Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

AFL——基于覆盖的模糊测试工具

硕士研究生 刘力源

2021年03月28日

内容提要

- 背景简介
- 基本概念
- 算法原理
- 优劣分析
- 应用总结
- 参考文献

背景介绍

- 预期收获
  - 1. 了解模糊测试的定义以及分类
  - 2. 了解AFL工具的工作原理
  - 3. 了解AFL的改进策略

背景简介

- 漏洞攻击事件
  - WannaCry勒索病毒
    - 利用了服务器消息块(SMB)协议中的漏洞
    - 一天内感染了150多个国家的23万多台计算机
    - 给金融、能源、医疗等行业造成了巨大的损失

背景介绍

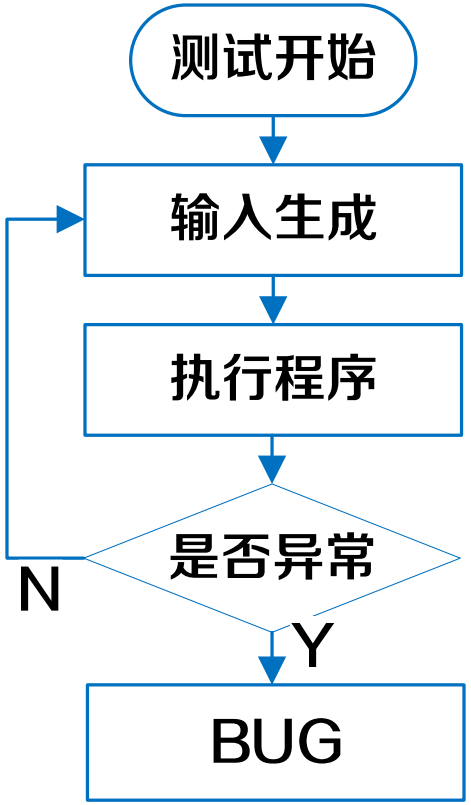
- 漏洞：
  - 定义：系统设计、实现或操作和管理中的缺陷或弱点，可以被利用来违反系统的安全策略
  - 对软件漏洞的攻击，特别是对零日漏洞的攻击，可能会导致严重的破坏。
- 漏洞检测主要技术
  - 静态分析：分析词法、语法、语义特征
  - 动态分析：在真实的系统或模拟器中执行目标程序检测程序的bug。
  - 符号执行：符号化程序输入，为每个执行路径维护一组约束。约束求解器求解约束，确定导致执行的输入。
  - 模糊测试（Fuzzing）

基本概念

基本概念

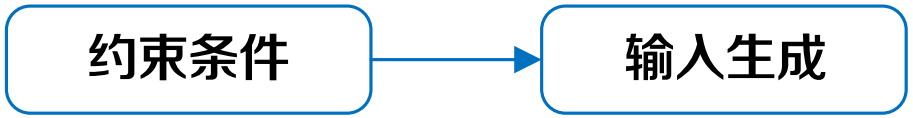
- 模糊测试
  - 定义：不断构造非预期的输入，并将其发送到目标程序中，监视目标程序的执行状态，检测可能存在的异常

  - 工作流程
    1. 构造非预期的畸形数据
    2. 程序执行构造的输入
    3. 发生异常则返回监视信息，否则执行1
    4. 分析测试结果



基本概念

- 测试样例生成方式
  - 基于生成
    - 定义：根据预先定义的约束条件生成输入
    - 优点：代码覆盖率高、容易通过程序验证



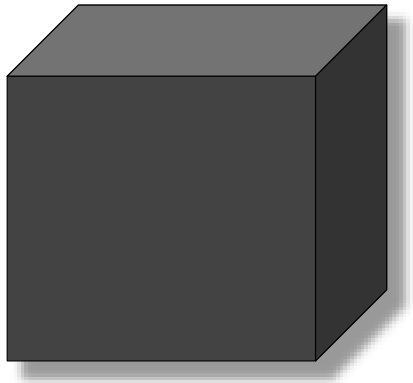
  - 基于变异
    - 定义：对初始输入的突变来生成测试用例
    - 优点：准备工作简单、几乎不需要先验知识

