计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(11):45−51 [doi: 10.15888/j.cnki.csa.006051]
©中国科学院软件研究所版权所有.
E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

污点导向型模糊测试的限制与性能分析①

张　岑[1], 庄　严[2], 程绍银[2]

[1](中国科学技术大学 计算机科学与技术学院, 合肥 230027)
[2](中国科学技术大学 信息安全测评中心, 合肥 230027)

摘　要: 在黑盒环境下, 污点导向型模糊测试是挖掘指定脆弱代码区域潜在漏洞的重要技术. 该技术将程序的输入当作污点值, 使用动态污点跟踪技术定位与脆弱区域相关联的部分输入. 随后的模糊测试阶段将只变异锁定的输入, 从而避免了与脆弱区域无关的大量变异. 然而, 以往的研究并未对该技术的实际使用限制、效率提升做出系统、数学的分析. 为了解决这一问题, 本文采用 14 个 CVE 漏洞实例对该技术的适用范围进行基准测试与限制分析, 将模糊测试过程抽象成几何分布以估计其效率提升与性能变化趋势. 分析表明, 在有元数据污点传播关系的漏洞挖掘中, 该技术存在限制. 同时, 实验验证了效率估计公式具有良好的参考价值.

关键词: 动态污点传播; 黑盒模糊测试; 漏洞分析; 伯努利实验; 性能分析

引用格式: 张岑,庄严,程绍银.污点导向型模糊测试的限制与性能分析.计算机系统应用,2017,26(11):45–51. http://www.c-s-a.org.cn/1003-3254/6051.html

Analysis on the Limitation and Performance of Taint-Based Directed Fuzzing

ZHANG Cen[1], ZHUANG Yan[2], CHENG Shao-Yin[2]

[1](School of Computer Science and Technology, University of Science and Technology of China, Hefei 230027, China)
[2](Information Technology Security Evaluation Center, University of Science and Technology of China, Hefei 230027, China)

Abstract: The tainted-based directed fuzzing is an important technology to find bugs towards several given suspicious vulnerable code areas in black-box scenarios. It sets program's input as initial taints, uses dynamic taint tracing to locate the regions of input related to suspicious code areas. Then it only fuzzes the located input, thus avoids a large part of testing unrelated to the suspicious areas. But the existing researches haven't analyzed its real world challenges systematically and haven't evaluated its performance enhancement mathematically. To solve this problem, this paper uses 14 CVEs as benchmark to do its limitation analysis, abstracts the fuzzing as shifted geometric distribution to get performance enhancement equation and analyzes the performance variation trend. Analyses show that the tainted-based directed fuzzing has limitations on fuzzing bugs containing metadata relation in the taint propagation. And the experiments verify that the performance enhancement equation has a good reference value.

Key words: dynamic taint propagation; black-box fuzzing; vulnerability analysis; bernoulli trials; performance analysis

1　引言

1.1　背景

黑盒模糊测试由 Miller 教授首先提出[1], 是在不获取源代码的情形下进行未知漏洞挖掘的有效方法, 也是近十多年来漏洞挖掘领域的热门研究方向. 但在给定若干可疑代码区域的应用场景下, 黑盒模糊测试对输入随机变异的特性显得过于盲目, 产生的测试例对特定代码区域来说, 不够具有针对性. Vijay Ganesh 和 Tielei Wang 等人独立同时期分别提出白、黑盒下污点导向型模糊测试[2,3], 改善了模糊测试的这一弱点. 该技

