MVD: Memory-Related Vulnerability Detection Based on Flow-Sensitive Graph Neural Networks

Sicong Cao[1], Xiaobing Sun[1], Lili Bo[1], Rongxin Wu[2], Bin Li[1], Chuanqi Tao[3]

[1]Yangzhou University
[2]Xiamen University
[3]Nanjing University of Aeronautics and Astronautics

44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA

Background

Memory-related vulnerabilities can result in performance degradation and program crash, severely threatening the security of modern software.

Background

Memory-related vulnerabilities can result in performance degradation and program crash, severely threatening the security of modern software.

Rank	ID	Name	Score	2020 Rank Change
[1]	CWE-787	Out-of-bounds Write	65.93	+1
[2]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.84	-1
[3]	CWE-125	Out-of-bounds Read	24.9	+1
[4]	CWE-20	Improper Input Validation	20.47	-1
[5]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	19.55	+5
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	19.54	0
[7]	CWE-416	Use After Free	16.83	+1
[8]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	14.69	+4
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	14.46	0
[10]	CWE-434	Unrestricted Upload of File with Dangerous Type	8.45	+5
[11]	CWE-306	Missing Authentication for Critical Function	7.93	+13
[12]	CWE-190	Integer Overflow or Wraparound	7.12	-1
[13]	CWE-502	Deserialization of Untrusted Data	6.71	+8
[14]	CWE-287	Improper Authentication	6.58	0
[15]	CWE-476	NULL Pointer Dereference	6.54	-2
[16]	CWE-798	Use of Hard-coded Credentials	6.27	+4
[17]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	5.84	-12
[18]	CWE-862	Missing Authorization	5.47	+7
[19]	CWE-276	Incorrect Default Permissions	5.09	+22
[20]	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	4.74	-13
[21]	CWE-522	Insufficiently Protected Credentials	4.21	-3
[22]	CWE-732	Incorrect Permission Assignment for Critical Resource	4.2	-6
[23]	CWE-611	Improper Restriction of XML External Entity Reference	4.02	-4
[24]	CWE-918	Server-Side Request Forgery (SSRF)	3.78	+3
[25]	CWE-77	Improper Neutralization of Special Elements used in a Command ('Command Injection')	3.58	+6

Existing Efforts

- Static Analysis-Based Approaches

Checkmarx

FORTIFY An HP Company

coverity

Infer

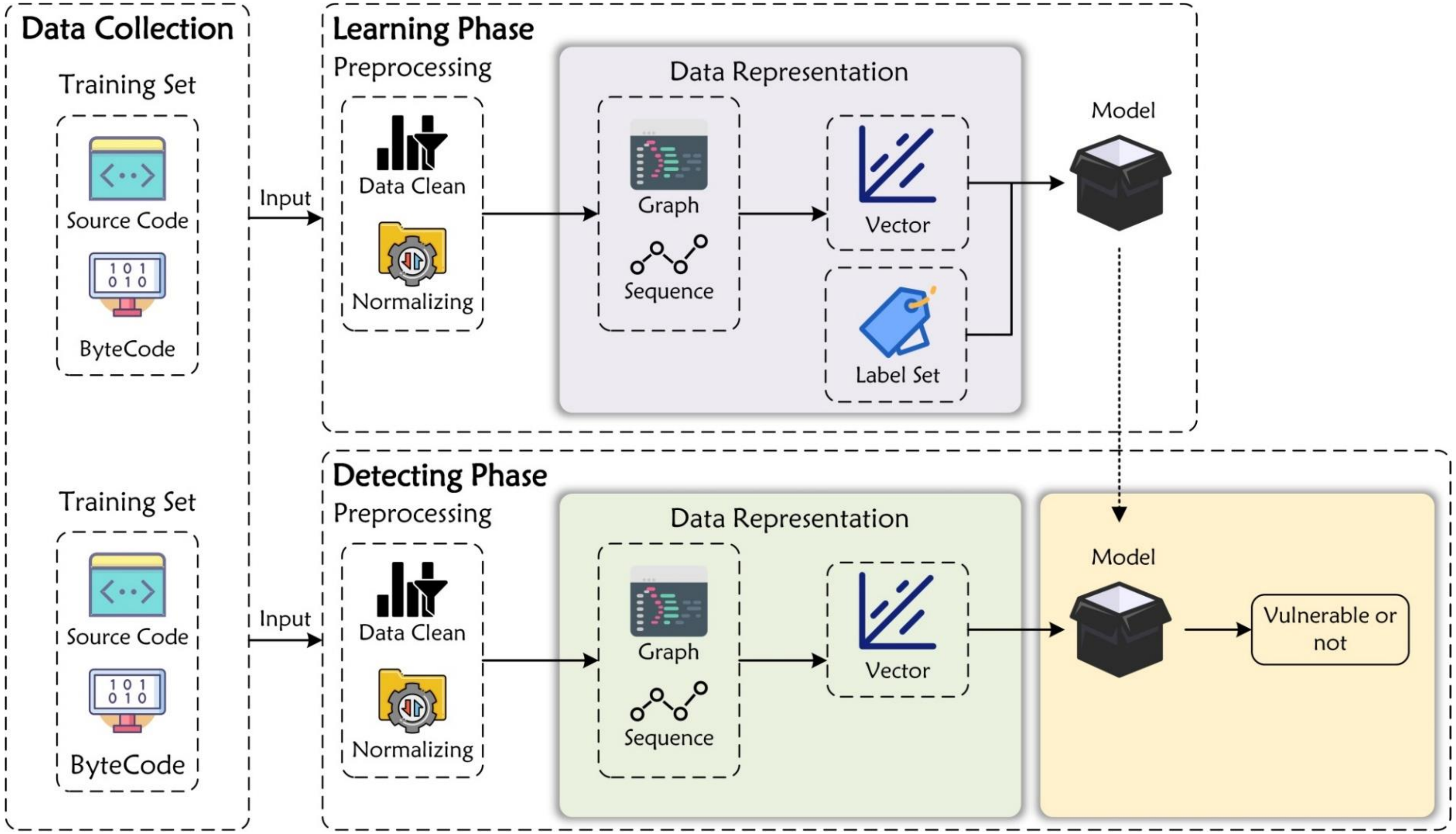
sonarqube

Limitations

- ☐ Highly dependent on pre-defined vulnerability rules/patterns crafted by security experts.
- ☐ The complex programming logic in real-world software projects gets in the way of the manual identification of the rules