- 程序的探索策略
  - 定向模糊算法：生成覆盖目标代码和程序的目标路径的测试用例
  - 基于覆盖的模糊算法：生成覆盖尽可能多的程序代码的测试用例

基本概念

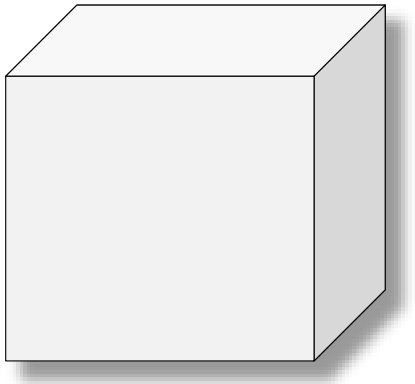
- 源代码的依赖程度

不了解内部结构



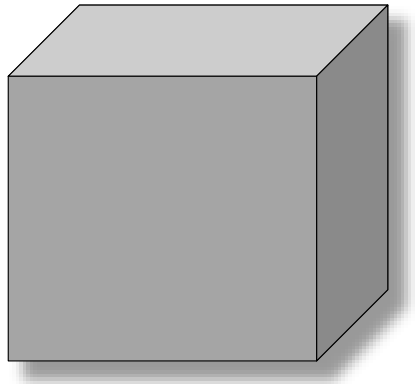
黑盒测试

可以访问源代码



白盒测试

借助程序分析工具

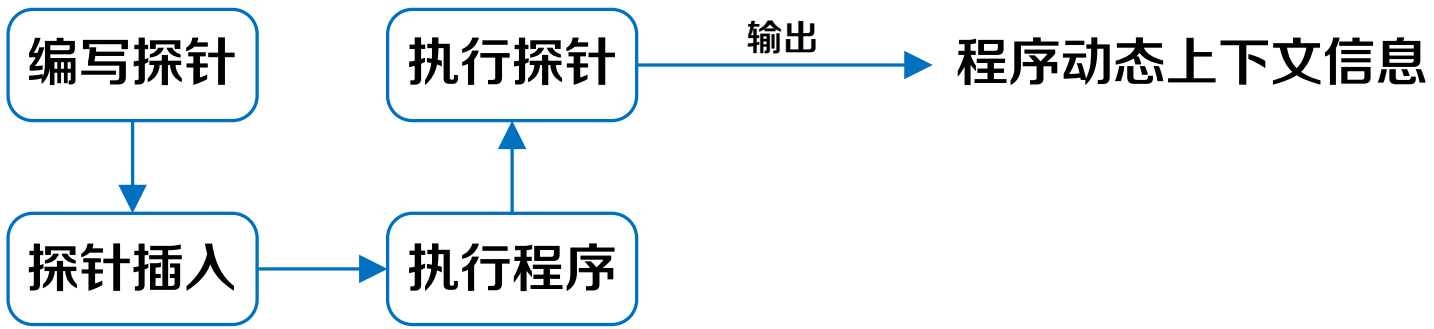


灰盒测试

注：灰盒不如白盒详细、完整，但相比黑盒更关注程序内部的逻辑

基本概念

- 程序监控信息与输入生成的关系
  - 哑模糊（dumb fuzz）：测试用例生成策略与程序行为信息无关
  - 智能模糊（smart fuzz）：根据测试用例如何影响程序行为的信息来调整测试用例的生成
- 程序插桩
  - 定义：保证被测程序原有逻辑完整性的基础上在程序中插入一些探针（进行信息采集的代码段）
  - 实现：



基本概念

- 模糊测试主要困难
  - 种子输入变异策略选择

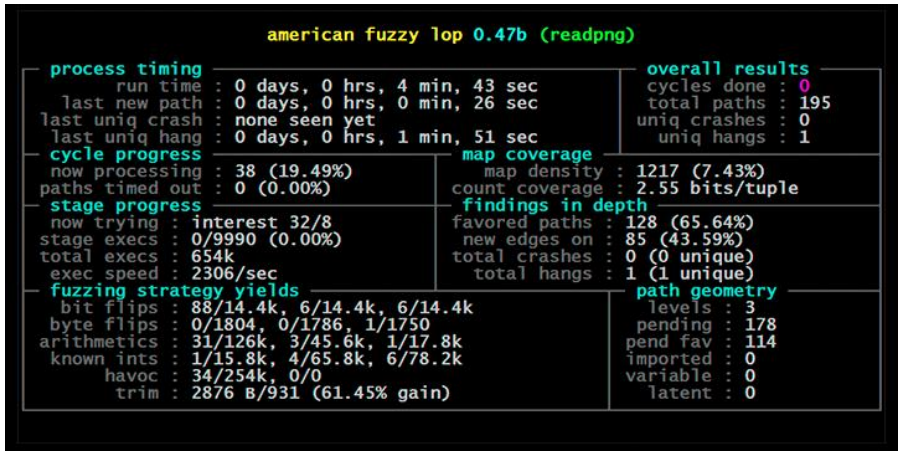
Where?

