Beijing Forest Studio
北京理工大学信息系统及安全对抗实验中心

基于深度学习的源代码漏洞挖掘

陈传涛　硕士研究生

2020年01月19日

内容提要

- 背景简介
- 基本概念
- 算法原理
- 优劣分析
- 应用总结
- 参考文献

背景简介

- 预期收获
  - 1. 了解源代码漏洞挖掘任务的基本概念和TIPO
  - 2. 了解深度学习与源代码漏洞挖掘的常见结合方式
  - 3. 理解一种基于树卷积神经网络的程序源代码嵌入方法

基本概念

基本概念

- 漏洞
  - 软件漏洞 (Vulnerability) 是由软件在设计、开发或配置过程中存在的错误 (Error) 而导致的缺陷 (Flaw) 的实例，利用漏洞可以违反某些显式或隐式的安全策略。


```
//STACK_OVER_FLOW
void stack_over_flow(unsigned char* data)
{
    //缓冲区
    unsigned char buffer[BUF_LEN];

    //拷贝数据至缓冲区
    strcpy((char*)buffer, (char*)data);
}
```

错误：拷贝动作没有约束边界

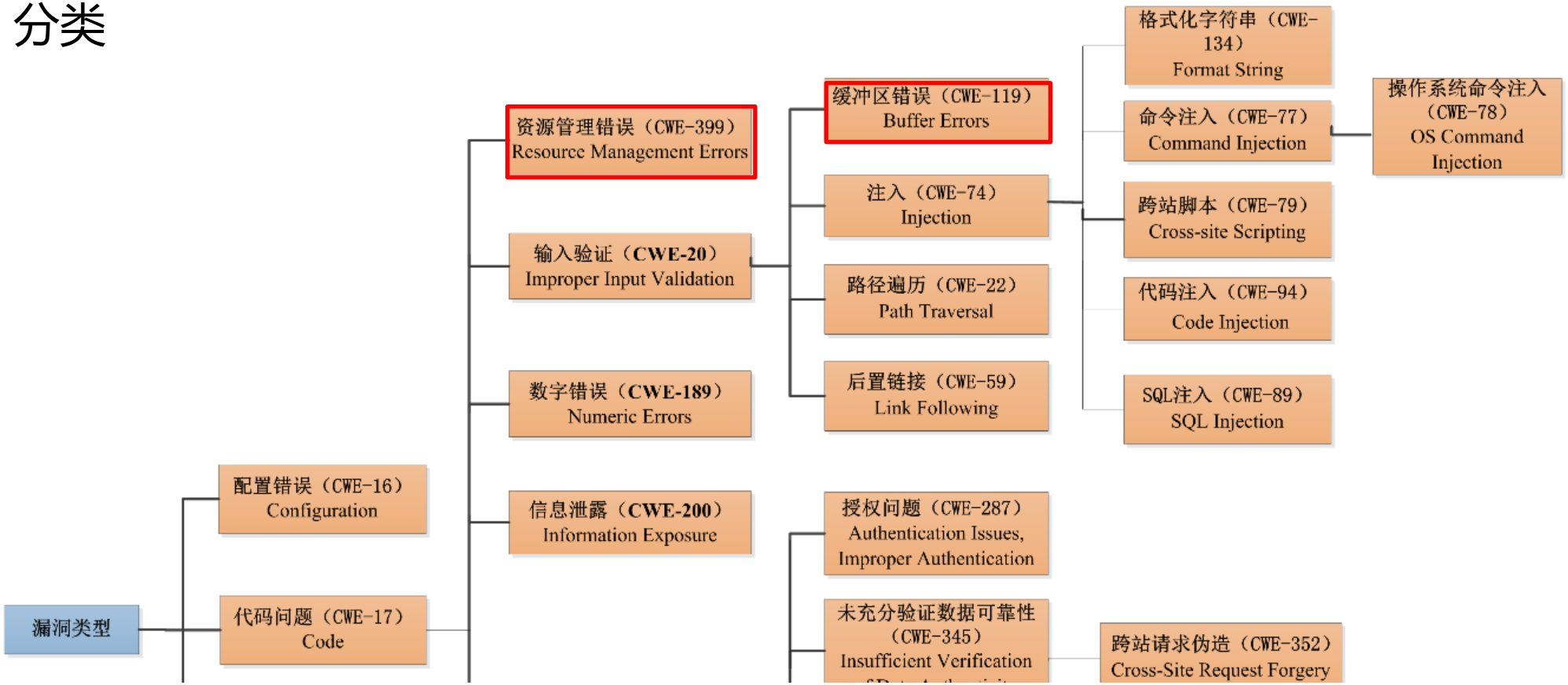
缺陷：可能导致栈缓冲区溢出

漏洞：可能导致恶意代码执行

参考：闫晗 - 基于深度学习的二进制软件漏洞挖掘 - 2019年12月15日

基础概念

- 漏洞类型
  - CWE，通用缺陷枚举(Common Weakness Enumeration)：根据漏洞原理进行分类



基础概念

- 缓冲区错误（CWE-119, Buffer Errors）
  - 软件在内存缓冲区上执行操作时读取或写入缓冲区的预定边界以外的内存位置。
  - 子类：
    - CWE-120: Buffer Copy without Checking Size of Input，复制时没有检查大小
    - CWE-121: Stack-based Buffer Overflow，栈溢出
    - CWE-122: Heap-based Buffer Overflow，堆溢出
    - CWE-125: Out-of-bounds Read，越界读
    - CWE-787: Out-of-bounds Write，越界写
    - ……

基础概念

- 缓冲区错误示例

```
int foo(char *str, size_t n)
{
    char buf[BUF_SIZE], *ar;
    size_t len = strlen(str);
    if(len >= BUF_SIZE) return ERROR;
    memcpy(buf, str, len);
    ar = malloc(n);
    if(!ar) return ERROR;
}
```

Non-vulnerable Program

```
int foo(char *str, size_t n)
{
    char buf[BUF_SIZE], *ar;
    size_t len = strlen(str);
    if(len >= 2*BUF_SIZE) return ERROR;
    memcpy(buf, str, len);
    ar = malloc(n);
    if(!ar) return ERROR;
}
```

