Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

软件漏洞注入技术

博士研究生 张浩然

2023 年 09 月 24 日

问题回溯

- 总结反思
  - 文献选择时间旧，新颖性较弱
  - 讲解领域和实验室研究方向关联度不足
  - 缺乏对基础概念、知识的讲解
- 相关学术报告
  - 2023.05.14-孔令迪：《源代码漏洞检测》
  - 2023.03.26-谢宁：《软件漏洞检测及其严重性评估》
  - 2023.10.30-孔令迪：《函数级漏洞检测》

内容提要

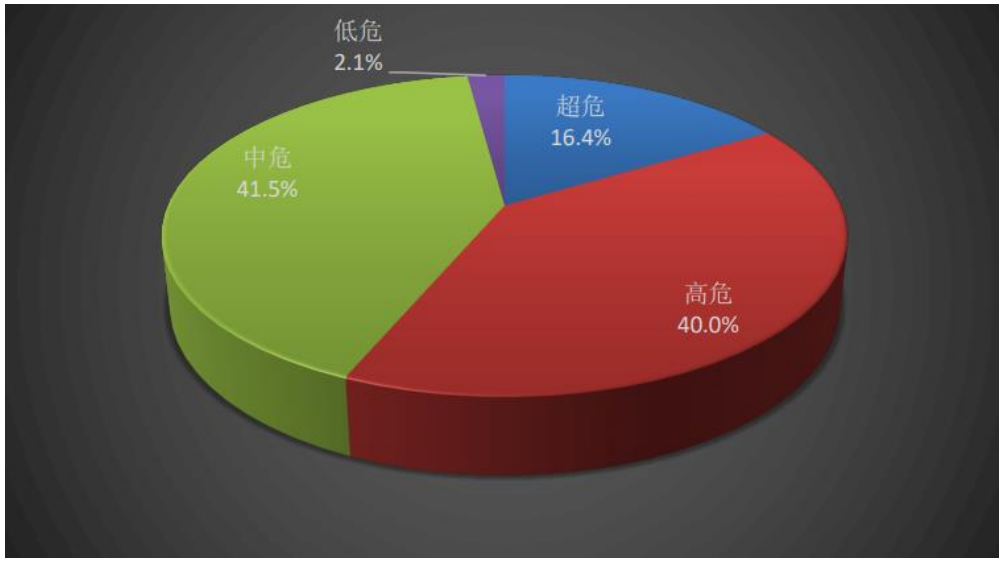
- 背景简介
- 基本概念
- 算法原理
- 总结
- 参考文献

背景简介

- 预期收获
  - 了解软件漏洞注入目的和常用方法
  - 理解软件漏洞注入真实漏洞方法
  - 理解语义类比方法在软件漏洞注入任务中的应用
  - 了解软件漏洞注入的前沿发展

背景简介

- 2022年安全漏洞态势报告
  - 自2017年每年公布的漏洞数量逐年上升，超危、高危漏洞占比56.4%
  - 2022年记录漏洞24892条，增幅达到23.2%



漏洞检测、防御方法至关重要

背景简介

- 漏洞检测算法
  - 基于深度学习的软件漏洞检测算法，需要大量漏洞数据作为训练集
  - 模糊测试方法需要准确的软件漏洞数据，用于评估挖掘效果
- 面临问题
  - 可以获取的带标记漏洞较少
  - 软件包含多少漏洞需要进行人为评估
  - 人为构建漏洞语料库成本高
    - 包含14个bug的完善语料库构建时间长达数月
    - 较新、较全的漏洞数据集具有商业价值
    - 需要经验非常丰富的专家来收集、构建

- 需要一种批量自动生成带漏洞软件的技术，降低检测算法的挑战性

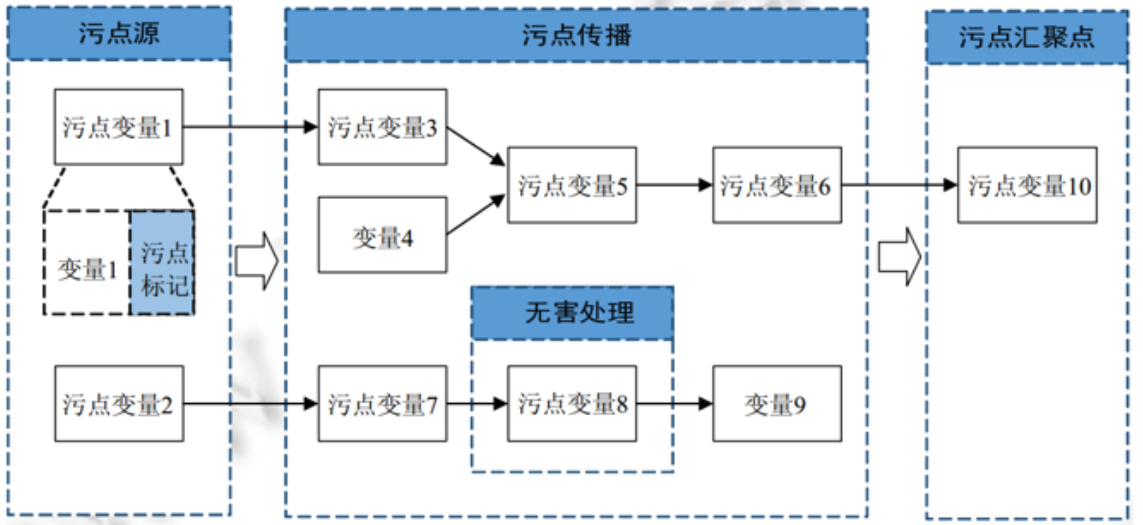
背景简介

- 软件漏洞注入技术
  - 通过插入、替换等方式，向正常代码中注入包含漏洞的代码语句、片段
    - 快速、自动化的批量生成漏洞
    - 部分软件包含不止一个漏洞，且有明确的输入可以进行触发
    - 包括缓冲区溢出(Buffer Overflow)、内存泄漏(Memory leak)等错误

```
1
   void foo(int a, int b, char *s, char *d, int n) {
3       int c = a+b;
        if (a != 0xdeadbeef)
5           return;
        for (int i=0; i<n; i++)
7           c+=s[i];
        memcpy(d,s,n+c); // Original source
        // BUG: memcpy(d+(b==0x6c617661)*b,s,n+c);
9   }
```