① 收稿时间: 2017-02-21; 修改时间: 2017-03-09; 采用时间: 2017-03-16

© 中国科学院软件研究所 http://www.c-s-a.org.cn

术以动态污点传播技术为基础, 在对程序执行的语句按语义设立相应的污点传播规则后, 将程序的输入设为初始污点值, 随着程序的执行传播污点. 程序执行至敏感区域时, 记录使用的污点. 这些污点所对应的原输入即为锁定的关联输入. 污点导向型模糊测试只对关联输入进行变异. 该技术排除了原输入中大量的目标代码块无关字节, 大大减少了变异空间, 使得模糊测试具有很强的针对性. 利用污点传播锁定关联输入的思想被研究人员广泛接纳并借鉴. 在模糊测试领域, 王金锭[4]、黄奕[5]等人建立目标函数集合, 根据函数参数是否被污染来评定集合内函数的不安全等级从而展开针对性模糊测试. 在符号执行领域, Wang 提出的 IntScope[6] 及 Haller 提出的 Dowser[7]针对特定漏洞类型建立检测模型, 若某代码块符合检测特征的同时又使用了被污染的数据, 再进行约束检测或启发式策略下的符号执行. 刘丙昌提出的 DSEE[8]则借鉴该思想, 通过只记录受污染指令块执行信息大幅度减少离线符号执行重放语句的数量.

1.2 研究内容

尽管污点导向型模糊测试取得了巨大的成功, 但以往的研究集中于展示该技术的优点, 或研究建立在该技术可行的范围之内. 对于该技术本身, 其实际运用是否有适用漏洞范围的限制, 如果有限制, 是否有较可行的解决方法, 且相比传统模糊测试, 究竟能带来多少效率的提升等这些问题并没有系统的分析. 为了改善这一现状, 我们基于 pin 与 BAP, 开发了离线重放式污点传播工具, 并结合 zzuf 实现了黑盒场景下污点导向型模糊测试的原型工具. 我们通过 14 个 CVE 漏洞对该技术进行实际限制的分析总结. 同时建立数学模型, 给出效率提升估计表达式及效率提升趋势分析. 实验表明, 该效率估计值有不错的参考价值.

2 重要概念

2.1 变异型模糊测试

模糊测试按是否需要预先提供样本可分为变异型模糊测试 (Mutational Fuzzing) 与生成型模糊测试 (Generation-based Fuzzing)[9]. 顾名思义, 变异型模糊测试需要使用者预先提供样本, 根据变异策略对样本进行变异来产生畸形输入.

若采取的变异策略不影响原样本的长度, 则可以将变异过程看成是针对原样本的若干比特进行了翻转. 这样的比特翻转式变异策略既能在能力上覆盖大多数已有的变异策略, 又使得模糊测试更易于建立数学模型, 是近年来模糊测试相关数学性研究的主要研究前提[10-12]. 这种模糊测试的参数配置可以用 (ρ, ζ, γ) 表示. ρ 代表目标程序, ζ 代表待变异的样本文件, γ 是变异率范围, 即输入总长度与每次翻转比特数比值的取值范围.

我们的研究基于变异型模糊测试, 同时实验与分析中均假设变异前后样本长度不变, 因此本文的结论在采取比特翻转式变异策略的变异型模糊测试中更为准确, 但对其它类型的模糊测试依然存在较高的参考价值.

2.2 污点传播

污点传播按其传播规则, 可分为基本污点传播 (Basic Tainting) 与高级污点传播 (Advanced Tainting)[13].

基本污点传播只关注操作数之间的数据依赖性. 它将用户指定的输入看作初始污染源, 当含有污点的数据被复制、作为源操作数参与算术或逻辑运算时, 污点传播给目的操作数. 当含有污点的数据被无污点的数据覆盖时 (包括 XOR 的清空), 污点被清除.

而高级污点传播考虑了指针和控制流下的传播. 它认为参与寻址的寄存器与对应地址的内容之间, 参与分支条件判断的污点与随后跳转至的基本块之间也存在依赖关系, 同样需要进行污点传播. 高级污点传播以纯数据依赖之外的维度记录了操作数之间的关系, 但它难以精确实施, 容易引起过污染 (Overtainting)[14]. 比如指针的污点传播, 一旦 EBP 寄存器被污染, 随后的所有函数栈帧内取临时变量的操作都将被污染, 引起污点爆炸.

