McGill UNIVERSITY Data Mining and Security Lab

Representation Learning for Vulnerability Detection on Assembly Code

Ashita Diwan, Miles Q. Li, Benjamin C. M. Fung

Software Vulnerability

Software vulnerability is a defect or weakness in system design, implementation or operation management that if exploited, can lead to various attacks or can even cause the systems to crash. (Krsul et al., 1998)

Buffer Overflow Attack

```
main(int argc, char *argv[]) {
  func(argv[1]);
}
void func(char *v) {
  char buffer[ 8 ];
  strcpy(buffer, v);
}
```

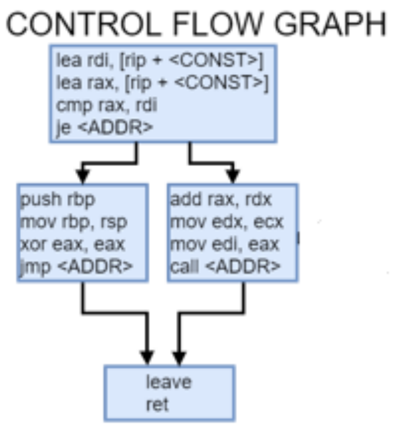
Why is it important to detect vulnerabilities?

- Codes are prone to attack by hackers and if vulnerabilities are not detected in time, they can be exploited for malicious use.
- Most researchers detect vulnerabilities at the source code level.
- What if the source code is unavailable? PROBLEM!

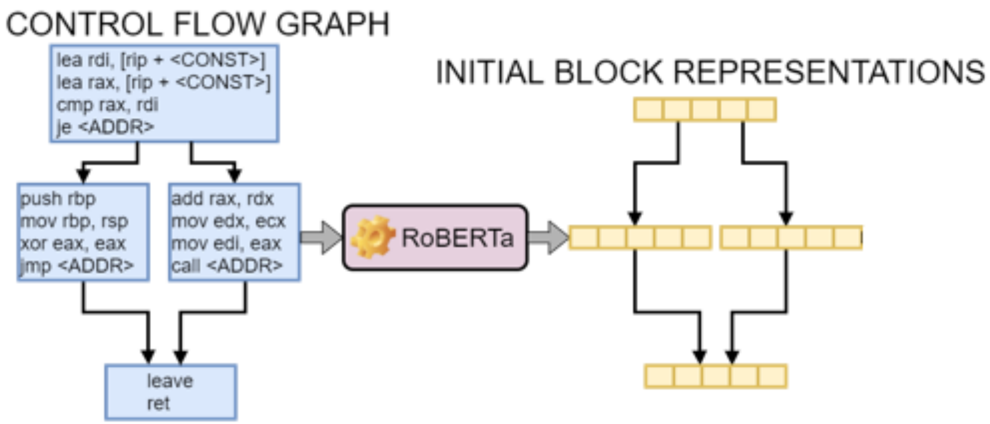
Solution - Detection at the assembly code level

- If the source code is unavailable, we can perform all our analysis at the assembly code level.
- Traditionally, the process of understanding the software from its binary executables is known as Reverse Engineering (MANUALLY INTENSIVE AND TIME CONSUMING)
- We need to automate the process.