背景简介

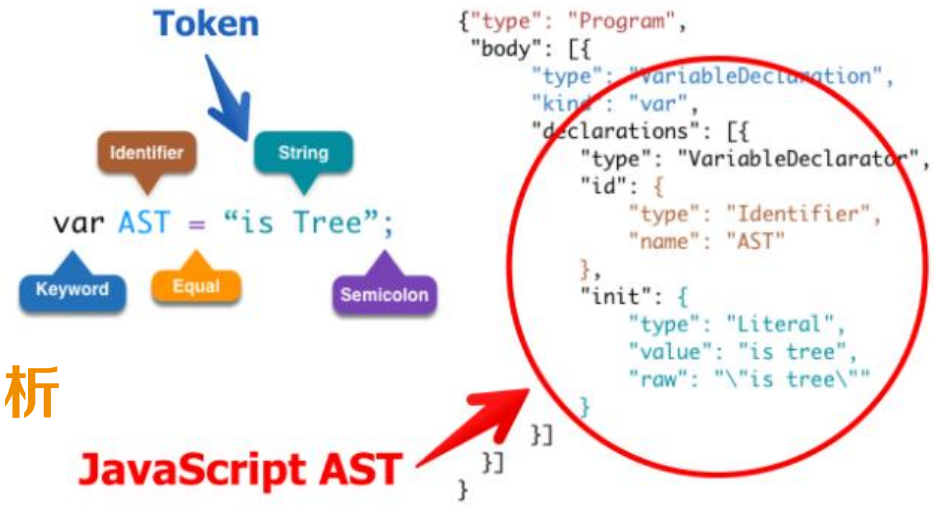
- 污点分析(Taint Analysis)
  - 污点源–直接引入不信任数据的位置
    - 程序输入参数、外部读取操作
  - 污染传播–数据流、控制流传播污点
  - 污染点–被污染源控制的安全敏感操作
    - 数据操作：数据库操作、文件操作
    - 内存操作：内存、指针的访问、修改、释放
    - 权限操作：敏感信息访问、身份认证、网络通信
  - 通过追踪程序执行流程，修改污染点逻辑，保留程序功能的同时植入漏洞
- 代码变异(Mutation)
  - 构造变异算子（值、语句、运算符），变更程序执行逻辑，实现漏洞注入



生成的漏洞与现实漏洞相似性不高

基础概念 抽象语法树&词嵌入

- 抽象语法树 (Abstract Syntax Tree)
  - 源代码语法结构的一种抽象表示
  - 树上的每个节点都表示源代码中的一种结构
  - 解释型语言：词法分析->语法分析->语法树
  - 使编程工具更好的理解源代码，有助于研究者分析

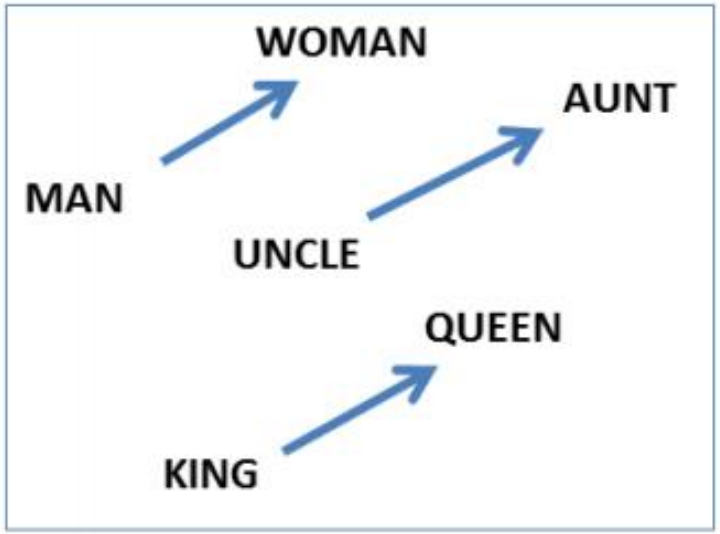


- 词嵌入(Word embedding)
  - 将自然语言中的词语映射为数值（稠密向量）表示的方式
  - 区别于OneHot、N-gram，可以有效表示词之间的关系

$$King = \begin{bmatrix} -0.95 \\ 0.93 \\ 0.7 \\ . \\ . \end{bmatrix} \quad Queen = \begin{bmatrix} 0.97 \\ 0.95 \\ 0.69 \\ . \\ . \end{bmatrix} \quad Apple = \begin{bmatrix} 0.00 \\ -0.01 \\ 0.03 \\ . \\ . \end{bmatrix} \quad Orange = \begin{bmatrix} 0.01 \\ 0.00 \\ -0.02 \\ . \\ . \end{bmatrix} \quad Man = \begin{bmatrix} -1 \\ 0.01 \\ 0.03 \\ . \\ . \end{bmatrix} \quad Woman = \begin{bmatrix} 1 \\ 0.02 \\ 0.02 \\ . \\ . \end{bmatrix}$$

基础概念 词类比

- 词类比（Word Analogy）
  - 自然语言中存在关系的词，其对应嵌入向量在向量空间存在运算关系
  - What word is to 'king' as 'woman' is to 'man'?
  - $\overrightarrow{king} - \overrightarrow{man} + \overrightarrow{woman} \approx \overrightarrow{queen}$
- 成立条件：共现位移PMI定理（预训练集特征）
  - $csPMI(x,y) = PMI(x,y) + \log p(x,y)$，出现关联性
    - $PMI(x,y)$：x,y的点互信息，代表x,y的关联性
    - $p(x,y)$：x,y同时出现概率
  - $csPMI(king, queen) = csPMI(man, woman)$时成立
  - 更直观解释：$\frac{p(\omega|a)}{p(\omega|b)} \approx \frac{p(\omega|x)}{p(\omega|y)}$，$\omega$是所有单词
  - 词对具有相似上下文分布特征



发现现象→证明结论

SemSeed

Semantic Bug Seeding: A Learning-Based Approach for Creating Realistic Bugs

算法原理 SemSeed

T	目标	在目标程序中仿照真实情况注入漏洞
I	输入	目标程序源代码
P	处理	1. 从已有代码变更中提取bug注入模式 2. 在目标程序中进行匹配，选择可注入bug位置 3. 在选定位置中依照bug注入模式生成代码，输出注入漏洞结果
O	输出	带有漏洞的代码

