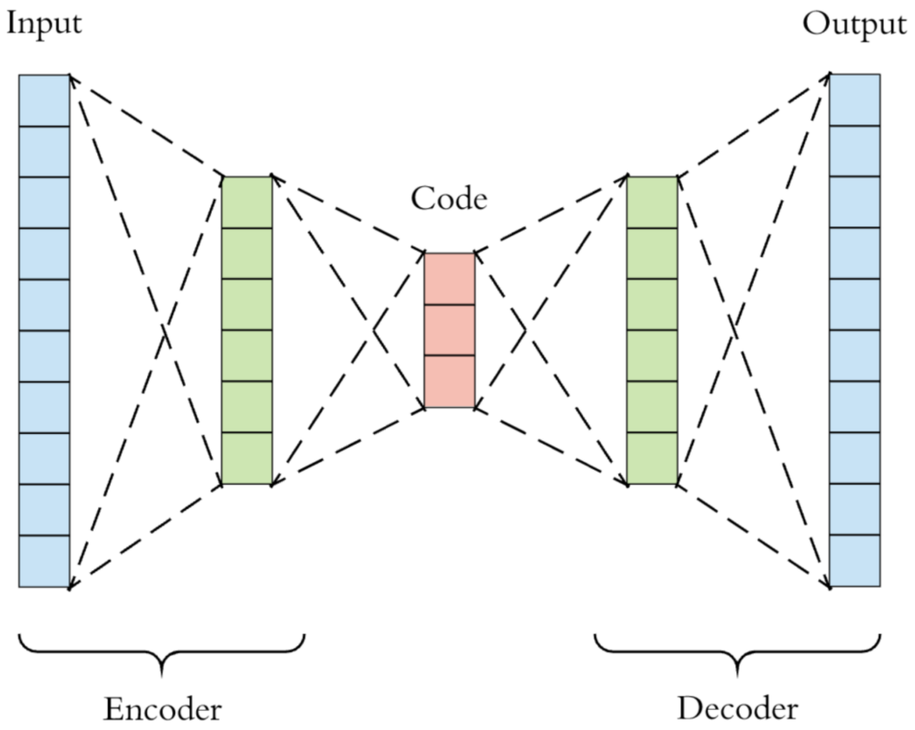
- **File Structure**: Indicates which binary functions belong to a source file.
- **Homologous Functions**: Binary functions originating from the same source file.
- **Continuous Distribution**: Homologous functions are placed continuously in a binary by the compiler.
- **Static Function**: Defined with the static keyword, visible only within the same file.
- **External Function**: Implemented in other binaries, such as APIs and system calls.