Existing Efforts

- Deep learning-Based Approaches

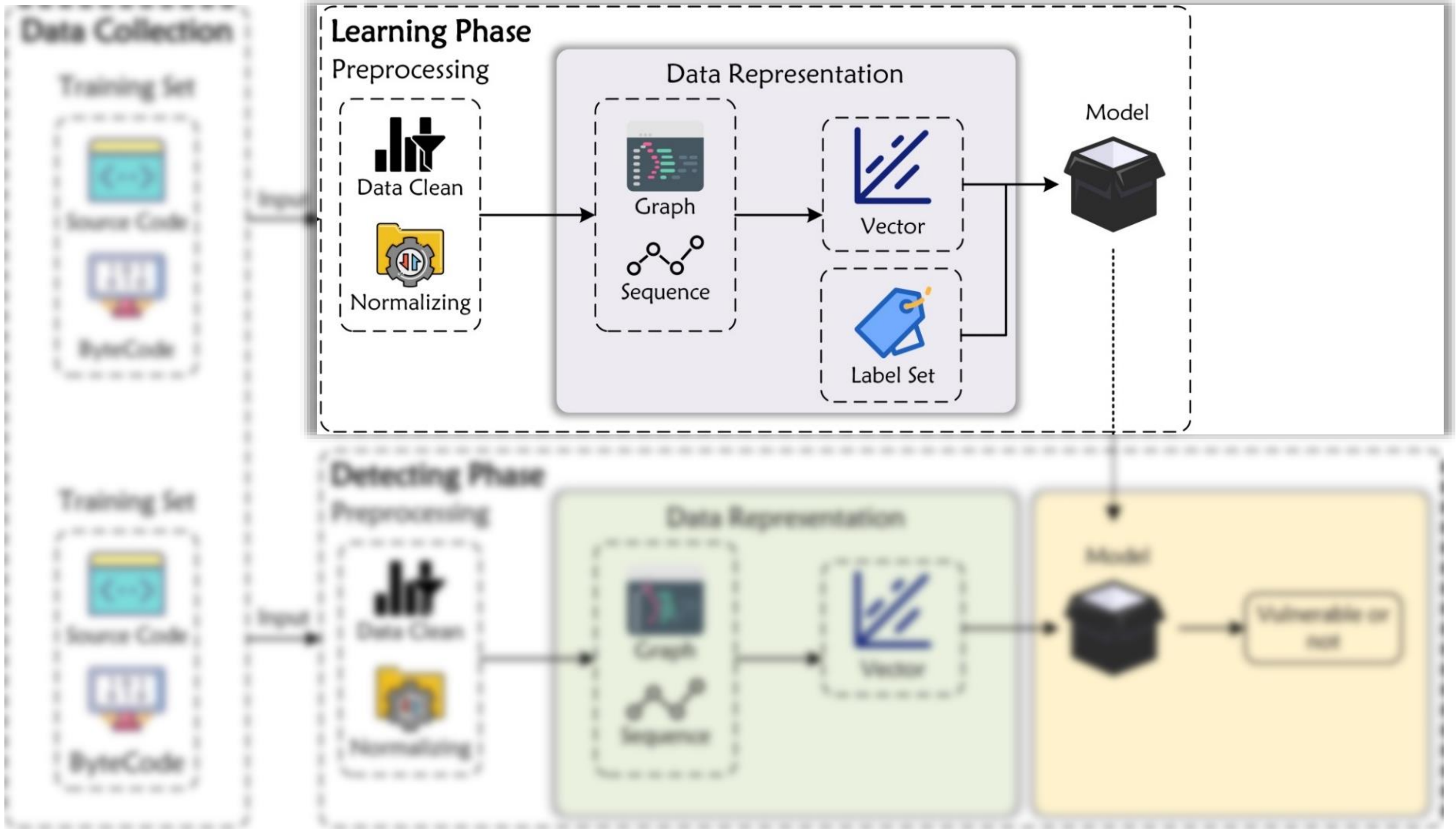


Existing Efforts

- Deep learning-Based Approaches

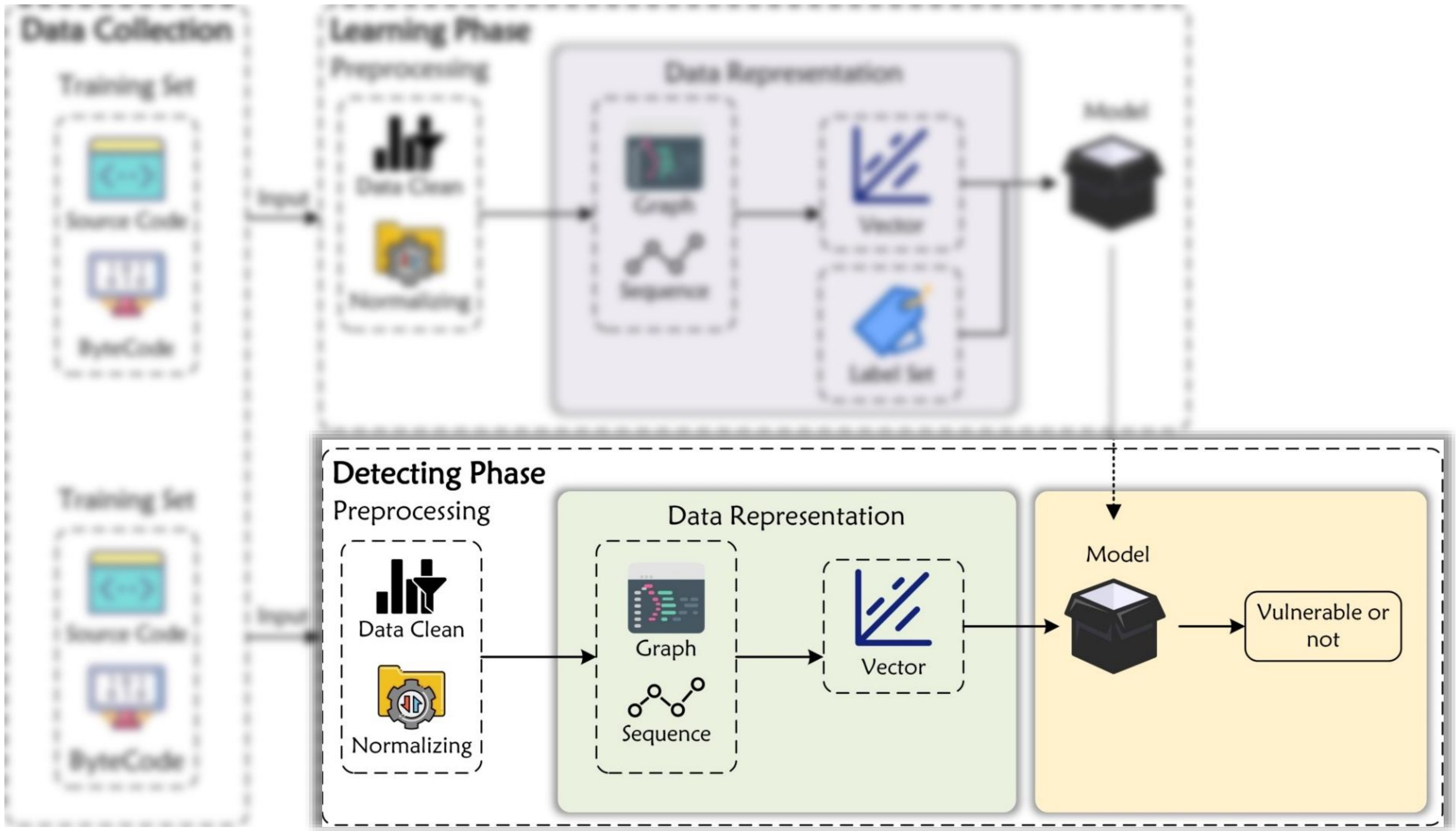
Existing Efforts

- Deep learning-Based Approaches



Existing Efforts

- Deep learning-Based Approaches



Limitations

```c
int SMB2_read(const unsigned int xid, struct cifs_io_parms
      *io_parms, unsigned int *nbytes, char **buf, int *buf_type)
{
    struct smb2_read_plain_req *req = NULL;
    ...
 -  cifs_small_buf_release(req);
    if (rc) {
      if (rc != -ENODATA) {
         trace_smb3_read_err(xid,req->PersistentFileId,
             io_parms->tcon->tid, ses->Suid, io_parms->offset,
             io_parms->length, rc);
      } else
           trace_smb3_read_done(xid,req->PersistentFileId,
             io_parms->tcon->tid, ses->Suid, io_parms->offset, 0);
