Vulnerability Detection with Fine-Grained Interpretations

来源信息

- ESEC/FSE(**CCF A**)
- 机构：新泽西理工学院（USA）
- 作者：Yi Li; Shaohua Wang; Tien N. Nguyen

摘要

现有的漏洞检测技术大多只能检测代码片段是否存在漏洞，但是不能完成漏洞的定位。论文提出了IVDetect方法，通过PDG提取，并提取与漏洞语句有关的控制和依赖关系语句，利用FA-GCN对代码进行表征并进行分类。利用GNNExplainer对分类结果进行解释，从PDG中选取子图进行解释，如果评判子图对分类结果的重要性，来确定漏洞的具体的位置。

动机

```
1   static long ec_device_ioctl_xcmd(struct cros_ec_dev *ec, void __user *arg)
2   {
3       long ret;
4       struct cros_ec_command u_cmd;
5       struct cros_ec_command *s_cmd;
6       if (copy_from_user(&u_cmd, arg, sizeof(u_cmd)))
7           return -EFAULT;
8       if ((u_cmd.outsize > EC_MAX_MSG_BYTES) || (u_cmd.insize > EC_MAX_MSG_BYTES))
9           return -EINVAL;
10      s_cmd = kmalloc(sizeof(*s_cmd) + max(u_cmd.outsize, u_cmd.insize), GFP_KERNEL);
11      if (!s_cmd)
12          return -ENOMEM;
13      if (copy_from_user(s_cmd, arg, sizeof(*s_cmd) + u_cmd.outsize)) {
14          ret = -EFAULT;
15          goto exit;
16      }
17  +   if (u_cmd.outsize != s_cmd->outsize ||
18  +       u_cmd.insize != s_cmd->insize) {
19  +       ret = -EINVAL;
20  +       goto exit;
21  +   }
22      s_cmd->command += ec->cmd_offset;
23      ret = cros_ec_cmd_xfer(ec->ec_dev, s_cmd);
24      /* Only copy data to userland if data was received. */
25      if (ret < 0)
26          goto exit;
27  -   if (copy_to_user(arg, s_cmd, sizeof(*s_cmd) + u_cmd.insize))
28  +   if (copy_to_user(arg, s_cmd, sizeof(*s_cmd) + s_cmd->insize))
29          ret = -EFAULT;
30  exit:
31      kfree(s_cmd);
32      return ret;
33  }
```

Figure 1: CVE-2016-6156 Vulnerability in Linux 4.6

上图是Linux内核中的一个漏洞，line-6，line-13分别利用指针从用户空间获取数据，在多线程情况下，可能出现竞争条件，另一个用户的线程可能会更改第二次取出的新值，当使用不一致的值时，会出现bug。

之前的方法，虽然可以检测出这段代码中的漏洞，但是无法定位出漏洞的具体位置，论文提出的IVDetect，不仅可以检测到漏洞，同时利用子图解释模型可以定位到具体的漏洞位置line-13.