Background

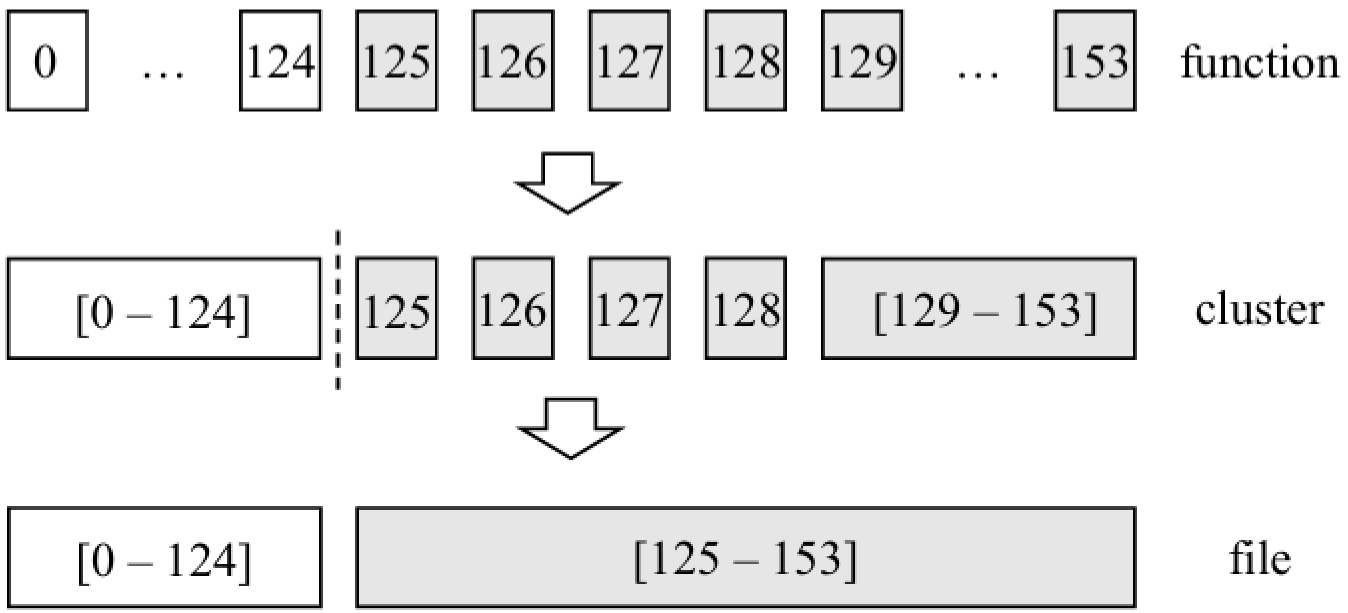
Key Techniques

- **Dynamic Programming (DP)**: Used to locate file boundaries by optimizing the file structure recovery process.

- **Encoder-Decoder Model**: Utilizes a graph neural network (GNN) to encode graph structure features for file name prediction.



Motivating Example

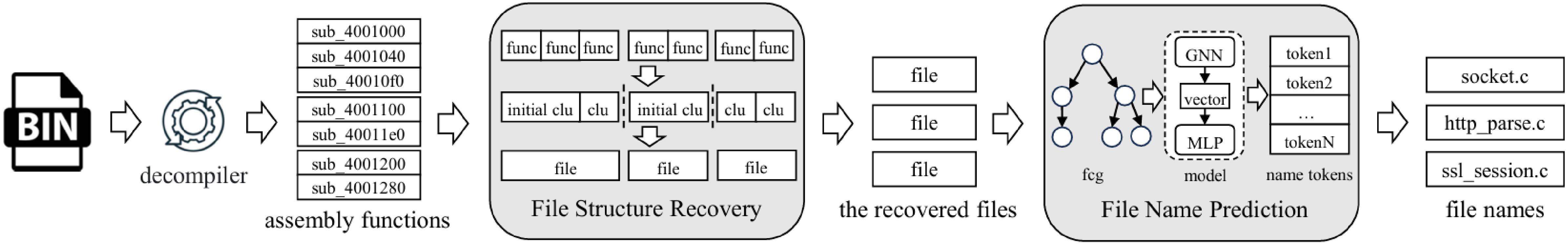


BCD utilizes a greedy strategy to cluster

An asynchronous event library *libev.so* is compiled from two source files, with one file containing 125 out of 154 functions, making it difficult to accurately identify all binary functions belonging to this file within the library.

