简介

近年来，反馈引导的Fuzzing工具被广泛应用于自动软件检测方面，然而，这类Fuzzer通常存在两个路障问题：魔数（magic number）和嵌套校验（nested checksum）。一些工作提出使用污点跟踪和符号化执行解决上述两个问题，但是这类工具会需要一系列的条件，如：程序源码、精准的环境建模、准确的指令集语义等等。

本文提出了一个无需污点跟踪和符号化执行的轻量级方法解决这两个问题。作者观察到在被测试程序执行期间，输入数据的一部分与程序内部数据状态是同步的，例如寄存器、内存。作者将这一现象命名为输入到状态同步（input-to-state correspondence）。利用输入到状态同步的性质，本文基于KAFL工具，实现了一个名为REDQUEEN的工具，可以用高效的方法克服fuzzing中的两个路障。

一些现阶段其他的常用工具介绍

就介绍现有的一些研究情况，不涉及论文自身东西

输入到状态同步

通过简单而直观的观察发现，对于大部分的程序，输入数据中的一部分数值会被直接地使用在执行时的多个状态中。例如，一些程序直接使用输入数据的一部分与硬编码的上下文条件进行匹配，如魔数检查。通过观察这些数值，可以进行有根据的猜测，以确定要替换输入中的哪些偏移位置（类似于轻量级的污点跟踪），以及替换成哪些数值（类似于基于符号化执行的方法）。利用这一关系可以解决Fuzzing中的魔数检查和校验和问题。

魔数

```
/* magic number example */
if(u64(input)==u64("MAGICHDR"))
  bug(1);
```

1. 追踪：每次遇到新的路径时，就`hook`所有比较指令并进行一次追踪执行。若比较指令的两个参数不同，则提取参数，生成一个自定义的突变模式`<pattern→repl>`。例如，图中的`input`为`TestSeedInput`，观察比较指令发现`input`中的`"deeStesT"`与`"RDHCIGAM"`进行了比较，因此input中的`"deeStesT"`应突变为`"RDHCIGAM"`即可获得新的`fuzzing interesting input`。此例中`pattern`为`"deeStesT"`，`repl`为`"RDHCIGAM"`，获得突变模式`<"deeStesT"→"RDHCIGAM">`。注意，此处提取的参数为程序运行时经过了大小端及编码处理的参数。

2. 变化：考虑到比较后的标志位检查，因此对比较的值应用`+1/-1`的变化。例如`"RDHCIGAM"`经过变化可得到`"RDHCIGAL"`、`"RDHCIGAN"`。

3. 编码：在到达实际比较之前，输入可能经过不同的编解码方式处理，因此提出了最常见的几种编解码方式创建更多的候选突变。例如零扩展、大小端转换、数字与`ASCII`码转换等。

4. 应用：使用`<pattern→repl>`的模式来识别需要用`mutation`替换`repl`的部分并应用于输入数据。

5. 染色：有时发现输入数据中的可候选替换位置数量非常大，我们设计了一个算法来增加输入中的随机字节数，输入的熵增使候选替换位置以数量级为单位减少。染色后，我们只会在染色前后两个输入的相同偏移处应用突变。

校验和

```
if(u64(input)==sum(input+8, len-8))
  if(u64(input+8)==sum(input+16, len-16))
    if(input[16]=='R' && input[17]=='Q')
      bug(2);
```

Listing 3: Fuzzing problem (2): finding valid input to bypass checksums.

现有方法中解决校验和问题的思路为：先移除困难检查，稍后再修复它们。REDQUEEN借鉴了这一思想，但无需使用符号化执行和污点跟踪。解决如图2中的校验和的方法分为以下几步：

1. 识别：首先使用三个启发式规则确定与校验和检查类似的比较。即，比较指令的一侧pattern是输入到状态同步的数值，另一侧repl经常变化。

a) 在所有的输入中，突变模式左侧的pattern都使用同样的编码方法。此处的编码指原始输入数据到比较指令中参数的编码，同2.1-3小节。

b) 比较指令的两个参数都不是立即数。

c) 在染色阶段，由于输入数据变化，pattern作为输入数据的一部分也发生变化，同时，由pattern计算校验和得到的repl也会发生改变。

2. 打补丁：使用一定会返回true的指令替换识别到的比较指令。这一步后继续进行fuzzing，将获得的interesting input放入一个预备队列中。

3. 验证：对于打补丁阶段获得的预备队列中的输入，对其应用基于输入到状态的突变，尝试修复这些预备输入，突变方式同2.1节魔数处理中所述。然后将修复后的输入喂给未打补丁的程序执行，如果修复后的输入仍然能覆盖到新的basic block，则将输入放入fuzzing的真实队列。否则，就丢弃补丁。将预备队列中的所有input处理完毕后，将预备队列清空。

嵌套校验和的处理：对于被patch的比较指令，创建指令间的依赖图，根据依赖关系进行拓扑排序获得有效的指令顺序。

评估

TABLE II: Listed and (+unlisted bugs) found after 5 hours of fuzzing on LAVA-M (numbers taken from the corresponding papers).