Vulnerable Program

基础概念

- 资源管理错误（ CWE-399, Resource Management Errors）
  - 在软件执行过程中对系统资源（如内存、磁盘空间、文件等）的错误管理。
  - 子类：
    - CWE-401: Missing Release of Memory after Effective Lifetime，内存泄漏
    - CWE-415: Double Free，二次释放
    - CWE-416: Use After Free，释放后使用
    - CWE-590: Free of Memory not on the Heap，释放并不在堆上的内存
    - CWE-761: Free of Pointer not at Start of Buffer，释放指针不在缓冲区头
    - CWE-762: Mismatched Memory Management Routines，不匹配的内存管理例程（ malloc/free 与 new/delete 混用）
    - ……

基础概念

- 资源管理错误示例

```
char* ptr = (char*)malloc (SIZE);
...
if (abrt) {
    free(ptr);
}
...
free(ptr);
```

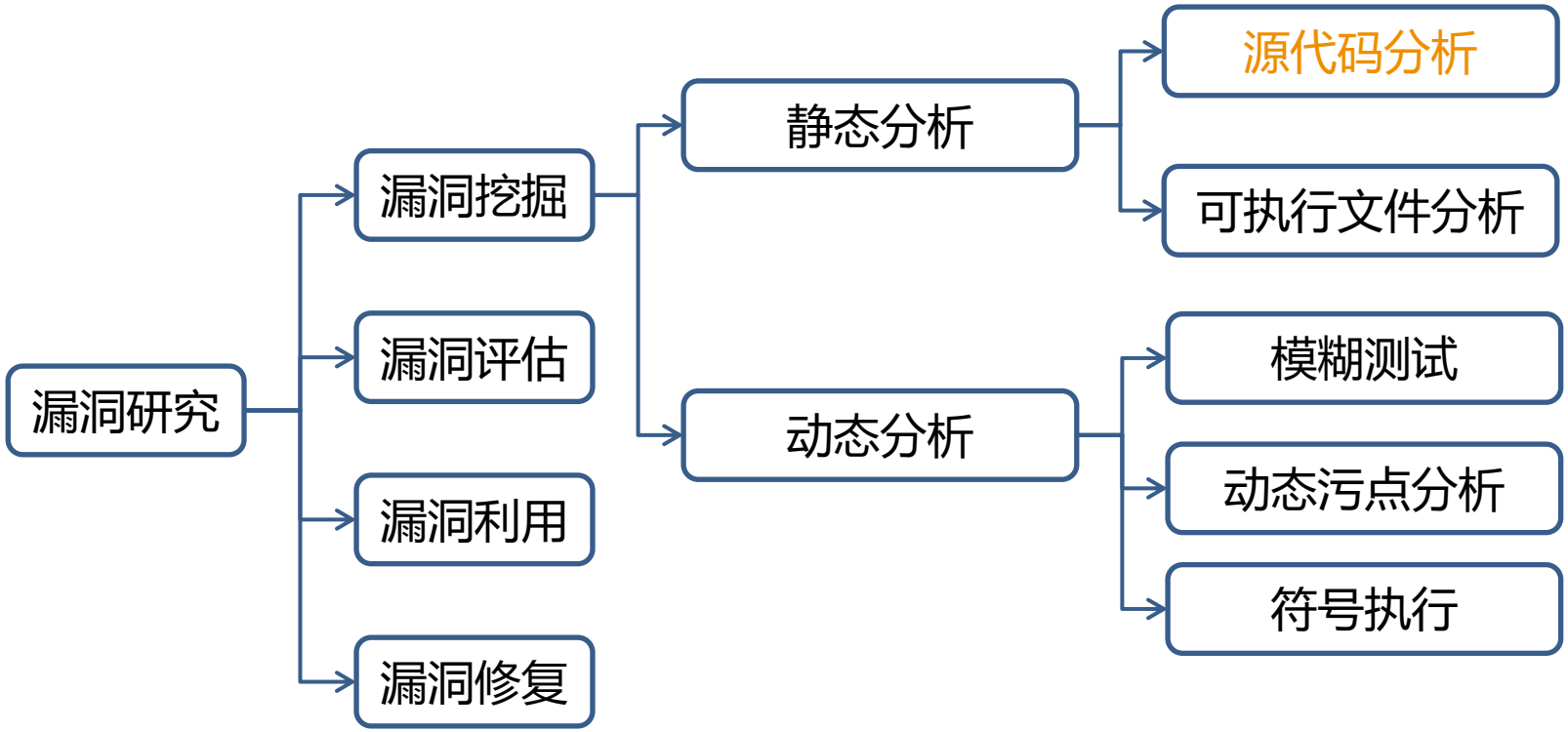
Double Free

```
char* getBlock(int fd) {
    char* buf = (char*) malloc(BLOCK_SIZE);
    if (!buf) {
        return NULL;
    }
    if (read(fd, buf, BLOCK_SIZE) != BLOCK_SIZE) {
        return NULL;
    }
    return buf;
}
```

