McGill
School of Information Studies

Data Mining and Security Lab

VulEXplaineR: XAI for Vulnerability Detection on Assembly Code

Samaneh Mahdavifar[1](✉), Mohd Saqib[1], Benjamin C. M. Fung[1], Philippe Charland[2], and Andrew Walenstein[3]

[1] Data Mining & Security Lab, McGill University, Montreal, Canada, H3A 1X1
samaneh.mahdavifar@affiliate.mcgill.ca, mohd.saqib@mail.mcgill.ca, ben.fung@mcgill.ca

[2] Defence Research and Development Canada philippe.charland@drdc-rddc.gc.ca

[3] BlackBerry Limited, Waterloo, Canada walenste@ieee.org

Outline

- Motivation
- Related Work
- Problem
- Objective
- Contributions
- Framework
- Experimental Results
- Conclusion
- Future work

Motivation

- **Software vulnerability**
  - Weaknesses in system design, implementation, or operation management that, if exploited, can lead to various attacks or can even cause the systems to crash
  - Software vulnerabilities are being detected and released through the CVE database
- **Vulnerability detection methods**
  - Reverse engineering
    - Manually intensive, making it unfeasible, especially for mitigation of zero-day vulnerabilities
  - Deep learning and machine learning
    - Automated the process of identifying software flaws

Related Work

- **Vulnerability Detection on Assembly Code**
- Instruction2Vec, (Lee et al., 2019)
  - Word2vec for embedding
  - Text-CNN to extract features
- ❑ Capture only the semantics of the binaries
- ❑ Function-level
- VDGraph2Vec, (Diwan et al., 2021)
  - RoBERTa for embedding
  - Message passing neural networks for capturing the control flow
- VulANalyzeR, (Li et al., 2022)
  - RNN embedding and Graph convolutions
  - Attention mechanism
- ❑ Structural and semantic representation learning
- ❑ Entire assembly code and function-level

Problem

- AI-based binary vulnerability detection methods
  - Not explainable
    - Does not explain why a binary is vulnerable to reverse engineers
    - If they rarely exist, the explanations are low-level and not abstract enough
  - Does not consider Edge features
    - Treat all jumps and calls equally
    - Prevents the GNN from modeling the edge distributions correctly

Objective

- Design and implement an XAI model for binary vulnerability detection
  - Providing evidence and predictions for reverse engineers
  - Identifying vulnerable behaviors rather than solely focusing on features

Contributions

- We provide explainability for vulnerability detection in terms of subgraph of the CFG based on a graph explanation model called PGExplainer
  - Uses block jumps and function calls as the edge distribution for the GCNN
- We use operand type frequency and TFIDF that provide a lightweight feature vector for detecting and explaining vulnerability.
- We augment operand type frequency and TFIDF with BERT for block embedding
- We evaluate explainability in terms of fidelity, provide a case study to analyze the extracted graph explanation, and validate it using expert knowledge

Method of communication

- **Input feature explanation**
  - Highly adaptable
  - Limited to provide abstract explanations and control flow
- **Rules**
  - Highest level of (global) explanations
  - Complex when directly applied to assembly code
- **Graph visualization**
  - Represent the structure and semantic information of assembly code
  - More human understandable
  - An intermediary for rule explanation

Assembly Code

```
push    ebp
mov     ebp, esp
push    ecx

mov     [ebp+key], 1A9Ch
mov     eax, [ebp+msg]
push    eax
push    offset Format
call    ds:printf
add     esp, 8
mov     [ebp+msg], 1
cmp     [ebp+msg], 0
jz      short loc_40103E
```

Graph Explanation

- Graph Neural Networks (GNNs)
  - Employed for representation learning in graph-structured data
  - Use a message-passing scheme to enable each node in the graph to capture the feature vector of the neighbor nodes

- Perturbation-based methods
  - GNNExplainer
    - Compact subgraph and a small subset of node features
    - Local explainability
  - PGExplainer
    - Global explainability
    - Approximated discrete masks for edges to explain the predictions

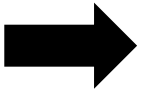
VulEXplaineR

- Inspired by PGExplainer
  - Use edge distribution to consider the edge type in CFG
    - intra-function jumps between blocks
    - inter-function calls between functions