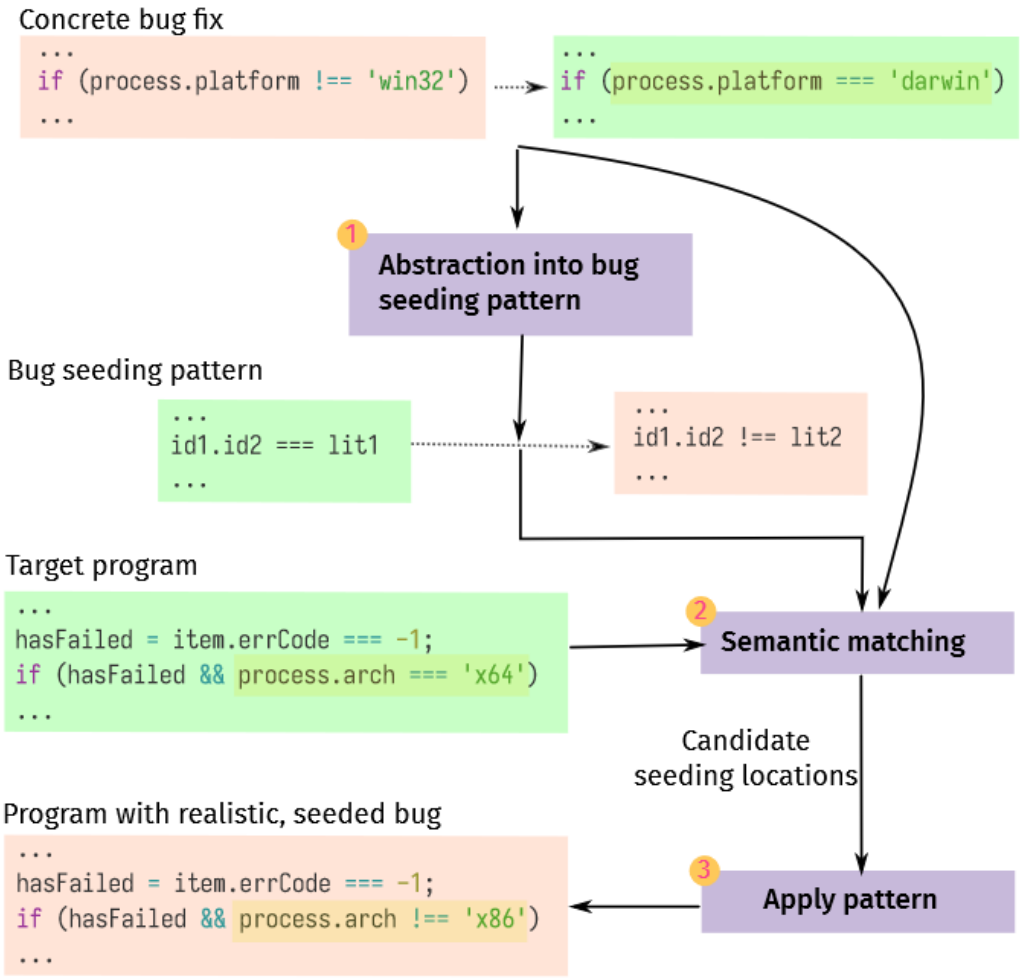
P	问题	现有方法创建的bug不够贴近现实
C	条件	训练集和测试集使用相同语言代码，词嵌入满足共现位移PMI
D	难点	1. 如何定位目标程序可以注入bug的位置 2. 如何依照目标程序语境生成恰当代码片段
L	水平	ESEC/FSE 2021，软件工程顶会，Distinguished Paper Award

算法原理 SemSeed

- 算法流程
  - 从已有代码变更中提取bug修复模式
  - 在目标代码中进行匹配，选择注入点
  - bug代码生成，输出注入漏洞结果
- 算法思想
  - 提取已有变更中的bug修复模式
  - 在相似的正常代码中执行相反过程

