# 基于适应度和输入约束模型的内核驱动漏洞挖掘 *

佘庚达，付　才†，岑泽威，吕建强

(华中科技大学 分布式系统安全湖北省重点实验室 湖北省大数据安全工程技术研究中心 网络空间安全学院, 武汉 430074)

**摘　要：**针对驱动程序在运行过程中难以监控和输入复杂的问题，提出并实现基于适应度和输入约束模型的驱动程序模糊测试工具 DrgenFuzzer。该工具利用内核跟踪技术结合二进制程序的静态分析实现驱动运行的信息监控；分析驱动接口参数，设计了样本约束的方案；提出了新型适应度计算方案和交叉变异方案。实验证明：与常用的内核模糊测试工具对比，该工具经过输入约束模型之后生成的样本测试成功率达到了其他工具的 10 倍以上，生成的样本质量更高。该工具对驱动程序进行模糊测试，挖掘到 i2c 驱动中的空指针引用漏洞。DrgenFuzzer 能有效引导和规范样本生成，提高了样本测试成功率和运行效率，增强了漏洞挖掘能力。

**关键词：**模糊测试；遗传算法；适应度；输入约束模型

**中图分类号：**TP311　　doi: 10.19734/j.issn.1001-3695.2022.11.0772

# Kernel driver vulnerability mining based on fitness and input constraint model

She Gengda, Fu Cai†, Cen Zewei, Lyu Jianqiang

(*Hubei Key Laboratory of Distributed System Security*, *Hubei Engineering Research Center on Big Data Security*, *School of Cyber Science & Engineering*, *Huazhong University of Science & Technology*, *Wuhan* 430074, *China*)

**Abstract:** In order to solve the problem that the driver is difficult to monitor and input complex during the running process, this paper proposes and implements a driver fuzzing tool DrgenFuzzer based on the fitness and input constraint model. The tool utilizes the kernel tracing technology combined with the static analysis of binary programs to realize the information monitoring of the driver operation; analyzes the driver interface parameters, designs a sample constraint scheme; proposes a new fitness calculation scheme and a crossover mutation scheme. Experiments show that compared with the commonly used kernel fuzzing tools, the test success rate of samples generated by this tool after the input constraint model is more than 10 times that of other tools, and the quality of the generated samples is higher. This tool performs a fuzzy test on the driver and exploits the null pointer reference vulnerability in the i2c driver. DrgenFuzzer can effectively guide and standardize sample generation, improve the success rate and operating efficiency of sample testing, and enhance the vulnerability mining ability.

**Key words:** fuzzing; genetic algorithm; fitness; input constraint model

## 0　引言

驱动程序作为操作系统内核与外界硬件设备的通信媒介[1]，运行在内核态，因此驱动程序的安全漏洞具有强大的破坏力。从漏洞的位置看，驱动程序是大部分内核漏洞的来源[2]。近年来，研究人员越来越重视驱动程序的安全问题[3]，很多相关漏洞被公开。因此，如何高效准确的挖掘驱动程序漏洞对操作系统安全具有重要意义。

模糊测试是非常有效的漏洞挖掘技术[4]。近年来，工业界和学术界都投入大量资源来研究内核模糊测试，已有相关工作开发或改造出优秀的内核模糊测试工具[5]，如 HFL，Syzkaller，PeriScope。其中 HFL 将静态分析和符号执行结合来进行混合模糊测试[6]，相比其他工具提升了性能和代码覆盖率。Syzkaller 是目前最先进的内核模糊器[7]，其借助虚拟化技术实现模糊测试，具有通用性强、应用范围广的优点。PeriScope 利用外部设备传输数据来进行内核模糊测试[8]，在驱动与内核之间存在监视器监视数据流，该工具不需要虚拟化技术。

以上工具都存在共同的不足，即面对驱动程序时测试样本成功率不高，生成的样本在执行过程中没有通过驱动中的参数有效性检查。根本原因是 ioctl 调用的参数不稳定，不同功能的驱动设备参数类型的变化较大，导致测试用例构造的效率和质量明显下降。因此，研究针对驱动程序的模糊测试具有重要现实意义。针对参数不稳定问题，设计了输入约束模型，通过模型有效规范参数类型，保证了参数的稳定性。同时引入了遗传算法和适应度引导测试样本的生成，保障了驱动程序模糊测试效率。

基于上述现状和思路，本文提出了基于适应度和输入约束模型的模糊测试方法，通过寻找函数中的理想基本块来计算适应度，适应度函数的设计综合考虑其后继关系和影响层数，通过输入约束模型约束输入参数格式，不断变异生成高质量测试样本来进行漏洞挖掘。

**收稿日期：**2022-11-12；**修回日期：**2023-02-01　　**基金项目：**国家自然科学基金资助项目(62072200，6217071437)

**作者简介：**佘庚达(1999-)，男，湖南长沙人，硕士研究生，主要研究方向为软件安全、漏洞挖掘等；付才(1976-)，男(通信作者)，湖北通城人，教授，系主任，博士，主要研究方向为漏洞分析、恶意代码分析(fucai@hust.edu.cn)；岑泽威(1997-)，男，浙江慈溪人，硕士，主要研究方向为信息安全、漏洞挖掘等；吕建强(1981-)，男，湖北黄冈人，博士研究生，主要研究方向为软件安全、漏洞挖掘和利用等.

