33rd USENIX Security Symposium (USENIX Security '24)

# FVD-DPM: Fine-grained Vulnerability Detection via Conditional Diffusion Probabilistic Models

**Miaomiao Shao**, Yuxin Ding

Harbin Institute of Technology, Shenzhen, China

Email: 21B951007@stu.hit.edu.cn

August 16, 2024

# CONTENTS

# Part 1 Research background

# Automatic Software Vulnerability Detection

- ➢ Software vulnerabilities pose a significant threat to software security
- ➢ Existing vulnerability detection approaches
  - Symbolic execution
  - Rule-base techiniques
  - Code similarities
  - Deep Learning

# Drawbacks of existing deep learning-based approaches

## Program semantics have not been fully leveraged

- Token sequence ignores the structural information of programs
- Graph-based representations, e.g., AST, CFG, DFG, PDG, extract program semantics from individual functions, disregarding call relationships between functions

## Detection granularity is coarse-grained

- Detection granularity is mostly at the file-level, function-level, slice-level
- Vulnerabilities always involve only a few statements

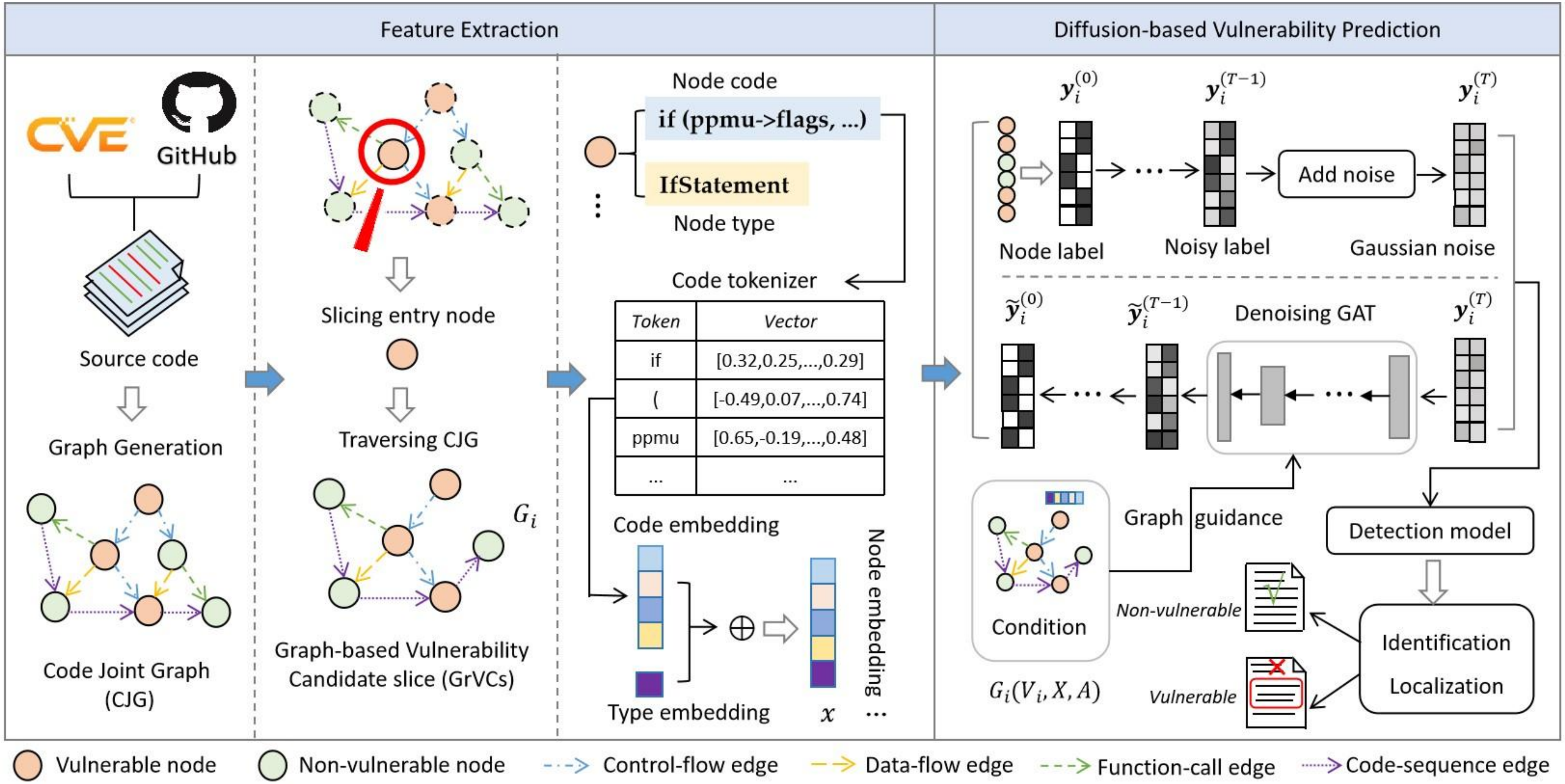
```
  int ksmbd_conn_handler_loop(void *p)
  {
      struct ksmbd_conn *conn = (struct ksmbd_conn *)p;
      unsigned int pdu_size, max_allowed_pdu_size;
      ...
      conn->request_buf = kvmalloc(size, GFP_KERNEL);
      if (!conn->request_buf)
          break;
      memcpy(conn->request_buf, hdr_buf, sizeof(hdr_buf));
-     if (!ksmbd_smb_request(conn))
-         break;
      ...
      if (size != pdu_size) {
          pr_err("PDU error. Read: %d, Expected: %d\n", size, pdu_size);
          continue;
      }
+     if (!ksmbd_smb_request(conn))
+         break;
      ...
  }
  bool ksmbd_smb_request(struct ksmbd_conn *conn)
  {
-     return conn->request_buf[0] == 0;
+     __le32 *proto = (__le32 *)smb2_get_msg(conn->request_buf);
+     if (*proto == SMB2_COMPRESSION_TRANSFORM_ID) {
+         pr_err_ratelimited("smb2 compression not support yet");
+         return false;
+     }
+     if (*proto != SMB1_PROTO_NUMBER &&*proto != SMB2_PROTO_NUMBER &&
+         *proto != SMB2_TRANSFORM_PROTO_NUM)
+         return false;
+     return true;
  }
```