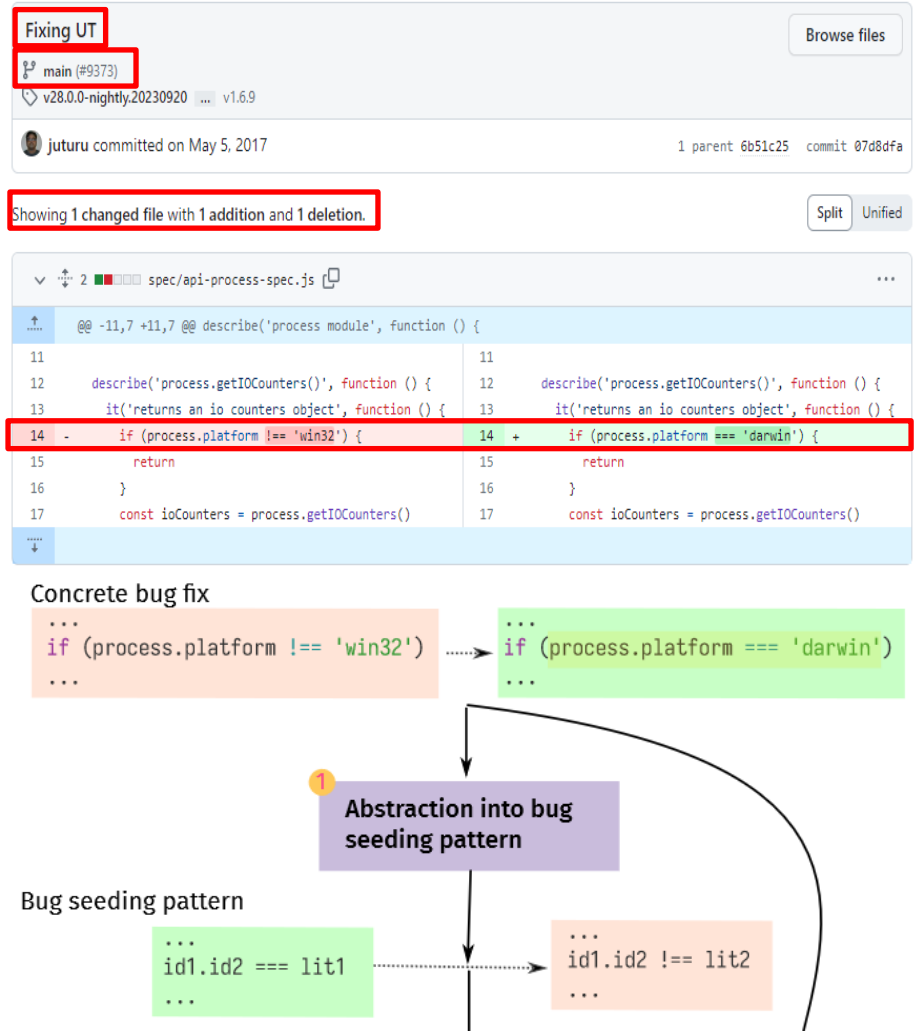
bug代码 + 修复模式 = 正常代码

正常代码 + (-修复模式) = bug代码



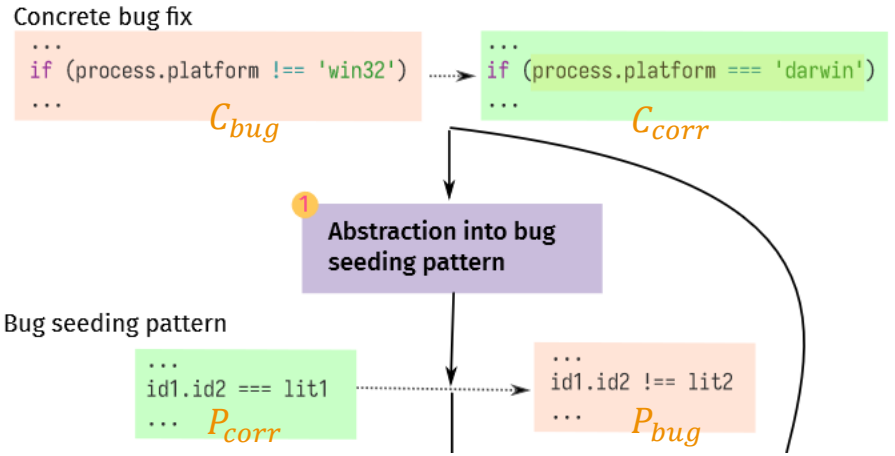
算法原理 Step1.Bug修复模式提取

- 获取GitHub上bug修复信息
  - commit message筛选
    - 关键字筛选bug/fix/error/issue/problem
  - 不存在分支，防止合并提交
  - 编程语言匹配
  - 文件变更数等于1，代码变更行数等于1
- 提取bug具体修复
  - 将代码表示为AST，修剪与更改行无关子树
  - bug具体修复对(C_{bug}, C_{corr})
    - $C_{bug} = [process.platform\ !==\ 'win32']$
    - $C_{corr} = [process.platform\ ===\ 'darwin']$



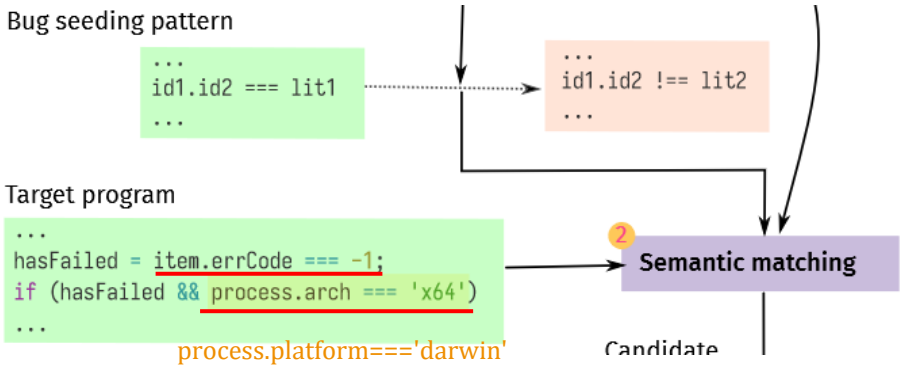
算法原理 Step1.Bug修复模式提取

- 提取bug修复模式
  - 编程代码中的3类标识
    - identifier(id)-标识符，变量名、函数名
    - literal(lit)-字面量，常量
    - 保留不变–其他，运算符、关键字
  - 获得bug修复模式对
    - $(C_{bug}, C_{corr}) \rightarrow (P_{corr}, P_{bug})$，具体到抽象
    - 提取bug修复模式，迁移至可能产生类似问题的代码片段，实现漏洞注入
  - 引导注入目标选择
    - 以P_{corr}匹配目标代码中的结构相同位置
    - 以C_{corr}匹配目标代码中的语义相同位置



算法原理 Step2.漏洞注入点选择

- 结构匹配
  - 通过AST提取目标代码模式token序列
  - token均为$[id1.id2 === lit1]$
  - 快速定位与C_{corr}结构相同的代码语句集合
- 语义匹配
  - 计算候选语句与C_{corr}的语义相似度
  - 词嵌入算法：FastText
    - 更有效处理OOV问题
    - 更精准的判别语义相似性
  - 余弦相似度$simil(v,w) = \frac{v \cdot w}{\|v\| \cdot \|w\|}$，阈值$m$
  - 精准定位与C_{corr}语义相似的代码语句片段



Algorithm 1 Semantically match a token sequence against a bug seeding pattern.