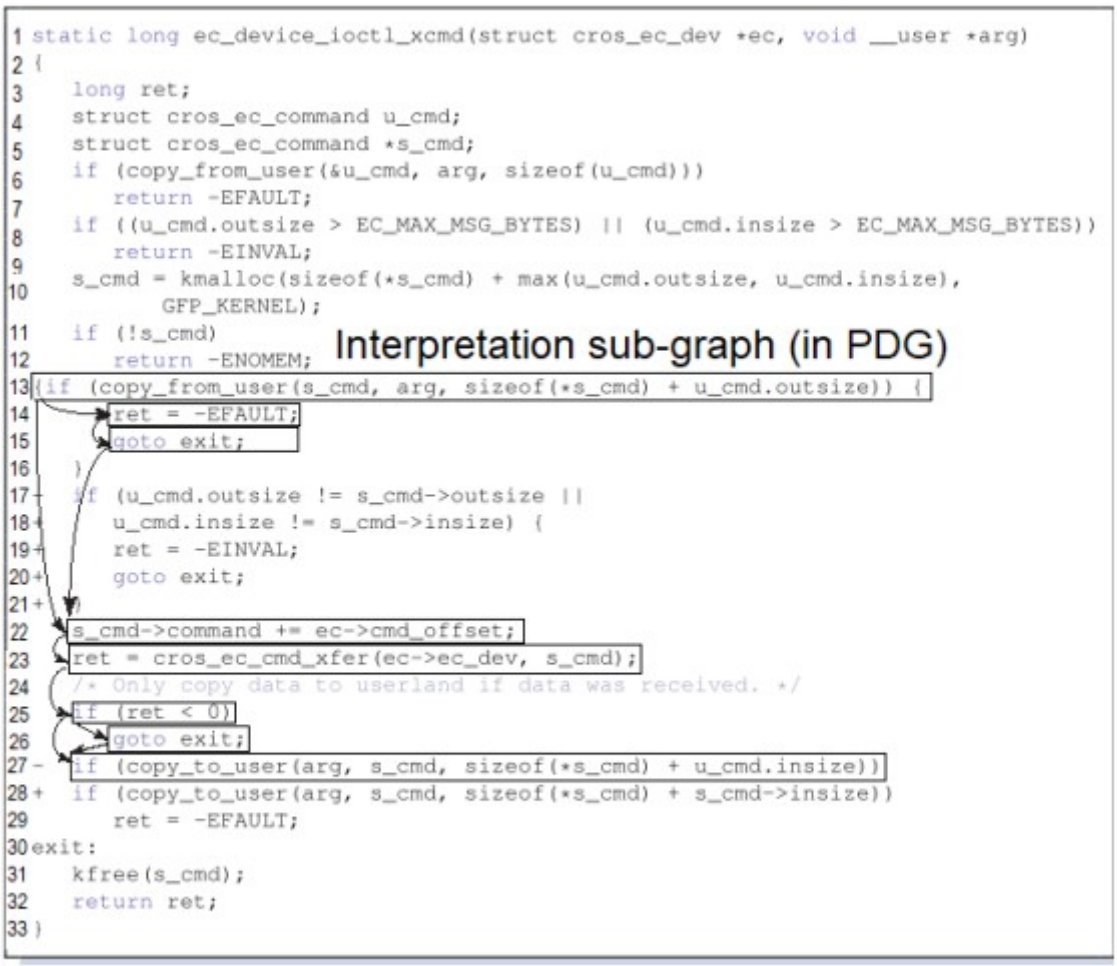

Figure 2: Interpretation Sub-Graph for Figure 1

论文方法

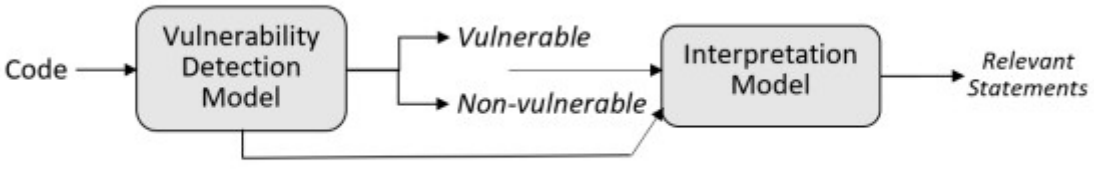

Figure 3: Overview of IVDetect

论文方法分为两个模块，漏洞检测模型：通过PDG提取漏洞语句相关的上下文，并利用GloVe和GRU完成源代码的特征表征。基于图的解释模块：主要任务是完成漏洞的定位，通过GNNExplainer对PDG子图进行解释并按照重要程度进行排序，对模型分类重要的语句就是漏洞产生的位置。

