第 45 卷 第 8 期 计 算 机 工 程 2019 年 8 月
Vol. 45 No. 8 Computer Engineering August 2019

·网络空间安全专题· 文章编号：1000-3428(2019)08-0022-03 文献标志码：A 中图分类号：TP391

基于改进 PCFG 的语言解释器模糊测试

刘志昊，孙晓山，张 阳

（中国科学院软件研究所，北京 100190）

摘 要： 为在语言解释器的模糊测试中构造符合语言规范的样本，并尽可能地得出异常测试结果以便发现漏洞，采用改进的概率上下文无关语法模型控制样本的变异过程，对变异结果中的未定义变量进行修正以提高符合语言规范的样本比率。在此基础上，对语言解释器进行模糊测试，结果表明，该测试所生成样本中符合语法、语义规范的比率高达 96% 。

关键词： 模糊测试；马尔科夫模型；概率上下文无关语法；机器学习；语言解释器

中文引用格式： 刘志昊，孙晓山，张阳. 基于改进 PCFG 的语言解释器模糊测试[J]. 计算机工程，2019，45(8)：22-24，30.

英文引用格式： LIU Zhihao，SUN Xiaoshan，ZHANG Yang. Fuzzy testing of language interpreter based on improved PCFG[J]. Computer Engineering，2019，45(8)：22-24，30.

Fuzzy Testing of Language Interpreter Based on Improved PCFG

LIU Zhihao，SUN Xiaoshan，ZHANG Yang

（Institute of Software，Chinese Academy of Sciences，Beijing 100190，China）

【Abstract】 In order to construct samples that conform to language norms in fuzzy testing of language interpreter，and get abnormal test results as far as possible to find vulnerabilities，the improved Probabilistic Context Free Grammar（PCFG）model is used to control the variation process of samples，and the undefined variables in the variation results are modified to increase the ratio of samples that conform to language norms. On this basis，the language interpreter is tested by fuzzy testing. Results show that the ratio of samples generated by the test that conform to the grammatical and semantic norms is as high as 96% .

【Key words】 fuzzing testing；Markov model；Probabilistic Context Free Grammar（PCFG）；machine learning；language interpreter

DOI： 10.19678/j.issn.1000-3428.0051008

0 概述

在网络安全领域，浏览器中的漏洞通常被用来实现网络攻击。JavaScript 解释器的漏洞是浏览器漏洞中的重要组成部分。由于 JavaScript 解释器存在于浏览器的底层，较多应用（包括手机应用）都使用浏览器提供的引擎解析网页内容。因此，挖掘 JavaScript 解释器中的漏洞得到研究者的广泛关注。目前，模糊测试是发现该漏洞的主要方法[1-2]。与针对二进制文件的模糊测试 AFLGo[3]、AFLFast[4]、Vuzzer[5]相比，由于解释器代码与输入结构更加复杂，路径组合空间更大，解释器模糊测试需要性能更高的策略。

目前，语言解释器模糊测试所采用的方法主要包括随机变异、随机组合、启发引导生成等。随机变异方法的典型代表是 JsFunFuzz[6]，其采用随机改变输入样本字符的方法生成一些既满足一部分规则同时有意违反另一部分规则的新样本。JsFunFuzz 语言规则由人工编写，维护成本较高。LangFuzz[7]针对多种语言的模糊测试需求以及复杂样本的构造特点，采用语法解析器将样本解析为语法树形式，然后对语法树的节点进行随机变换，包括插入、交换等操作。文献[8]提出一种 SampleFuzz 方法，其通过机器学习生成更可能导致解释器崩溃的样本。SampleFuzz 利用大量样本对模型进行训练，使模型可以生成较符合语言规范的样本。但是 SampleFuzz 在生成过程中，并非选择概率最大的元素，其思想是低概率的序列更可能覆盖新的未测试代码路径从而发现新的缺陷。Skyfire[9]采用上下文敏感语法描述语义特征，通过大规模样本学习元素与语法规则的概率分布，选择低概率的元素与规则，从而提高发现缺陷的概率。Ifuzzer[10]通过遗传算法生成新样本，遗传算法采用基

基金项目： 国家自然科学基金(61471344)。

作者简介： 刘志昊(1993—)，男，博士研究生，主研方向为网络与系统安全；孙晓山，助理研究员；张 阳，副研究员。

收稿日期： 2018-03-29 **修回日期：** 2018-06-29 **E-mail：** lzhha2006@126.com

于代码复杂度引导的适应函数,以产生更加复杂的脚本。

JavaScript 语言具有复杂的语法规则与语义特征。由于常规脚本很难触发漏洞,需要利用异常脚本提高模糊测试的效率。从统计学的角度分析,异常脚本属于低概率脚本,可以利用脚本语言的非均匀性特征[11-13]。