An out-of-bounds read vulnerability (CVE-2023-38430)

Part 2

# Main research content

# Overview of FVD-DPM



# Step I: Feature extraction

## Generating Code Joint Graph (CJG)

- Control Flow Graph (CFG)
- Data Flow Graph (DFG)
- Call Graph (CG)
- Code Sequence (CS)

Joern, Neo4j

## Extracting Slicing Entry Nodes

- API/library function calls
- Sensitive variables (array and pointer variables)
- Arithmetic expressions

# Step I: Feature extraction

## Program Slicing

- Start from the slicing entry node
- Iteratively perform forward and backward slicing until all nodes in the CJG are traversed

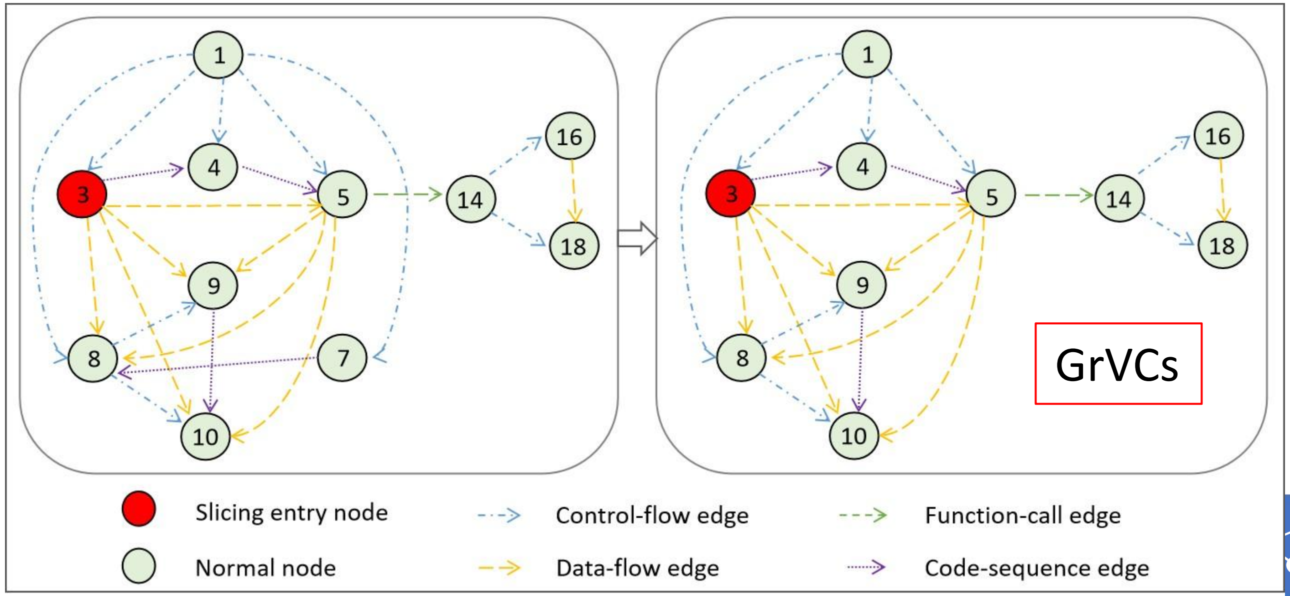
## Node Embedding

- Node type
- Node value

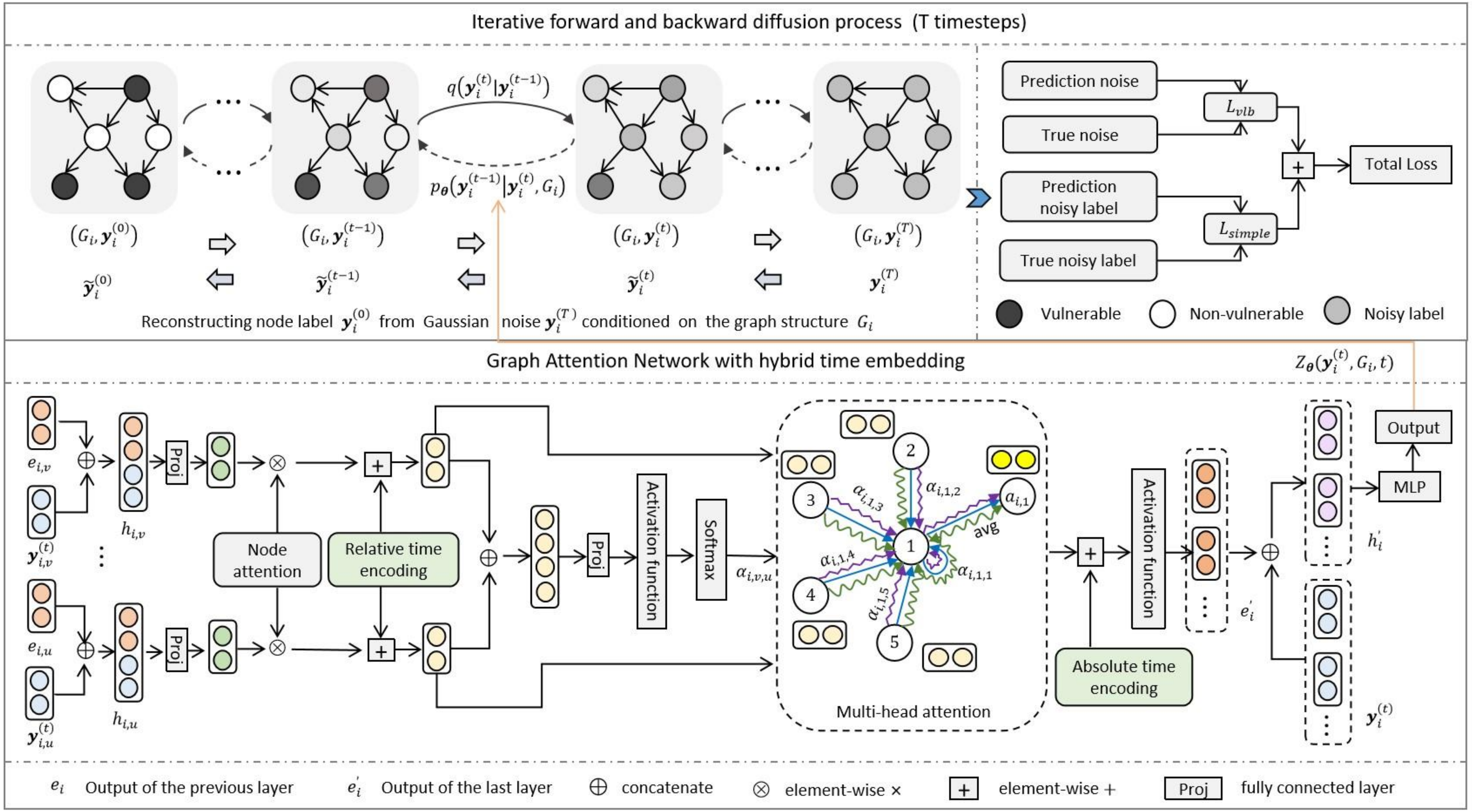
```
struct vfsmount *collect_mounts(struct path *path)
{
      struct mount *tree;
      namespace_lock();
      tree = copy_tree(real_mount(path->mnt), path->dentry, CL_COPY_ALL | CL_PRIVATE);

      namespace_unlock();
      if (IS_ERR(tree))
            return ERR_CAST(tree);
      return &tree->mnt;
}


struct mount *copy_tree(struct mount *mnt, struct dentry *dentry, int flag)
{
      struct mount *res, *p, *q, *r, *parent;
      ...
      return res;
}
```