Methodology

- **Overview**

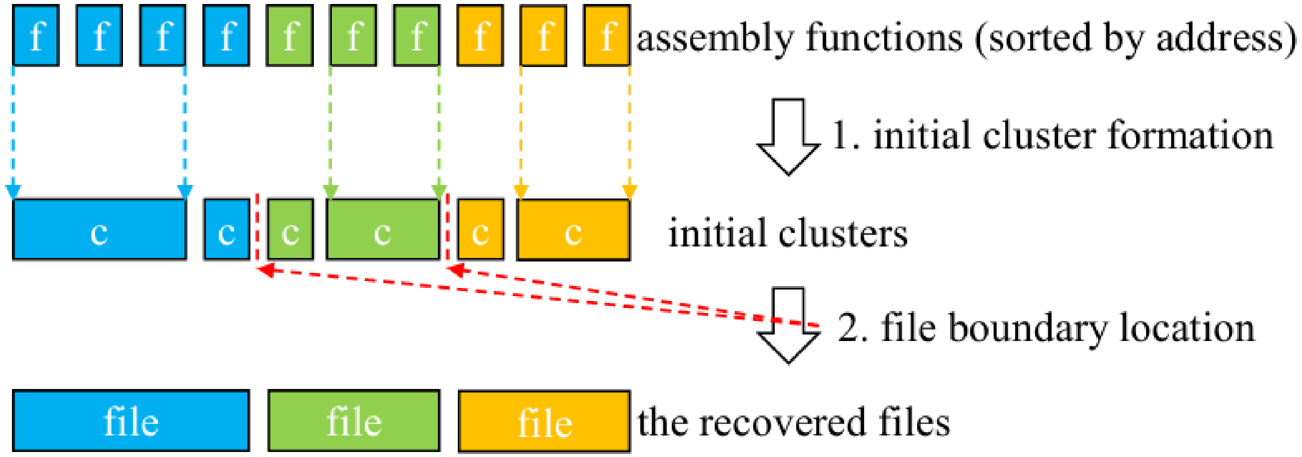


Overall Workflow of DeLink

Methodology

- **File Structure Recovery**
- **Core Idea**
  - Static Function Visibility
  - Continuous Function Placement

All binary functions between a static callee function and its caller function are from the same source file.

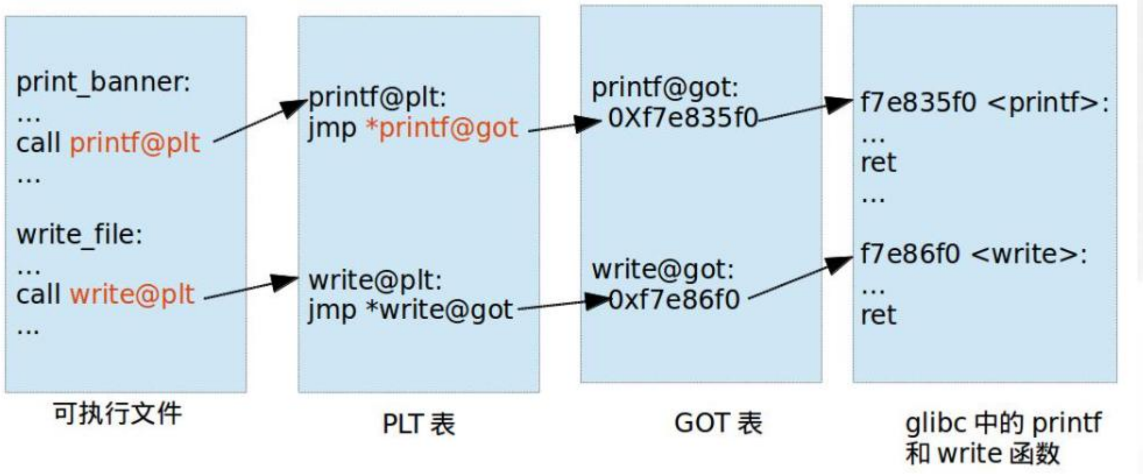


Overview of File Structure Recovery

Methodology

- **Identify static functions within dynamic libraries**

  - **Non-static functions** are typically called through the **Procedure Linkage Table (PLT)** and **Global Offset Table (GOT)**
  - **A static global variable** has the same visibility as a static function, meaning that functions which access this variable are **homologous**



Methodology

- **Identify static functions within executable program**

  - **A three-step heuristic approach**
  - 1. Function Set Construction
  - 2. Filtering Based on Index Distance
  - 3.Outlier Removal Using API References

$D = \{d_1, d_2, ..., d_n\}$, where d_i is the function index distance from the caller function f_i^+ to its callee function f^- (i.e., $d_i = i_i^+ - i^-$). Besides, average function index distance *aver_dis* is calculated by $\frac{1}{n}\sum_{i=1}^{n}|d_i|$.

Key Insights

•**Static functions** tend to be located close to their caller functions.

•**Non-static functions** might have larger, sometimes negative distances due to cross-file calls.