2.3 脆弱区域

污点导向型模糊测试需要事先以人工或自动化的方法指定一些可能存在漏洞的可疑区域作为目标. 本文中使用脆弱区域指代这些目标. 黑盒场景下选择脆弱区域最简单的做法是建立危险库函数列表, 通过静态二进制分析工具找出调用了危险库函数的汇编语句.

3 基于漏洞实例的限制分析

3.1 问题归约与研究思路

污点导向型模糊测试实际分为利用污点技术锁定关联输入与针对关联输入进行模糊测试两阶段. 而能力限制分析问题即为分析该技术在何种情况下能够挖掘出漏洞. 假设对于某漏洞的每个触发点, 从该点触发漏洞存在一个最小触发集 (触发漏洞最少应翻转的比

© 中国科学院软件研究所 http://www.c-s-a.org.cn

特集合), 那么污点导向型模糊测试能挖掘出该漏洞的充分必要条件是锁定的关联输入是该漏洞某最小触发集的超集.

通过上一段的分析, 我们可以将污点导向型模糊测试限制分析问题研究思路确定如下: 我们集中研究该技术的锁定关联输入阶段. 首先我们收集若干漏洞实例建立测试漏洞集, 以漏洞的任一最小触发集能否被完全锁定为衡量标准来度量该技术的能力. 若存在该技术无法挖掘出的漏洞, 则分析、总结限制存在的原因, 并讨论缓解措施.

3.2 漏洞挖掘能力评估

本文分析 Linux 著名 PDF 渲染库 poppler 近十年的 CVE 以建立测试漏洞集. 在近十年的 CVE 漏洞中, 能够从 NVD、Mitre、Bugzilla、SecurityFocus 等网站提供的信息中找到足够多细节重现 POC 的有 14 个. 我们分析这些漏洞的根本原因 (Root Cause), 将根源所在的整个函数设立为脆弱区域, 手工制作能测试到该函数的样本文件并找出所有最小触发集. 这 14 个漏洞在表 1 中第一列列出, 作为研究的测试漏洞集.

表 1 测试漏洞集评估结果

CVE编号	漏洞类型	是否完整收集	欠污染类型
20082950	指针未初始化	否	①
20090755	设计错误	否	①③
20093603	整数溢出	是	
20093604	整数溢出	是	
20093606	整数溢出	是	
20093608	整数溢出	是	
20093609	整数溢出	是	
20093938	缓冲区溢出	是	
20094035	缓冲区溢出	否	
20103704	不安全库函数使用	是	
20134472	条件竞争	否	④
20134473	栈溢出	是	
20134474	格式化字符串	是	
20158868	数组越界写	是	①③

锁定关联输入阶段的具体配置分成污点传播规则与污点收集规则的配置两部分. 我们使用基本污点传播规则, 并将污点收集规则设置为收集从进入脆弱区域的第一条指令起, 直至退出脆弱区域时执行的所有汇编语句中出现过的但不是即将被清除的污点. 这些被收集的污点对应的输入被视为关联输入.

在上述准备下, 我们使用测试漏洞集得出表 1 中结果. 14 个漏洞有 4 个锁定的关联输入不是最小触发集的超集, 这说明该技术在具体应用中存在能力限制.

3.3 限制原因与缓解措施分析

针对不能挖掘的漏洞实例, 我们结合污点传播系统细粒度调试、对照 poppler 源代码与汇编代码等多种分析方法, 认为污点传播与污点收集两阶段均存在限制.

3.3.1 污点传播的限制

污点传播阶段存在限制的根本原因是基本污点传播无法传递操作数之间存在的元数据关系, 而高级污点传播在现有技术下又难以有效实施. 具体来说, 在传播过程中, 若只采用基本污点传播规则, 传播过程会存在欠污染, 即有些应该被污染的变量没有被污染, 从而导致污点导向型模糊测试无法锁定到足够的关联输入, 而高级污点传播难以精准实施甚至极易过污染的特性可能会使得最终锁定的关联输入远多于真实关联输入以至失去锁定的意义. 下文将首先展示元数据关系模型, 并结合模型详细讨论基本污点传播的不足, 接着讨论高级污点传播精确实施的可行性.