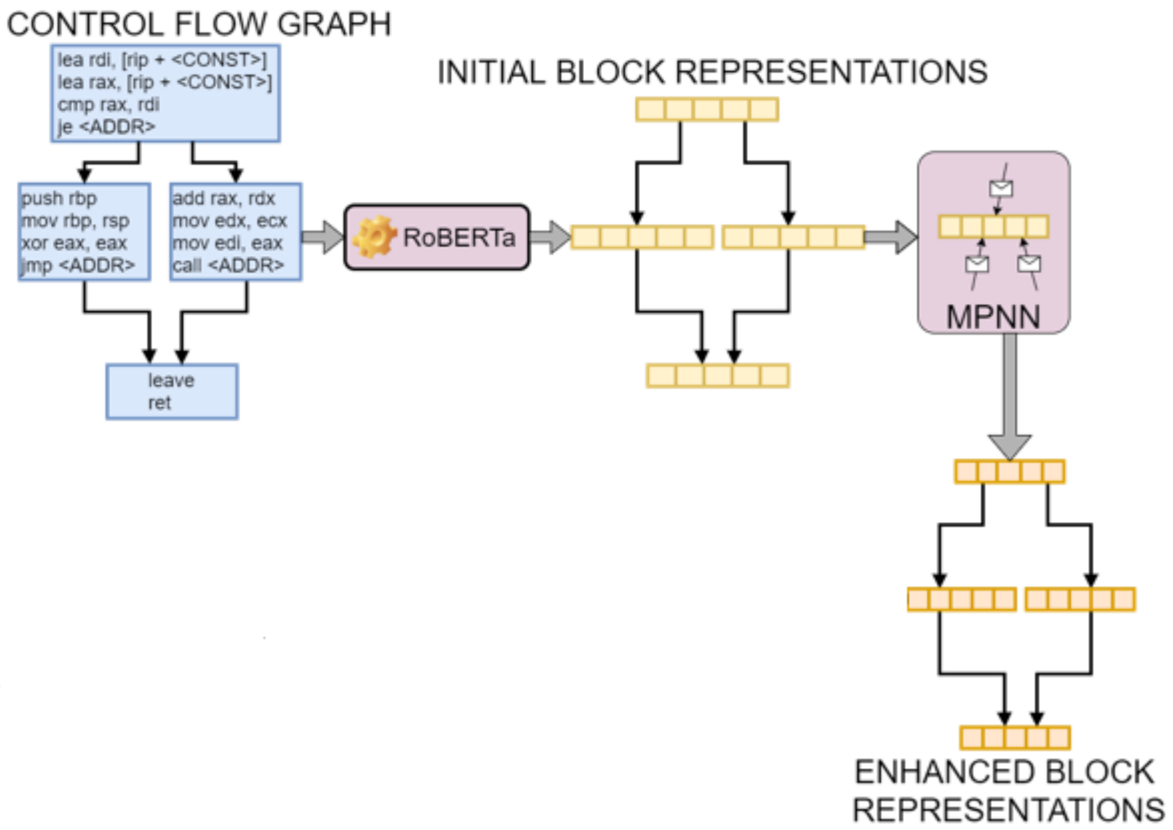
Workflow


CONTROL FLOW GRAPH
lea rdi, [rip + <CONST>]
lea rax, [rip + <CONST>]
cmp rax, rdi
je <ADDR>
push rbp
mov rbp, rsp
xor eax, eax
jmp <ADDR>
add rax, rdx
mov edx, ecx
mov edi, eax
call <ADDR>
leave
ret