脚本语言可以解析为语法树形式,而概率上下文无关语法(Probabilistic Context Free Grammar, PCFG)[14]适用于描述树形结构,且其参数训练速度较快,通常可以获得数 G 级的脚本库。因此,本文基于改进的 PCFG 模型,尽可能生成异常的脚本片段,将其输入到解释器中以进行模糊测试。

1　改进的 PCFG 模型

本文采用的模糊测试框架包括样本变异模块、执行测试模块等。样本变异模块从 Mozilla 的回归测试用例中随机选择脚本文件,采用 PCFG 提供的信息随机选择语法树的节点进行变异,并对变异结果中的未定义变量进行修正,以获得新的样本,然后观测执行结果,如果发生解释器崩溃现象,则记录结果并进行汇报。本文模糊测试系统结构如图 1 所示。

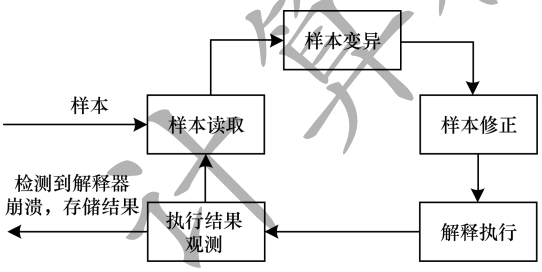

图 1　模糊测试系统结构

1.1　PCFG 模型

PCFG 模型包括的元素可以用五元组(V,N,S,R,D)进行描述。各元素定义如下:

V:一组终端元素,包括字符串、整数以及变量等。

N:一组非终端元素,包括语句、循环结构等。

S:程序的开始符号。

R:一组形式规则 $N_i \rightarrow \beta_m$,其中,$\beta_m = M_1 M_2 \cdots M_n$ 是由非终端元素和终端元素混合组成的序列。

D:概率分布描述函数。

因此,有:

$$\sum_m D(N_i \rightarrow \beta_m) = 1, \forall i$$

PCFG 较早提出用于解析自然语言的结构。由于自然语言具有歧义,因此一个句子可能存在 2 种及以上的语法树,PCFG 通常被用来判断最可能的语法树。程序语言的语法树具有唯一性,对于一个解析所得的语法树 t,本文定义其概率为:

$$P(t) = D(N_t \rightarrow \beta) \prod_1^k P(t_{M_i})$$

其中,N_t 是语法树 t 的根节点,t_{M_i} 是根为 M_i 的子树的根节点。

1.2　模型改进

规则 $N_i \rightarrow \beta_m$ 描述了 N_i 与 β_m 间的概率关系。β_m 可能包含多个符号或者元素,他们之间的依赖关系在模型中未得到反映,但是该依赖关系为脚本片段概率的一部分。为引入这种依赖关系,本文结合 JavaScript 的语法,对 PCFG 模型进行改进。图 2 所示为一段脚本的语法树描述,图 3 所示为 JavaScript 部分语法描述。

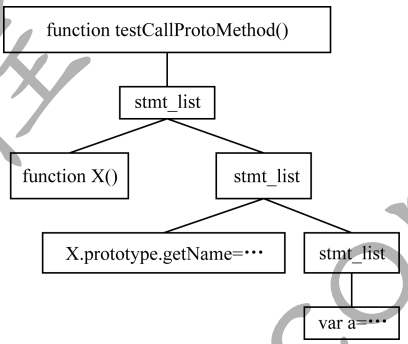

图 2　一个代码片段的语法树描述

```
sun_def :=param_list stmt_list
stmt_list:=stmt | stmt stmt_list
param_list:=ε| param | param param_list
stmt:=assign | if | while |return |…
```

图 3　JavaScript 部分语法描述

图 2 中的 stmt_list 为不定长,其对应的 PCFG 规则为 $N_i \rightarrow \beta_m$,但 β_m 需要明确定义。因此,如果采用原始 PCFG 定义程序语言,则会引入大量的规则,这将增加 PCFG 模型的参数量,导致参数学习难度加大。例如,β_m 的具体例子包括:fun _ defassignif,iffun_defassignassign,whileifwhileassign assign。

由于规则组合的数量极大,语料库无法提供所有可能的组合方案,这将导致对很多组合的频度估计偏差较大。该问题在自然语言处理中并不存在,只来源于脚本等程序语言。本文通过马尔科夫模型直接对序列进行建模,以处理上述问题。