图 1 展示了程序执行时的污点传播模型. 正方形代表了变量, 如寄存器, 某内存地址的内容. 含有阴影的正方形表示该变量被污染. 圆形表示超出污点传播讨论范围的某些数据, 比如某条语句的执行时间. 椭圆代表了脆弱区域. 需要注意的是图中原始的输入污点有 Ta、Tb、Tc, 在基本污点传播规则下, 我们只收集到了 Tb.

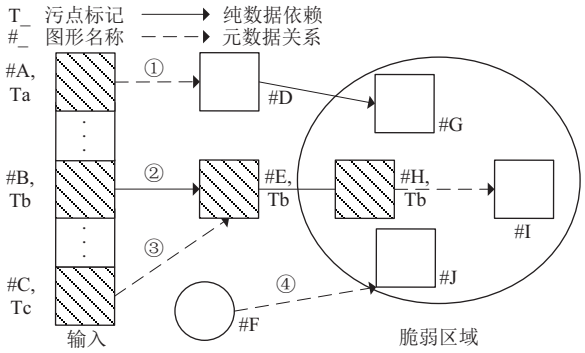

图 1 污点传播模型图

图中虚线①③④分别代表了本文总结出的三种操作数之间可能存在的元数据关系. 我们依次举例说明并展示该种元数据关系为何会在基本污点传播过程中丢失. ①代表目的操作数是源操作数的元数据. 假设#A 是某整数, #D 是 bool 值, 记录了#A 是否是质数. 这在程序中可能是通过类似 if(is_prime(#A)){…}的语句实现#D 的赋值, 但#D 不会带有#A 的污点信息. ③代表源操作数是目的操作数的元数据. 比如在 PDF 文件中, string 对象被表示为“(This is a string)”, 其中内容两

© 中国科学院软件研究所 http://www.c-s-a.org.cn

边的括号标识着其存储 string 类型的内容. 假设#B 是字符串内容, #C 是括号, 那么在解析过程中, #C 的值决定了程序以字符串解析代码继续运行, 但同样的, 其污点标记 Tc 并不会传递给#E. ④同样代表源操作数是目的操作数的元数据, 但此时源操作数的数据超出了污点传播的模型涉及范围. 比如#F 代表了某代码块的执行时间, 该元数据对于条件竞争漏洞至关重要, 但现有的污点传播模型难以处理这样的关系.

理论上来说, 高级污点传播可以正确处理部分元数据关系的污点传播——①③. 但难点在于如何在二进制情形下正确区分某操作是否建立元数据关系. 有一些研究讨论了高级污点传播的实现, Slowinska 等人通过键盘记录器污点传播的实验验证了指针传播极易过污染的特点, 并对该技术的精确实施持悲观态度[13]. 而随后, Bao Tao 和 Kang 分别给出了白/黑盒环境下用强控制依赖条件 (Strict Control Dependency) 选择性进行污点传播的缓解措施[15,16]. 具体到我们的研究, Kang 提出的 DTA++是目前唯一可能的缓解措施但其实际性能开销过大. DTA++借助 SMT 求解器找出具有强控制依赖的分支, 认为这些分支存在欠污染 (Undertainting), 并专门生成污点传播规则. 但 DTA++的工作模式是首先人为指定污点, 随后分析这些污点是否存在欠污染, 若是则生成补充性规则. 观察图 1, 我们并不能事先确定哪些输入是 Ta 与 Tc. 而假若我们有数十个样本, 每个文件的大小为若干 KB, 逐一验证每一个污点是否存在欠污染将为模糊测试额外附加很大的性能开销.

综上, 当前污点导向型模糊测试更实际可行的污点传播策略是基本污点传播, 但该传播规则存在较大的欠污染问题, 现有的技术难以很好的处理污点导向型模糊测试在污点传播阶段的限制.