代码表征

在论文中的方法，作者提取了一下四种类型特征作为代码的表征

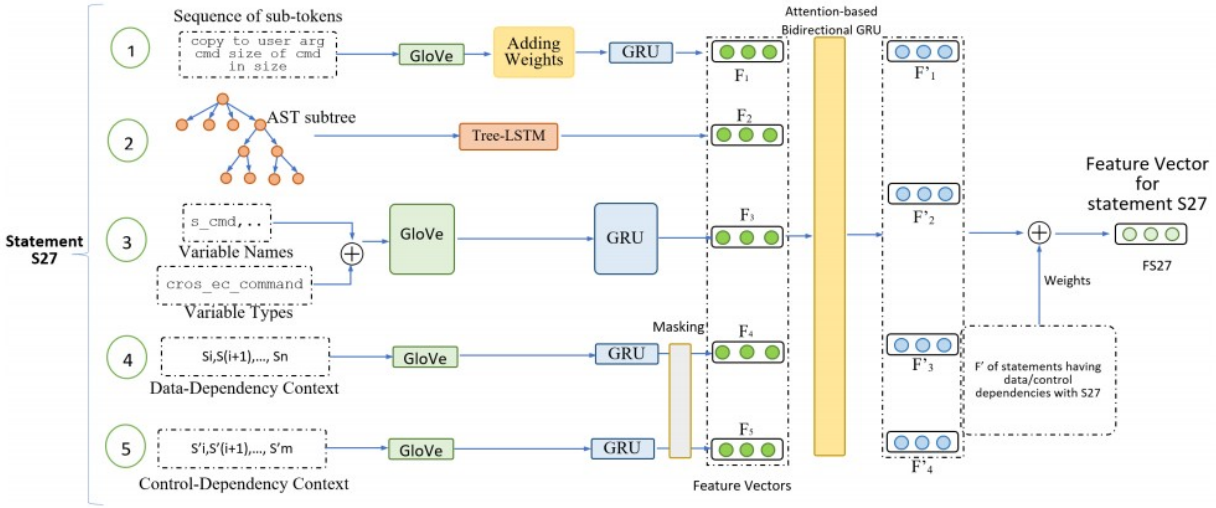

Figure 4: Code Representation Learning for Statement S27 in Graph-based Vulnerable Code Detection

- subtoken的标记序列：将语句转成token序列，然后利用驼峰命名法进一步将token拆分成sub-token。利用GloVe将Token编码成向量，使用GRU生成subtoken标记序列的特征向量。GloVe可以很好地捕获subtoken之间的语义相似性。GRU可以很好的输出整条语句的特征向量。Token化的过程中只考虑变量、方法和类名，其他的都移除。
- 语句的代码结构：AST子树，从AST中提取该语句的子树，然后利用Tree-LSTM生成语句结构的特征向量。
- 变量和类型：对于每一条语句，收集变量的名称和类型，利用1的方法拆分成sub-token，并使用GloVe编码成向量，GRU输出变量和类型的sun-token的特征向量。
- 上下文信息：对语句上下文有数据依赖和控制依赖的语句利用GloVe和GRU提取特征向量，
- 特征融合：提取了上面的4个特征之后，利用带有注意力机制的BGRU将这些特征融合起来，作为该语句的代码表征F'，对于上下文每一个依赖语句都要计算一个F',最后将这些F'加权求和得到最终的表征。