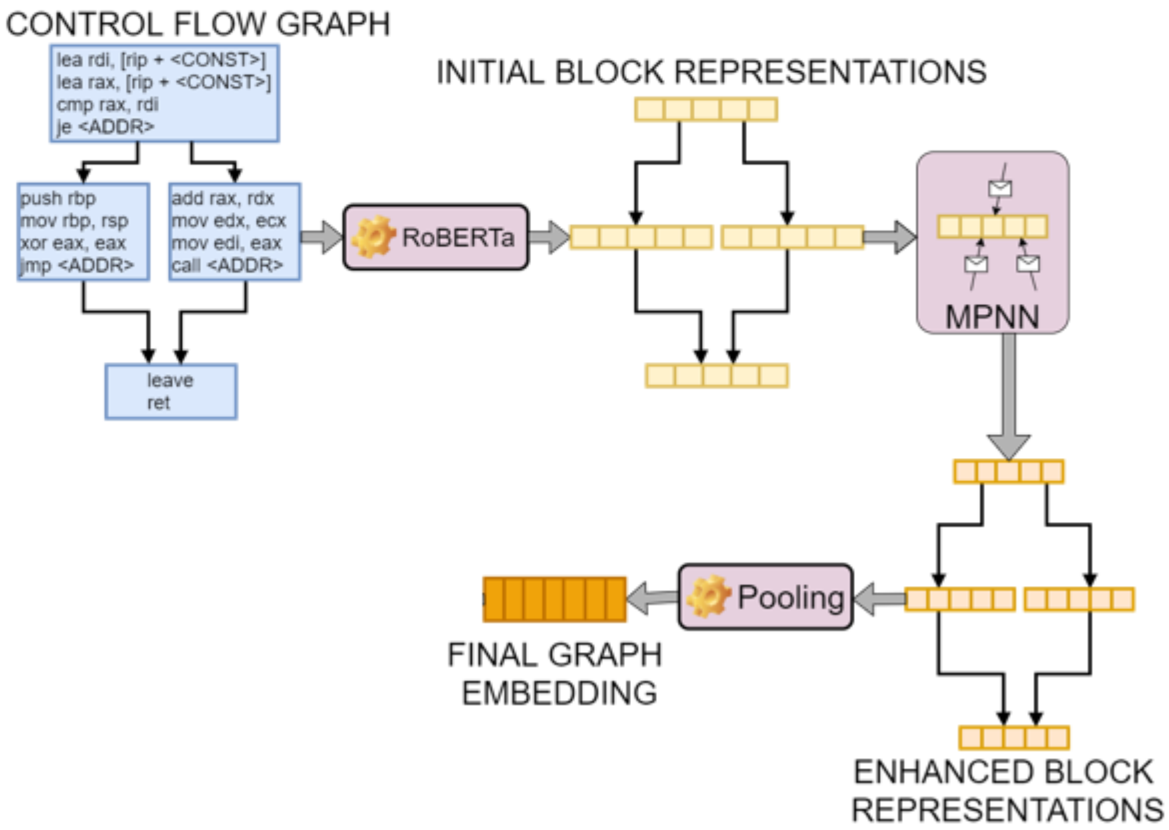
Workflow


CONTROL FLOW GRAPH
lea rdi, [rip + <CONST>]
lea rax, [rip + <CONST>]
cmp rax, rdi
je <ADDR>
push rbp
mov rbp, rsp
xor eax, eax
jmp <ADDR>
add rax, rdx
mov edx, ecx
mov edi, eax
call <ADDR>
leave
ret
RoBERTa
INITIAL BLOCK REPRESENTATIONS

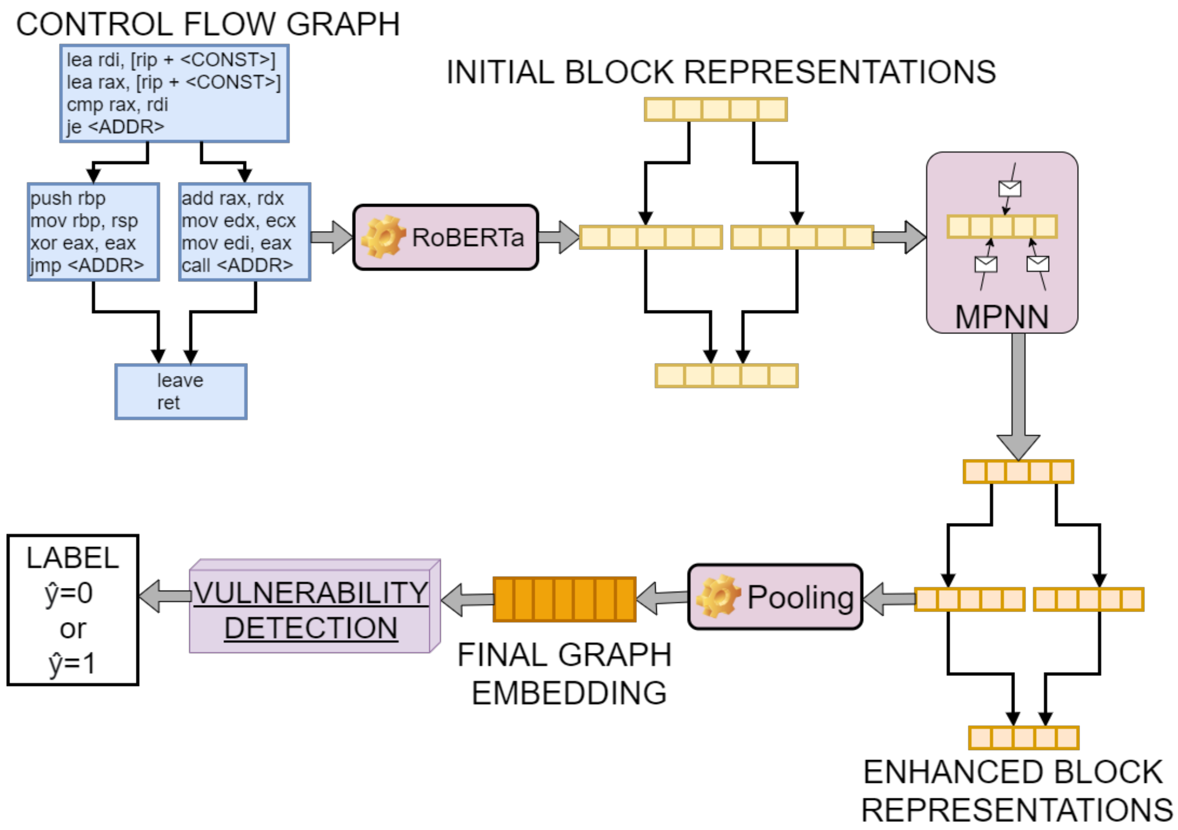
Workflow


CONTROL FLOW GRAPH
lea rdi, [rip + <CONST>]
lea rax, [rip + <CONST>]
cmp rax, rdi
je <ADDR>
push rbp
mov rbp, rsp
xor eax, eax
jmp <ADDR>
add rax, rdx
mov edx, ecx
mov edi, eax
call <ADDR>
leave
ret
RoBERTa
INITIAL BLOCK REPRESENTATIONS
MPNN
ENHANCED BLOCK REPRESENTATIONS