      return rc == -ENODATA ? 0 : rc;
    } else
        trace_smb3_read_done(xid,req->PersistentFileId,
            io_parms->tcon->tid, ses->Suid, io_parms->offset,
            io_parms->length);
 +  cifs_small_buf_release(req);
    ...
     return rc;
  }
void cifs_small_buf_release(void *buf_to_free)
{
     ...
     mempool_free(buf_to_free, cifs_sm_req_poolp);
     ...
}
```

A Use-After-Free Vulnerability in Linux Kernel

- Flow Information Underutilization
  - Lack of interprocedural analysis.
  - Partial flow information loss in model training.
- Coarse Granularity
  - Focus on function-level or slice-level detection.

Observation 1. Comprehensive and precise inter-procedural flow analysis is necessary.

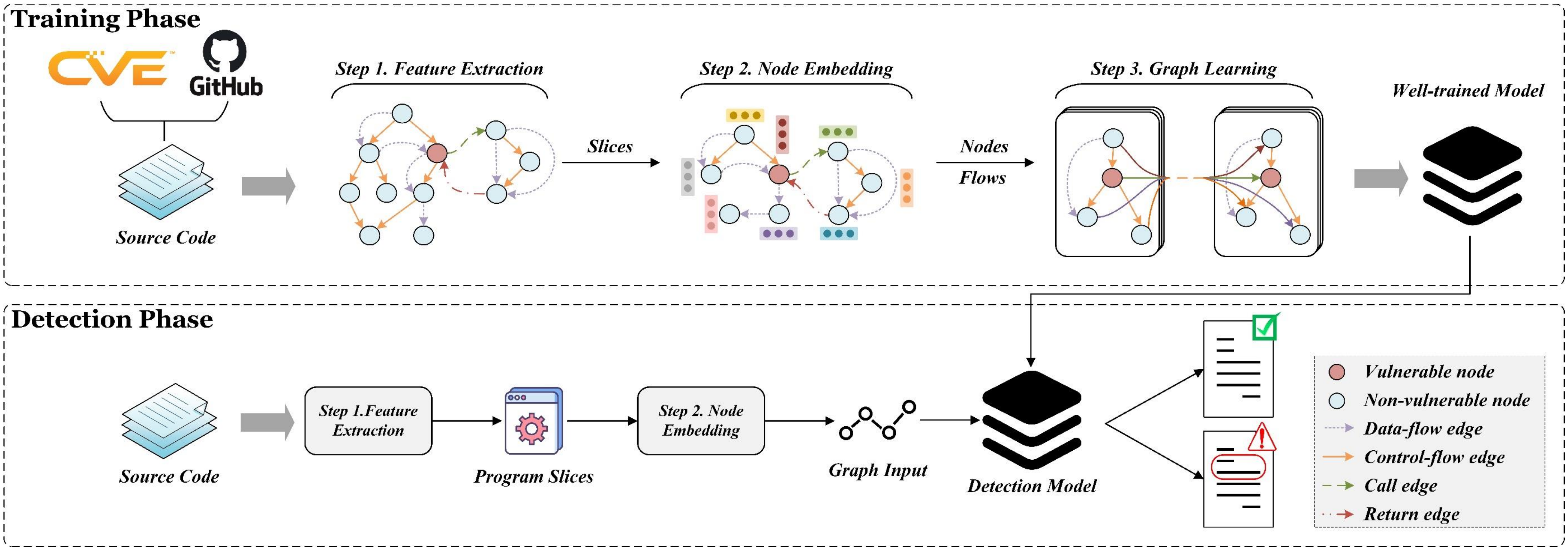
Observation 2. Sensitive contextual information within flows helps to refine detection granularity.