利用FA-GCN进行漏洞检测

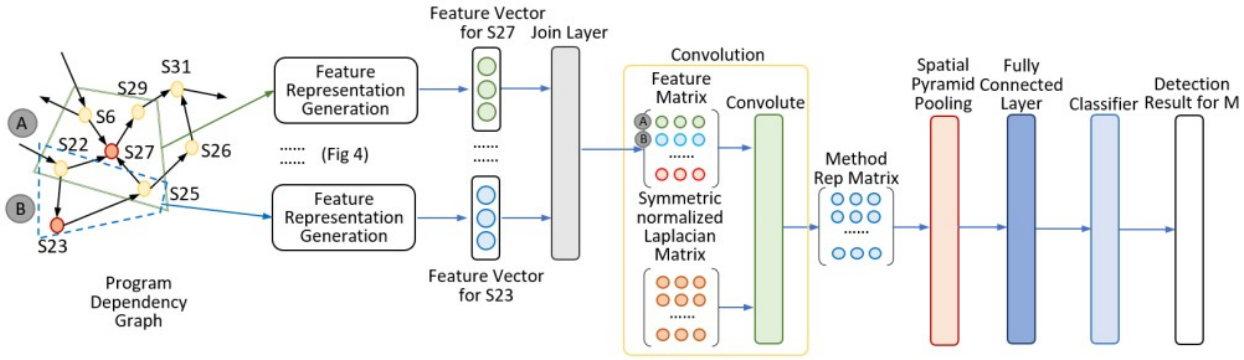

Figure 5: Vulnerability Detection with FA-GCN

首先，构建函数的PDG，FA-GCN会沿着PDG的所有节点（语句）执行一个滑动窗口，窗口内的内容就是当前节点关联的语句，例如S27的窗口内容为{S6，S22，S25，S29}，S23的窗口内容为{S22，S25}，根据代码表征的方法，迭代窗口的节点生成最终的节点特征表示，最后将每条语句的特征向量连接起来，构成了代码片段的特征矩阵。

接着计算对称归一化拉普拉斯矩阵并利用CNN进行卷积，生成方法的表示矩阵，并连接其到一个全连接层将矩阵转换成向量，最后通过两个隐藏层和一个softmax函数进行分类

在CNN步骤中，传统的CNN需要输入的shape是一致的，但是漏洞挖掘场景中shape会有不同，因此论文采用的空间金字塔池化层解决这个问题。

基于图的解释模型

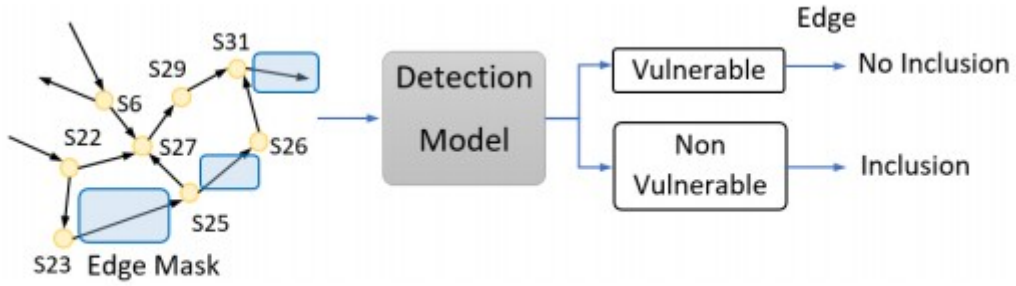

Figure 6: Masking to Derive Interpretation Sub-Graphs

论文利用GNNExclaner和edge-mask技术来实现漏洞的定位。其思想是最大化mask之后FA-GCN模型的输出差异。通过屏蔽掉PDG中的一些edge来判断这些edge对模型的检测是否重要。如果删除的边之后对模型的检测结果影响很大，说明该条edge很重要，反之对漏洞检测帮助不大。最后输出这些maskd edge的子图作为最终的模型解释。

实验

论文的数据集如下表所示

Table 1: Three Datasets

Dataset	Fan	Reveal	Devign
Vulnerabilities	10,547	1,664	10,067
Non-vulnerabilities	168752	16505	12,294
Ratio (Vul:Non-vul)	1:16	1:9.9	1:1.2

与现有的方法对比发现IVDetect在所有指标中都表现得更好。

Table 3: RQ1. Method-Level VD on FFMPeg+Qemu Dataset

	VulDee--Pecker	SySeVR	Russell *et al.*	Devign	Reveal	IVDetect
nDCG@1	0	0	0	0	0	1
nDCG@3	0	0	0	0	0	0.63
nDCG@5	0	0	0.43	0.45	0.5	0.65
nDCG@10	0.37	0.44	0.45	0.46	0.5	0.68
nDCG@15	0.45	0.48	0.49	0.52	0.55	0.75
nDCG@20	0.48	0.51	0.54	0.56	0.6	0.82
MAP@1	0	0	0	0	0	1
MAP@3	0	0	0	0	0	0.83
MAP@5	0	0	0.20	0.20	0.25	0.80
MAP@10	0.22	0.31	0.30	0.32	0.38	0.78
MAP@15	0.29	0.33	0.34	0.37	0.41	0.72
MAP@20	0.32	0.35	0.37	0.42	0.45	0.69
FR@1	n/a	n/a	n/a	n/a	n/a	1
FR@3	n/a	n/a	n/a	n/a	n/a	1
FR@5	7	6	5	5	4	1
FR@10	7	6	5	5	4	1
FR@15	7	6	5	5	4	1
FR@20	7	6	5	5	4	1
AR@1	n/a	n/a	n/a	n/a	n/a	1
AR@3	n/a	n/a	n/a	n/a	n/a	2
AR@5	n/a	n/a	5	5	4	3.3
AR@10	8.7	7.8	7.8	7.4	7.4	4.7
AR@15	11.2	10	9.5	10	9.1	7.6
AR@20	13.3	12.1	12.6	12.1	12.4	10.3
AUC	0.68	0.72	0.79	0.77	0.79	0.84