Workflow

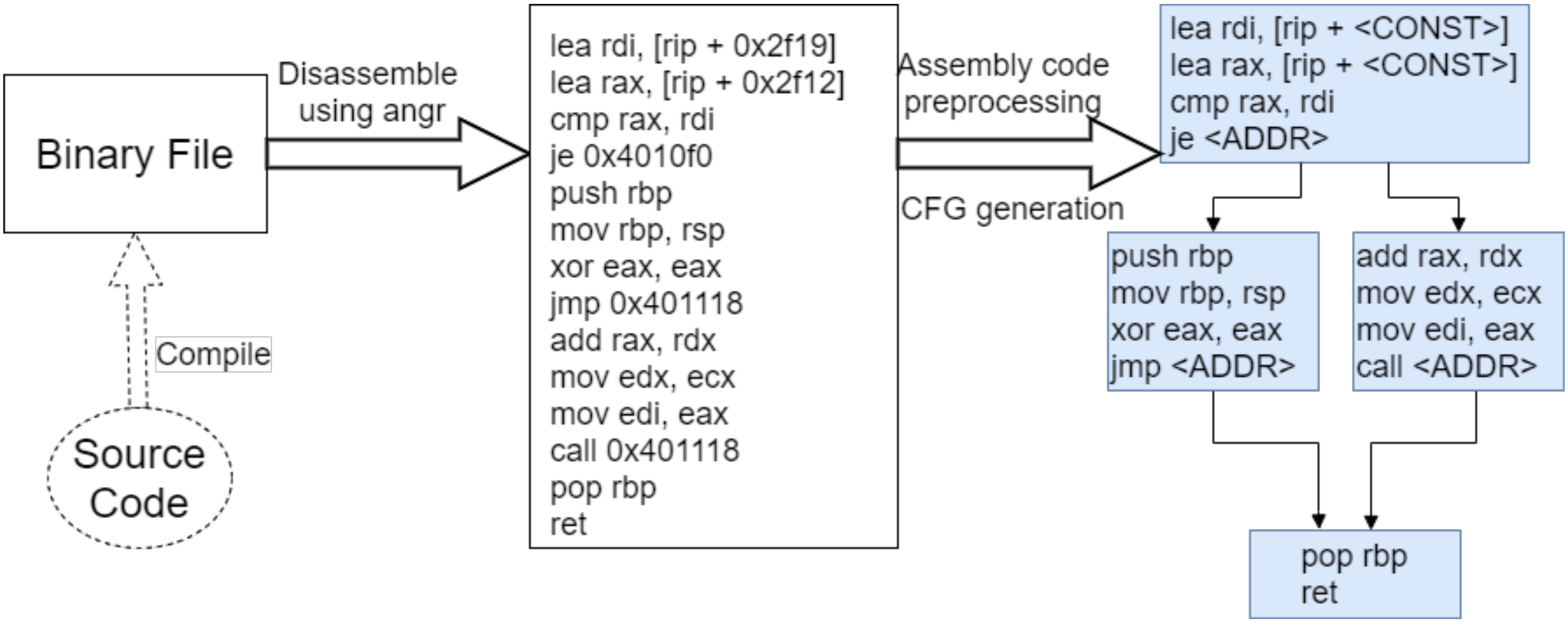

CONTROL FLOW GRAPH
lea rdi, [rip + <CONST>]
lea rax, [rip + <CONST>]
cmp rax, rdi
je <ADDR>
push rbp
mov rbp, rsp
xor eax, eax
jmp <ADDR>
add rax, rdx
mov edx, ecx
mov edi, eax
call <ADDR>
leave
ret
RoBERTa
INITIAL BLOCK REPRESENTATIONS
MPNN
Pooling
FINAL GRAPH EMBEDDING
ENHANCED BLOCK REPRESENTATIONS