Memory leaks

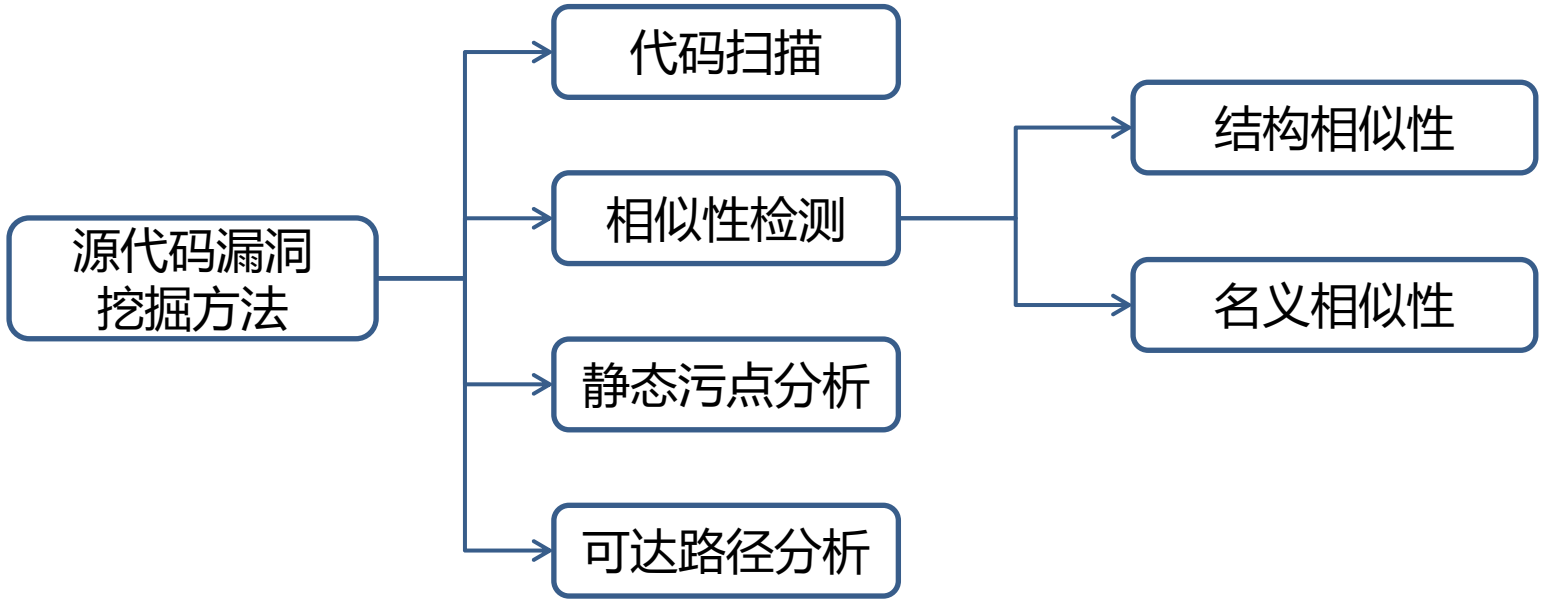
基础概念

- 安全漏洞的研究



基础概念

- 源代码漏洞挖掘的经典技术



算法原理

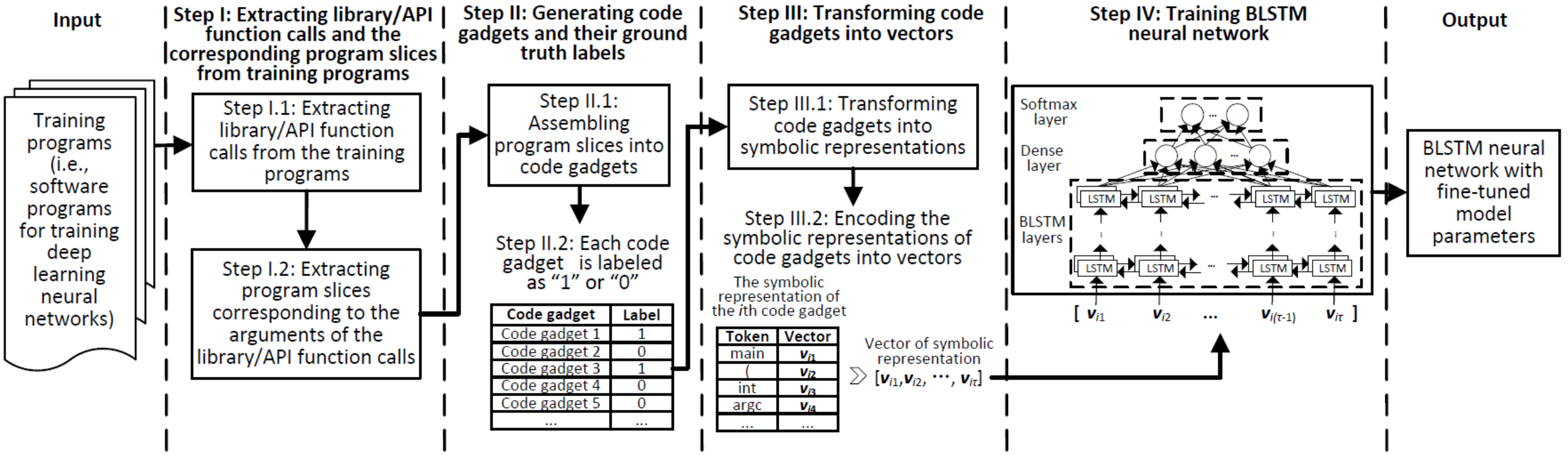
算法原理

T	从漏洞样本中自动学习漏洞模式
I	程序源代码
P	1. 数据预处理 2. 特征构建 3. 模型训练 4. 模型测试
O	源代码漏洞检测模型

P	对程序源代码进行适当的特征表示
C	源代码漏洞数据集
D	如何表示与漏洞相关的语法语义特征
L	CCF A类会议

算法原理

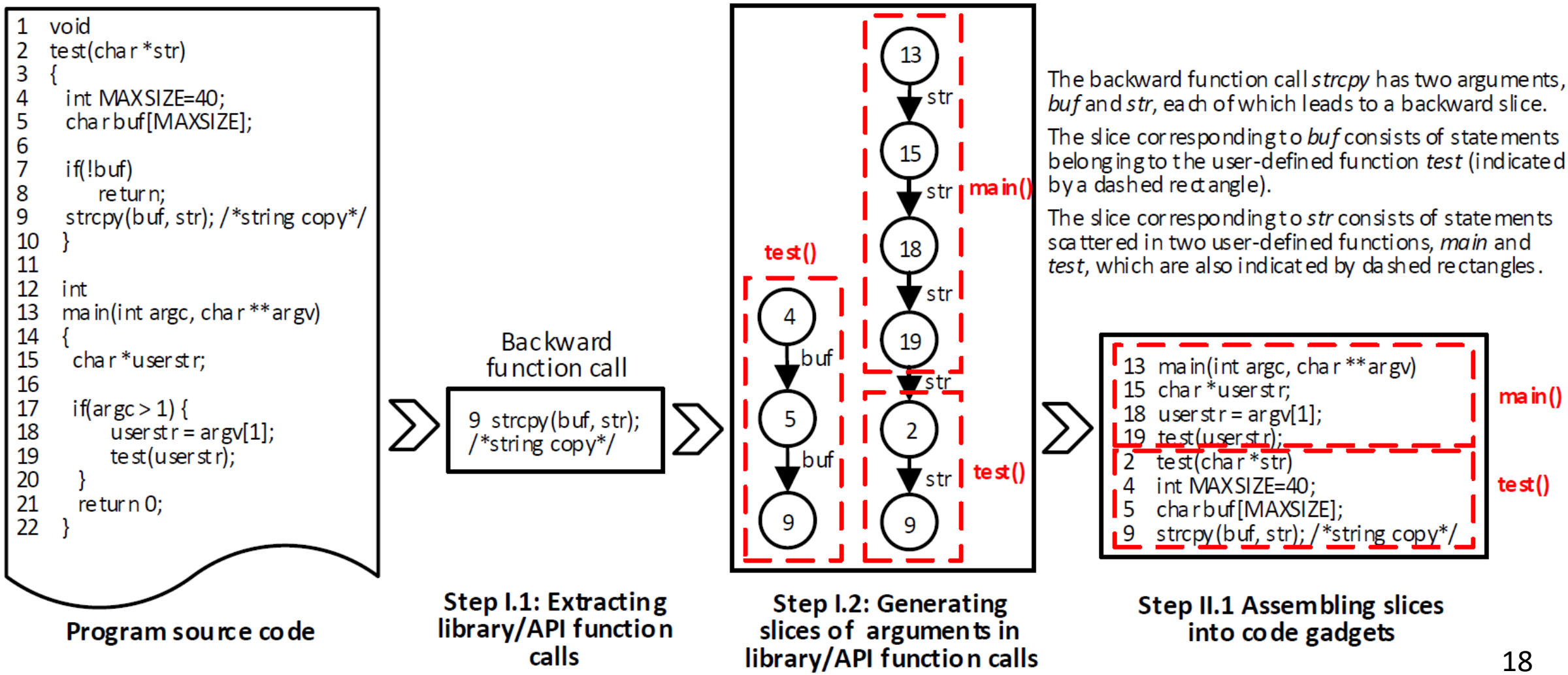