## 1　相关理论与技术

### 1.1　eBPF 和 Kprobes

eBPF 技术可以内核监控、跟踪和调试。该技术是由伯克利包过滤(Berkerley Packet Filter，BPF)优化而来。BPF 主要功能是抓取和过滤特定规则的网络包，来避免从内核到用户空间无用的数据包复制行为。Alexei Starovoitov 在原 BPF 基础上，增加了性能分析、系统追踪等新功能，增强了其性能[9]，命名为 eBPF。对于内核信息的监测和追踪，eBPF 技术提供了 Kprobes 方案。

Kprobes 是一种跟踪内核函数执行状态的调试工具。通过内核函数符号表，Kprobes 可以在任意一个导出函数的入口位置插入探测点进行监控[10]。如果需要粒度比函数级更小的探测点，可以通过加上合适的偏移量来进行探测，所以理论上 Kprobes 可以定位到内核里几乎所有的地址空间。Kprobes 生成监测模块并将其加载到内核，同时设置探针到目的地址，设置探针不会破坏内核结构和原有的任何功能。当内核执行到探针位置时，会调用监测模块中的回调函数。探针位置可由函数或具体地址表示。在不需要监测内核时，用户可以移除这些探针，不会对原驱动运行产生任何影响。

Kprobes 提供了 kprobe、jprobe 和 kretprobe 三种不同类型的探针。其中 kprobe 是该内核调试技术最基础的探测手段，其余两个都基于 kprobe 实现，都只是在 kprobe 的基础上进行优化封装。在实际使用过程中，用户通过设置内核函数名、相对偏移量或者直接设定具体的地址来在对应的位置设置探针。

### 1.2　模糊测试技术

模糊测试技术简单描述就是模拟用户输入的过程，其核心思想在短时间内产生大量的随机样本输入到目标程序中，以触发内存中可能存在的漏洞[11]。按生成数据的方式划分，主流的模糊测试技术有基于生成的模糊测试和基于变异的模糊测试[12]。

基于生成的模糊测试技术通过模型控制生成样本，生成的输入满足目标程序的格式需求，所以需要了解程序。同时在程序逻辑漏洞挖掘上，这类技术表现不好。因为输入很规范，一些非法输入造成的漏洞被忽略掉了。

基于变异的模糊测试技术是灰盒模糊测试技术。这项技术在工业界和学术界内很常用。AFL 就是基于变异的模糊测试技术。这类技术通常会计算样本的变异价值，在变异生成新样本时，会对所有的样本文件计算适应度，从中选择一个适应度高的的样本来进行变异。适应度高意味着变异潜力高。其中潜力通常指的是产生新的路径和产生新的覆盖的能力。

这两个模糊测试技术都是基于目标程序对输入验证的不完整性假设[13]，采用随机的方法，所以输入状态空间往往很大，又因为其随机特性，使得模糊测试效果很大程度依赖于生成的测试用例数量。

## 2　模糊测试方案

本文的模糊测试方法是基于驱动程序特征。传统的模糊测试方法不能很好的监视内核驱动的运行状况，生成的测试样本有效率较低，存在大量的无效样本，导致模糊测试的效率低。

为了更好的监控内核驱动程序的运行状态，同时保证样本的有效率高，本文提出了新型适应度并引入到模糊测试中，用于样本的生成和变异。

### 2.1　内核监控

eBPF 中的技术 Kprobes 几乎可以在内核任何位置加入插桩代码，但需要提供插桩点具体内存地址。因此内核运行监控的问题转换为插桩点的具体地址获取。内核导出函数的地址 $Funadd_i$ 可以通过导出符号表(linux 系统文件/proc/kallsyms)来获取。

驱动程序中函数由若干个基本块组成，覆盖率统计粒度以基本块为单位，所以对每个基本块都要设置探测点。这样就需要获取每个基本块的首地址用于打点，获取首地址采取相对偏移量计算的方式。

对于相对偏移量 $Offset_i$，IDA Pro 这个工具能够帮助研究人员自动化的完成二进制程序的基本块划分，在 FlowChart 模式下，IDA Pro 不仅能够给出直观的基本块流程图，还能输出每个函数、每个基本块的相对地址。故而有以下公式。

$$Address_i = Funadd_i + bbl\_reladd_i - fun\_reladd_i \quad (1)$$

其中，$Address_i$ 表示第 $i$ 个插桩分析点的具体地址，$Funadd_i$ 表示第 $i$ 个插桩分析点对应的内核函数首地址，$bbl\_reladd_i$ 为第 $i$ 个基本块的相对地址，$fun\_reladd_i$ 为第 $i$ 个基本块对应的函数相对地址。

### 2.2　驱动接口参数分析

驱动程序所有接口中 ioctl 是最为复杂的一个接口，其原型为 int ioctl(int fd, unsigned int cmd, unsigned long arg)。其中 fd 是 open 函数打开设备文件返回的文件描述符；cmd 是驱动程序对应的命令码，负责功能控制；arg 是需要的参数[14]。