# Step II: Diffusion-based Vulnerability Prediction



# 2 Main research content

We formalize the diffusion process using a GrVCs, denoted as $G_i(V_i, E_i)$. The graph $G_i(V_i, E_i)$ consists of a node set $V_i$ and an edge set $E_i$. The node label of the graph $G_i(V_i, E_i)$ is represented by $y_i$, with values of 0 (vulnerable) and 1 (non-vulnerable). Given that the node label $y_i$ is discrete, we relax it into an one-hot vector to yield continuous values.

## Forward Diffusion Process

- Node label $y_i^{(0)}$ conforms to the initial data distribution $q(y)$
- Gaussian noise is continuously injected into the data distribution during the forward diffusion process

$$q(y_i^{(1)}, \cdots, y_i^{(T)} | y_i^{(0)}) = \prod_{t=1}^{T} q(y_i^{(t)} | y_i^{(t-1)})$$

$$q(y_i^{(t)} | y_i^{(t-1)}) = N(y_i^{(t)}; \sqrt{1-\beta_t} y_i^{(t-1)}, \beta_t I)$$

## Conditional Reverse Process

- Reconstruction of the node label $y_i^{(0)}$ from Gaussian noise conditioned on the graph structure $G_i$ and $y_i^{(T)}$
- $y_i^{(T)}$ is sampled from the Gaussian distribution $N(0, I)$

$$p_\theta(y_i^{(0)}, \cdots, y_i^{(T-1)} | y_i^{(T)}, G_i) = \prod_{t=1}^{T} p_\theta(y_i^{(t-1)} | y_i^{(t)}, G_i)$$

$$p_\theta(y_i^{(t-1)} | y_i^{(t)}, G_i) = N(y_i^{(t-1)}; \mu_\theta(y_i^{(t)}, G_i), \Sigma_\theta)$$

## Learning the mean and variance

- Calculate the inverse distribution $q(y_i^{(t-1)}|y_i^{(t)}, y_i^{(0)})$
- Bayes theorem

$$q(y_i^{(t-1)}|y_i^{(t)}, y_i^{(0)}) = q(y_i^{(t)}|y_i^{(t-1)}, y_i^{(0)}) \frac{q(y_i^{(t-1)}|y_i^{(0)})}{q(y_i^{(t)}|y_i^{(0)})}$$

