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

Sicong Cao¹, Xiaobing Sun¹, Lili Bo¹, Rongxin Wu², Bin Li¹, Chuanqi Tao³

¹Yangzhou University
²Xiamen University
³Nanjing University of Aeronautics and Astronautics
Memory-related vulnerabilities can result in performance degradation and program crash, severely threatening the security of modern software.
Memory-related vulnerabilities can result in performance degradation and program crash, severely threatening the security of modern software.
Existing Efforts

• Static Analysis-Based Approaches

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

- **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 interprocedural flow analysis is necessary.

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

A Use-After-Free Vulnerability in Linux Kernel
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

Training Phase

Source Code

Step 1. Feature Extraction

Step 2. Node Embedding

Step 3. Graph Learning

Well-trained Model

Detection Phase

Source Code

Step 1. Feature Extraction

Program Slices

Step 2. Node Embedding

Graph Input

Detection Model
Workflow of MVD
Details of MVD

- Feature Extraction
  - Program Dependence Graph + Call Graph
  - Program slicing
  - System API Calls
  - Pointer Variable

### Exemplary Code Sample

```c
1  void memory_leak ()
2  {
3      char *str = “This is a string”;
4      char *str1;
5      memory_leak_func (strlen(str), &str1);
6      strcpy (str1, str);
7  }
8  void memory_leak_func (int len, char **stringPtr)
9  {
10     char *p = malloc (sizeof(char) * (len + 1));
11     *stringPrt = p;
12  }
```

### Program Slicing

(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 Tomas 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 Tomas 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 Tomas 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 Tomas 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.

```
1 static bool try_merge_free_space(...){
2   ... 
3   right_info = tree_search_offset(ctl, offset + bytes, 0, 0);
4   if (right_info & rb_prev(&right_info->offset_index))
5       left_info = rb_entry(rb_prev(&right_info->offset_index),
6                              struct btrfs_free_space, offset_index);
7   else
8       left_info = tree_search_offset(ctl, offset - 1, 0, 0);
9   if (...) {...
10      kmem_cache_free(btrfs_free_space_cachep, right_info);
11     Merged = true;
12    if (...) {...
13       info->offset += left_info->offse;
14       info->bytes += left_info->bytes)
15     return merged;
16 }
```

(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   |
| Flawfinder | 61.1  | 18.2  | 23.5  | 20.5   |
| 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?

```c
static int l2tp_ip_bind(struct sock *sk, struct sockaddr *uaddr,
                         int addr_len){
...
3 - if (!sock_flag(sk, SOCK_ZAPPED))
4 -    return -EINVAL;
...
6   read_unlock_bh(&l2tp_ip_lock);
7   lock_sock (sk);
8   + if (!sock_flag(sk, SOCK_ZAPPED))
9   +    goto out;
10 ...
11 }
```

**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   | N | S | V | E |
|------------|-----------|---|---|---|---|
| LibHunt    | 2.6-2.8   | 66 | 20,172 | 29,412 |
| Hepig      | 0.5-1.0   | 75 | 1,517 | 2,160 |
| Android    | 0.4-0.6   | 10 | 66 | 382 |
| LibWing    | 0.8-0.8   | 63 | 215 | 216 |
| LibTIFP    | 0.6-0.8   | 74 | 184 | 639 |
| LibFir     | 0.5-0.6   | 16 | 950 | 573 |
| LAPI      | 0.6-0.6   | 54 | 842 | 847 |
| GEOM      | 0.6-0.6   | 123 | 4,711 | 7,300 |
| Wazuh     | 0.6-0.6   | 37 | 2,066 | 2,390 |
| VVD        | 0.6-0.6   | 4,045 | 11,217 | 15,869 |
| Total      | 0.6-0.6   | 4,045 | 49,234 | 56,705 |

Approach | A (%) | P (%) | R (%) | F1 (%) |
|-----------|-------|-------|-------|--------|
| ValDDeePecker | 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 |

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.

(b) A Vulnerability Missed by Devign

1. static bool try_merge_free_space(...) {
2.   right_info = tree_search_offset(ctl, offset + bytes, 0, 0);
3.   if (right_info && right_info->offset_index)
4.   { left_info = try_merge_div_free(right_info->offset_index),
5.     struct bitfs_free_space_offset_index;}
6.   else
7.   { left_info = tree_search_offset(ctl, offset - 1, 0, 0);
8.     if (...) 
9.     { merge = true;
10.   }
11.   }
12.   info_bytes = left_info->offset
13.   info_bytes += left_info->offset
14.   return merge;
15. }

24
Thank you!