由于 ioctl 的特性，fd、cmd、arg 这三个参数并非互不相关的，不同的 fd 对应有不同的 cmd 命令集合，不同的 cmd 值所对应的 arg 参数类型也是不一样的。在驱动程序实际运行的过程中，驱动程序会进行参数有效性检查。如果采取随机参数生成的形式，需要测试的样本组合就会非常巨大，同时这些庞大的样本组合中包含着大量无法通过有效性检查的测试样本，因此，对于 ioctl 函数的测试并不能简单的通过三个随机参数来完成。

在模糊测试的过程中需要将三个参数进行关联约束。

a)fd。根据 open 函数打开的的设备确定 fd。

b)cmd。每一个驱动，对应的 cmd 命令集合也是确定的。可以从内核驱动代码中提取出设备对应的 cmd 命令集合。一般 cmd 命令集合保存在各驱动的头文件中，表示方式分为宏定义和十六进制。

c)arg。驱动的每一个命令，都会有对应的的参数 arg。其中，大部分参数的类型是结构体。所以在约束过程中会涉及到结构体里的多个基础类型的变量。多数情况下，cmd 参数的宏定义中就包含了 arg 类型。少数的 cmd 宏定义中不包含参数信息，这时候要到驱动中的 ioctl 函数寻找。通过查看参数传递函数 copy_from_user 和 get_user 来了解参数类型。

在分析出函数 ioctl 参数 fd、cmd、arg 参数的对应关系，可以总结出输入约束模型。

该模型用于约束函数 ioctl 的输入参数，其内容是对驱动程序源码静态分析所得，以 json 文件格式总结每个驱动的设备名、所有命令和所有参数。通过对该输入约束模型的提取分析，可以生成输入样本。

代码 1　输入约束模型文件格式

```
dev: 设备名
cmd: {
cmd1, //本设备的命令码集合
cmd2,
…
}
```

```
type: {//参数意义对应命令码集合
arg1{type},      //基础参数类型
arg2{type,element{arg2_1,arg2_2…}},//结构体类型
…
}
```

由于约束的存在，生成的待测样本质量更高，大多能够通过驱动程序初步的逻辑监测，这些待测样本就能够进入到驱动程序更深层次的逻辑，发现程序更多的问题。

### 2.3 基于新型适应度的模糊测试

在遗传算法中评价个体优劣程度的指标是适应度，适应度越高则个体越优秀，遗传生成下一代的可能性也越大[15]。在模糊测试过程中，样本适应度越高说明覆盖率提升的概率越大，则样本越优质。因此将样本产生新覆盖的可能性数值化为新型适应度函数，保证样本变异方向是覆盖更多的未访问基本块。本文定义理想基本块和覆盖率的概念，设计了与覆盖率提升可能性对应的适应度计算函数。

#### 2.3.1 定义概念

本文定义了模糊测试中的三个概念。

a) 理想基本块。该基本块是模糊测试过程中已覆盖基本块，但是还存在至少一个后继块未覆盖。

图 1 中灰色节点表示执行中已覆盖的基本块，B 是图中唯一理想基本块。

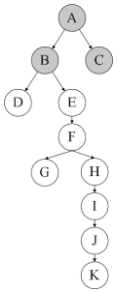


图 1　基本块调用关系图

Fig. 1　Basic block call diagram

b) 覆盖率。用于表示驱动程序在模糊测试过程中的已执行代码在总代码中的占比率 *Coverage* 。

$$Coverage = \frac{BBL_{executed}}{BBL_{all}} \times 100\% \tag{2}$$

其中， $BBL_{executed}$ 表示模糊测试进行到目前为止已经执行过的基本块数量，$BBL_{all}$ 表示整个驱动程序中所有的基本块数量。

c) 适应度。本文中的适应度分为理想基本块适应度和样本适应度。理想基本块适应度是指该理想基本块产生新覆盖的可能性。样本适应度是指样本覆盖率提升的可能性。适应度越高，新覆盖增加可能性越大。

#### 2.3.2 适应度计算

在遗传算法中，优质样本意味着覆盖率越高；适应度高的样本代表着生成优质样本的可能性越高。样本的适应度代表着覆盖率提升的可能性。

a) 理想基本块适应度。

根据定义，新覆盖的基本块都是理想基本块的子块，所以计算理想基本块的适应度来反映样本的优劣。根据漏洞挖掘实验和经验，本文总结了理想基本块适应度计算的三条规则。

规则 1　理想基本块的后继块数量越多，产生新覆盖的可能行就越大，这两者成正比关系。

规则 2　调用图中越往后，产生新覆盖和理想基本块的关联越小，需设置层数上限。后继块数量不变的情况下，理想基本块被执行的次数和产生新覆盖的能力成反比关系。

规则 3　同等情况下，适应度计算中后继块数量比基本块执行次数影响更大。

可得理想基本块适应度函数，公式如下：

$$Fitness_i = Succs_i + \frac{Succs_i}{Vis\_num_i} \tag{3}$$

其中，$Fitness_i$ 表示基本块 $i$ 的适应度，$Succ_i$ 表示基本块 $i$ 后继未被访问的基本块数(最多 4 层)，$Vis\_num_i$ 表示整个模糊测试过程中基本块 $i$ 被访问次数。