- 基于深度学习的源代码漏洞检测方法 - VulDeePecker
  - 总体框架



输入 → 数据预处理 → 代码向量表示 → 模型训练 → 输出

算法原理

- 数据预处理：程序切片



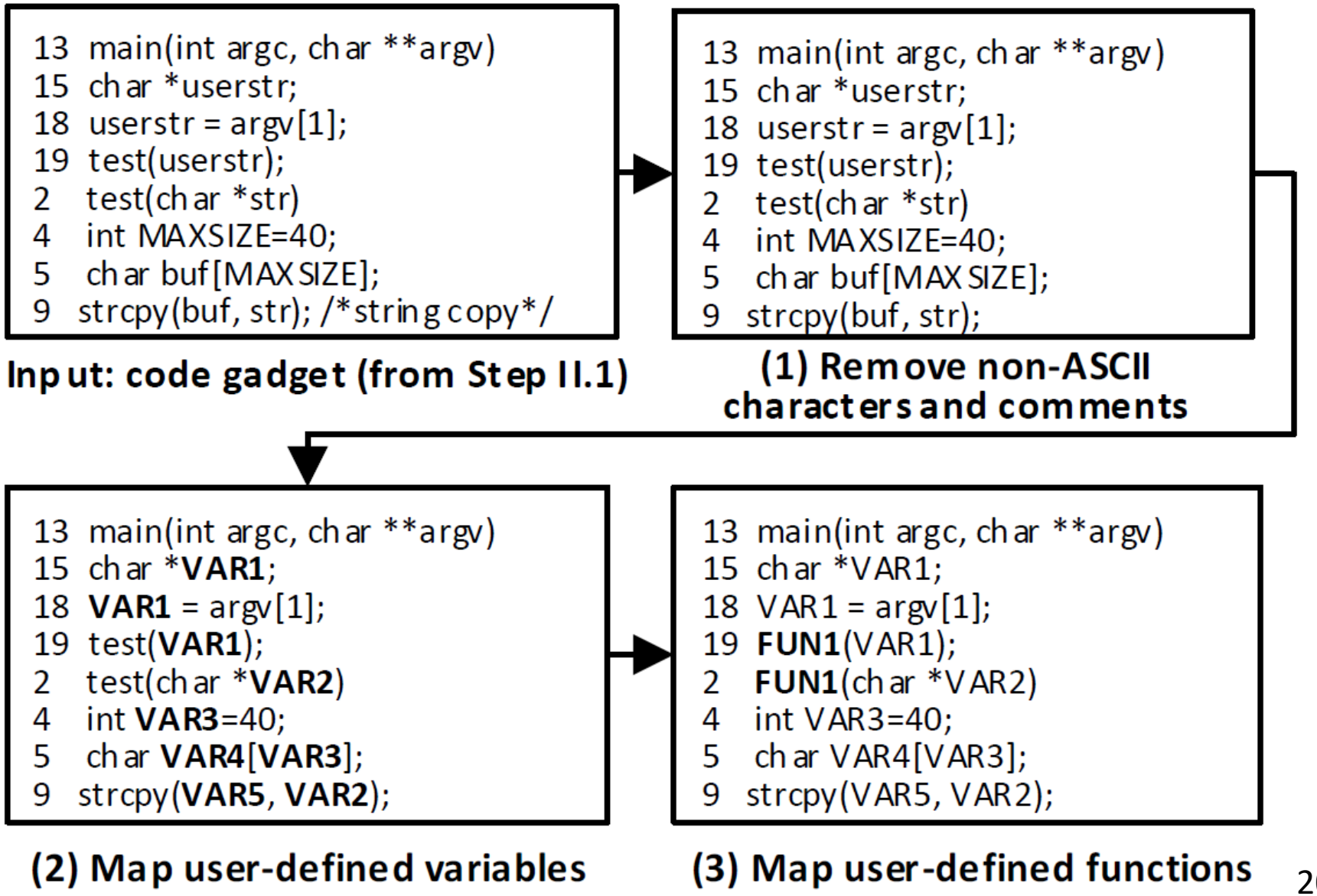
算法原理

- 数据预处理：关键API
  - CWE-119（缓冲区错误）
    - gets, scanf, memcpy, memmove,
    - istream.read*, SendMessage,
  - CWE-399（资源管理错误）
    - free, delete, new, malloc,
    - strdup, sprintf,