Our Solutions

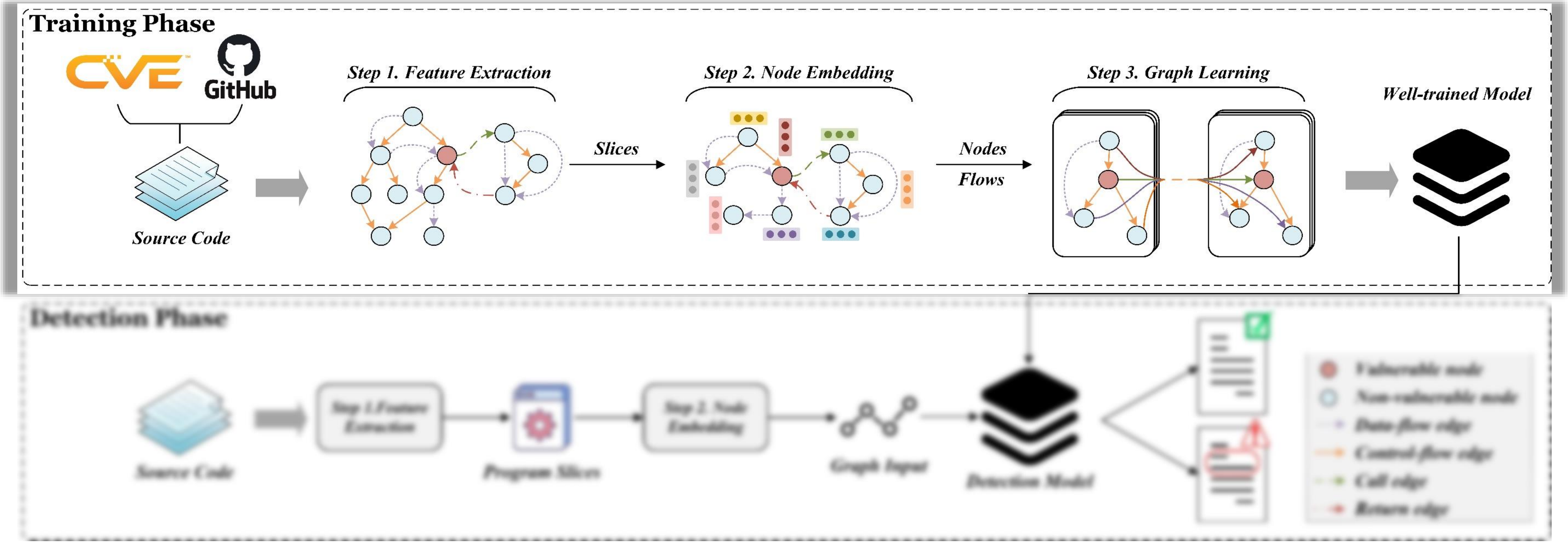
- Flow Information Underutilization
  - ☐ Lack of interprocedural analysis.
  - ☐ Partial flow information loss in model training.
- Coarse Granularity
  - ☐ Focus on function-level or slice-level detection.

- Fully Utilizing Flow Information
  - ✓ Combining Program Dependence Graph (PDG) with Call Graph (CG).
  - ✓ A novel Flow-Sensitive Graph Neural Networks (FS-GNN).
- Fine Granularity
  - ✓ Formalizing the detection of vulnerable statements as a node classification problem.