图 1 中，B 是唯一的理想基本块，B 的后继块有 7 个 D、E、F、G、H、I、J 和 K，但最多纪录 4 层，所有 $Succ_B = 6$。

b) 样本适应度。

样本的适应度和样本执行的所有理想基本块相关，所以样本的适应度等于所有理想基本块适应度的和。一般来说测试次数不会影响样本覆盖率的提升，但如果测试次数到达了较大值，说明样本开发潜力已尽。这里乘一个以测试次数为自变量的函数对适应度进行调整。

由此可得待测样本的适应度函数，计算方法如下：

$$FITNESS_i = \sum_{j=1}^{num} Fitness_j \times F(x) \tag{4}$$

其中， $FITNESS_i$ 表示第 $j$ 个待测样本的适应度， $\sum_{j=1}^{num} Fitness_j$ 表示第 $j$ 个待测样本所覆盖的所有希望节点的适应度之和，$x$ 表示该待测样本测试次数， $F(\mathrm{x})$ 为调整函数。

根据模糊测试的实际情况，得到如下公式：

$$F(\mathrm{x}) = \frac{1}{1 + e^{\frac{x - Maxnum/2}{A}}} \tag{5}$$

其中，$x$ 表示该待测样本测试次数，Maxnum 为设置的模糊测试次数上限，A 为调整参数。实验时设置 Maxnum 为 2000，A 为 100。

#### 2.3.3 样本选择

样本选择是从上次测试群体中选择优秀样本，之后将优秀样本进行交叉变异后生成下一代，将劣质样本淘汰，这种选择将样本的优秀特征保存了下来。完成样本池的保优去劣。本文选择的样本方式是轮盘赌选择。样本被选中的概率与其适应度大小成正比。

根据样本适应度计算样本被选中的概率：

$$p_i = \frac{FITNESS_i}{\sum_{j=1}^{N} FITNESS_j} \tag{6}$$

其中， $p_i$ 表示样本 $i$ 被选中的概率， $FITNESS_i$ 表示样本 $i$ 的适应度。

和自然界中一样，需要设置样本池容量上限和样本淘汰机制，淘汰的概率也和样本适应度相关，适应度越高淘汰的几率越低：

$$Dis_i = \frac{\sum_{j \neq i}^{N} FITNESS_j}{(\mathrm{N}-1)\sum_{j=1}^{N} FITNESS_j} \tag{7}$$

其中， $Dis_i$ 表示样本 $i$ 被淘汰的概率， $FITNESS_i$ 表示样本 $i$ 的适应度。为了维持待测样本池的稳定性，每生成一个新的样本进入样本池，样本池都会选择一个样本丢弃。

## 3 DrgenFuzzer 的设计和实现

### 3.1 系统架构

DrgenFuzzer 系统由内核监控、接口分析和模糊测试三个

部分组成，如图 2 所示。

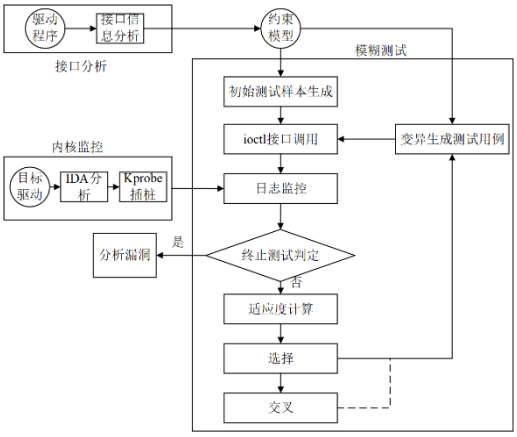


图 2　DrgenFuzzer 结构图

Fig. 2　Dgenfuzzy Structure Diagram

a)内核监控。以 IDA Pro 作为静态分析工具分析目标驱动程序，划分基本块并提取每个函数、每个基本块的相对地址；计算探测点地址之后结合 kprobes 技术完成对目标驱动程序的插桩。

b)接口分析。首先，找到目标驱动程序的命令码集合，记录备用。然后，通过每一个命令确定对应的参数类型。最后，总结出输入约束模型。

c)模糊测试。首先，根据输入约束模型中的信息生成输入样本。其次，将样本传入测试程序进行 ioctl 接口调用。接下来，分析监控日志中是否有漏洞存在。然后，计算适应度作为选择样本的标准并进行样本选择。之后，对选择的样本进行交叉变异。以上过程将一直循环，直到发现漏洞。

## 3.2　功能设计

### 3.2.1　内核监控

内核监控主要基础是在驱动中插桩。利用 IDA Pro，提取出每一个基本块和对应函数的相对地址，计算出每一个基本块相对于函数首地址的偏移量，如图 3 所示。结合符号表中驱动函数首地址的实际地址，以及式(1)，计算得到每个基本块的实际地址信息。整理准备用于后续的插桩操作。

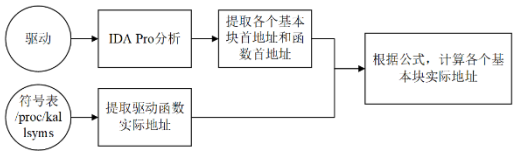


图 3　计算探测点地址

Fig. 3　Calculate the address of the detection point

kprobes 能够以探测点具体地址进行插桩。因为已经得到了所有基本块的实际地址信息，就可以对驱动程序进行插桩。通过自动化脚本生成监测模块，将其载入内核。这样当驱动运行时，内核日志会输出记录信息。