Workflow

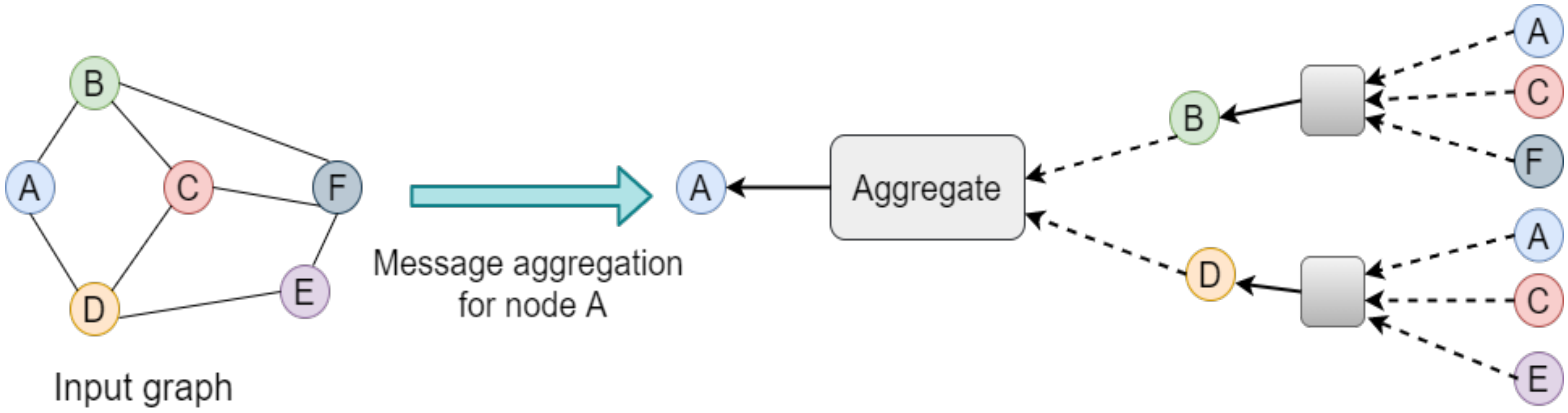


VDGraph2Vec learns both the structural and semantic aspects of the assembly code.