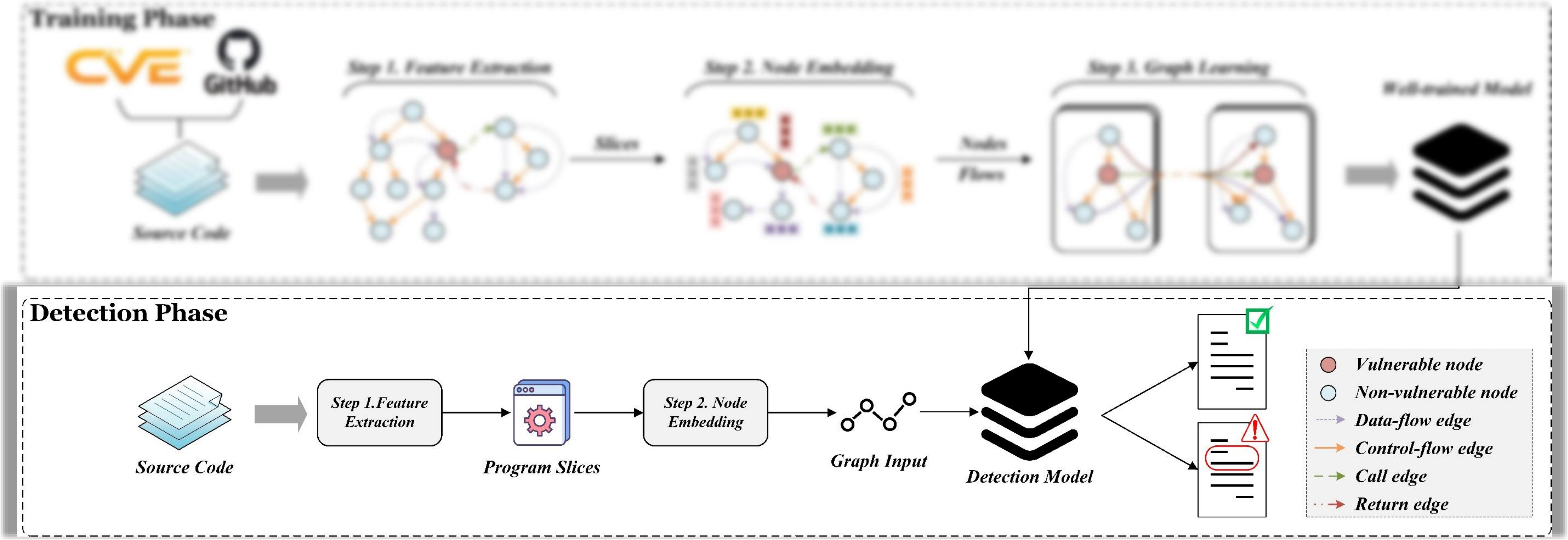
Workflow of *MVD*



Workflow of *MVD*



Workflow of *MVD*



Details of *MVD*

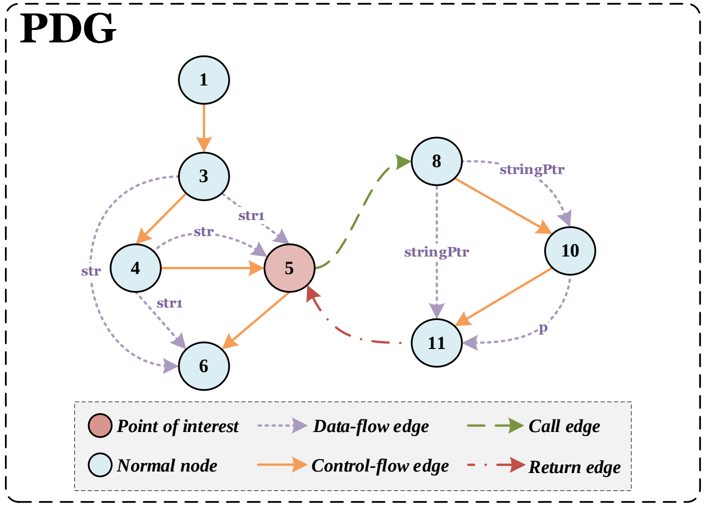
- Feature Extraction

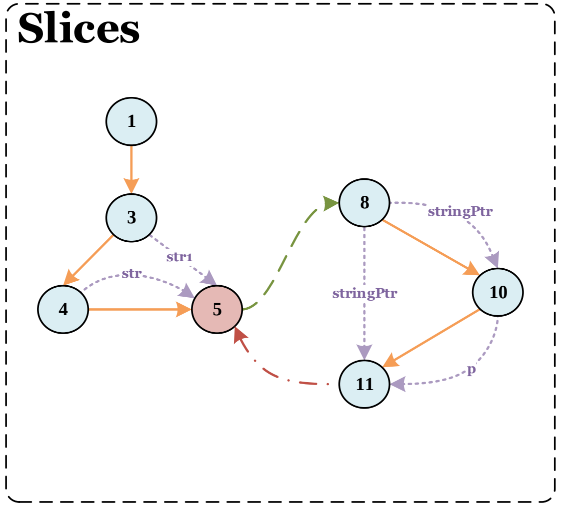
※ Program Dependence Graph + Call Graph

※ Program slicing → System API Calls → Pointer Variable

```
void memory_leak ()
{
    char *str = “This is a string”;
    char *str1;
    memory_leak_func (strlen(str), &str1);
    strcpy (str1, str);
}
void memory_leak_func (int len, char **stringPtr)
{
    char *p = malloc (sizeof(char) * (len + 1));
    *stringPrt = p;
}
```

(a) Exemplary Code Sample





(b) Program Slicing

Details of *MVD*

- Node Embedding

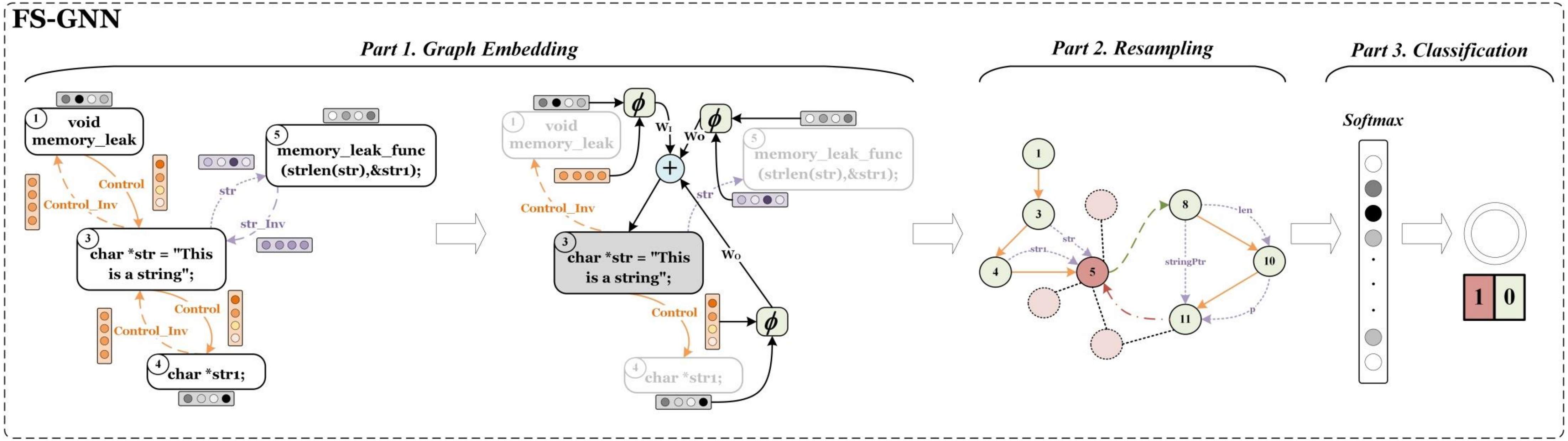
  ※ Doc2Vec [1]

- Graph Learning

  ※ Graph Embedding

  ※ Resampling

  ※ Classification



[1] Quoc V. Le and Tomá s Mikolov. Distributed Representations of Sentences and Documents. ICML 2014.