•**Header file analysis** helps differentiate valid function relationships from outliers.

Methodology

- **File Boundary Location**

Algorithm 1: File Boundary Location

```
Input   : the target binary B, the cluster set C.
Output  : the file boundary results R.
Function file_boundary_location(B, C):
    func_feats ← get_func_feature(B)
    feat_matrix ← gen_feat_matrix(func_feats)
    while e_clu not at the end of C do
        s_clu, e_clu ← get_clusters(C, file_bdy_pos)
        bdy_array ← dynamic_programming(s_clu, e_clu,
          feat_matrix)
        file_bdy_pos ← backtrack_pos(bdy_array)
        if number(file_bdy_pos) > 0 then
            R ← add_boundary(file_bdy_pos)
        end
    end
    return R
```

Steps

1. **Cluster Formation**:
2. **Function Relationship Matrix**
3. **Subset Processing**
4. **Dynamic Programming (DP) Optimization**
5. **Backtracking for Final Boundaries**:

$$Q = \frac{1}{2W} \sum_{i,j} \left[w_{ij} - \frac{k_i^{out} k_j^{in}}{2W} \right] \delta(C_i, C_j)$$

Methodology

- **File Name Prediction**
  - Code Feature Selection(22,458 source files from 663 open-source software projects)

Total Files	Files Named with Func Tokens	Proportion
22, 458	20, 387	90.78%

Statistics on Files Named with Function Name Tokens.

Function Level	File Token Number	Proportion
First Level	32, 738	58.57%
Second Level	28, 609	51.18%
Third Level	27, 240	48.73%
Fourth Level	23, 098	41.32%

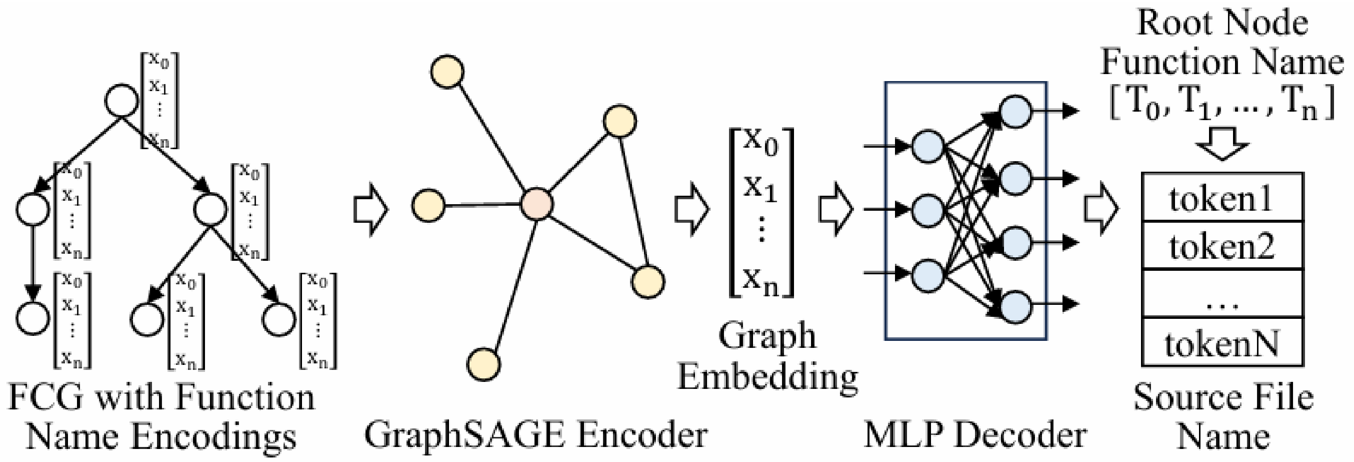
Statistics on File Name Token Occurrences across FunctionLevels.