Objective function

$$\min_{G_s} H(Y_0|G = G_s) = \min_{G_s} \mathbb{E}_{G_s}\left[H(Y_0|G = G_s)\right] \approx \min_{\Theta} \mathbb{E}_{G_s \sim q(\Theta)}\left[H(Y_0|G = G_s)\right],$$

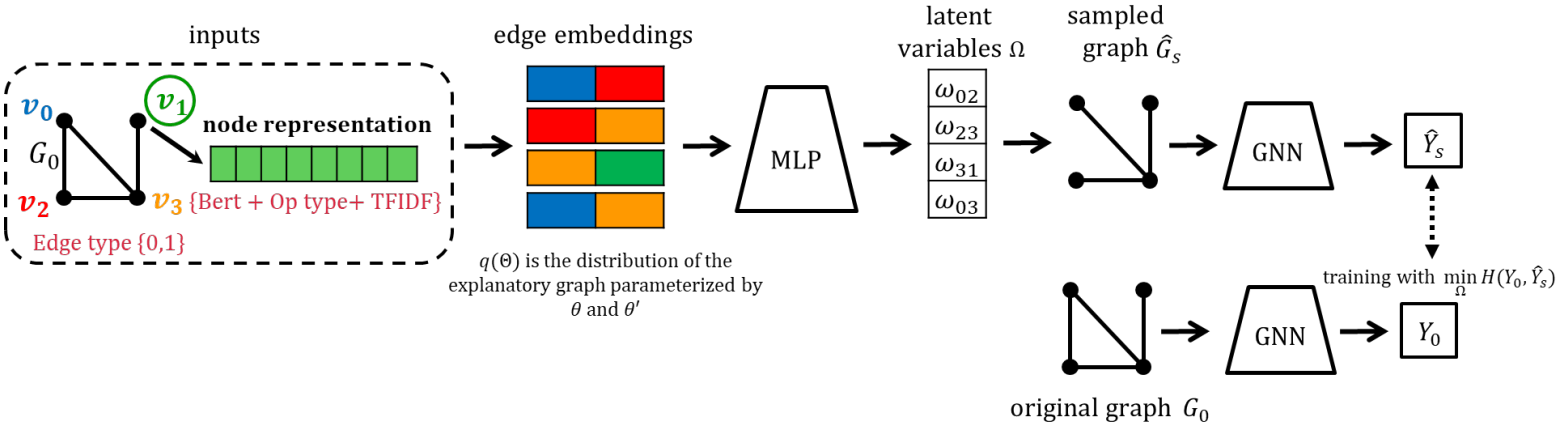
reparametrization technique



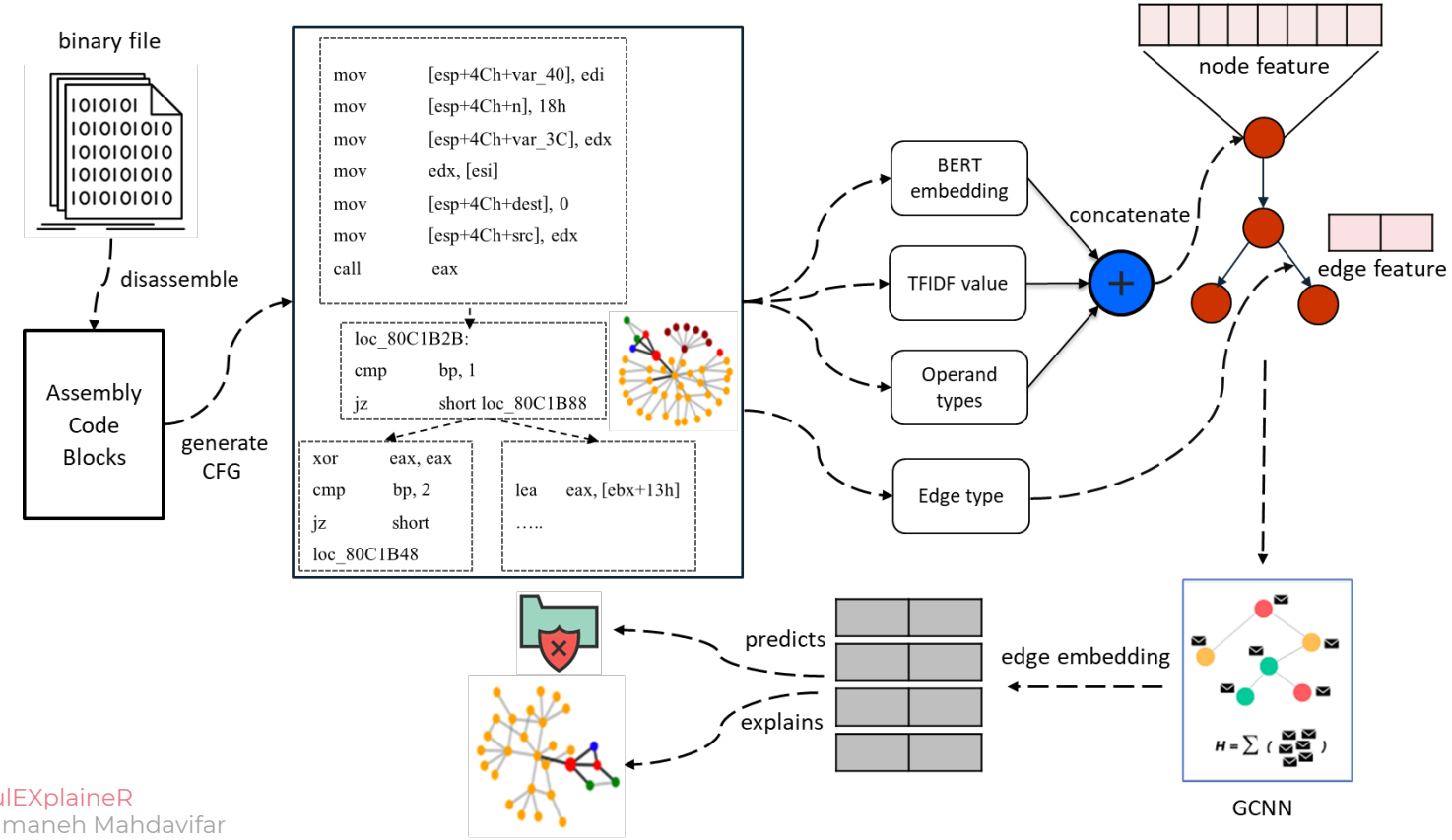
$$\min_{\Omega} \mathbb{E}_{\epsilon \sim \text{Uniform}(0,1)} H(Y_0|G = G_s).$$

- Block embedding
  - BERT, augmented with TFIDF and operand types
- Provide explainability in terms of sub-graph G_s of GCNNs
  - Takes a trained GCNN and its predictions
  - Returns an explanation in the form of a small subgraph of the input graph

Extracting Explanation Graph



VulEXplaineR Framework



Experimental Results

- **Datasets**
- **NDSS18 Vulnerability Dataset (NIST)**
  - 32,281 binaries (Vulnerable and Non-vulnerable)
  - Windows and Linux
  - CWE-119 and CWE-399
    - CWE-119
- **Juliet Test Suite**
  - 83,624 binaries (Vulnerable and Non-vulnerable)