Details of *MVD*

- Node Embedding

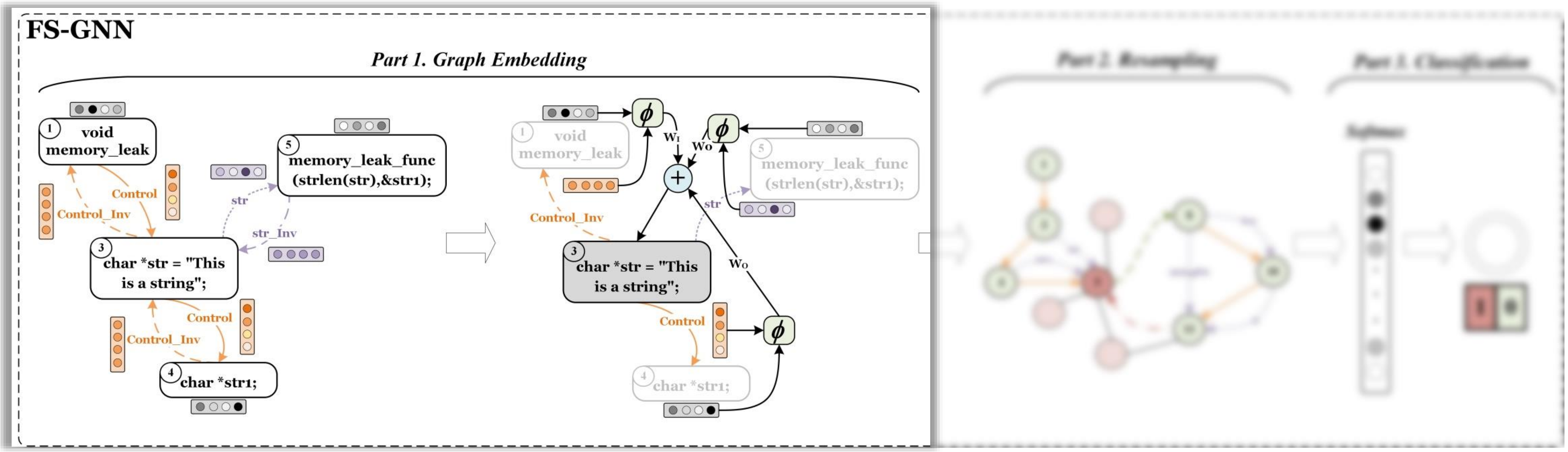
  ※ Doc2Vec [1]

- Graph Learning

  ※ Graph Embedding

  ※ Resampling

  ※ Classification



[1] Quoc V. Le and Tomás Mikolov. Distributed Representations of Sentences and Documents. ICML 2014.

Details of *MVD*

- Node Embedding

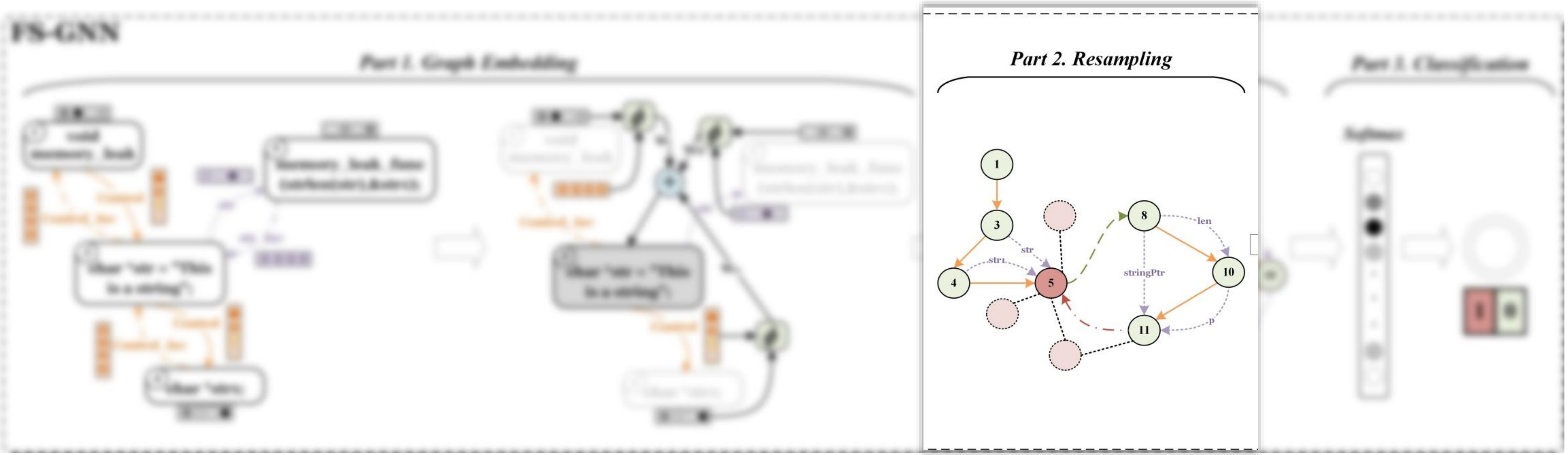
  ※ Doc2Vec [1]

- Graph Learning

  ※ Graph Embedding

  ※ Resampling

  ※ Classification



[1] Quoc V. Le and Tom á s Mikolov. Distributed Representations of Sentences and Documents. ICML 2014.