3.3.2 污点收集的限制

本文在研究中将脆弱区域假定为脆弱点所在函数, 并收集整个函数执行中涉及的污点. 该策略的依据在于函数是被设计来完成某一项特殊的功能, 其逻辑是完整的. 因此整个函数的使用污点集显然比某几条敏感语句的使用污点集更可能是最小触发集的超集. 这一收集策略对测试集中除了 CVE-2009-4035 之外的漏洞都成立 (同时排除超出范围的条件竞争漏洞). 该漏洞所在函数 parse 用于读取 PDF 内嵌字体文件头部信息, 因此对 parse 函数进行污点收集, 关联输入是字体文件头部. 但触发该漏洞需要字体文件除了头部以外的剩余部分都是‘\t’, ‘\n’等无意义的分隔符, 即字体文件被截断. 因此对于正常格式的样本文件, 我们无法收集到足够的关联输入. 这一问题的根本原因在于程序单次执行所获取的信息是有限的, 脆弱区域更全面的信息需要辅以静态分析或综合多次有价值的执行信息. 因此, 针对与本段例子类似的测试目标, 以单次执行信息为指导收集关联输入的污点导向型模糊测试的实际效果依赖于样本的“好坏”.

4 基于伯努利概型的性能分析

4.1 性能分析模型的建立

4.1.1 伯努利概型定义

伯努利概型是 n 重伯努利试验 (Bernoulli Trail) 的简称, 符合伯努利概型需满足如下四点:

(1) 在其他条件不变的前提下重复地做一种试验;

(2) 各次重复试验的结果互相独立;

(3) 每次试验的结果只有某事件 (称为事件 A) 发生与不发生两种;

(4) 每次实验中事件 A 发生的概率 p 均相等.

4.1.2 问题抽象与建模

为了估计污点导向型模糊测试相对于传统模糊测试的性能提升, 我们做如下的抽象、假设与建模.

我们在比特翻转式变异策略的变异型模糊测试的范围内研究污点导向型模糊测试. 如 2.1 节所述, 此时模糊测试的参数配置为 (ρ, ζ, γ). 假设目标脆弱区域确实含有一个漏洞, 且我们拥有能够通过翻转比特变异方式触发该漏洞的样本 ζ. 我们将变异一次并测试的完整过程叫做一轮模糊测试, 那么一轮的结果有触发和不触发漏洞两种, 且每轮模糊测试可以看成是互相独立的, 每一轮触发漏洞的概率也相等. 结合上文定义, n 轮模糊测试满足伯努利概型, 事件 A 发生意味着触发漏洞, A 发生的概率为 p.

进一步的, 我们再将触发漏洞定义成被翻转的比特集是该漏洞某最小触发集的超集. 假设对于目标脆弱区域, 样本中有 x_t 个独立的触发点, 而我们收集到的关联数据覆盖了 x_f 个. 同时, 假设每一个触发点的最小触发集大小均为 $h_{\min}$, 即, 给定的样本与其任一最小化后的触发文件的汉明距离为 $h_{\min}$. 同时, 我们再将变异率范围 γ 设为固定值而不是范围, 设每次翻转 $h_{\min}+h_a$ 比特. 文件 ζ 的比特长度为 l_t, 锁定的关联数据的比特长度为 l_f.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

4.2 效率提升分析

对于传统模糊测试, 用 p_t 代表 p, 对于污点导向型, 用 p_f 代表 p, 同时由于 γ 在实际测试中通常很小, 触发点总数 x_t 相对于 l_t 与 l_f 也很小, 我们忽略一次变异中覆盖了多个最小触发集时的重复计数情况, 则可以得出如下公式:

$$p_t=\frac{x_t\cdot\binom{l_t-h_{min}}{h_a}}{\binom{l_t}{h_{min}+h_a}},\quad p_f=\frac{x_f\cdot\binom{l_f-h_{min}}{h_a}}{\binom{l_f}{h_{min}+h_a}} \tag{1}$$