Control Flow Graph

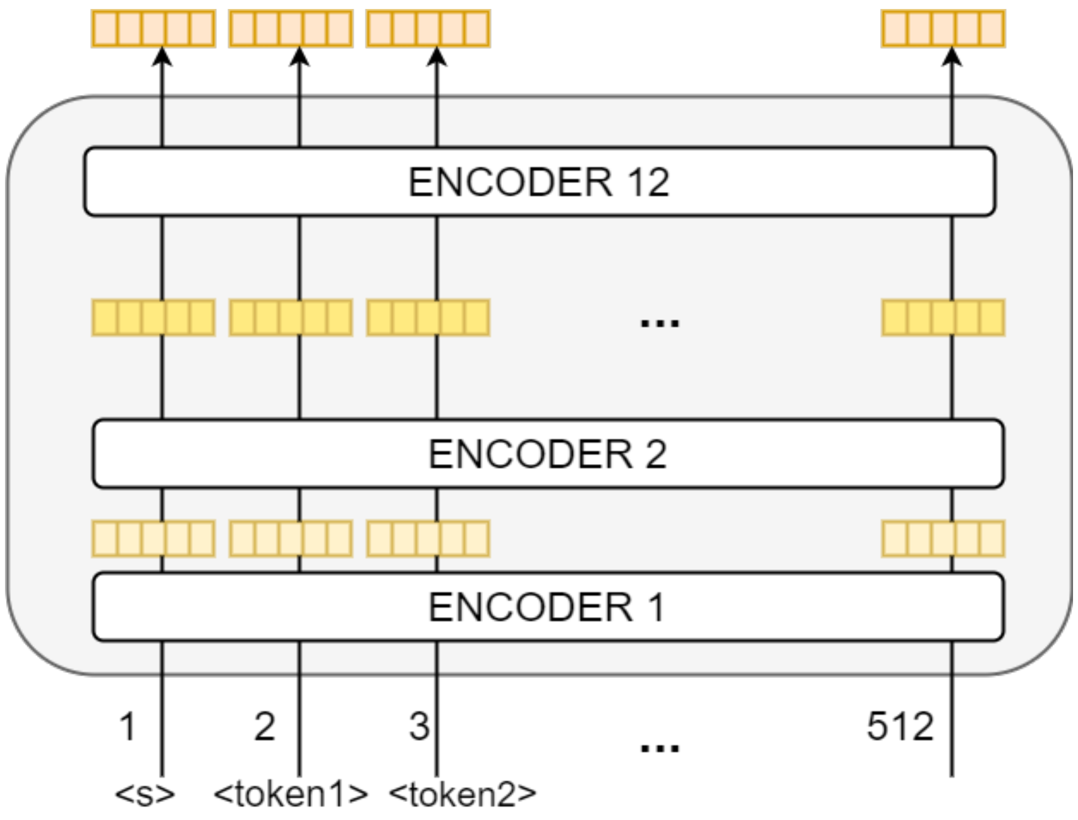


Message Passing Neural Networks

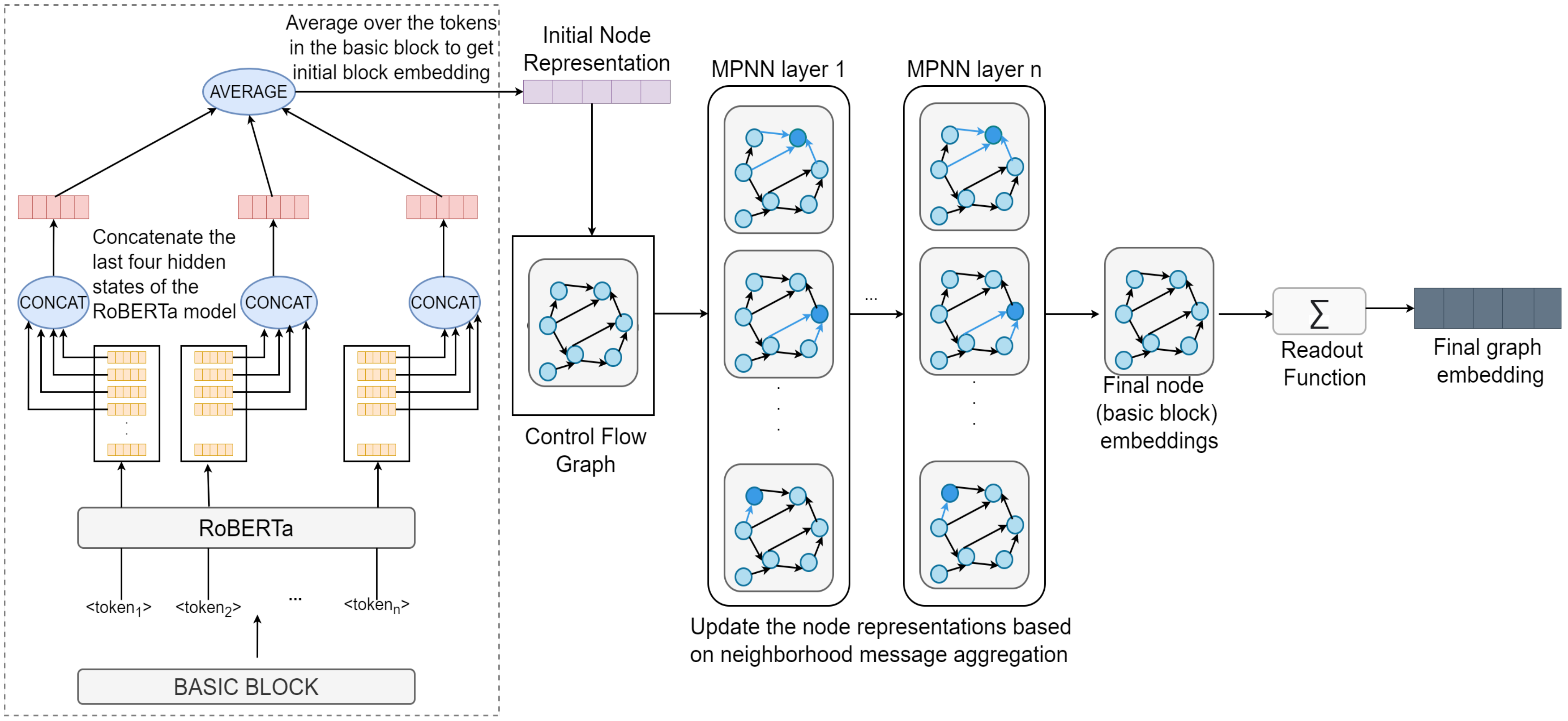


RoBERTa

- It is one of the models that has achieved the state-of-the-art results in NLP.
- Takes into consideration both left and right context to generate embeddings.