Details of *MVD*

- Node Embedding

  ※ Doc2Vec [1]

- Graph Learning

  ※ Graph Embedding

  ※ Resampling

  ※ Classification

[1] Quoc V. Le and Tom á s Mikolov. Distributed Representations of Sentences and Documents. ICML 2014.

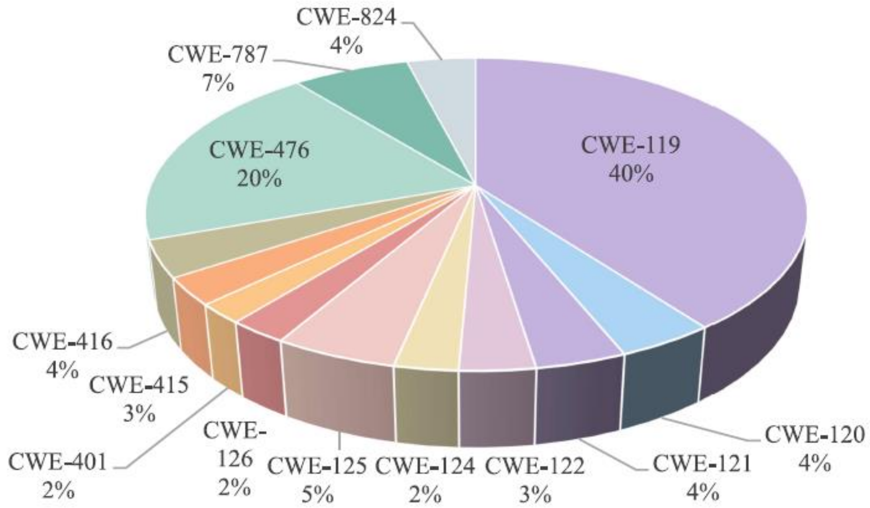
Evaluation

Research Questions

- **RQ1**: How effective is *MVD* compared to deep learning-based vulnerability detectors?
- **RQ2**: How effective is *MVD* compared to static analysis-based vulnerability detectors?
- **RQ3**: How effective is FS-GNN for memory-related vulnerability detection?
- **RQ4**: How efficient are *MVD* and baselines in terms of their time cost for detecting memory-related vulnerabilities?

DataSet

Project	#Version	#Samples	#Vertices	#Edges
Linux Kernel	2.6-4.20	868	26,917	29,512
FFmpeg	0.5-4.1	73	1,971	2,168
Asterisk	1.4-16.14	18	468	502
Libarchive	2.2-3.4	11	235	269
Libming	0.4.7	7	119	141
LibTIFF	3.8-4.0	24	584	639
Libav	12.3	16	526	573
LibPNG	1.0.x-1.6.x	13	392	447
QEMU	0.9-4.3	121	4,711	5,308
Wireshark	1.2-3.2	57	2,056	2,190
SARD	-	3,145	11,237	13,049
Total	-	4,353	49,216	54,798



Evaluation

- **RQ1**: How effective is *MVD* compared to deep learning-based vulnerability detectors?

Approach	A (%)	P (%)	R (%)	F1 (%)
VulDeePecker[1]	60.9	51.4	35.1	41.7
SySeVR [2]	63.4	53.3	62.9	57.7
Devign [3]	68.3	54.8	66.1	59.9
MVD	**74.1**	**61.5**	**69.4**	**65.2**

Answer to RQ1: In comparison with the popular DL-based approaches, *MVD* achieves better detection performance by fully utilizing flow information via interprocedural analysis and FS-GNN.

```
static bool try_merge_free_space(...){
    ...
    right_info = tree_search_offset(ctl, offset + bytes, 0, 0);
    if (right_info && rb_prev(&right_info->offset_index))
        left_info = rb_entry(rb_prev(&right_info->offset_index),
                    struct btrfs_free_space, offset_index);
        else
            left_info = tree_search_offset(ctl, offset - 1, 0, 0);
        if (...) {...
            kmem_cache_free(btrfs_free_space_cachep, right_info);
            Merged = true;
        if (...) {...
            info->offset = left_info->offse;
            info->bytes += left_info->bytes}
        return merged;
}
```

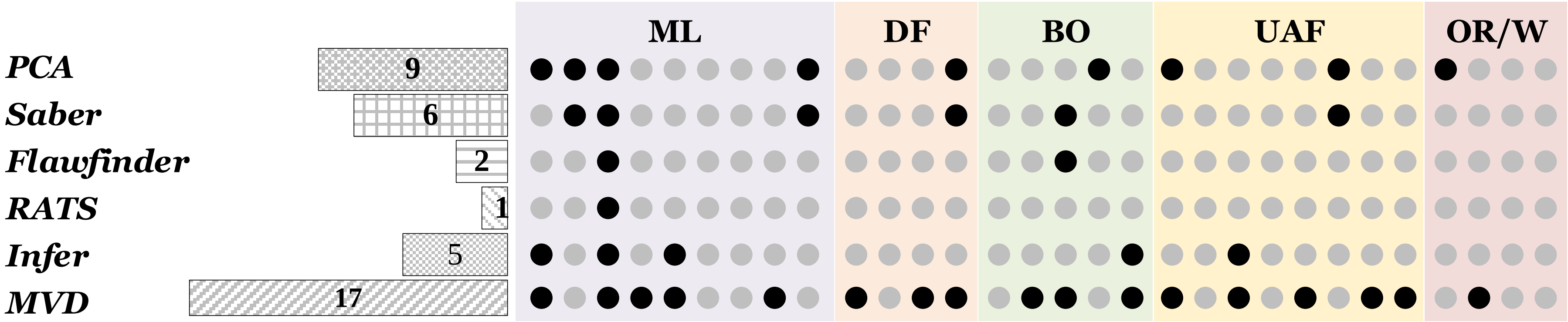
(a) A Vulnerability Missed by *Devign*

[1] Z. Li et al. VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. NDSS 2018.
[2] Z. Li et al. SySeVR: A Framework for Using Deep Learning to Detect Software Vulnerabilities. TDSC 2021.
[3] Y. Zhou et al. 2019. Devign: Effective Vulnerability Identification by Learning Comprehensive Program Semantics via Graph Neural Networks. NeurIPS 2019.