- $q(y_i^{(t-1)}|y_i^{(t)}, y_i^{(0)})$ is a Gaussian distribution denoted as $N(\hat{\mu}_t, \widehat{\Sigma}_t)$

$$\hat{\mu}_t = \frac{1}{\sqrt{\alpha_t}}(y_i^{(t)} - \frac{\beta_t}{\sqrt{1-\overline{\alpha}_t}})\overline{Z}_t$$

$$\widehat{\Sigma}_t = \frac{1-\overline{\alpha}_{t-1}}{1-\overline{\alpha}_t}\beta_t$$

- $p_\theta(y_i^{(t-1)}|y_i^{(t)}, G_i)$ is also a Gaussian distribution denoted as $N(\mu_\theta(y_i^{(t)}, G_i), \Sigma_\theta)$

$$\mu_\theta(y_i^{(t)}, G_i) = \frac{1}{\sqrt{\alpha_t}}(y_i^{(t)} - \frac{\beta_t}{\sqrt{1-\overline{\alpha}_t}})Z_\theta(y_i^{(t)}, G_i)$$

$$\Sigma_\theta = \exp(\kappa \log\beta_t + (1-\kappa)\log\widehat{\Sigma}_t)$$

## GAT with Hybrid Time Encoding

- Absolute time encoding
- Relative time encoding

$$\alpha_{i,v,u}^m = \frac{\exp(\phi(\omega^T[(W^m h_{i,v} \oplus W^m h_{i,u}) + rel(t)]))}{\sum_{l\in N_v} \exp(\phi(\omega^T[(W^m h_{i,v} \oplus W^m h_{i,l}) + rel(t)]))}$$

$$a_{i,v} = \sigma\left(\frac{1}{M}\sum_{m=1}^{M}\sum_{u\in N_v} \alpha_{i,v,u}^m (W^m h_{i,u} + rel(t))\right)$$

$$h_{i,v}^{'} = \varphi(a_{i,v} + abs(t)) \oplus y_{i,v}^{(t)}$$

$$Z_\theta(y_i^{(t)}, G_i) = \mathrm{MLP}(h_i^{'})$$

$\phi(\cdot)$ LeakyReLU

$\varphi(\cdot)$ ELU

# Part 3

# Research result

# Research questions

- How effective is FVD-DPM when compared to state-of-the-art vulnerability detection approaches?
- How effective is CJG in vulnerability detection compared to existing code representations?
- Can FVD-DPM perform better in vulnerability detection by incorporating hybrid time encoding into GAT, and simultaneously learning mean and variance of the noisy label distribution?
- How effective and precise is FVD-DPM in locating different types of vulnerabilities?

## Datasets

| Dataset | #Version | #Vul. Fs | #Fs | #Vul. GrVCs | #Non-Vul. GrVCs | #GrVCs | #Nodes | #Edges |
|---|---|---|---|---|---|---|---|---|
| NVD | - | 937 | 2,011 | 4,355 | 8,526 | 12,881 | 870,855 | 4,633,355 |
| SARD | | 2,851 | 5,879 | 4,742 | 22,720 | 27,462 | 240,202 | 580,908 |
| OpenSSL | 0.9.6-3.0.7 | 2,009 | 2,302 | 6,677 | 3,362 | 10,039 | 221,262 | 684,357 |
| Libav | 0.6-11.5 | 1,666 | 1,956 | 7,710 | 4,334 | 12,044 | 334,964 | 1,372,749 |
| Linux Kernel | 2.6-5.17 | 1,178 | 1,528 | 4,036 | 2,287 | 6,323 | 272,267 | 1,099,651 |
| Total | - | 8,641 | 13,676 | 27,520 | 41,229 | 68,749 | 1,939,550 | 8,371,020 |

## Metics

Recall (R) F1 score (F1) Area Under Curve (AUC)

Matthews Correlation Coefficient (MCC) Intersection over Union (IoU)

## Baselines

- Vulnerability identification (slice-level detection):

  *Cppcheck, Flawfinder, Devign, VulDeePecker, SySeVR, VulDeeLocator, MVD*

- Vulnerability localization (statement-level detection):