CWE ID	C/C++ library/API function calls related to vulnerabilities
CWE-119	cin, getenv, getenv_s, _wgetenv, _wgetenv_s, catgets, gets, getchar, getc, getch, getche, kbhit, stdin, getdlgtext, getpass, scanf, fscanf, vscanf, vfscanf, istream.get, istream.getline, istream.peek, istream.read*, istream.putback, streambuf.sbumpc, streambuf.sgetc, streambuf.sgetn, streambuf.snextc, streambuf.sputbackc, SendMessage, SendMessageCallback, SendNotifyMessage, PostMessage, PostThreadMessage, recv, recvfrom, Receive, ReceiveFrom, ReceiveFromEx, Socket.Receive*, memcpy, wmemcpy, _memccpy, memmove, wmemmove, memset, wmemset, memcmp, wmemcmp, memchr, wmemchr, strncpy, _strncpy*, lstrcpyn, _tcsncpy*, _mbsnbcpy*, _wcsncpy*, wcsncpy, strncat, _strncat*, _mbsncat*, wcsncat*, bcopy, strcpy, lstrcpy, wcscpy, _tcscpy, _mbscpy, CopyMemory, strcat, lstrcat, lstrlen, strchr, strcmp, strcoll, strcspn, strerror, strlen, strpbrk, strrchr, strspn, strstr, strtok, strxfrm, readlink, fgets, sscanf, swscanf, sscanf_s, swscanf_s, printf, vprintf, swprintf, vsprintf, asprintf, vasprintf, fprintf, sprint, snprintf, _snprintf*, _snwprintf*, vsnprintf, CString.Format, CString.FormatV, CString.FormatMessage, CStringT.Format, CStringT.FormatV, CStringT.FormatMessage, CStringT.FormatMessageV, syslog, malloc, Winmain, GetRawInput*, GetComboBoxInfo, GetWindowText, GetKeyNameText, Dde*, GetFileMUI*, GetLocaleInfo*, GetString*, GetCursor*, GetScroll*, GetDlgItem*, GetMenuItem*
CWE-399	free, delete, new, malloc, realloc, calloc, _alloca, strdup, asprintf, vsprintf, vasprintf, sprintf, snprintf, _snprintf, _snwprintf, vsnprintf

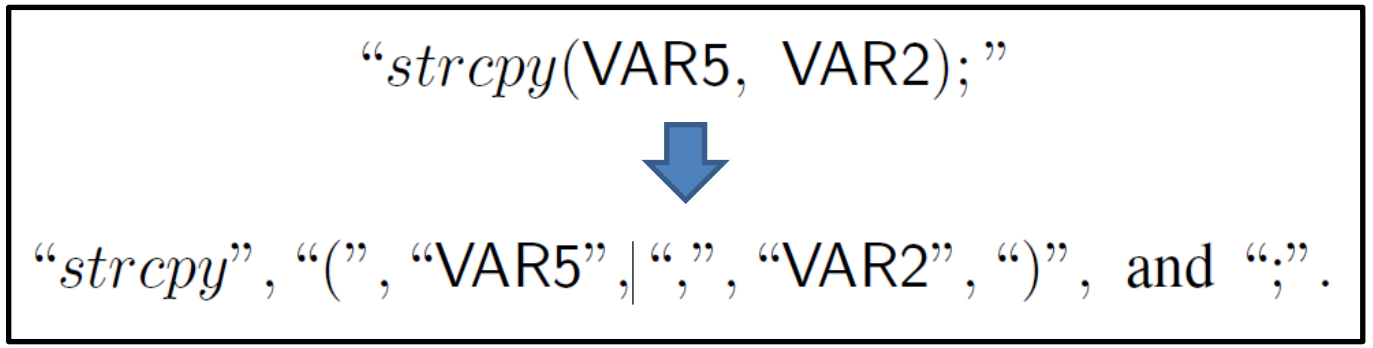
算法原理

- 数据预处理
  - 字符串替换



算法原理

- 代码向量表示
  - 主要步骤：
    - tokens 序列
    - 生成词嵌入
  - 优点
    - 实现简单
  - 缺点
    - 忽略了代码语句和代码块的逻辑独立性



源代码 tokens 序列

算法原理

- 实验结果
  - 在BE（Buffer Errors）和RM（Resource Management）数据集上的实验结果

Dataset	FPR(%)	FNR(%)	TPR(%)	P(%)	F1(%)
BE-ALL	2.9	18.0	82.0	91.7	86.6
RM-ALL	2.8	4.7	95.3	94.6	95.0
HY-ALL	5.1	16.1	83.9	86.9	85.4

  - 与其它工作的对比结果

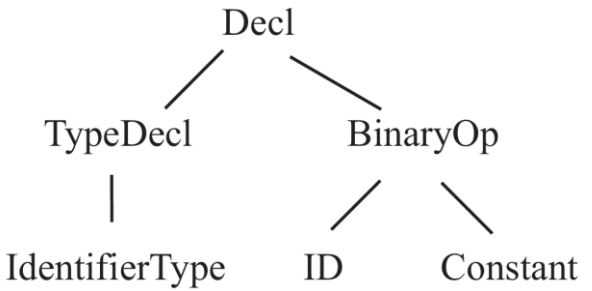
Method	FPR (%)	FNR (%)	A (%)	P (%)	F1 (%)
Flawfinder	21.6	70.4	69.8	22.8	25.7
RATS	21.5	85.3	67.2	12.8	13.7
Checkmarx	20.8	56.8	72.9	30.9	36.1
VUDDY	4.3	90.1	71.2	47.7	16.4
VulDeePecker	2.5	41.8	92.2	78.0	66.6
SySeVR-BGRU	1.4	5.6	98.0	90.8	92.6