定义 X 为我们运行测试直至第一次触发脆弱区域漏洞所需运行轮数, X 服从几何分布. 设 $E[X]$为 X 的期望值, 则 $E[X]=1/p_t$. 类似的, 传统测试使用 X_t 表示 X, 污点导向型使用 X_f 来表示, 用 r 表示两者期望值之比即效率提升. 我们得到:

$$E[X_t]=\frac{1}{p_t},\quad E[X_f]=\frac{1}{p_f},\quad r=\frac{E[X_t]}{E[X_f]}=\frac{p_f}{p_t} \tag{2}$$

实际情形下, 样本文件与关联输入的比特长度之差远大于最小触发集的大小, 即$l_t-l_f\geq h_{\min}+1$, 此外, 我们还有$l_t\geq l_f>0$及$h_{\min}\geq 1$, 结合上述条件, 代入 p_t 与 p_f, 得:

$$\begin{aligned}r&=\frac{x_f\binom{l_f-h_{\min}}{h_a}}{\binom{l_f}{h_{\min}+h_a}}\frac{\binom{l_t}{h_{\min}+h_a}}{x_t\binom{l_t-h_{\min}}{h_a}}=\frac{x_f l_t!\left(l_f-h_{\min}\right)!}{x_t l_f!\left(l_t-h_{\min}\right)!}\\&=\frac{x_f}{x_t}\frac{l_t}{\left(l_t-h_{\min}\right)}\frac{\left(l_t-1\right)}{\left(l_t-h_{\min}-1\right)}\cdots\frac{\left(l_f+1\right)}{\left(l_f-h_{\min}+1\right)}\\&=\frac{x_f}{x_t}\frac{l_t}{l_f}\frac{\left(l_t-1\right)}{\left(l_f-1\right)}\cdots\frac{\left(l_t-h_{\min}+1\right)}{\left(l_f-h_{\min}+1\right)}\end{aligned} \tag{3}$$

由此我们得到了效率提升的估计表达式. 以测试集中的 CVE-2010-3704 为例, 我们有 $h_{\min}=1$, l_t=13339×8=106712, l_f=1510×8=12080, x_f/x_t=1, 则:

$$r=\frac{106712}{12080}\approx 8.834 \tag{4}$$

在实验中同配置下, 我们测得的真实值为 9.887, 与估计值比较接近. 需要注意的是在我们的假设中有三条并不是一直成立——覆盖了最小触发集的变异都视为触发漏洞, 所有的最小触发集大小都为 $h_{\min}$, 计算 p 时多次计数同时覆盖多个触发集的情形. 后两条假设在漏洞触发点很少或为 1 个时都是成立的. 这表明估计值在漏洞触发点较少时更为准确.

我们还可进一步化简 r 以总结效率变化趋势:

$$r\geq\frac{x_f}{x_t}\frac{l_t}{l_f}\frac{l_t}{l_f}\cdots\frac{l_t}{l_f}=\frac{x_f}{x_t}\left(\frac{l_t}{l_f}\right)^{h_{\min}} \tag{5}$$

r 可以看成指数型表达式 $k\cdot a^b$. 我们得出如下结论: 污点传播型模糊测试抓取关联数据的能力越强, k 越大, 效率比随 k 线性增长; 样本文件与关联输入大小之比越大, a 越大, 效率比随 a 幂增长; 漏洞越难以触发, $h_{\min}$ 越大, b 越大, 效率比随之指数增长.