  *Cppcheck, DeepLineDP, VulDeeLocator*

# Research result

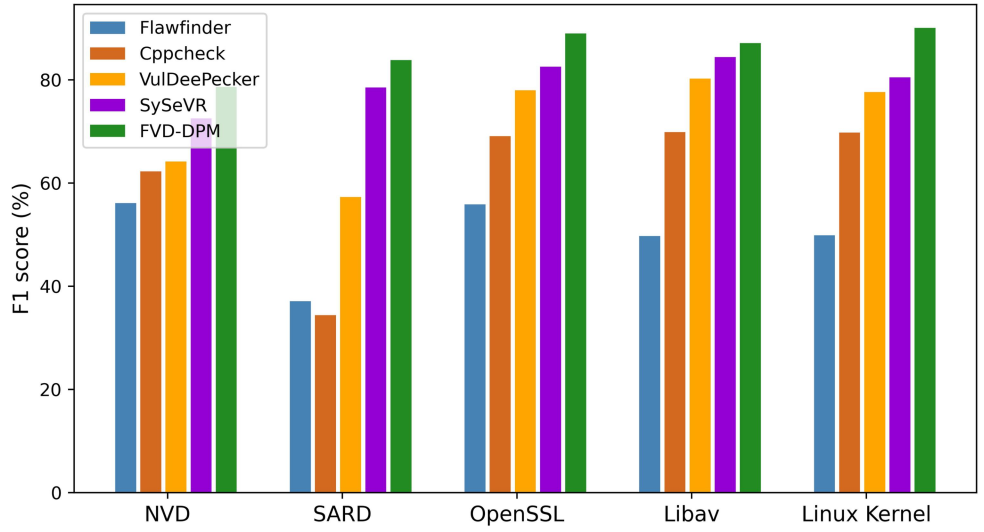
## Identification results (%)

| Method | F1 | R | AUC | MCC |
|---|---|---|---|---|
| Flawfinder | 49.73 | 52.86 | - | 10.07 |
| Cppcheck | 61.09 | 71.43 | - | - |
| MVD | 65.20 | 61.50 | - | - |
| VulDeePecker | 71.48 | 77.62 | 77.65 | 51.20 |
| Devign | 73.26 | - | - | - |
| SySeVR | 79.72 | 81.26 | - | 60.49 |
| VulDeeLocator | **85.90** | 82.07 | - | - |
| FVD-DPM (ours) | 85.73 | **82.93** | **86.40** | **72.14** |

## Localization results (IoU: %)

| Method | NVD | SARD | OenSSL | Libav | Linux Kernel |
|---|---|---|---|---|---|
| Cppcheck | 15.27 | 9.89 | 48.79 | 42.82 | 27.33 |
| DeepLineDP | 31.05 | 14.67 | 18.53 | 24.31 | 30.02 |
| VulDeeLocator | 32.60 | 36.30 | - | - | - |
| FVD-DPM | **59.04** | **72.35** | **63.13** | **62.95** | **72.70** |

## Results for RQ1




## FVD-DPM VS. VulChecker

| Method | CWE190 | CWE121 | CWE122 | CWE415 | CWE416 |
|---|---|---|---|---|---|
| VulChecker | 97.00 | 85.40 | 79.00 | 100.00 | 90.90 |
| FVD-DPM | 97.87 | 88.30 | 90.93 | 94.83 | 88.23 |

- FVD-DPM outperforms most existing state-of-the-art vulnerability detection approaches

# RQ2: Effectiveness of Code Joint Graph

Contributions of different edge types in Code Joint Graph (%)

| **Code representation** | **Vulnerability Identification** | | | | **Vulnerability Localization** | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **F1** | **R** | **AUC** | **MCC** | **F1** | **R** | **AUC** | **MCC** | **IoU** |
| CFG | 82.45 | 76.33 | 82.72 | 60.03 | 71.81 | 55.97 | 82.68 | 72.17 | 60.76 |
| CFG+DF | 82.69 | 79.22 | 84.73 | 69.16 | 79.29 | 77.90 | 88.91 | 79.22 | 61.14 |
| CFG+DF+CG | 82.74 | 80.02 | 85.02 | 69.10 | 78.95 | **78.94** | **89.41** | 78.88 | 61.77 |
| CFG+DF+CG+CS (CJG) | **85.28** | **82.28** | **85.91** | **70.51** | **79.60** | 77.15 | 88.53 | **79.55** | **64.90** |