算法原理

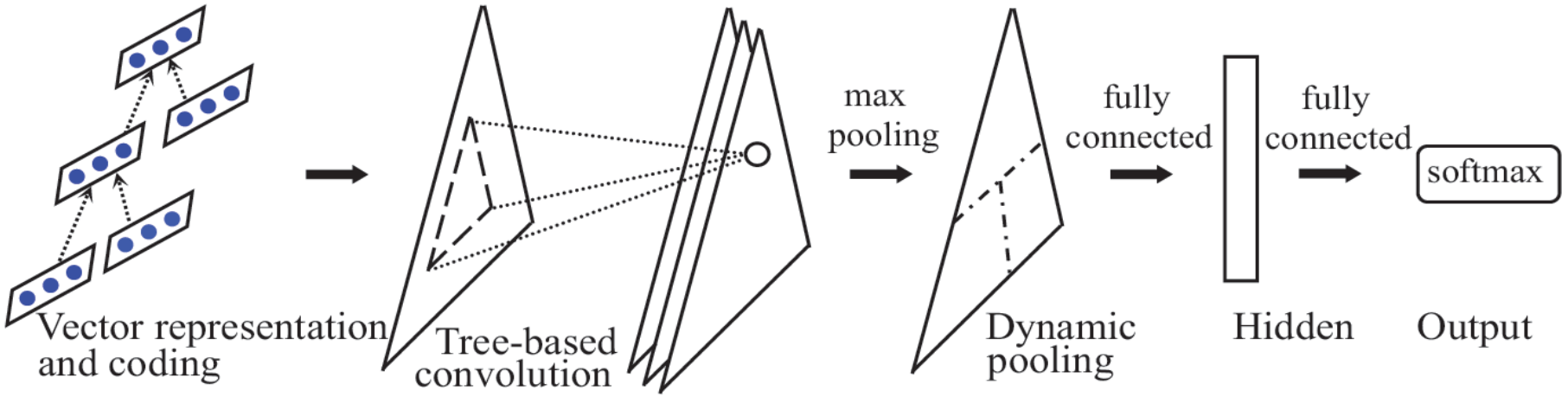
- 基于树卷积神经网络的程序分类 - TBCNN
  - 总体框架

数据预处理

int a=b+3;



模型训练



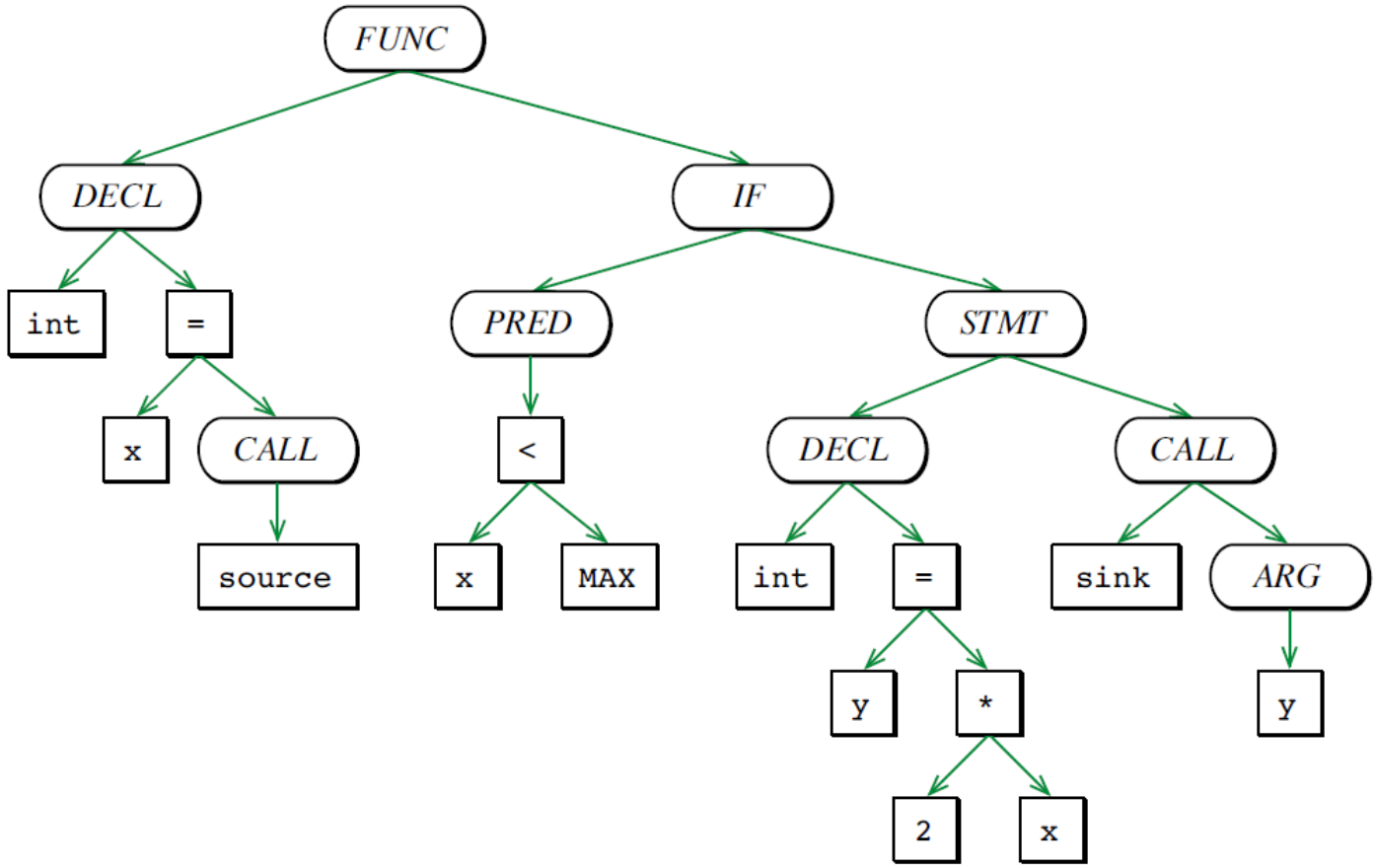
预训练 → 树卷积 → 池化 → 全连接 → 输出

算法原理

- 抽象语法树
  - 示例

```
void foo()
{
    int x = source();
    if (x < MAX)
    {
        int y = 2 * x;
        sink(y);
    }
}
```