对于 stmt_list 语法规则生成的语句序列,能够将其视为一个由马尔科夫过程生成的序列,即可以使用马尔科夫模型对其进行建模。本文假设每个语句直接受上一个语句的影响(此处的假设只是近似假设,实际程序中的语句依赖关系较复杂),对于一个 stmt_list,其概率可以表示为:

$$P(M_1 | N_t) \prod_2^k P(M_i | M_{i-1})$$

因此,整个 $N_t \rightarrow \beta_k$ 的概率可以描述为:

$$D(N_t) P(M_1 | N_t) \prod_2^k P(M_i | M_{i-1})$$

在训练过程中,只需计算相邻语句之间的转移概率即可,而相邻语句的组合在较大语料库中出现

的频率不会产生稀疏问题,对于概率计算结果的影响较小。因此,一个语法树的概率可以表示为:

$$P(t)=D(N_t)P(M_1|N_t)\prod_{2}^{k}P(M_i|M_{i-1})\prod_{1}^{k}P(t_{M_i})$$

1.3 参数训练

在 $P(t)$ 的计算公式中,需要训练的参数包括 $D(N_t)$、$P(M_1|N_t)$、$P(M_i|M_{i-1})$ 等。本文利用语料库来确定参数值。解析语料库中的脚本片段,获得语法树,对语法树进行遍历,可以得到如下的统计信息:每种语法树的数量 $\#N_i$,每个 stmt_list 中相邻语句组合的数量 $\#M_{i-1}M_i$,语法树中第 1 个元素的数量等。对于 $D(N_i)$,其频率估算为:

$$D(N_i)=\frac{\#N_i}{\#N}$$

其中,$\#N$ 是语料库中出现的语法树与子树的总数量。

$$P(M_i|M_{i-1})=\frac{\#M_i}{\#M_{i-1}M_i}$$

其中,$\#M_i$ 是 $\#N_t$ 语法树中 M_i 出现的数量。

$$P(M_i|N_i)=\frac{\#M_1}{\#N_i}$$

其中,$\#M_1$ 是 M_1 作为第 1 个元素出现的数量。

因为可以快速实现语句的数量统计,所以上述训练过程的速度较快。

1.4 变量引用修正

由于脚本生成过程中会采用新的节点进行替换,最终生成脚本中的变量可能没有定义,且该现象是导致脚本被解释器拒绝的主要原因。本文通过对语法树进行遍历,发现其中的未定义变量,然后选择脚本中最近定义的变量名替代原未定义变量,以提高脚本的准确率。本文通过栈保留变量定义域,减少由于定义域问题导致的变量未定义现象。

2 实验结果与分析

本文采用 SpiderMonkey JavaScript 引擎作为测试对象。从 Gtihub 上获取共 50 000 个 JavaScript 文件作为训练语料,将 SpiderMonkey 中用于回归测试的脚本文件作为模糊测试的初始样本。实验中所使用的原型系统由 Python 语言开发,代码规模大约为 3 000 行。其中,JavaScript 语法通过 Acorn 进行解析,结果存储于磁盘,通过读取文件的方式重新被 Python 程序加载。实验环境为 Intel i7-6700 CPU(频率 3.40 GHz),内存 8 GB,操作系统是 Ubuntu 16.04,通过 js 命令行工具进行测试。实验结果如图 4 所示,表 1 所示为不同类型的语法树出现的频度。可以看出,每种语句的分布并不均匀,具有特定的幂律分布规律[15]。其中,最常见的子树类型是成员表达式、调用表达式等。这种分布规律是采用机器学习方法改进模糊测试的统计基础。

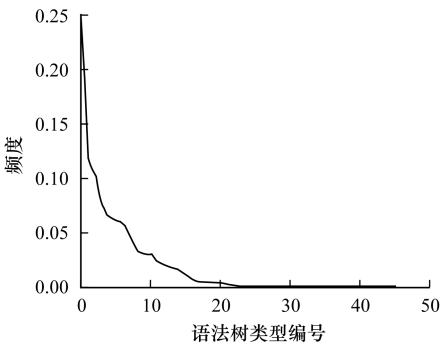

图 4 语法树类型的频度统计结果

表 1 最高频的 5 种语法结构占比情况 %

语法树/子树类型	占比
MemberExpression	25.39
CallExpression	11.56
ExpressionStatement	10.43
BlockStatement	7.43
Property	6.40