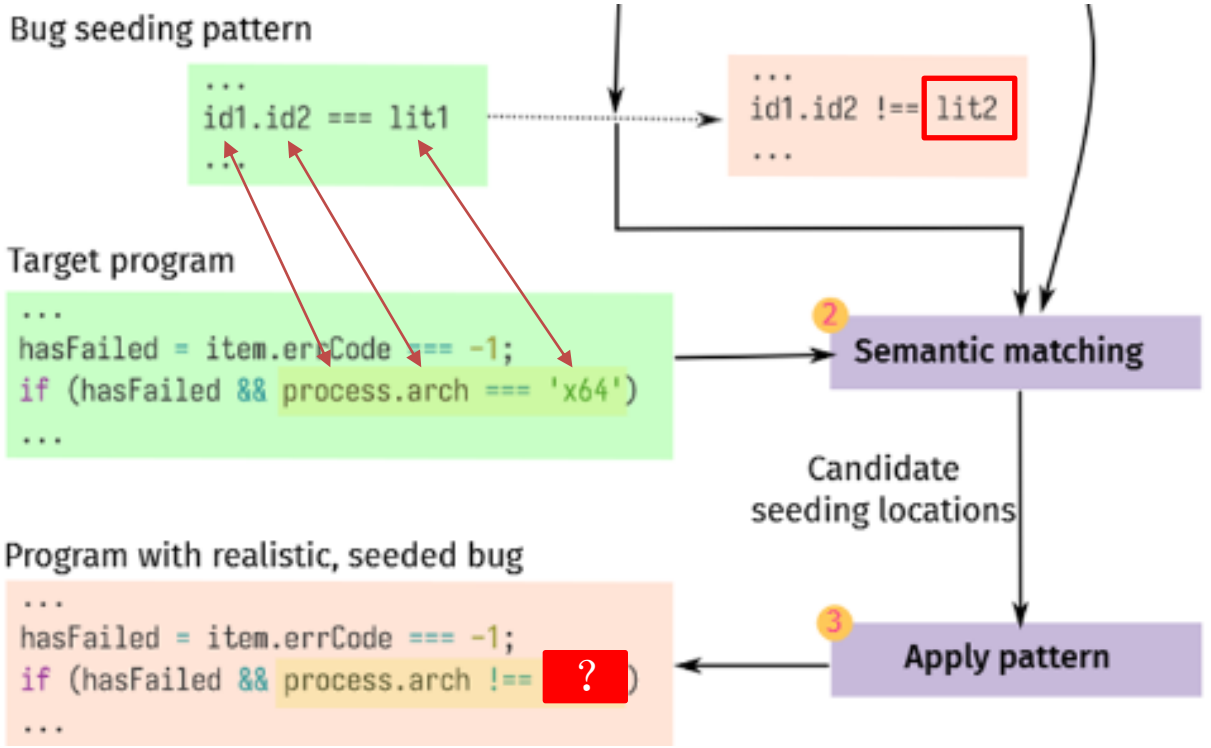
Input: Token sequence C and concrete bug fix (C_{bug}, C_{corr})
Output: *True* if C is a semantic match, *False* otherwise

1: $[t_1, \ldots, t_n] \leftarrow C$ ▷ Tokens of target location
2: $[t'_1, \ldots, t'_n] \leftarrow C_{corr}$ ▷ Tokens where real bug occurred
3: $S \leftarrow []$
4: **for** $i = 1$ **to** n **do**
5: **if** $kind(t_i) \in \{Identifier, Literal\}$ **then**
6: $v \leftarrow emb(t_i)$
7: $v' \leftarrow emb(t'_i)$
8: Append $simil(v, v')$ to S
9: **return** $avg(S)$ >= matching threshold m

算法原理 Step3.bug代码生成

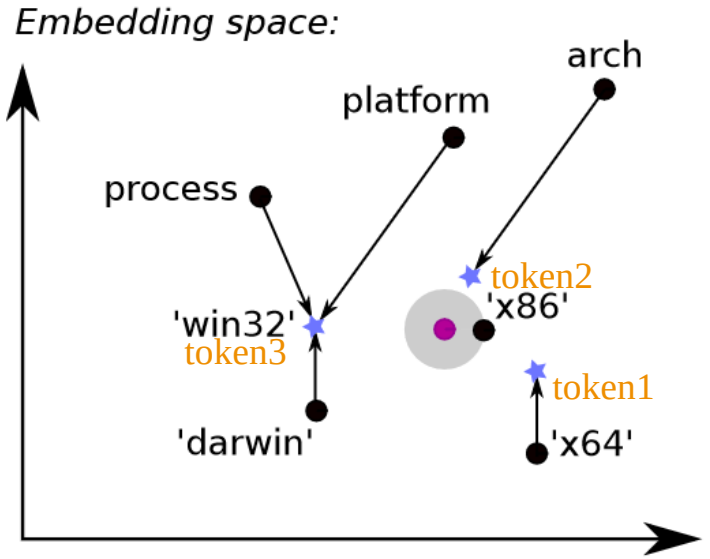
- 未绑定token处理
  - 仅出现在错误模式中的token，在插入bug的过程中无法确定对应的文本
  - token绑定关系：$(P_{bug}, target)$
    - $id1 \leftrightarrow process,\ id2 \leftrightarrow arch$
    - $lit2$：未绑定token
  - 使用原本C_{bug}的token文本?
    - 未定义的变量
    - 变量类型错误
  - 使用上下文中出现过的token?
    - 可靠性无法保证

如何确保未绑定token的合理有效?



算法原理 Step3.bug代码生成

- 类比查询(Analogy Queries)
  - 词类比通过语义感知(semantics-aware)未绑定token
  - 通过代码语境类比查询token
    - What token1 is to *'x64'* as *'win32'* is to *'darwin'*?
    - What token2 is to ***arch*** as *'win32'* is to ***platform***?
    - What token3 is to ***process*** as *'win32'* is to ***process***?
  - 多个token取平均值得到未绑定token最可能的位置V_{tgt}
  - 在候选token集合T中，选择最接近V_{tgt}的结果
    - T包含id和lit
    - 上下文token+常见token



	correct	bug
concrete	***process.platform==='darwin'***	process.platform!=='win32'
pattern	***id1.id2===lit1***	id1.id2!==lit2
target	***process.arch==='x64'***	process.arch!==???

已知token共同决定

SemSeed 算法原理

- 算法特点
  - 使用模式提取方法学习真实bug出现方式
  - 使用语义匹配方法定位可注入bug的位置
  - 模仿曾经存在过的错误，本地化后插入
  - 使用类比查询方法选择最真实的token替换
- 实际应用时的调整优化
  - 未绑定token采取k近邻，选取最可能的k个
    - 产生k个含bug程序
    - 插入bug为熵增行为，增加程序混乱度
    - 以是否能提升检测器性能衡量注入漏洞算法
  - 插入bug后自动进行代码审查，避免语法错误

Algorithm 2 Apply bug seeding pattern to candidate token sequence

Input: A candidate token sequence C, a concrete bug fix (C_{bug}, C_{corr}) and its corresponding bug seeding pattern (P_{corr}, P_{bug}), a set T of identifier and literal tokens.
Output: Tokens C_{seed} of seeded bug

$C_{seed} \leftarrow []$
for $i \leftarrow 1$ **to** $length(C_{bug})$ **do**
  if $kind(C_{bug}[i]) \notin \{Identifier, Literal\}$ **then**
    Append $C_{bug}[i]$ to C_{seed} ▷ Copy token
  else if $P_{bug}[i]$ bound to t_{bound} **then**
    Append t_{bound} to C_{seed} ▷ Use bound token
  else
    $V_{tgt} \leftarrow \varnothing$ ▷ Bind token via analogy queries
    for $t_{abstr} \in P_{corr}$ **do**
      $t_{orig} \leftarrow$ token that t_{abstr} is bound to in C_{bug}
      $t_{seed} \leftarrow$ token that t_{abstr} is bound to in C
      $v_{tgt} \leftarrow emb(t_{seed}) + emb(C_{bug}[i]) - emb(t_{orig})$
      Add v_{tgt} to V_{tgt}
    $t_? \leftarrow \arg\max_{t \in T} simil(emb(t), avg(V_{tgt}))$
    Append $t_?$ to C_{seed}