FCG && The root node function

eg : in ssl_session.c, first level :ssl_session_init()

Methodology

- **Prediction Model**



- **Training**
  use source code directly

- **Prediction**
  when faced with binary with no symbols,use SymLM to predicate function name

Evaluation

- **File Structure Recovery Quality Evaluation**

metrics ：

1.Precision(P),Recall(R),and F_1score(F_1) to evaluate the **file structure recovery results**

$$P = \sum_{i=1}^{N_f} \frac{n_i}{n_f} P_i, \quad R = \sum_{i=1}^{N_f} \frac{n_i}{n_f} R_i, \quad F_1 = \sum_{i=1}^{N_f} \frac{n_i}{n_f} F_1^i$$

2.Jaccard(J),Fowllkes-Mallows(FM),Rand(RI) to evaluate ~ **from clustering perspective**

$$JC = \frac{a}{a+b+c}$$

$$FMI = \sqrt{\frac{a}{a+b} \cdot \frac{a}{a+c}}$$

$$RI = \frac{a+d}{a+b+c+d}$$

Evaluation

- **File Structure Recovery Quality Evaluation**

Dataset 1: consists of 106C/C++ executable programs from ModX and 14 C++ executable programs from BCD

e

Dataset 2: composed of 45 dynamic libraries,with 6 of them compiled using the“-fvisibility=hidden” optimization.

Approach	Results on Dataset I (%)						Results on Dataset II (%)					
	Precision	Recall	F_1 score	JC	FMI	RI	Precision	Recall	F_1 score	JC	FMI	RI
BCD	58.28	69.51	63.17	41.81	60.34	95.51	67.02	75.53	70.79	37.37	55.79	92.36
ModX	69.71	62.88	66.01	8.81	17.47	92.94	75.99	67.39	71.23	11.56	22.73	89.61
DeLink	**77.77**	**83.63**	**80.51**	**49.03**	**66.12**	**96.02**	**82.70**	**86.59**	**84.54**	**72.27**	**82.53**	**97.25**

Evaluation

- **File Name Prediction Accuracy**

Approach	Precision(%)	Recall(%)	F_1 score(%)
TF-IDF	57.22	47.44	51.87
Graph2Seq	63.37	51.32	56.71
DeLink	**70.09**	**63.91**	**66.86**

File Name Prediction Accuracy Comparison.

- **Limitation Analysis**

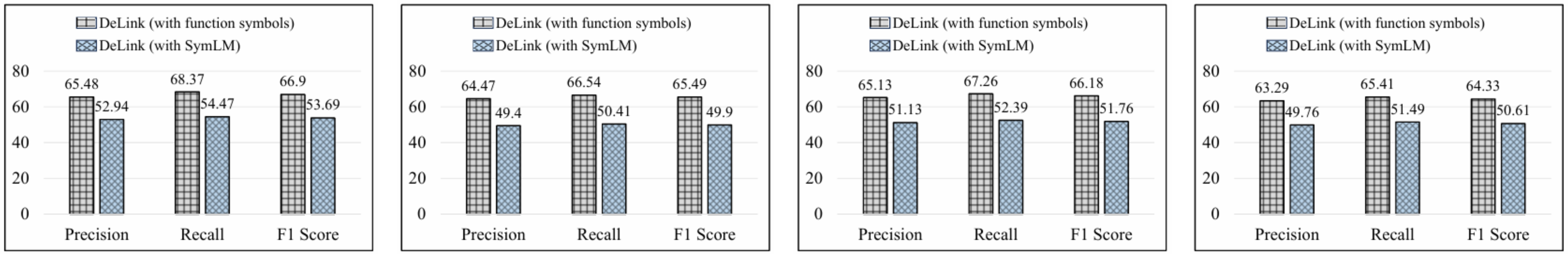
TF-IDF : Fixed Output Length && Vocabulary Coverage Issue

Graph2Seq: Error Propagation && Local Perspective Limitation

Evaluation

- **File Name Prediction Without Acess to function symbols**

  Dataset IV : consists of 1,118 files from 60 stripped binaries.