VDGraph2Vec Model



Datasets used

- **Juliet Test Suite for C/C++** is a collection of test cases in the C/C++ language. It contains the good (patched) and bad (vulnerable/non-patched) examples organized under 118 different CWEs.
- **It contains more synthetic examples.**

CWE36_Absolute_Path_Traversal
CWE121_Stack_Based_Buffer_Overflow
CWE126_Buffer_Overread
CWE188_Reliance_on_Data_Memory_Layout
CWE195_Signed_to_Unsigned_Conversion_Error
CWE223_Omission_of_Security_Relevant_Infor...
CWE247_Reliance_on_DNS_Lookups_in_Securit...
CWE259_Hard_Coded_Password
CWE319_Cleartext_Tx_Sensitive_Info
CWE328_Reversible_One_Way_Hash
CWE367_TOC_TOU
CWE391_Unchecked_Error_Condition
CWE78_OS_Command_Injection
CWE122_Heap_Based_Buffer_Overflow
CWE127_Buffer_Underread
CWE190_Integer_Overflow
CWE196_Unsigned_to_Signed_Conversion_Error
CWE226_Sensitive_Information_Uncleared_Bef...
CWE252_Unchecked_Return_Value
CWE272_Least_Privilege_Violation
CWE321_Hard_Coded_Cryptographic_Key
CWE338_Weak_PRNG
CWE369_Divide_by_Zero
CWE396_Catch_Generic_Exception

Datasets used

- **NDSS18 dataset** is extracted from the National Vulnerability Database (NVD) and Software Quality Assurance Dataset.
- **It represents a more realistic scenario.**
- It contains binary files for CWE-119 and CWE-322 over Windows OS and Linux OS platforms.

Statistics of the datasets

- We use CWE-121 (Stack-based buffer overflow) and CWE-190 (Integer Overflow) from Juliette Test Suite, and CWE-119 (Improper restriction of operations within the bounds of a memory buffer) from the NDSS18 dataset.

CWE	Vulnerable samples	Non-vulnerable samples	Average # of nodes	Average # of edges
121	3100	3100	55.81	71.42
190	3960	3960	52.36	67.21
119	6521	5861	14.71	17.72

Evaluation metrics

- Accuracy
- Precision
- Recall
- F1-score
- AUC-ROC score

Experimentation

- ❖ Utilizing 2 variants of MPNN – Gated Graph Neural Network (VDGraph2Vec-GGNN) and Graph Convolution Network (VDGraph2Vec-GCN).
- ❖ Implement the state-of-the-art binary vulnerability detection models to compare with our VDGraph2Vec model.
- ❖ Compare the potential of our model at the node embedding and classification level.

Graph Convolution Network (Kipfl et al., 2016)
Gated Graph Neural Network (Li et al., 2015)

Baseline models

- ❖ Handcrafted features with GCN (HF-GCN)
- ❖ Handcrafted features with GGNN (HF-GGNN)
- ❖ Instruction2Vec with TextCNN (i2V-TCNN)
- ❖ Word2Vec with Structure2Vec (w2v-s2v)
- ❖ Word2Vec with GCN (w2v-GCN)
- ❖ Word2Vec with GGNN (w2v-GGNN)
- ❖ RoBERTa with Structure2Vec (RoS2v)

Results

- These experiments have been performed on CWE-121 examples.

Model	Accuracy	Precision	Recall	F1-score	AU-ROC score
HF-GCN	70.96	79.28	60.85	68.85	71.55
HF-GGNN	71.77	66.81	92.35	77.53	70.58
i2v-TCNN	94.83	97.12	92.96	95.0	94.94
w2v-s2v	95.32	94.08	97.24	95.63	95.21
w2v-GCN	95.81	94.13	98.16	96.11	95.66
w2v-GGNN	97.58	98.14	97.24	97.69	97.59
RoS2v	97.90	**100.0**	96.02	97.97	98.01
VDGraph2Vec-GCN*	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**
VDGraph2Vec-GGNN*	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**