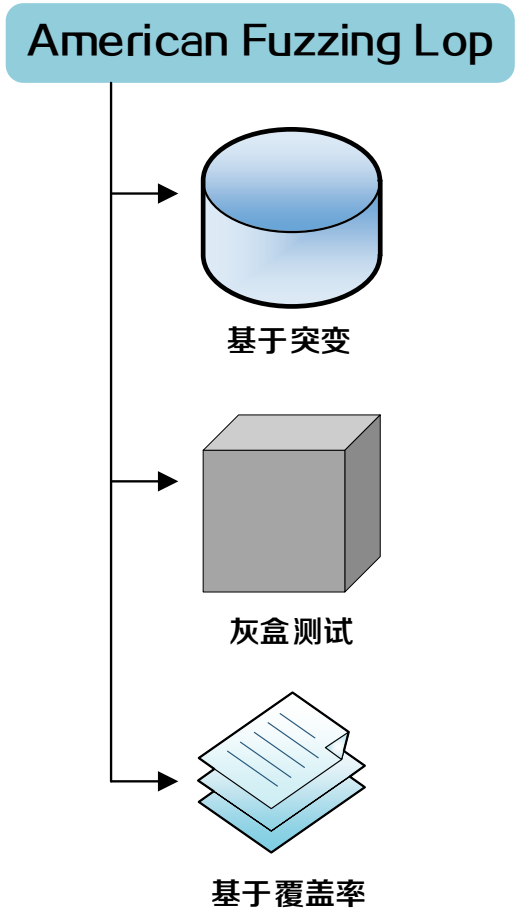
How?

→ 减少测试资源的浪费

  - 低代码覆盖率
    - 触发更多的路径
  - 如何通过程序验证
    - 程序通常在解析和处理之前验证输入

- American Fuzzing Lop（AFL）
  - 基于覆盖的模糊测试工具
  - 两种检测模式（源代码访问）
    - 编译式检测：gcc、llvm
    - 外部检测：qemu





算法原理

算法原理

T	检测目标程序潜在的漏洞
I	初始测试用例
P	1.待测程序插桩 2.种子选取和突变 3.程序监视跟踪 4.崩溃信息收集
O	触发bug的输入和程序漏洞

P	实现较高的代码覆盖率
C	源代码检测
D	代码覆盖率的度量方法
L	Zalewski 2013-2017

算法原理

- AFL主要工作过程
  - T：由种子构造的输入集合
  - Tx：引发崩溃的测试样例集合
    - ① 将给定/构造的种子加入T
    - ② 从T中随机选择一个种子进行突变
    - ③ 监视代码覆盖和安全违规情况
    - ④ 收集触发崩溃的样例Tx
    - ⑤ 将触发新路径的样例加入到T中
    - ⑥ 执行②
    - ⑦ 测试结束后得到Tx