Program	Listed Bugs	FUZZER	SES	VUzzer	STEELIX	T-Fuzz	Angora		Redqueen	
uniq	28	7	0	27	7	26	28	(+ 1)	28	(+ 1)
base64	44	7	9	17	43	43	44	(+ 4)	44	(+ 4)
md5sum	57	2	0	-	28	49	57	(+ 0)	57	(+ 4)
who	2136	0	18	50	194	63	1443	(+ 98)	2134	(+ 328)

作者用REDQUEEN与其他多个工具对LAVA-M数据集进行测试，各工具检测到的bug数如上图所示。REDQUEEN找到了几乎所有LAVA-M开发者列出的bug，只有2个遗漏bug，另外还找到了337个额外bug。

为测试REDQUEEN对真实世界程序的测试效果，作者使用一系列工具对binutils进行测试并统计了各工具的基本块覆盖率，REDQUEEN对于大部分被测试程序的效果都优于其他benchmark。

作者还对REDQUEEN的性能开销进行了测试。如图4所示，条形图表示每秒的平均执行次数。与LAF-INTEL和AFL-FAST相比，KAFL和REDQUEEN的性能影响在25-50%范围内。

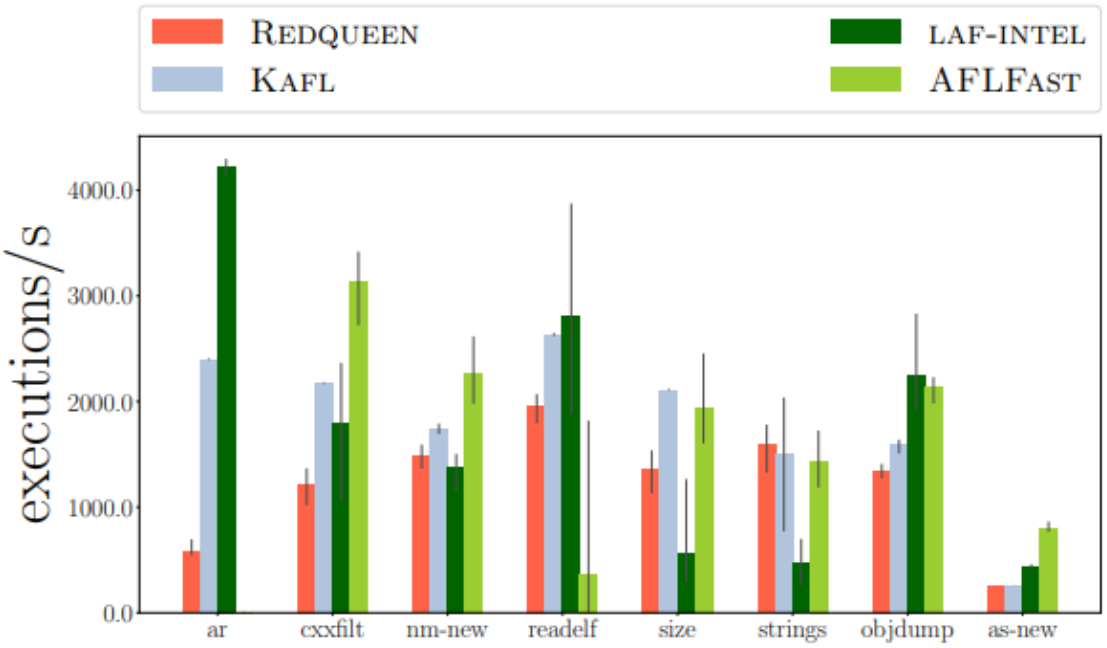

Fig. 4: Evaluating the execution speed on our **binutils** targets.

局限性

无法解决输入与转换发生后的状态不对应的情况。例如，binutils测试集的一些程序中，来自输入的字符串用于索引哈希映射，而后使用整数返回。