  - 118 distinct CWEs
    - CWE-121 and CWE-190

Table 1: Dataset distributions

Dataset	# Vulnerable samples	# Non-vulnerable samples
NDSS18	8978	8999
Juliet Test Suite	7060	7060

Experimental Results

Table 2: Performance results on the NDSS18 dataset

Models	Accuracy	Recall	Precision	F1
VulEXplaineR$_{FS1}$	**96.02**	**96.02**	**97.31**	**95.51**
VulEXplaineR$_{FS2}$	78.75	78.75	79.67	78.35
VulGCNN$_{FS1}$	93.02	93.02	95.81	92.99
VulTFIDF$_{RF}$	70.59	70.59	70.59	70.46
VDGraph2Vec	95.48	96.21	95.65	95.92
i2v-TCNN	81.41	82.50	83.72	83.11
VulANalyzeR	89.53	94.18	85.36	90.10

Table 3: Performance results on the Juliet Test dataset

Models	Accuracy	Recall	Precision	F1
VulEXplaineR$_{FS1}$	**99.80**	**99.80**	**99.81**	**99.80**
VulEXplaineR$_{FS2}$	94.79	96.65	93.40	95.00
VulGCNN$_{FS1}$	98.55	98.55	98.56	98.55
VulTFIDF$_{RF}$	87.47	87.47	88.25	87.42
VDGraph2Vec	100	100	100	100
i2v-TCNN	92.81	93.1	93.59	93.31
VulANalyzeR	99.68	100	99.38	99.69