MAP多类平均精度，nDCG归一化折损累计增益（Normalized Discounted cumulative gain）

在三个数据集上的精度和召回率结果分析

Table 6: RQ1. Precision and Recall Results of Method-Level VD on Three Datasets (P: Precision; R: Recall; F: F score)

	FFMPeg+Qemu			Fan			Reveal		
	P	R	F	P	R	F	P	R	F
VulDeePecker	0.49	0.27	0.35	0.12	0.49	0.19	0.19	0.14	0.17
SySeVR	0.50	0.66	0.56	0.15	**0.74**	0.27	0.24	0.42	0.31
Russell *et al.*	0.55	0.41	0.45	0.16	0.48	0.24	0.26	0.12	0.16
Devign	0.52	0.63	0.57	0.18	0.52	0.26	0.33	0.32	0.32
Reveal	0.55	**0.73**	0.62	0.19	**0.74**	0.30	0.31	**0.58**	0.40
IVDetect	**0.60**	0.72	**0.65**	**0.23**	0.72	**0.35**	**0.39**	0.52	**0.45**

细粒度的VD解释比较

Table 7: RQ2. Fine-grained VD Interpretation Comparison

Interp. Model	Accuracy										MFR	MAR
	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10		
ATT	0.01	0.16	0.41	0.54	0.59	0.60	0.62	0.63	0.64	0.65	4.8	6.3
GRAD	0.01	0.19	0.43	0.54	0.59	0.62	0.63	0.65	0.66	0.67	4.2	5.6
GE	0.05	0.30	0.54	0.63	0.67	0.68	0.70	0.72	0.72	0.73	3.5	5.0

GE: GNNExplainer; Nx: x is the number of nodes in the interpretation

不同特征表征对模型的影响

Table 9: RQ4. Evaluation for the Impact of Internal Features.

	ST (A)	(A)+**SST** (B)	(B)+**AST** (C)	(C)+**Var** (D)	(D)+**CD** (E)	(E)+**DD** (F)
nDCG@15	0.25	0.27	0.29	0.35	0.42	0.45
nDCG@20	0.26	0.27	0.29	0.37	0.44	0.46
MAP@15	0.07	0.11	0.12	0.19	0.26	0.28
MAP@20	0.09	0.11	0.13	0.19	0.26	0.28
FR@15	14	12	11	7	5	4
FR@20	14	12	11	7	5	4
AR@15	14	13.5	11	10.3	9	8.5
AR@20	19.5	15	13.5	12.5	11.2	10.4
AUC	0.75	0.76	0.77	0.83	0.85	0.9

ST: sequence of tokens; **SST**: sequence of sub-tokens; **AST**: sub-AST; **Var**: variables; **CD**: control dependencies; **DD**: data dependencies; **F = IVDetect**

总结

论文的核心目标是实现漏洞的检测和定位。相较于以前的方法，不同之处首先是在特征的表征过程中，利用漏洞上下文的数据和控制依赖信息、语句的Token语义，可以更加全面的表征漏洞的信息，挖掘漏洞语句的上下文依赖关系。第二点在于对漏洞的定位，现有的方法大部分都是在解决识别和分类的问题，很少有做定位的。文章利用基于图的解释模型实现漏洞的定位，通过GNNExplainer结合edge-mask找到漏洞触发的子图，实现漏洞的定位。这在现实场景中有很大的意义，可以帮助开发人员快速定位漏洞完成修复。