SemSeed 实验设计

- 实验设置
  - bug注入效果评估
    - RQ1：重现真实bug的有效性；RQ2：语义感知的必要性
  - 参数影响评估
    - RQ3：参数设置的影响
  - 应用性评估
    - RQ4：生成bug对检测器可用性；RQ5：与传统方法对比；RQ6：注入bug效率

- 数据集设置
  - GitHub上star最多的100个JavaScript项目，包含3600个具体错误
    - 2880个训练(guiding)用例，包含2201种错误模式，服从长尾分布
    - 720个测试(held-out)用例
- 匹配阈值$m = 0.2$，选择数量$k = 10$，候选集合$T =$ 同文件所有token T_{file} + 训练集最常见token top_{1000}

算法实验 RQ1.重现真实BUG的有效性

- 在测试集代码中使用SemSeed注入漏洞
  - 训练集中提取错误模式，向测试集正常代码注入漏洞
  - 比较注入结果与测试集实际更改是否相同
- 复现测试集bug的可验证条件：共53个可验证bug
  - 测试集中的模式出现在训练集中
  - 如果需要处理未绑定token，其必须在T中
- 实验效果
  - 16个涉及重新排列，均成功
  - 37个涉及未绑定token，6个失败
    - 代码中包含单个、多个未绑定token均成功查询
  - 漏洞注入成功率88.7%

可以注入真实漏洞，算法有效

Table 2: Examples of reproduced real-world bugs.

Correct code	Buggy code
Bug to imitate: Commit b776e2b7 of jQuery	
`var opt = speed &&` `  typeof speed === "object"`	`var opt =` `  typeof speed === "object"`
Seeded bug: Commit b94532c2 of Chart.js	
`if ( style &&` `  typeof style === 'object') {`	`if (typeof style === 'object') {`
Bug to imitate: Commit ad708ca5 of Meteor	
`catalog. complete .` `  getReleaseVersion`	`catalog. official .` `  getReleaseVersion`
Seeded bug: Commit bd74fb4c of Node.js	
`parent. stderr .on('data',` `  function() { ... });`	`parent. stdout .on('data',` `  function() { ... });`
Bug to imitate: Commit 1027871e of webpack	

`optimization: {`
`  chunkIds : "named"`
`}`
	`optimization: {`
	`  namedChunks : true`
	`}`

Seeded bug: Commit 28f346e8 of freeCodeCamp	

`db: {`
`  connectionTimeout : 15000`
`}`
	`db: {`
	`  timeout : 10000`
	`}`

算法实验 RQ2.语义感知的必要性

- 消融实验：未绑定token不使用语义感知
  - 随机种子重复10次实验
  - 随机选择T中的token作为结果
  - 漏洞注入成功率仅为30.2%
    - 仅不包含未绑定token的目标成功
- 结果分析
  - 包含未绑定token的模式占比为62%
  - 最常见10个模式中，均有未绑定token
    - 代码的修改是常见行为
    - 代码的删除、顺序调整较不常见

语义感知对生成漏洞十分重要

Table 3: Ten most frequent and five randomly selected bug seeding patterns. Unbound tokens are highlighted.

Correct	Buggy	Nb.
`id1 : lit1`	`id1 : lit2`	99
`lit1 : lit2`	`lit1 : lit3`	71
`id1.id2(lit1);`	`id1.id2( lit2 );`	40
`var id1 = lit1;`	`var id1 = lit2 ;`	33
`id1 : lit1`	`id2 : lit1`	18
`id1 = lit1`	`id1 = lit2`	18
`throw new id1(lit1);`	`throw new id1( lit2 );`	17
`id1.id2 = lit1 ;`	`id1.id2 = lit2 ;`	13
`id1(lit1) ;`	`id1( lit2 );`	13
`return lit1;`	`return lit2 ;`	11
`id1 = lit1 in id2`	`id1 = !!id2. id3`	1
`id1.id2(lit1 + id3).id4);`	`id1.id2(lit1 + id3);`	2
`id1.id2(id3[id4.id5]);`	`id1.id2(id4.id5)`	2
`var id1 = id2.id3(id4);`	`var id1 = id2.id3;`	1
`var id1 = id2.id1;`	`var id1=id2. id3 ;`	5

算法实验 RQ3.参数设置验证

- 语义匹配阈值m对结果影响
  - $m = 0$时，不考虑语义匹配，仅结构匹配
  - $m = 1$时，目标语句与C_{corr}完全相同
  - 蓝线：算法选取目标点数量占总目标点数量比例
  - 红线：RQ1中的47个bug复现成功率
    - m越高，匹配难度越高，部分潜在目标被略过
  - 红线高于蓝线，证明语义匹配筛选的有效性
- 未绑定token选择数量k，候选集合T对结果影响
  - 未绑定token的$TOP-k$近邻
  - 同函数的token集合T_{fct}，同文件的token集合T_{file}
  - T_{file} +所有文件最常见的1000个token= T_{common}

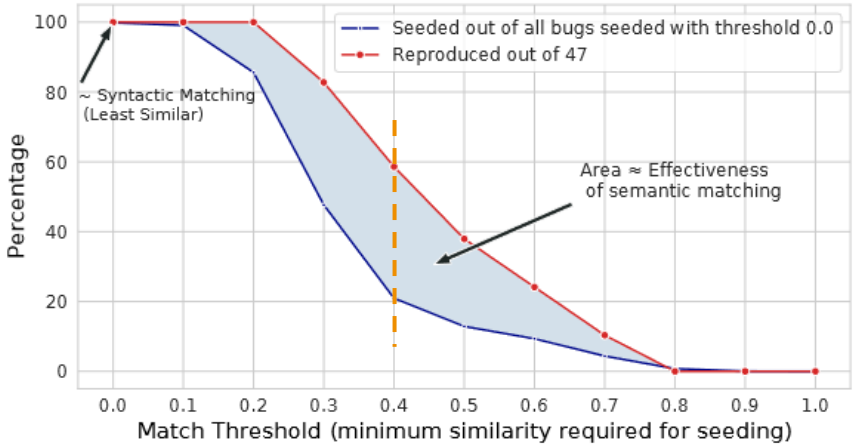


Figure 3: Influence of matching threshold m on seeded bugs.