Graph Explanation

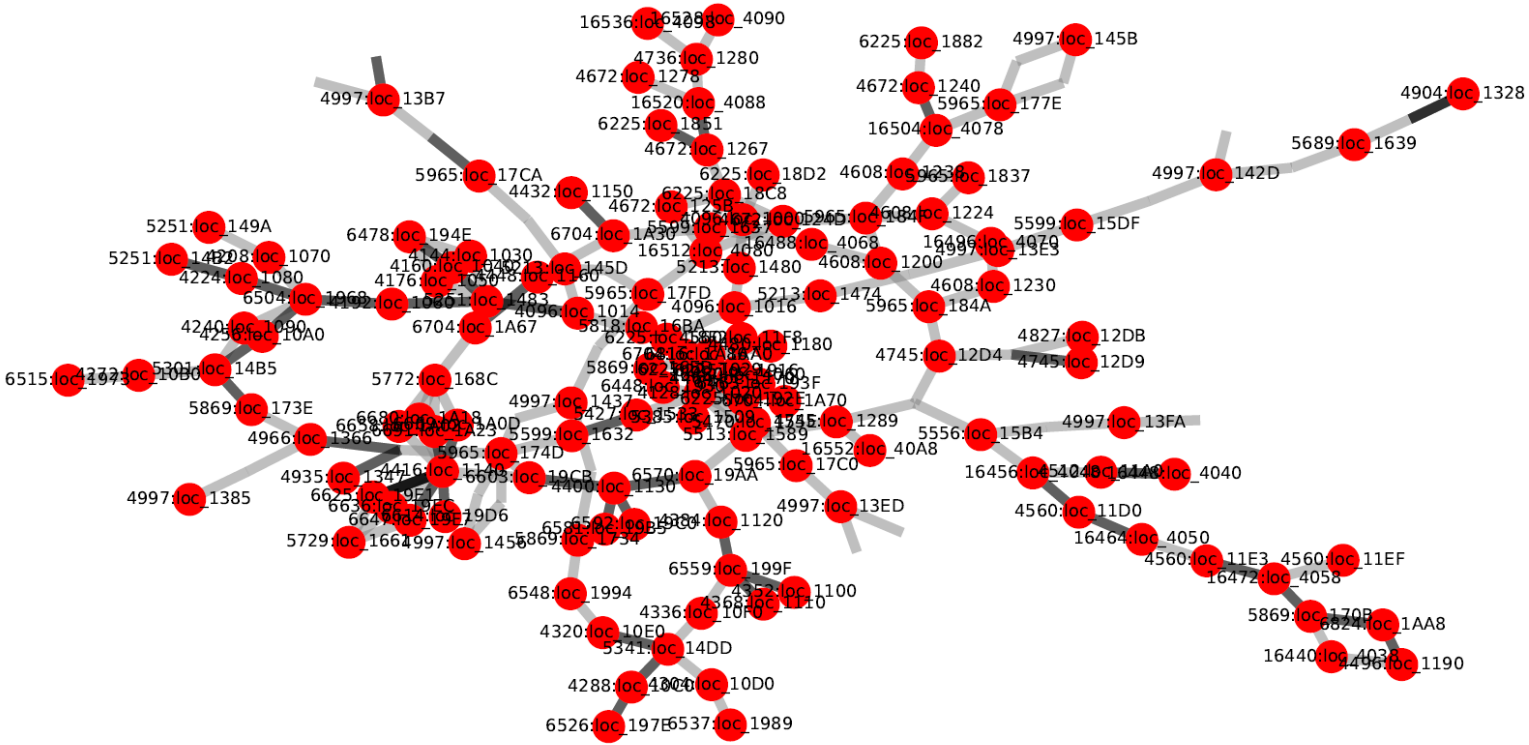
- Fidelity analysis
  - refers to the degree to which the predictions of the extracted subgraphs align with the actual GNN

Table 4: Fidelity of VulEXplaineR

Dataset	VulEXplaineR$_{FS1}$	VulEXplaineR$_{FS2}$
NDSS18	0.95	0.80
Juliet Test Suite	0.99	0.99

How does an explanation look like?