Source code



Abstract syntax tree (AST)

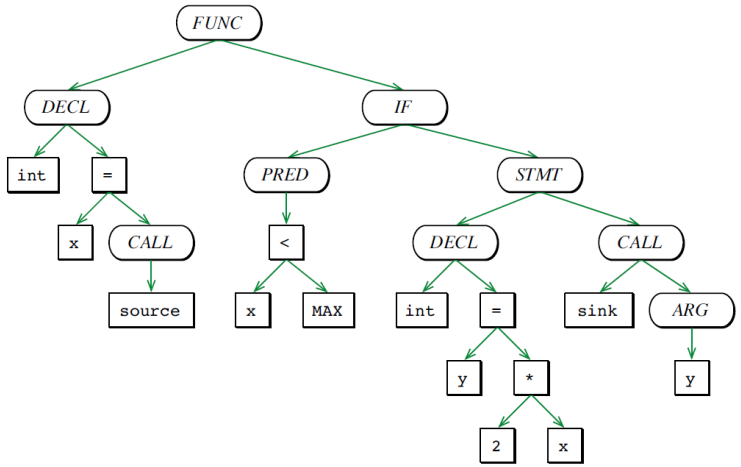
算法原理

- 预训练
  - 目的：获取所有 tokens 的向量表示。
  - 希望：一个节点能由其所有子节点进行表示

$$\mathrm{vec}(p) \approx \tanh\left(\sum_i l_i W_{\mathrm{code},i} \cdot \mathrm{vec}(c_i) + \boldsymbol{b}_{\mathrm{code}}\right)$$

  - 公式说明：
    - p：非叶子节点
    - c：p的直接子节点
    - vec()：词嵌入
    - Nf：特征维度
    - W：权重矩阵
  - 输出：

$$\boldsymbol{p} = W_{\mathrm{comb1}} \cdot \mathrm{vec}(p) + W_{\mathrm{comb2}} \cdot \tanh\left(\sum_i l_i W_{\mathrm{code},i} \cdot \mathrm{vec}(x_i) + \boldsymbol{b}_{\mathrm{code}}\right)$$

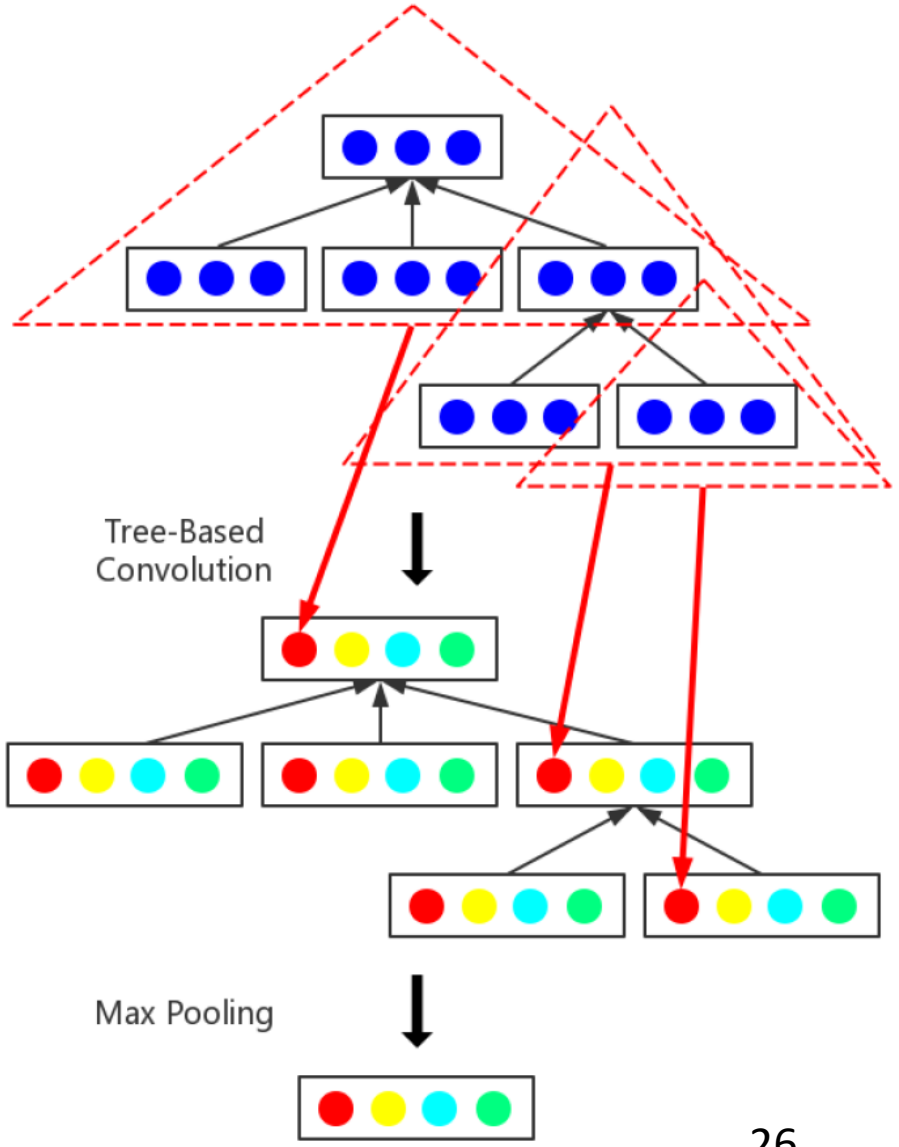


Abstract syntax tree (AST)