### 3.2.2　接口分析

首先，确定并记录备用驱动程序所有命令码。然后，确定每一个命令对应的参数类型。最后，根据上述的分析过程给出输入约束模型。如图 4，表示的是接口分析的过程。

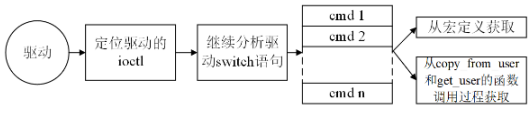


图 4　接口信息分析

Fig. 4　Interface information analysis

### 3.2.3　模糊测试

针对驱动设备的模糊测试系统可以专注在待测样本的生成、测试、选择和变异上。

1)生成样本

输入约束模型中包含 fd，cmd 命令信息，直接填入样本。只有参数 arg 需要根据模型内部信息生成。arg 参数分为基本参数类型和结构体参数类型。

对于基本参数类型来说，比如 int、long int、bool、char、int[]、long int[]、char[]等，基本参数类型的构造可以直接填充随机初始值。非数组类型以 int 为例，它可以选择-2147483648 到 2147483647 之间的任意一个数。对于数组类型，填入取值范围内的任意值，定义声明范围内的数组长度。以 char[16]为例，可能为 NULL，也有可能是长度小于 16 的随机字符串。

对于结构体参数类型来说，需要按结构体内的每个基本参数类型生成数据，然后将其组合到一起，然后通过 ioctl 调用传给驱动程序。

2)执行样本

接受到生成的样本后，根据样本生成测试程序。之后测试程序通过函数 ioctl 进行模糊测试。

3)适应度计算

计算适应度需要统计新样本运行的记录信息。记录信息包含基本块被访问次数、理想基本块序列、所有基本块的后继块序列和样本适应度。当新样本测试后，监控日志不断生成，此时就需要准备计算适应度。如果覆盖的基本块没有变多，增加访问次数。如果有新增覆盖基本块，增加访问次数，更新理想基本块序列，更新其附近的后继块信息。根据以上数据和式(3)~(5)，计算出当前测试样本的适应度。

算法 1　适应度计算算法

输入：本次测试所有基本块集合 v；覆盖基本块集合 *vis*；基本块访问次数 *vis_num*；基本块关系树 *dc_tree*;理想基本块集合 *h*；后继基本块集合 *s*；样本测试次数 *x*。

输出：样本适应度 *p*。

```
def fiteness(v,vis,vis_num,dc_tree,h,s,x):
flag = False
for item in v://遍历所有基本块
  for j in range(len(v[item])):
    node=v[item][j]
    if node==1://筛选本次测试覆盖的基本块
      if vis[item][j]==0:
        flag = True
        vis[item][j]=1 //更新覆盖基本块集合
        vis_num[item][j]+=1//更新测试次数
        dc_tree[item].visit(TreeNode(str(j)))
      else:
        continue
      if flag:
        updatehopeseq(v,dc_tree,h)//更新理想基本块集合
        updatesuccseq(v,dc_tree,h,s)//更新后继块集合
        sum=0
        for it in h:
          for node in h[it]:
            sum=sum+calffitness(it,node,vis_num,s)
//计算适应度，F(x)为调整函数
            p=float(sum*F(x))
```

```
return p
```

4)样本选择

依据式(6)和(7)，计算出各个样本被选择和被淘汰的概率。根据轮盘赌方法，生成随机数实现对样本的选择。根据测试参数的类型，对样本有不同的处理。

a)基本类型。选择完成。

b)结构体类型。暂时从样本池剔除当前选择样本，重新选择一个样本，并和当前样本一起进行交叉变异。

5)交叉变异

样本中参数类型不同，交叉变异也采取相应的策略。

a)对于 int,float,long 等基础参数类型，不进行交叉操作，只将数据变异。这里引入几种不同的变异算子。有 6 种变异算子：$\{x=x^2, x=-x, x=2x, x=\frac{x}{100}, x=x+1, x=\sqrt{x}\}$。生成一个 0 至 5 的随机数，对应 6 种变异算子。数值根据变异算子进行相应的运算得到新数据，作为下一轮的新样本输入。而对于 int、float 等类型的数组，首先生成随机新长度。新长度比原有长度更短时，按新长度给原数组进行变异。新长度更长时，将原有数组按基础类型逐个进行变异。而空白部分按类型随机生成。对于指针类型，保持不变或者置空。

b)对于结构体类型，除了变异还需要对两个样本进行交叉变异操作。对其进行二进制编码，统计结构体内参数变量数量为 $m$。生成 0 至 $2^m-1$的随机数，然后将这个数以二进制的形式表示，二进制位为 1 则对应的变量作为交叉的部分。比如 $(5)_2=101$，选中了第 1 和第 3 个参数作为交叉变量。将选中的参数交换。生成一个 1 至 500 的随机数，当随机数小于等于 100 时，对交叉结果中所有参数进行变异。否则，不对交叉结果进行操作。

## 4　实验及结果分析

本章主要通过实验测试框架的功能实现和运行效率。功能实现方面是指测试和验证功能是否正确实现。运行效率方面是比较 DrgenFuzzer 与其他的模糊测试框架，总结出优势和不足。