本文的模糊测试需要对原样本进行变异。在随机修正的情况下,生成样本中符合语法、语义规范的比率(即样本准确率)低于 20%(变异率为 15%),其余被解释器报告为存在语法、语义错误。因此,生成样本是否存在语法、语义问题也是重要的性能评价指标。本文模糊测试准确率结果如表 2 所示。从表 2可以看出,加入修正后的模糊测试可以使准确率提升至 96% 。

表 2 模糊测试生成样本的准确率结果

实验编号	生成样本数	准确样本数	准确率/%
1	10 000	9 622	96.22
2	10 000	9 608	96.08
3	10 000	9 613	96.13

在实际应用中,经过一周的运行,本文系统发现了一个 SpiderMonkey 漏洞,报告给 mozilla 得到确认,其编号为 bugzilla1415291。产生该漏洞的原因是 js::WasmTableObject::getImpl 中存在缓冲区溢出问题。

3 结束语

本文建立一种改进的 PCFG 模型,以对语言解释器进行模糊测试。利用大规模开源代码中存在的概率分布特征,启发模糊测试的样本生成,并通过变量修正等措施提高样本中符合语法、语义规范的比率。实验结果表明,该模糊测试系统的准确率高达 96% ,并能够发现 JavaScript 解释器中的新漏洞。将本文模型应用于 PHP 等其他脚本语言解释器的模糊测试,将是下一步的研究方向。

(下转第 30 页)

(上接第 24 页)

参考文献

[1] 李进东,王韬,吴杨,等. 基于协议分析与模糊测试的 SIP 漏洞挖掘研究[J]. 计算机工程,2016,42(8):117-122.

[2] 苏晓艳,武东英,刘龙,等. 基于 Fuzzing 的 Cisco IOS 漏洞挖掘方法[J]. 计算机工程,2012,38(16):117-120.

[3] BÖHME M, PHAM V T, NGUYEN M D, et al. Directed greybox fuzzing[C]//Proceedings of 2017 ACM SIGSAC Conference on Computer and Communications Security. New York, USA: ACM Press, 2017: 2329-2344.

[4] PHAM V T, ROYCHOUDHURY A. Coverage-based greybox fuzzing as Markov chain[C]//Proceedings of 2016 ACM SIGSAC Conference on Computer and Communications Security. New York, USA: ACM Press, 2016: 1032-1043.

[5] RAWAT S, JAIN V, KUMAR A, et al. VUzzer: application-aware evolutionary fuzzing[EB/OL]. [2018-03-10]. https://www.cs.vu.nl/~herbertb/download/papers/vuzzer_ndss17.pdf.

[6] RUDERMAN J. Introducing JsFunfuzz[EB/OL]. [2018-03-10]. http://www.squarefree.com/2007/08/02/introducing-jsfunfuzz/.

[7] HOLLER C, HERZIG K, ZELLER A. Fuzzing with code fragments[C]//Proceedings of the 21st USENIX Conference on Security Symposium. Berkeley, USA: USENIX Association, 2012: 445-458.

[8] GODEFROID P, PELEG H, SINGH R. Learn and fuzz: machine learning for input fuzzing[EB/OL]. [2018-03-11]. https://arxiv.org/pdf/1701.07232.pdf.

[9] WANG Junjie, CHEN Bihuan, WEI Lei, et al. Skyfire: data-driven seed generation for fuzzing[C]//Proceedings of 2017 IEEE Symposium on Security and Privacy. Washington D.C., USA: IEEE Press, 2017: 579-594.

[10] VEGGALAM S, RAWAT S, HALLER I, et al. IFuzzer: an evolutionary interpreter fuzzer using genetic programming[C]//Proceedings of the 21st European Symposium on Research in Computer Security. Berlin, Germany: Springer, 2016: 581-601.

[11] RAYCHEV V, BIELIK P, VECHEV M, et al. Learning programs from noisy data[C]//Proceedings of ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. New York, USA: ACM Press, 2016: 761-774.

[12] TU Zhaopeng, SU Zhengdong, DEVANBU P. On the localness of software[C]//Proceedings of ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York, USA: ACM Press, 2014: 269-280.

[13] Learning from "big code"[EB/OL]. [2018-03-10]. http://learnbigcode.github.io/datasets/.

[14] MANNING C D, HINRICH S. Foundations of statistical natural language processing[M]. Cambridge, USA: MIT Press, 1999.

[15] VIRKAR Y, CLAUSET A. Power-law distributions in binned empirical data[J]. SIAM Review, 2012, 51(4): 661-703.

编辑 吴云芳