算法原理

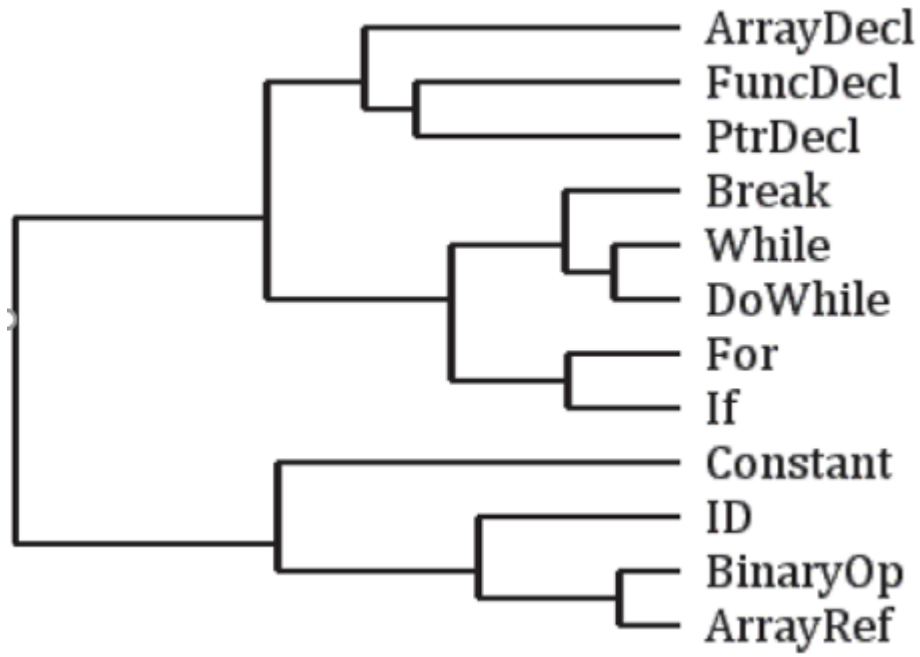
- 树卷积
  - 目的：获取程序的向量表示
  - 希望：每个树节点能够包含其附近节点的信息
  - 输出：

$$\boldsymbol{y} = \tanh\left(\sum_{i=1}^{n} W_{\text{conv},i} \cdot \boldsymbol{x}_i + \boldsymbol{b}_{\text{conv}}\right)$$

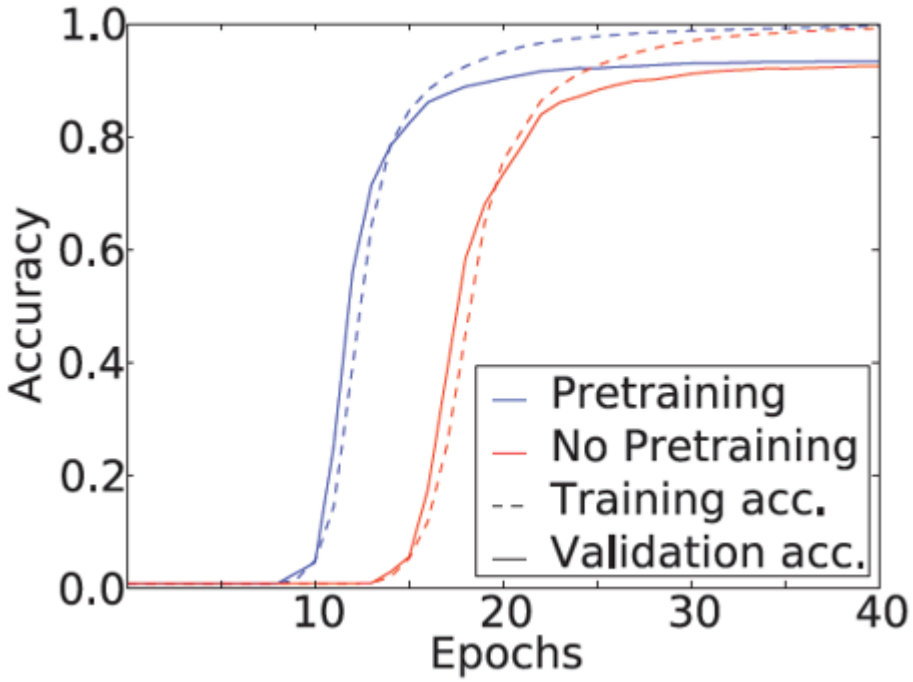


算法原理

- 实验结果



AST节点向量表示的分层聚类



有无预训练时的学习曲线对比

优劣分析

- 横向对比
  - 基于源代码tokens序列的漏洞检测方法 – VulDeePecker
    - 优点：baseline
    - 缺点：根据数据依赖对程序切片，可能会误切漏洞相关代码；tokens 序列不能很好表示代码语句的上下文结构
  - 基于树卷积神经网络的程序分类 – TBCNN
    - 优点：能够捕捉代码结构信息
    - 缺点：树深度过大时产生梯度消失；生成抽象语法树需要代码可编译；
- 纵向对比
  - 经典方法：漏报率高、准确率低、效率低、难以适应新样本
  - 深度学习：漏报率低、准确率高、效率高、可以适应新样本

应用总结

- 算法的应用领域
  - 应用深度学习处理程序分析相关任务：函数相似性检测、恶意行为检测、漏洞检测、代码分类

- 未来的发展
  - 代码属性图（CPG）+图神经网络（GCN）

参考文献

[1]. Li, Z., et al., VulDeePecker: A Deep Learning-Based System for Vulnerability Detection. 2018 NETWORK AND DISTRIBUTED SYSTEM SECURITY SYMPOSIUM, 2018.

[2]. Mou, L., et al., Convolutional Neural Networks over Tree Structures for Programming Language Processing. 2016.

[3]. CWE - CWE List Versison 3.4.1 https://cwe.mitre.org/data/index.html

谢谢！

大成若缺，其用不弊。大盈若冲，其用不穷。大直若屈。大巧若拙。大辩若讷。静胜躁，寒胜热。清静为天下正。