实验的硬件平台为 Intel(R) Core(TM) i7-9700 CPU @ 3.00GHz，内存 4.00GB，Ubuntu20.04；开发语言为 Python2.7。

### 4.1　内核监控性能测试

监控功能的实现基于在内核实现 kprobe 插桩，所以需要测试插桩是否造成大量的性能损失。以函数执行时间大小评估性能损失。

本次实验对象是内核 Linux-5.17-rc8 中的 e1000、psmouse 和 usbhid 等在内的 12 个驱动程序。实验中分别测试 DrgenFuzzer 插桩、kcov 插桩和空白组。实验原理是在驱动模块中选择一个驱动函数，在调用该函数语句的前后加入函数 ktime_get_boottime_ts64，两者的差值是函数执行时间，计算时间并输出。为了排除抵消特殊情况对实验整体造成的影响，取多次执行的时间对比分析。

以 e1000 驱动作为样例说明实验的过程，在模块中选定 e1000_clean_rx_irq 作为目标函数，在不同实验组下计算该函数执行时间。

a) DrgenFuzzer 插桩实验。根据 IDA Pro 分析，数据包接收函数 e1000_clean_rx_irq 被划分成 46 个基本块。在每个基本块的开始地址进行插桩，总共 46 个探测点。

b) kcov 插桩实验。kcov 插桩是在驱动编译时插桩。

c) 空白对照组。不对驱动程序进行插桩。

三个实验都从内核日志中收集函数时间消耗，取测试三十次的平均值作为最终结果。

如图 5 所示：测试不同的驱动，空白对照组执行时间都远远小于其他两组。对于 DrgenFuzzer 插桩和 kcov 插桩，这两组实验的曲线没有很大的差距，非常接近。根据图中的执行时间来说，DrgenFuzzer 插桩组的执行时间略高于 kcov 插桩组，甚至在 snd 驱动处，kcov 插桩反超了前者。综合所有数值，两者的差值没有超过 150ns。这说明 DrgenFuzzer 插桩对实现模糊测试带来的时间消耗是可以接受的，ns 级别的执行时间远小于应用处理消耗的时间。

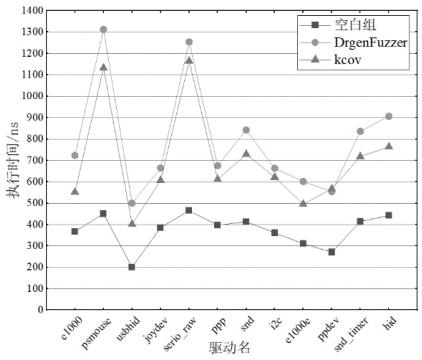


图 5　不同驱动的执行时间

Fig. 5　Execution time of different drivers

综合两者实现的结果来看，DrgenFuzzer 插桩能反馈基本块关系来指导生成高质量样本。但是 kcov 组只提供覆盖率，并没有反馈各个基本块的关系，不能指导对样本变异方向。DrgenFuzzer 的插桩方式在收集覆盖基本块关联关系上优于 kcov 插桩。在性能损失方面上，两者接近。

### 4.2　输入约束模型有效性

对于不同的模糊测试架构，如果生产的输入样本数量一致，能够成功被执行的样本数越多，代表架构生成样本的质量越高。

测试输入约束模型生成的样本有效率，定义样本测试成功率：

$$Succ=\frac{Num}{All} \tag{8}$$

其中，*Succ* 表示样本测试成功率，*Num* 表示成功执行的测试样本数，*All* 表示生成样本总数。

为了证明 DrgenFuzzer 输入约束模型的有效性，将其与内核模糊测试工具 Trinity 比较。Trinity 没有约束样本的输入约束模型。比较两者的样本测试成功率。

测试对象为 linux 系统下 snd、input、i2c、ppp 和 e1000 等十个驱动程序。分别使用两个工具测试，将产生的所有待测样本和有效执行的待测样本进行记录，然后根据式(8)计算得到样本测试成功率。

如表 1、2 所示：以测试驱动 i2c 为例，Trinity 样本测试成功率仅为 0.03%。DrgenFuzzer 样本测试成功率有 6.8%，远远高于前者。这是因为 Trinity 没有有效的约束样本的生成和变异，在 Trinity 生成的待测样本中许多测试用例的 fd 和 cmd 不匹配。虽然在中途进行了优化，但是本质上还是随机变异，测试效率依然很低。多数的无效样本基本都是因为生成 arg 参数的格式有效但语义无效，关于语义的判断很难只通过输入约束模型来完成，超过了本次实验的范畴。

在图 6 中，虽然不同的驱动程序成功率不同，但大多数情况下 DrgenFuzzer 的成功率远远高于 Trinity，是后者的数倍。说明输入约束模型让测试样本的有效性和高质量样本生

成效率提高。

表 1 样本执行总数和执行成功数

Tab. 1　Total number of sample executions and successful executions

| 驱动程序名 | DrgenFuzzer | | Trinity | |
|---|---|---|---|---|
| | 总数 | 成功数 | 总数 | 成功数 |
| snd | 35827 | 2142 | 40344 | 172 |
| input | 24362 | 2571 | 10008 | 22 |
| i2c | 25451 | 1719 | 174752 | 47 |
| ppp | 36584 | 2478 | 267906 | 255 |
| e1000 | 34654 | 2183 | 29231 | 152 |
| psmouse | 23246 | 1418 | 42387 | 89 |
| joydev | 31457 | 2359 | 31956 | 169 |
| usbhid | 21392 | 1583 | 24587 | 64 |
| e1000e | 33659 | 2793 | 34416 | 76 |
| hid | 20648 | 1879 | 23864 | 103 |