5 效率验证实验

5.1 实验设计

我们搭建了分布式模糊测试平台用于验证第 4 节中的效率估计. 对被选中测试的每个 CVE, 手动设置污点导向型及传统型模糊测试的参数配置, 包括样本文件、变异率范围、脆弱区域、关联输入等. 我们修改了 zzuf 作为实验的变异型模糊测试器, 将软件崩溃视为触发漏洞的形式, 并编写脚本识别崩溃是否来自脆弱区域. 每个 CVE 对应的脆弱区域的实验流程见图 2. 由于模糊测试具有很大的随机性, 实验对每一个 CVE 进行 100 组测试, 并使用 100 组的平均轮数与平均时间进行传统型与污点导向型的效率比较. 同时, 我们手工调整了样本文件 ζ 及变异率范围 γ, 以让模糊测试较容易发现漏洞——如果样本文件变异触发漏洞的难度过高, 做 100 组传统型模糊测试的耗时将是难以接受的.

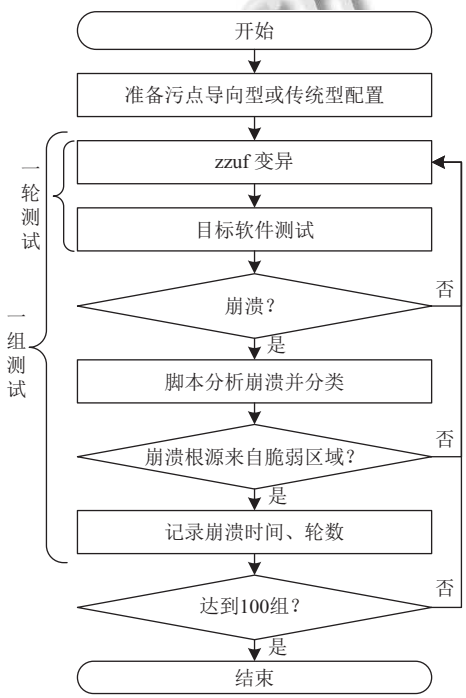

图 2 单脆弱区域模糊测试实验流程图

© 中国科学院软件研究所 http://www.c-s-a.org.cn

测试漏洞集中共包含了 9 类漏洞, 但我们从适合验证的 6 类漏洞中每类各选择一个进行验证. 不适合的漏洞的原因有——CVE-2013-4472 属于条件竞争漏洞, 超出了常规污点传播的范畴, CVE-2013-4473/4474 属于命令行参数的漏洞而非文件输入格式, 输入本身大小就非常小, 锁定的关联数据与其原输入几乎一样, 对比效果不够明显.

5.2 结果与分析

表 2 给出了实验参数和结果. I 代表一组测试平均轮数, T 代表一组测试平均时间, 剩余的列名含义则与第 4 节数学分析中对应符号含义一致. 结果含有含有 x 的行表示该样本文件在实际测试中对于指定漏洞有很多难以统计的触发方式, 因此使用 x 代替, 且很明显的可以看出, 这些样本文件的实际效率提升相比其他漏洞是有限的. 实验数据中, 效率估计值与真实值大致相同, 表明效率估计是有效且正确的, 具有良好的参考价值.

表 2 效率验证实验参数及结果

CVE编号	γ_{trad}	γ_{taint}	$h_{\min}$	x_f/x_t	I_{trad}	I_{taint}	T_{trad}	T_{taint}	r_{math}	r_{real}
20082950	0.00000 0.00017	0.00000 0.00290	1	0.6	11,014	808	928 s	78 s	10.2	13.6
20090755	0.00000 0.00084	0.00000 0.01340	1	x	439	249	68 s	43 s	(16.0*x)	1.8
20093604	0.00009 0.00151	0.00560 0.09000	1	1	151,310	1,872	62,319 s	982 s	59.3	80.8
20094035	0.00000 0.00073	0.00000 0.00270	1	1	5,684	1,354	684 s	193 s	3.7	4.2
20103704	0.00000 0.00380	0.00000 0.03310	1	1	25,232	2,552	2,354 s	233 s	8.8	9.9
20158868	0.00000 0.00038	0.00000 0.00390	1	x	19,351	6,614	1,602 s	507 s	(10.2*x)	2.9

6 结语

污点导向型模糊测试是提高模糊测试针对性的重要方法. 我们通过漏洞实例的分析验证, 使用元数据关系对污点传播过程建模, 分析了污点导向型模糊测试的限制及使用现有技术改善的难点. 同时我们通过建立数学模型, 给出了该技术的效率估计式、影响效率提升的关键参数. 实验表明该效率估计有良好的参考价值. 本文是对污点导向型模糊测试技术的系统性分析, 是该技术在定量与定性分析方面的良好补充.