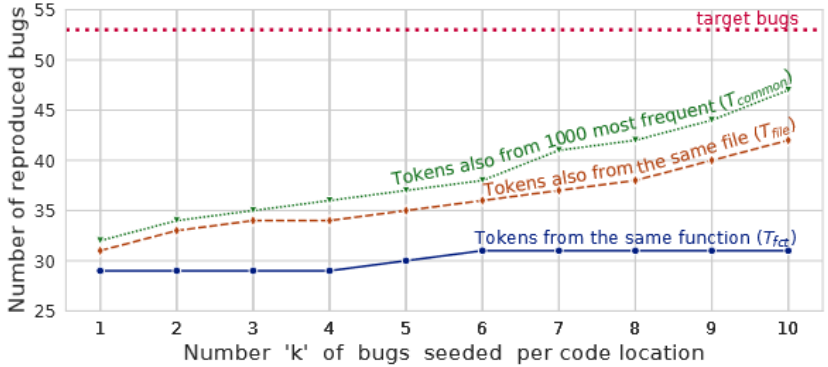
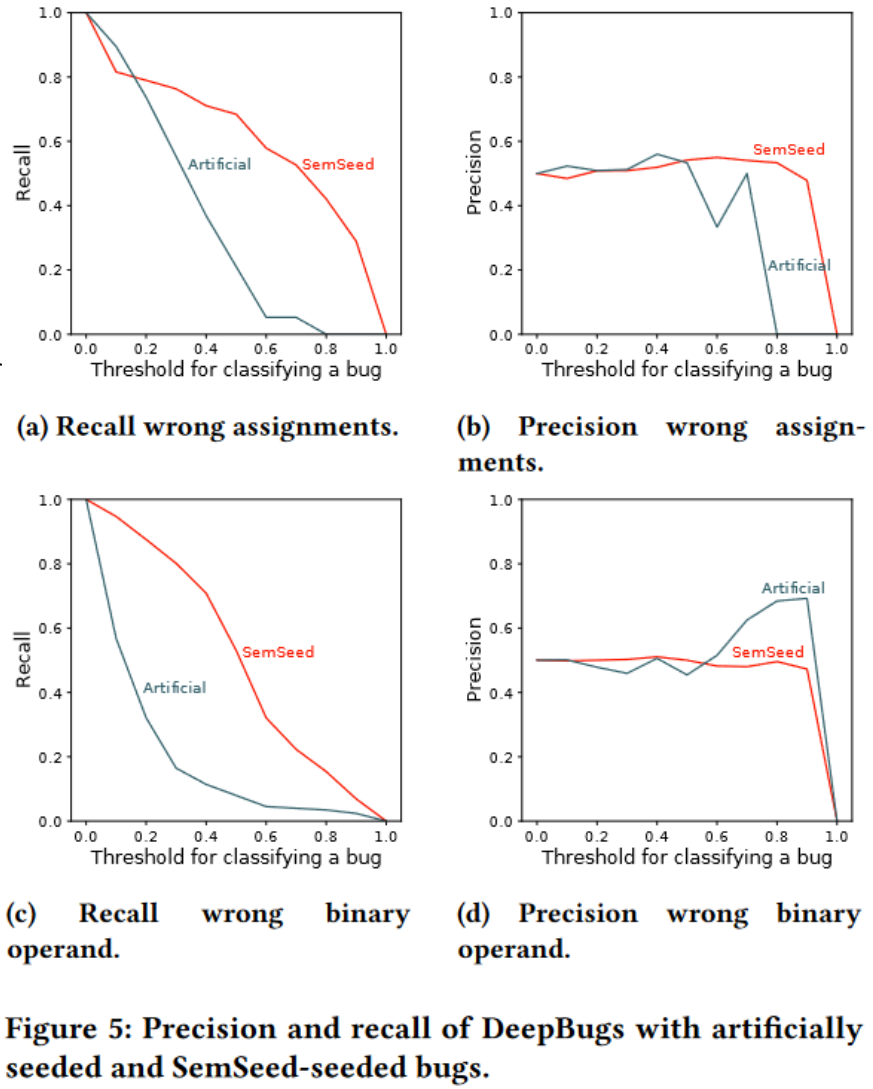


Figure 4: Reproduced real-world bugs depending on token set T and number k if bugs to seed.

算法实验 RQ4.生成bug对检测器可用性

- 生成bug作为检测算法训练数据
  - 检测算法DeepBugs（2018 CCF-C 高引）
    - 红线：SemSeed生成bug，T中类比查询token
    - 蓝线：DeepBugs默认方法，T_{file}中随机选择token
  - 实验错误模式均涉及未绑定token
    - 错误赋值(assignments)：赋值语句右侧变量错误
    - 错误二进制(binary)操作：运算量错误
  - 实验效果
    - 召回率有显著提升，更好检出漏洞
    - SemSeed生成训练集数据与真实bug偏差较小
    - 可分辨“有点错”的bug，提升检测器判断能力

算法生成的漏洞对检测器有提升效果



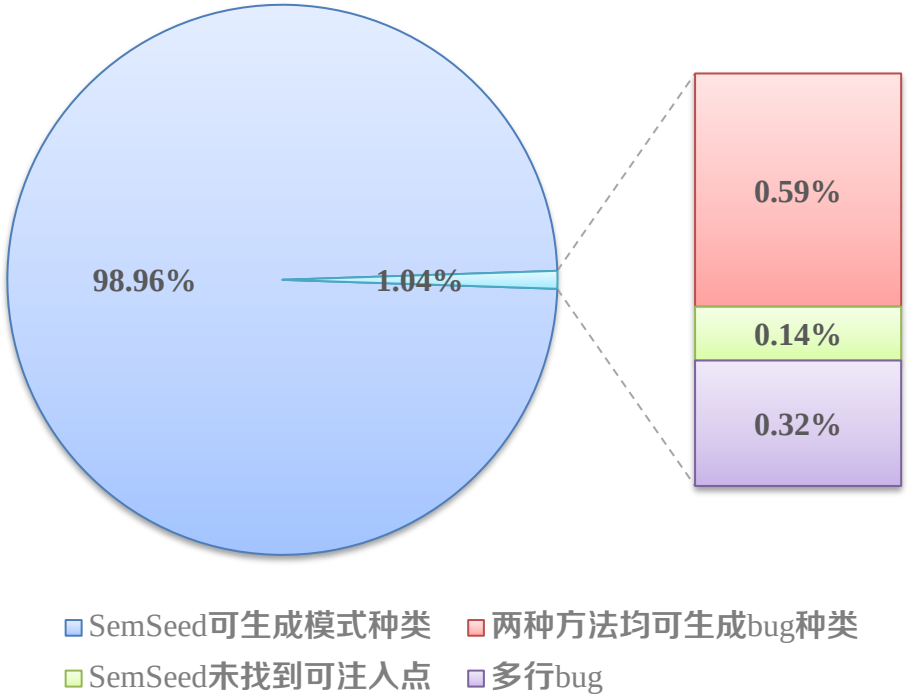
(a) Recall wrong assignments. (b) Precision wrong assignments.

