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

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

![Diagram of buffer overflow](image-url)
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
Workflow
Workflow
VDGraph2Vec learns both the structural and semantic aspects of the assembly code.
Control Flow Graph

Binary File

Disassemble using angr

lea rdi, [rip + 0x2f19]
lea rax, [rip + 0x2f12]
cmp rax, rdi
je 0x4010f0
push rbp
mov rbp, rsp
xor eax, eax
jmp 0x401118
add rax, rdx
mov edx, ecx
mov edi, eax
call 0x401118
pop rbp
ret

Assembly code preprocessing

lea rdi, [rip + <CONST>]
lea rax, [rip + <CONST>]
cmp rax, rdi
je <ADDR>

CFG generation

push rbp
mov rbp, rsp
xor eax, eax
jmp <ADDR>

add rax, rdx
mov edx, ecx
mov edi, eax
call <ADDR>

pop rbp
ret
Message Passing Neural Networks

Input graph

Message aggregation for node A

Aggregate

A

B

C

D

E

F

A

B

C

D

E

F

A

C

F

A

C

E

Data Mining and Security Lab
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.
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.

<table>
<thead>
<tr>
<th>CWE</th>
<th>Vulnerable samples</th>
<th>Non-vulnerable samples</th>
<th>Average # of nodes</th>
<th>Average # of edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>121</td>
<td>3100</td>
<td>3100</td>
<td>55.81</td>
<td>71.42</td>
</tr>
<tr>
<td>190</td>
<td>3960</td>
<td>3960</td>
<td>52.36</td>
<td>67.21</td>
</tr>
<tr>
<td>119</td>
<td>6521</td>
<td>5861</td>
<td>14.71</td>
<td>17.72</td>
</tr>
</tbody>
</table>
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.
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.

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>Precision</th>
<th>Recall</th>
<th>F1-score</th>
<th>AU-ROC score</th>
</tr>
</thead>
<tbody>
<tr>
<td>HF-GCN</td>
<td>70.96</td>
<td>79.28</td>
<td>60.85</td>
<td>68.85</td>
<td>71.55</td>
</tr>
<tr>
<td>HF-GGNN</td>
<td>71.77</td>
<td>66.81</td>
<td>92.35</td>
<td>77.53</td>
<td>70.58</td>
</tr>
<tr>
<td>i2v-TCNN</td>
<td>94.83</td>
<td>97.12</td>
<td>92.96</td>
<td>95.0</td>
<td>94.94</td>
</tr>
<tr>
<td>w2v-s2v</td>
<td>95.32</td>
<td>94.08</td>
<td>97.24</td>
<td>95.63</td>
<td>95.21</td>
</tr>
<tr>
<td>w2v-GCN</td>
<td>95.81</td>
<td>94.13</td>
<td>98.16</td>
<td>96.11</td>
<td>95.66</td>
</tr>
<tr>
<td>w2v-GGNN</td>
<td>97.58</td>
<td>98.14</td>
<td>97.24</td>
<td>97.69</td>
<td>97.59</td>
</tr>
<tr>
<td>RoS2v</td>
<td>97.90</td>
<td>100.0</td>
<td>96.02</td>
<td>97.97</td>
<td>98.01</td>
</tr>
<tr>
<td>VDGraph2Vec-GCN*</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
</tr>
<tr>
<td>VDGraph2Vec-GGNN*</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>
Example of CWE-121

```c
void CWE121_Stack_Base_Buffer_Overflow__char_type_overrun_memcopy_01_bad()
{
    char Void structCharVoid;
    structCharVoid.voidSecond - (void *)SRC_STR;
    /* Print the initial block pointed to by structCharVoid.voidSecond */
    printf((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 */
    printf((char *)structCharVoid.charFirst);
    printf((char *)structCharVoid.voidSecond);
}
#endif /* OMITBAD */

#ifdef OMITGOOD

static void good1()
{
    char Void structCharVoid;
    structCharVoid.voidSecond = (void *)SRC_STR;
    /* Print the initial block pointed to by structCharVoid.voidSecond */
    printf((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 */
    printf((char *)structCharVoid.charFirst);
    printf((char *)structCharVoid.voidSecond);
}
```

No issues found.
Results

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

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

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

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

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