Evaluation

- **RQ2**: How effective is *MVD* compared to static analysis-based vulnerability detectors?

Approach	A (%)	P (%)	R (%)	F1 (%)
PCA [1]	65.2	48.9	61.1	54.3
Saber [2]	64.4	47.6	59.2	52.8
Flawfinder	61.1	18.2	23.5	20.5
RATS	56.3	7.9	11.6	9.4
Infer	50.7	33.1	54.8	41.3
MVD	**67.6**	**54.8**	**63.6**	**58.9**



[1] W. Li et al. PCA: memory leak detection using partial call-path analysis. ESEC/FSE 2020.
[2] Y. Sui et al. Static memory leak detection using full-sparse value-flow analysis. ISSTA 2012.

Evaluation

- **RQ2**: How effective is *MVD* compared to static analysis-based vulnerability detectors?

```
static int l2tp_ip_bind(struct sock *sk, struct sockaddr *uaddr,
    int addr_len){
    ...
-   if (!sock_flag(sk, SOCK_ZAPPED))
-       return -EINVAL;
    ...
    read_unlock_bh(&l2tp_ip_lock);
    lock_sock (sk);
+   if (!sock_flag(sk, SOCK_ZAPPED))
+       goto out;
    ...
}
```

Answer to RQ2: With the advantage of deep learning models in mining implicit vulnerability patterns, *MVD* performs better in comparison with the popular static analysis-based approaches.

Evaluation

- **RQ3**: How effective is FS-GNN for memory-related vulnerability detection?

Approach	A (%)	P (%)	R (%)	F1 (%)
GCN	61.2	17.3	8.2	11.1
GGNN	69.4	41.8	52.5	46.5
RGCN	72.7	49.3	58.1	53.3
FS-GNN	**77.5**	**56.4**	**62.9**	**59.5**

Answer to RQ3: FS-GNN can effectively contribute to the performance of *MVD*, as it can better capture the structured information of vulnerable code.

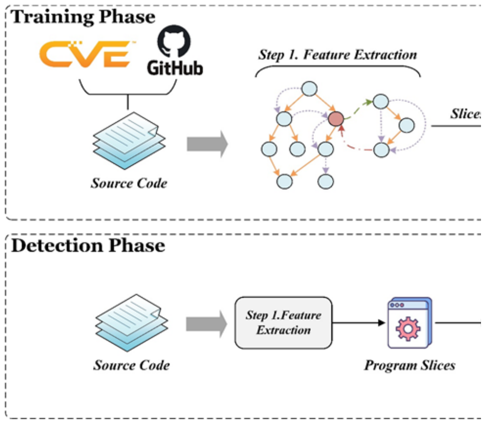
- **RQ4**: How efficient are *MVD* and baselines in terms of their time cost for detecting memory-related vulnerabilities?

Method	*MVD*	*VulDeePecker*	*SySeVR*	*Devign*	*PCA*	*Saber*	*Infer*	*Flawfinder*	*RATS*
Training Time(s)	2386.2	1019.5	1833.9	2583.7	N/A	N/A	N/A	N/A	N/A
Detection Time(s)	10.4	8.1	9.7	11.9	**9.2**	11.8	145.8	17.4	20.6

Answer to RQ4: In spite of a great deal of training time, *MVD* achieves relatively shorter detection time with better detection results, making a trade-off between accuracy and efficiency.

Conclusion

Workflow of *MVD*



Evaluation

Research Questions

- **RQ1**: How effective is *MVD* compared to deep learning-based vulnerability detectors?
- **RQ2**: How effective is *MVD* compared to static analysis-based vulnerability detectors?
- **RQ3**: How effective is FS-GNN for memory-related vul
- **RQ4**: How efficient are *MVD* and baselines in terms of

DataSet

Project	#Version	#Samples	#Vertices	#Edges
Linux Kernel	2.6-4.20	868	26,917	29,512
FFmpeg	0.5-4.1	73	1,971	2,168
Asterisk	1.4-16.14	18	468	502
Libarchive	2.2-3.4	11	235	269
Libming	0.4.7	7	119	141
LibTIFF	3.8-4.0	24	584	639
Libav	12.3	16	526	573
LibPNG	1.0.x-1.6.x	13	392	447
QEMU	0.9-4.3	121	4,711	5,308
Wireshark	1.2-3.2	57	2,056	2,190
SARD	-	3,145	11,237	13,049
Total	-	4,353	49,216	54,798

Evaluation

- **RQ1**: How effective is *MVD* compared to deep learning-based vulnerability detectors?

Approach	A (%)	P (%)	R (%)	F1 (%)
VulDeePecker	60.9	51.4	35.1	41.7
SySeVR	63.4	53.3	62.9	57.7
Devign	68.3	54.8	66.1	59.9
MVD	**74.1**	**61.5**	**69.4**	**65.2**

```
static bool try_merge_free_space(...){
    ...
    right_info = tree_search_offset(ctl, offset + bytes, 0, 0);
    if (right_info && rb_prev(&right_info->offset_index))
        left_info = rb_entry(rb_prev(&right_info->offset_index),
                struct btrfs_free_space, offset_index);
        else
            left_info = tree_search_offset(ctl, offset - 1, 0, 0);
        if (...) {...
            kmem_cache_free(btrfs_free_space_cachep, right_info);
            Merged = true;
        if (...) {...
            info->offset = left_info->offse;
            info->bytes += left_info->bytes}
        return merged;
}
```

(b) A Vulnerability Missed by *Devign*

Answer to RQ1: In comparison with the popular DL-based approaches, *MVD* achieves better detection performance by fully utilizing flow information via interprocedural analysis and FS-GNN.

15

Thank you!