A subgraph of the main CFG



Extracted Subgraph

CWE-805

```
4904:loc_1328
['endbr64', 'push rbp', 'mov rbp rsp', 'sub rsp const', 'mov addr rdi', 'mov
    rax addr', 'mov rdi rax', 'call
    cwe121_stack_based_buffer_overflow__cwe805_struct_declare_loop_54c_badsink
    ', 'nop', 'leave', 'retn']
```

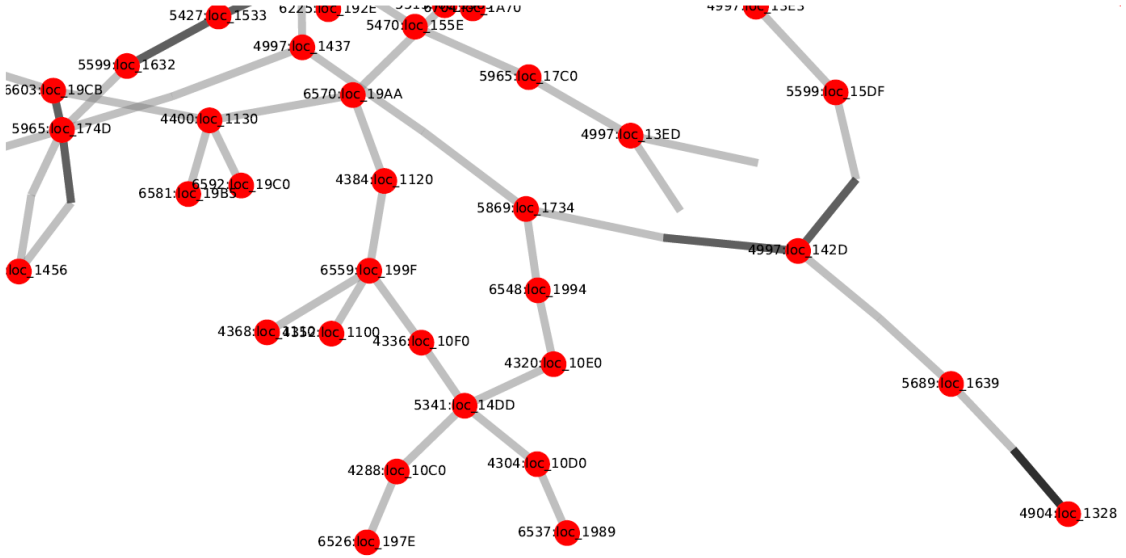


Fig. 8: 4904:loc_1328

Extracted Subgraph

```
4935:loc_1347
['endbr64', 'push rbp', 'mov rbp rsp', 'sub rsp const', 'mov addr rdi', 'mov
    rax addr', 'mov rdi rax', 'call
    cwe121_stack_based_buffer_overflow__cwe805_struct_declare_loop_54d_badsink
    ', 'nop', 'leave', 'retn']
```