(a) O0 Optimization Level **(b) O1 Optimization Level** **(c) O2 Optimization Level** **(d) O3 Optimization Level**

The Accuracy of DeLink with Original Function Symbols and with SymLM

Evaluation

- **File Structure Recovery Time Comparison**

Recovery Time (s)	Dataset I	Dataset II	Average
BCD	44.99	7.26	34.43
ModX	170.25	33.01	131.79
DeLink	**20.78**	**6.41**	**16.74**

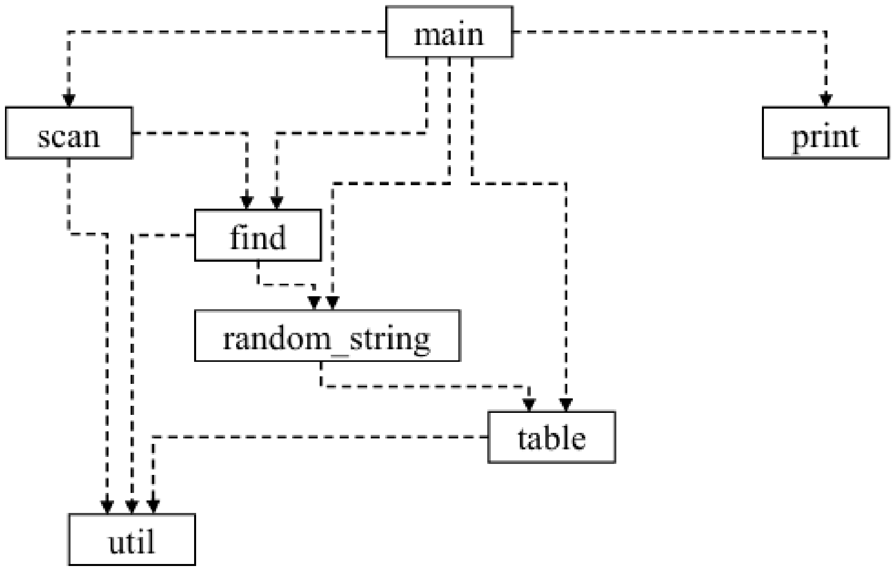
- **File Name Prediction Time Comparison**

Prediction Time (ms)	Average
TF-IDF	1.28
Graph2Seq	0.33
DeLink	**0.32**

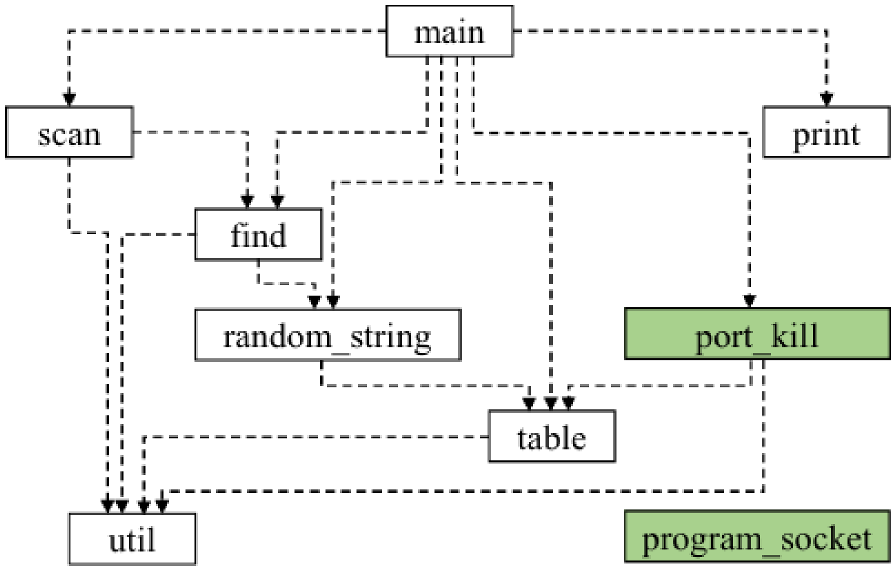
Application

- **Malware homology analysis**

Mirai: a well-known botnet program family that can be utilized for large-scale and destructive distributed denial of service (DDoS) attacks



(a) Mirai Sample A



(b) Mirai Sample B

End.