(c) Recall wrong binary operand. (d) Precision wrong binary operand.

Figure 5: Precision and recall of DeepBugs with artificially seeded and SemSeed-seeded bugs.

算法实验 RQ5&RQ6

- 算法与传统方法在生成bug种类方面对比
  - 向1000个随机JavaScript程序中注入bug
  - 变异算法预设的23种变异算子生成bug代码
  - SemSeed使用2880个训练用例
  - 实验结果：SemSeed注入bug种类优于传统方法
    - SemSeed可以生成2185个种类的bug
    - 传统方法注入bug种类的56.5%可以复现
- 漏洞注入效率
  - 140分钟注入67万个错误，平均每0.01秒生成一段bug代码
  - 词类比查询未绑定token占用时间最长



算法生成的漏洞的种类、数量、速度可观

算法总结 SemSeed

- 算法优势
  - 高效、大量的生成供检测器使用的训练数据
  - 自动定位目标程序匹配位置，批量生成漏洞
  - 漏洞种类丰富，贴近真实错误
- 算法劣势
  - 无明确bug触发器，无严重性、功能性区分
  - 插入bug仅限于代码行级，无代码上下文逻辑关系、显式/隐式信息流关系考虑
- 应用领域
  - 向已有软件中注入漏洞，辅助检测算法发展
  - 自动生成CTF题目(AutoCTF)

领域发展

- 软件漏洞自动生成技术
  - 方法层面创新
    - 基于深度学习方法
  - 研究对象层面创新
    - 向二进制程序中注入漏洞
    - 向不同编程语言的代码中注入漏洞
  - 技术层面创新
    - 提取目标软件旧变更，辅助生成漏洞
    - 基于动态分析，提取代码间联系，通过改动多行实现逻辑漏洞

以攻促防，攻防相长

参考文献

[1] Patra J, Pradel M. Semantic bug seeding: a learning-based approach for creating realistic bugs[C]. Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. 2021: 906-918.

[2] Ethayarajh K, Duvenaud D, Hirst G. Towards Understanding Linear Word Analogies[C]. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. 2019: 3253-3262.

[3] Pradel M, Sen K. Deepbugs: A learning approach to name-based bug detection[J]. Proceedings of the ACM on Programming Languages, 2018, 2(OOPSLA): 1-25.

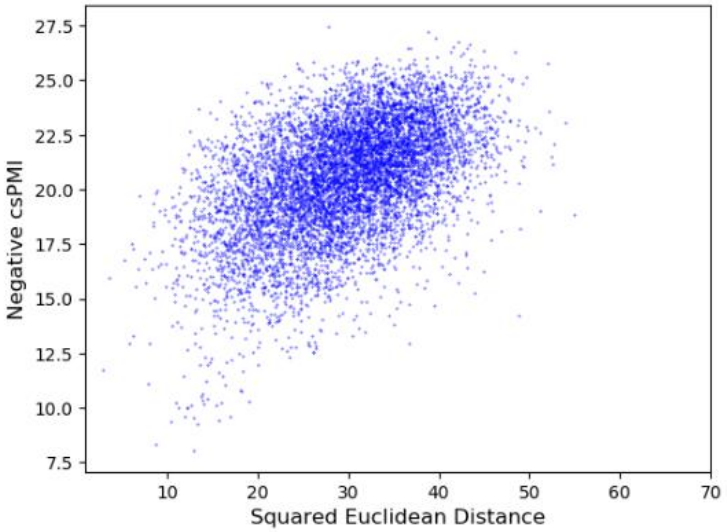
[4] Dolan-Gavitt B, Hulin P, Kirda E, et al. Lava: Large-scale automated vulnerability addition[C]. 2016 IEEE symposium on security and privacy (SP). IEEE, 2016: 110-121.

道可道，非常道。名可名，非常名。无名天地之始。有名万物之母。故常无欲以观其妙。常有欲以观其徼。此两者同出而异名，同谓之玄。玄之又玄，众妙之门。

谢谢！

附录：词类比

- $PMI(x,y) = \log \frac{p(x,y)}{p(x)p(y)}$
- 词类比主要适用于频繁出现的单词对
- 如果在没有重构错误的情况下，类比在一组单词对上完全成立，则每个单词对都具有相同的csPMI值



Analogy	Mean csPMI	Mean PMI	Median Word Pair Frequency	csPMI Variance	Accuracy
capital-world	−9.294	6.103	980.0	0.496	0.932
capital-common-countries	−9.818	4.339	3436.5	0.345	0.954
city-in-state	−10.127	4.003	4483.0	2.979	0.744
gram6-nationality-adjective	−10.691	3.733	3147.0	1.651	0.918
family	−11.163	4.111	1855.0	2.897	0.836
gram8-plural	−11.787	4.208	342.5	0.590	0.877
gram5-present-participle	−14.530	2.416	334.0	2.969	0.663
gram9-plural-verbs	−14.688	2.409	180.0	2.140	0.740
gram7-past-tense	−14.840	1.006	444.0	1.022	0.651
gram3-comparative	−15.111	1.894	194.5	1.160	0.872
gram2-opposite	−15.630	2.897	49.0	3.003	0.554
gram4-superlative	−15.632	2.015	100.5	2.693	0.757
currency	−15.900	3.025	19.0	4.008	0.092
gram1-adjective-to-adverb	−17.497	1.113	46.0	1.991	0.500