CWE-805

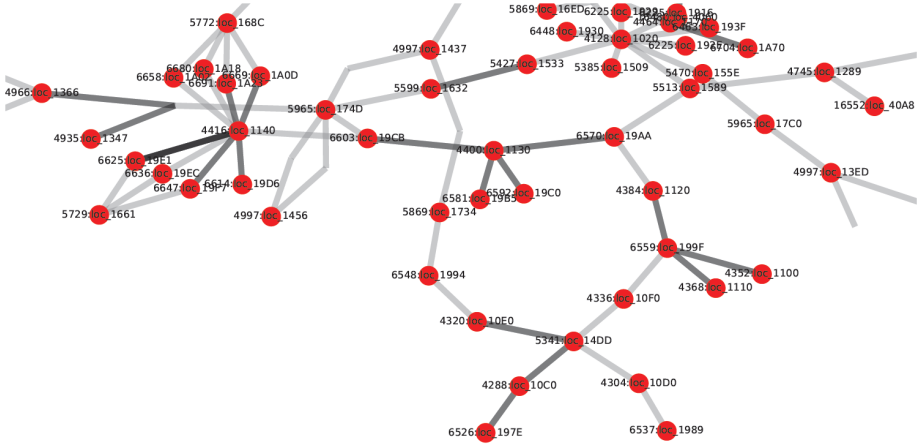
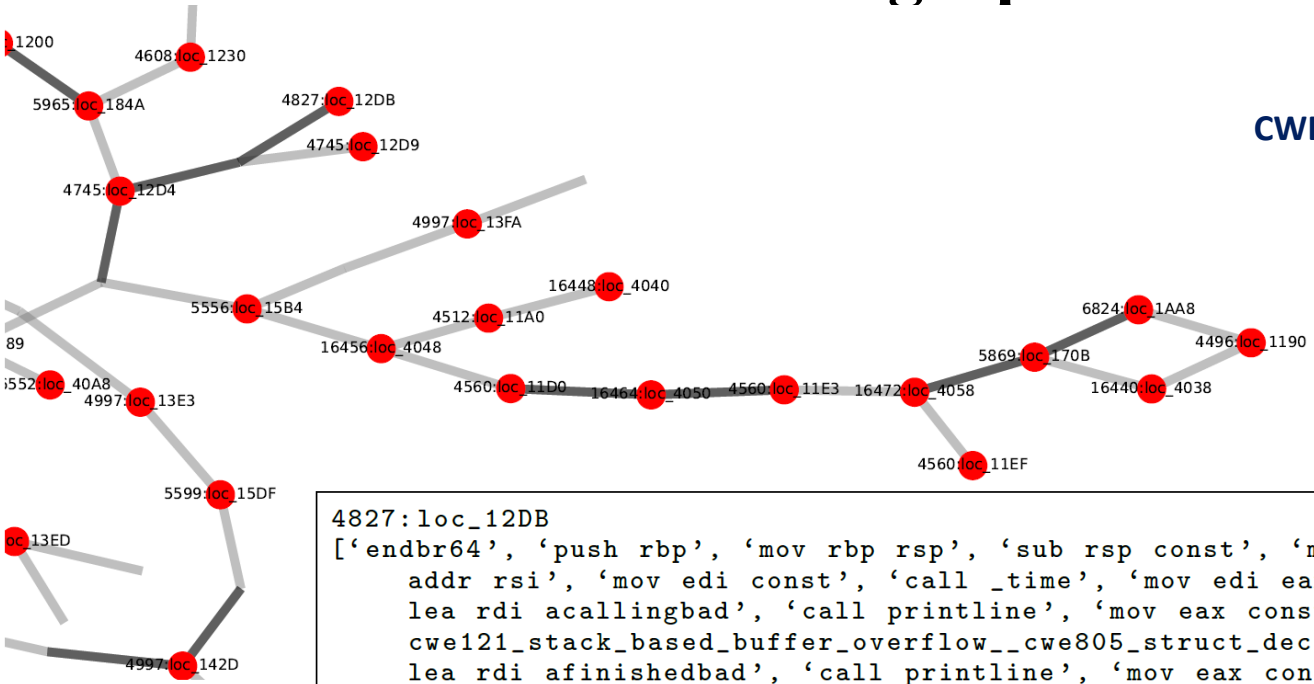


Fig. 5: 4966:loc_1366, 4935:loc_1347

```
4966:loc_1366
['endbr64', 'push rbp', 'mov rbp rsp', 'sub rsp const', 'mov addr rdi', 'mov
    rax addr', 'mov rdi rax', 'call
    cwe121_stack_based_buffer_overflow__cwe805_struct_declare_loop_54e_badsink
    ', 'nop', 'leave', 'retn']
```

Extracted Subgraph



CWE-121

```
4827:loc_12DB
['endbr64', 'push rbp', 'mov rbp rsp', 'sub rsp const', 'mov addr edi', 'mov
    addr rsi', 'mov edi const', 'call _time', 'mov edi eax', 'call _srand', '
    lea rdi acallingbad', 'call printline', 'mov eax const', 'call
    cwe121_stack_based_buffer_overflow__cwe805_struct_declare_loop_54_bad', '
    lea rdi afinishedbad', 'call printline', 'mov eax const', 'leave', 'retn']
```