Example of CWE-121

```
void CWE121_Stack_Based_Buffer_Overflow__char_type_overrun_memcpy_01_bad()
{
    {
        charVoid structCharVoid;
        structCharVoid.voidSecond = (void *)SRC_STR;
        /* Print the initial block pointed to by structCharVoid.voidSecond */
        printLine((char *)structCharVoid.voidSecond);
        /* FLAW: Use the sizeof(structCharVoid) which will overwrite the pointer voidSecond */
        memcpy(structCharVoid.charFirst, SRC_STR, sizeof(structCharVoid));
        structCharVoid.charFirst[(sizeof(structCharVoid.charFirst)/sizeof(char))-1] = '\0'; /* null terminate the string */
        printLine((char *)structCharVoid.charFirst);
        printLine((char *)structCharVoid.voidSecond);
    }
}

#endif /* OMITBAD */

#ifndef OMITGOOD

static void good1()
{
    {
        charVoid structCharVoid;
        structCharVoid.voidSecond = (void *)SRC_STR;
        /* Print the initial block pointed to by structCharVoid.voidSecond */
        printLine((char *)structCharVoid.voidSecond);
        /* FIX: Use sizeof(structCharVoid.charFirst) to avoid overwriting the pointer voidSecond */
        memcpy(structCharVoid.charFirst, SRC_STR, sizeof(structCharVoid.charFirst));
        structCharVoid.charFirst[(sizeof(structCharVoid.charFirst)/sizeof(char))-1] = '\0'; /* null terminate the string */
        printLine((char *)structCharVoid.charFirst);
        printLine((char *)structCharVoid.voidSecond);
    }
```

No issues found

Results

- These experiments have been performed on CWE-190 examples.

Model	Accuracy	Precision	Recall	F1-score	AU-ROC score
HF-GCN	67.67	69.02	72.89	70.90	67.21
HF-GGNN	69.19	71.19	72.19	71.69	68.92
i2v-TCNN	90.78	90.06	93.22	91.61	90.56
w2v-s2v	93.43	93.11	94.85	93.98	93.31
w2v-GCN	95.07	94.71	96.26	94.97	95.48
w2v-GGNN	95.41	95.58	96.02	95.81	95.40
RoS2v	94.57	94.25	95.79	95.01	94.46
VDGraph2Vec-GCN*	99.74	99.53	**100.0**	99.76	99.72
VDGraph2Vec-GGNN*	**100.0**	**100.0**	**100.0**	**100.0**	**100.0**

Results

- These experiments have been performed on CWE-119 (more realistic) examples.

Model	Accuracy	Precision	Recall	F1-score	AU-ROC score
HF-GCN	64.83	69.84	64.13	66.86	64.92
HF-GGNN	66.77	64.66	88.04	74.56	64.23
i2v-TCNN	81.41	83.72	82.50	83.11	81.32
w2v-s2v	85.0	85.91	87.17	86.54	84.74
w2v-GCN	89.03	90.32	89.79	90.05	88.94
w2v-GGNN	90.48	92.77	89.79	91.25	90.56
RoS2v	86.86	86.0	90.42	88.15	86.55
VDGraph2Vec-GCN*	92.9	93.08	94.16	93.61	92.58
VDGraph2Vec-GGNN*	**95.48**	**95.65**	**96.21**	**95.92**	**95.27**

Results

- We also performed cross-dataset evaluation on CWE-121 with the model trained on samples from the Juliet Test Suite and tested on samples from another dataset. This helps to assess the generalization capability of the model.

Model	Accuracy	Precision	Recall	F1-score	AU-ROC score
HF-GCN	61.0	62.73	54.2	58.15	60.99
HF-GGNN	62.2	71.32	40.8	51.98	62.2
i2v-TCNN	75.7	**100.0**	51.4	67.89	75.7
w2v-s2v	72.2	**100.0**	44.4	61.4	72.2
w2v-GCN	83.8	**100.0**	67.6	80.66	83.8
w2v-GGNN	89.3	**100.0**	78.6	88.01	89.3
RoS2v	78.7	**100.0**	57.4	72.93	78.69
VDGraph2Vec-GCN*	91.1	**100.0**	82.2	90.23	91.1
VDGraph2Vec-GGNN*	**94.9**	**100.0**	**89.8**	**94.6**	**94.9**

Our contributions

- Generating a latent representation for the entire assembly code, rather than for just an assembly function.
- We use RoBERTa for representing the assembly instructions.
- VDGraph2Vec can successfully detect vulnerabilities because both semantics and the hierarchical structure of assembly code are being taken into consideration.
- Our model is able to achieve high performance in different experimental settings, surpassing the recent works in this direction.

Future Work

- ❖ Incorporate more structural information of the graph with the dataflow dependencies.
- ❖ Extend the work for all target machine architectures.
- ❖ Spot the location of the vulnerabilities in code.
- ❖ Extend this study by proving the effectiveness of these vector representations for other downstream tasks such as binary clone detection.

Q
&
A