表 2　DrgenFuzzer 和 Trinity 样本测试成功率

Tab. 2　Test sample success rate of dgenfuzzy and Trinity

| 驱动程序名 | 成功率 | |
|---|---|---|
| | DrgenFuzzer | Trinity |
| snd | 0.059 | 0.0042 |
| input | 0.106 | 0.0022 |
| i2c | 0.068 | 0.0003 |
| ppp | 0.068 | 0.0009 |
| e1000 | 0.063 | 0.0051 |
| psmouse | 0.061 | 0.0021 |
| joydev | 0.075 | 0.0053 |
| usbhid | 0.074 | 0.0026 |
| e1000e | 0.083 | 0.0022 |
| hid | 0.091 | 0.0043 |

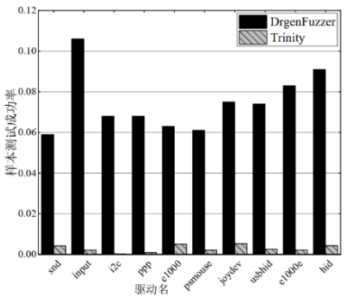


图 6　DrgenFuzzer 和 Trinity 的样本测试成功率

Fig. 6　Test sample success rate of dgenfuzzy and Trinity

### 4.3　漏洞挖掘

本次漏洞挖掘的测试对象是内核 Linux-5.17-rc8 和 Linux-5.8.4 中的各个驱动(4.2 节中的 10 个驱动)。为了说明 DrgenFuzzer 的优势，实验采用了几款常用开源工具对驱动程序来进行漏洞挖掘。

在现有的工具中，Trinity 是最早的内核模糊测试工具。Syzkaller 是最先进的内核模糊测试工具，通用性强且应用范围广。Peach 能以自定义文件格式的形式进行模糊测试。在这些方面上，其他工具不具备优势。上述三种工具都在驱动程序中发现过许多漏洞。本文分别使用这几个工具和 DrgenFuzzer 对内核驱动程序进行漏洞挖掘测试。

如表 3 所示。在两种不同内核版本下，DrgenFuzzer 的测试用例数量都是最少，但挖掘到的漏洞数量最多，具有较大的优势。对结果和工具原理进行分析可知，因为 Peach 和 Trinity 不存在输入约束模型约束生成样本，存在大量的无效的样本。Syzkaller 是从内核中系统调用设计出发的，内核功能复杂，存在许多无关系统调用测试，所以对单个驱动程序的测试效率不高。DrgenFuzzer 中总结了输入约束模型来约束样本的生成，待测样本测试成功率更高；提出了新型适应度计算方案，样本向覆盖率提高的方向变异，漏洞挖掘效果更好。

表 3　驱动漏洞挖掘结果对比

Tab. 3　Comparison of driver vulnerability mining results

| 内核版本 | 工具 | 漏洞数 | 测试用例数 |
|---|---|---|---|
| Linux-5.17-rc8 | Peach | 0 | 9163541 |
| | Syzkaller | 1 | 7153870 |
| | DrgenFuzzer | 4 | 5823260 |
| | Trinity | 1 | 10849524 |
| Linux-5.8-4 | Peach | 2 | 816789 |
| | Syzkaller | 4 | 654891 |
| | DrgenFuzzer | 8 | 621538 |
| | Trinity | 3 | 964532 |

### 4.4　漏洞总结

对 i2c、e1000、ppp、input、snd 等驱动进行模糊测试。通过接口分析，得到如表 4 的信息。

表 4　接口信息

Tab. 4　Interface information

| 驱动 | cmd 命令数量 | 使用的结构体类型数 |
|---|---|---|
| i2c | 9 | 4 |
| e1000 | 5 | 1 |
| ppp | 19 | 1 |
| input | 38 | 14 |
| snd | 37 | 6 |

DrgenFuzzer 在四个驱动中发现多个漏洞，选取一个 i2c 漏洞说明。i2c 中有命令码 I2C_RDWR，当使用这个命令的过程中会调用函数 i2cdev_ioctl_rdwr，出错的位置在这个函数中。

与错误相关的结构体是 struct i2c_msg，结构体声明如下。

代码 2　结构体声明

```
struct i2c_msg {
__u16 addr;
__u16 flags;
__u16 len;
__u8 *buf;
};
```

结构体参数中 len 和 buf 分别代表缓冲区长度和缓冲区长度。驱动代码中有判断 i2c_msg 是否安全的语句。其代码如下：

代码 3　判断代码

```
if (msgs[i].len > 8192) {
res = -EINVAL;
break;
}
data_ptrs[i] = (u8 __user *)msgs[i].buf;
```

可以看到代码只考虑了 len 数值过大即缓冲区过长的问题，但是当指针 msgs[i].buf 值为 NULL 时，可以通过判断 len 的 if 语句。之后在执行 buf 赋值给 data_ptrs[i]时造成空指针引用的问题。