Fig. 7: 4827:loc_12DB

Extracted Subgraph

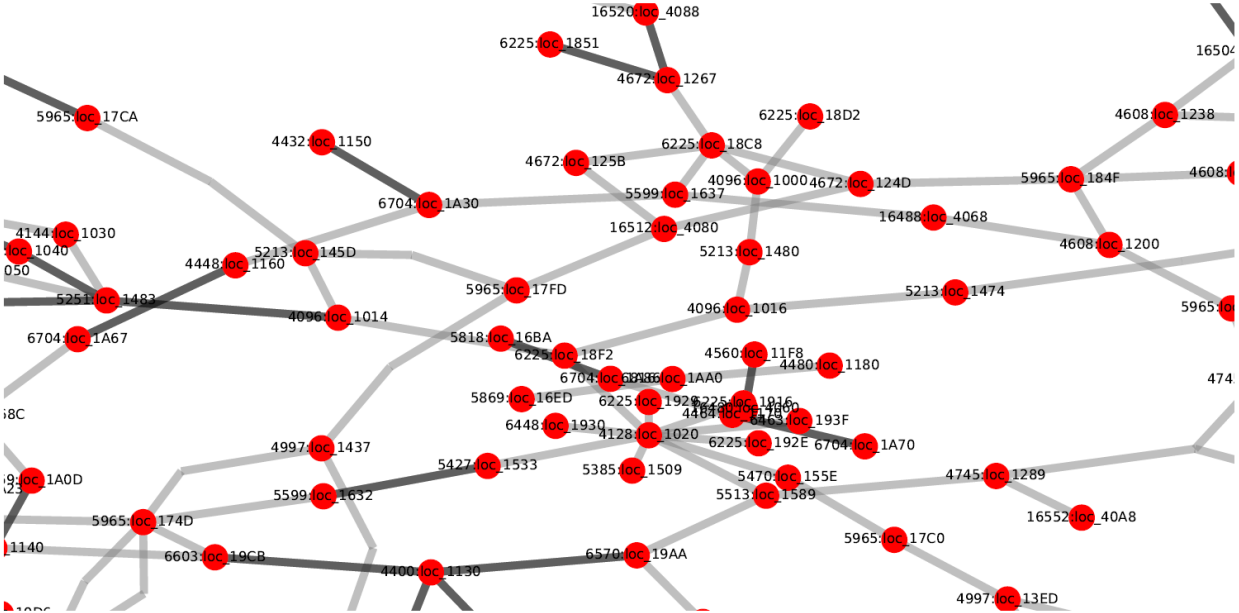


Fig. 4: 4672:loc_125B

```
4672:loc_125B
['mov rdi cs:__dso_handle', 'call __cxa_finalize']
```

Conclusion

- We propose VulEXplaineR, an XAI method for vulnerability detection in assembly code.
- Utilizing BERT and TFIDF, it offers an efficient framework to represent relationships between blocks and functions
- Inspired by PGExplainer,
  - VulEXplaineR produces explanations in the form of subgraphs of GCNNs
  - incorporating edge embeddings for enhanced accuracy
- Experimental results on the NDSS2018 and Juliet Test datasets show that VulEXplaineR
  - outperforms state-of-the-art baselines
  - provides high explainability that matches the graph nature of the assembly code and is valuable for reverse engineers
- Qualitative and quantitative evaluations, including fidelity metrics, demonstrate the method's effectiveness
- A case study highlights VulEXplaineR's ability to identify vulnerabilities and dependencies within the extracted subgraph

Future Work

- Design motifs as the ground truth
  - Conduct a quantitative evaluation of the extracted subgraph in the form of binary edge classification

- Model the underlying GNN as a directed graph
  - A directed graph imposes an ordering on a pair of nodes
  - Further describes the relationship between the nodes

Thank you

Please do not hesitate to ask

Method of communication

- **Input feature explanation**
  - Whole Instructions
  - Tokens
- Pros
  - Highly adaptable to different problems
- Cons
  - Limited to provide abstract explanations
  - Limited to provide control flow of the program

Assembly Code

```
push    ebp
mov     ebp, esp
push    ecx


mov     [ebp+key], 1A9Ch
mov     eax, [ebp+msg]
push    eax
push    offset Format
call    ds:printf
add     esp, 8
mov     [ebp+msg], 1
cmp     [ebp+msg], 0
jz      short loc_40103E
```

Method of communication

- **Rules**
  - Pros
    - Highest level of explanation
    - Powerful at approximating non-linear decision boundaries
    - Global explanations
  - Cons
    - Complex when directly applied to assembly code
    - Time complexity

Method of communication

- **Graph visualization**
  - Matches the nature of the graphs of assembly code
    - Represent the structure and semantic information of assembly code
    - Relationships and dependencies between blocks
  - Visual interpretation
    - More human understandable
    - An intermediary for rule explanation