参考文献

1 Miller BP, Fredriksen L, So B. An empirical study of the reliability of UNIX utilities. Communications of the ACM, 1990, 33(12): 32–44. [doi: 10.1145/96267.96279]

2 Ganesh V, Leek T, Rinard M. Taint-based directed whitebox fuzzing. Proc. of the IEEE 31st International Conference on Software Engineering. Vancouver, Canada. 2009. 474–484.

3 Wang TL, Wei T, Gu GF, *et al*. TaintScope: A checksum-aware directed fuzzing tool for automatic software vulnerability detection. Proc. of 2010 IEEE Symposium on Security and Privacy. Oakland, CA, USA. 2010. 497–512.

4 王金锭. 二进制程序漏洞挖掘技术的研究与工具实现[硕士学位论文]. 合肥: 中国科学技术大学, 2011.

5 黄奕. 基于模糊测试的软件安全漏洞发掘技术研究[硕士学位论文]. 合肥: 中国科学技术大学, 2010.

6 Wang TL, Wei T, Lin ZQ, *et al*. IntScope: automatically detecting integer overflow vulnerability in X86 binary using symbolic execution. Proc. of Network and Distributed System Security Symposium. San Diego, California, USA. 2009.

7 Haller I, Slowinska A, Neugschwandtner M, *et al*. Dowsing for overflows: A guided fuzzer to find buffer boundary violations. Proc. of the 22nd USENIX Conference on Security. Washington, DC, USA. 2013. 49–64.

8 刘丙昌. 模糊测试应用于二进制程序的漏洞发现方法研究[硕士学位论文]. 厦门: 厦门大学, 2014.

9 Miller C, Peterson ZNJ. Analysis of mutation and generation-based fuzzing. Technology Report. Independent Security Evaluators, 2007.

10 Woo M, Cha SK, Gottlieb S, *et al*. Scheduling black-box mutational fuzzing. Proc. of the 2013 ACM SIGSAC Conference on Computer & Communications Security. Berlin, Germany. 2013. 511–522.

11 Cha SK, Woo M, Brumley D. Program-adaptive mutational fuzzing. Proc. of 2015 IEEE Symposium on Security and Privacy. San Jose, CA, USA. 2015. 725–741.

© 中国科学院软件研究所 http://www.c-s-a.org.cn

12 Householder AD. Well there's your problem: Isolating the crash-inducing bits in a fuzzed file. Technical Note CMU/SEI-2012-TN-012. Software Engineering Institute, 2012.

13 Slowinska A, Bos H. Pointless tainting?: Evaluating the practicality of pointer tainting. Proc. of the 4th ACM European Conference on Computer Systems. Nuremberg, Germany. 2009. 61–74.

14 Schwartz EJ, Avgerinos T, Brumley D. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). Proc. of 2010 IEEE Symposium on Security and Privacy. Oakland, CA, USA. 2010. 317–331.

15 Bao T, Zheng YH, Lin ZQ, *et al*. Strict control dependence and its effect on dynamic information flow analyses. Proc. of the 19th International Symposium on Software Testing and Analysis. Trento, Italy. 2010. 13–24.

16 Kang MG, Mccamant S, Poosankam P, *et al*. DTA++: Dynamic taint analysis with targeted control-flow propagation. Proc. of the Network and Distributed System Security Symposium. San Diego, California, USA. 2011.

www.c-s-a.org.cn

www.c-s-a.org.cn

© 中国科学院软件研究所 http://www.c-s-a.org.cn