## 5　结束语

本文在分析驱动程序接口参数的基础上，总结出输入约

束模型来约束生成样本。同时考虑到内核驱动运行难以监测的问题，提出了基于 eBPF 的内核监控方案。并根据程序参数的特性，引入理想基本块的新定义，提出新的适应度计算方案，能够高效的指导样本进行交叉变异。从而设计并实现了 DrgenFuzzer 工具，并与其他常见工具进行对比，证明了 DrgenFuzzer 功能实现完备、运行效率优秀、生成样本更优质和漏洞挖掘能力强。下一步是分析结构体中的各个参数间的关联关系，接口静态分析分析不出这些关系。尝试其他方法进行分析，使系统能够更高效的生成高质量样本。对于不同类型的数据采取不同的变异手段，使测试样本更多样化。

## 参考文献：


[1] Zhao Bodong, Li Zheming, Qin Shisong, *et al.* Statefuzz: system call-based state-aware linux driver fuzzing [C]// Proc of the 31st USENIX Security Symposium. Berkeley, CA: USENIX Association, 2022: 3273-3289.

[2] Zhao Wenjia, Lu Kangjie, Wu Qiushi, *et al.* Semantic-informed driver fuzzing without both the hardware devices and the emulators [C]// Proc of Network and Distributed Systems Security Symposium. Rosten, VA, USA: Internet Society, 2022.

[3] Wang Daimeng, Zhang Zheng, Zhang Hang, *et al.* SyzVegas: beating kernel fuzzing odds with reinforcement learning [C]// Proc of the 30th USENIX Security Symposium. Berkeley, CA: USENIX Association, 2021: 2741-2758.

[4] 吴志勇, 王红川, 孙乐昌, 等. Fuzzing 技术综述 [J]. 计算机应用研究, 2010, 27 (3): 829-832. (Wu Zhiyong, Wang Hongchuan, Sun Lechang, *et al.* Overview of Fuzzing technology [J]. Application Research of Computers, 2010 (3): 829-832.)

[5] 张策, 孟凡超, 考永贵, 等. 软件可靠性增长模型研究综述 [J]. 软件学报, 2017, 28 (9): 2402-2430. (Zhang Ce, Meng Fanchao, Kao Yonggui, *et al.* Survey of software reliability growth model [J]. Journal of Software, 2017, 28 (9): 2402-2430.)

[6] Kim K, Jeong D R, Kim C H, *et al.* HFL: hybrid fuzzing on the linux kernel [C]// Proc of the 27th Annual Network and Distributed System Security Symposium. Rosten, VA, USA: Internet Society, 2020.

[7] Carabas C, Carabas M. Fuzzing the linux kernel [C]// Proc of Computing Conference. Piscataway, NJ: IEEE, 2017: 839-843.

[8] Song D, Hetzelt F, Das D, *et al.* Periscope: an effective probing and fuzzing framework for the hardware-os boundary [C]// Proc of Network and Distributed Systems Security Symposium. Rosten, VA, USA: Internet Society, 2019: 1-15.

[9] 姜欧涅. 基于 eBPF 的网络数据包捕获与分析系统的设计与实现 [D]. 武汉: 华中科技大学, 2020.

[10] Sun Hao, Shen Yuheng, Liu Jianzhong, *et al.* KSG: augmenting kernel fuzzing with system call specification generation [C]// Proc of USENIX Annual Technical Conference. Berkeley, CA: USENIX Association, 2022: 351-366.

[11] Liang Hongliang, Pei Xiaoxiao, Jia Xiaodong, *et al.* Fuzzing: state of the art [J]. IEEE Trans on Reliability, 2018, 67 (3): 1199-1218.

[12] 徐鹏, 刘嘉勇, 林波, 等. 基于循环神经网络的模糊测试用例生成 [J]. 计算机应用研究, 2019, 36 (9): 2679-2685. (Xu Peng, Liu Jiayong, Lin Bo, *et al.* Generation of fuzzing test case based on recurrent neural networks [J]. Application Research of Computers, 2019, 36 (9): 2679-2685.)

[13] 廉美, 邹燕燕, 霍玮, 等. 动态资源感知的并行化模糊测试框架 [J]. 计算机应用研究, 2017, 34 (1): 52-57. (Lian Mei, Zou Yanyan, Huo Wei, *et al.* Dynamic resource awareness framework for parallel fuzzing [J]. Application Research of Computers, 2017, 34 (1): 52-57.)

[14] 何远, 张玉清, 张光华. 基于黑盒遗传算法的 Android 驱动漏洞挖掘 [J]. 计算机学报, 2017, 40 (5): 1031-1043. (He Yuan, Zhang Yuqing, Zhang Guanghua. Android driver vulnerability discovery based on black-box genetic algorithm [J]. Chinise Journal of Computers, 2017, 40 (5): 1031-1043.)

[15] 邓一杰, 刘克胜, 朱凯龙, 等. 基于动态适应度函数的模糊测试技术研究 [J]. 计算机应用研究, 2019, 36 (5): 1415-1418. (Deng Yijie, Liu Kesheng, Zhu Kailong, *et al.* Research on fuzzing technique based on dynamic fitness function [J]. Application Research of Computers, 2019, 36 (5): 1415-1418.)