- Overall, the model's performance gradually improved as we added different types of edges to the CFG
- The model's performance with *CFG+DF* significantly surpassed that of the *CFG*, highlighting the substantial contribution of data flow to extracting vulnerability features

3 Research result

# Results for RQ3: Ablation Study

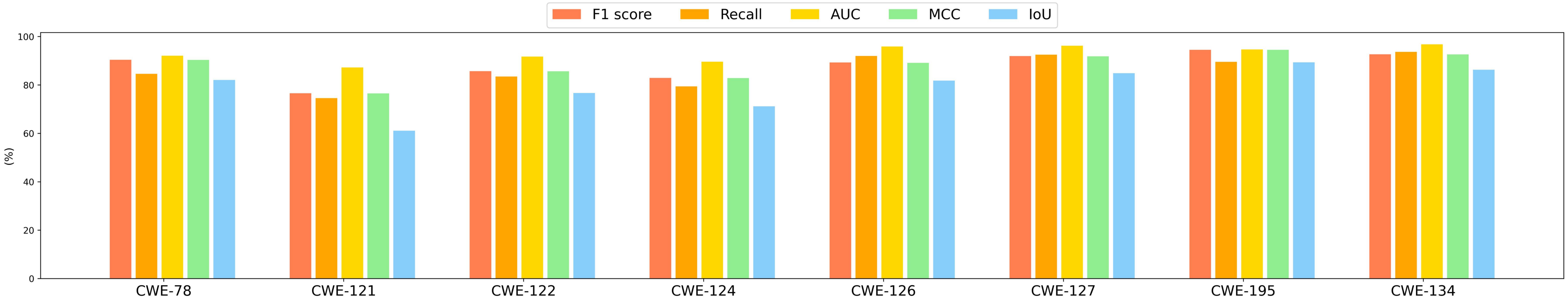
Comparative experiments on models with and without hybrid time encoding (%)

| **Time Encoding** | **Vulnerability Identification** | | | | **Vulnerability Localization** | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **F1** | **R** | **AUC** | **MCC** | **F1** | **R** | **AUC** | **MCC** | **IoU** |
| Without | 77.20 | 69.72 | 80.28 | 60.64 | 74.96 | 72.21 | 86.04 | 74.97 | 58.22 |
| With | 86.05 | 83.34 | 86.15 | 71.90 | 79.72 | 78.05 | 88.97 | 79.65 | 66.00 |

Experimental results achieved by different objectives

| **Objective** | **Vulnerability Identification** | | | | **Vulnerability Localization** | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | **F1** | **R** | **AUC** | **MCC** | **F1** | **R** | **AUC** | **MCC** | **IoU** |
| $L_{simple}$ | 84.98 | 81.64 | 85.32 | 71.05 | 77.82 | 74.76 | 87.32 | 77.88 | 63.67 |
| $L_{hybrid}$ | 86.41 | 83.62 | 86.30 | 72.61 | 79.62 | 77.08 | 88.48 | 79.63 | 66.05 |

# RQ4: Results on Different CWE Types




- FVD-DPM achieves good performance in locating vulnerable statements across different vulnerability types
- The vulnerability pattern of CWE-121 is complex and may involve multiple statements in various functions, making it more challenging to identify

# Part 4 Research prospect

3

## Research prospect

- Improve the interpretability of deep learning-based vulnerability detection approaches
- Explore the potential of leveraging popular large language models (LLMs), such as ChatGPT, DeepSeek Coder, in fine-grained vulnerability detection

# THANK YOU

**Miaomiao Shao**

Harbin Institute of Technology, Shenzhen,  China

Email: 21B951007@stu.hit.edu.cn

August 16, 2024