Algorithm 1 Coverage-based Fuzzing

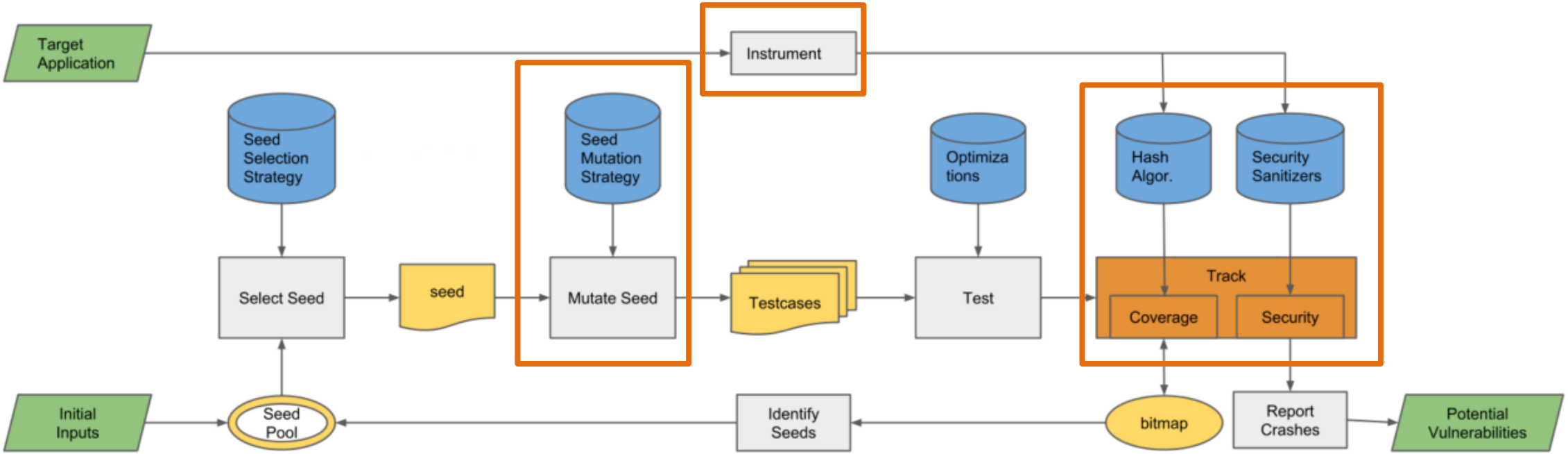
Input: Seed Inputs S

```
1:  T = S
2:  Tx = ∅
3:  if T = ∅ then
4:      T.add(emptyfile)
5:  end if
6:  repeat
7:      t = choose_next(T)
8:      s = assign_energy(t)
9:      for i from 1 to s do
10:         t' = mutate(t)
11:         if t' crashes then
12:             Tx.add(t')
13:         else if isInteresting(t') then
14:             T.add(t')
15:         end if
16:     end for
17: until timeout or abort-signal
```

Output: Crashing Inputs Tx

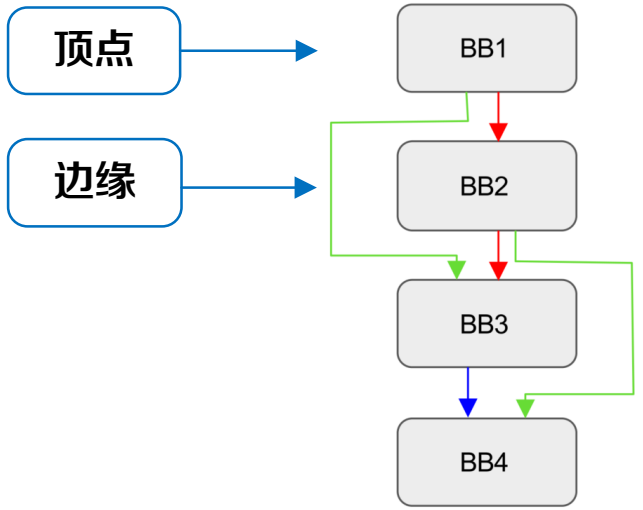
算法原理

- 工作流程图


Target Application
Instrument
Seed Selection Strategy
Seed Mutation Strategy
Optimizations
Hash Algor.
Security Sanitizers
Select Seed
seed
Mutate Seed
Testcases
Test
Track
Coverage
Security
Initial Inputs
Seed Pool
Identify Seeds
bitmap
Report Crashes
Potential Vulnerabilities

算法原理

- 代码覆盖率的度量
  - 粒度选择
    - 基本块（Basic Block，BB）：只有一个入口和出口点的代码片段，其中的指令按顺序执行，只执行一次
    - 理由：
      ① 程序执行的最小相干单位
      ② 测量功能或指令会导致信息丢失或冗余
      ③ 基本块可以被第一个指令的地址识别
      ④ 通过插桩可轻易提取基本块的信息
  - 测量方法选择
    - 计算基本块的数量：记录顶点
    - 计算基本块的过渡：记录边缘
    - 前者会丢失边缘信息

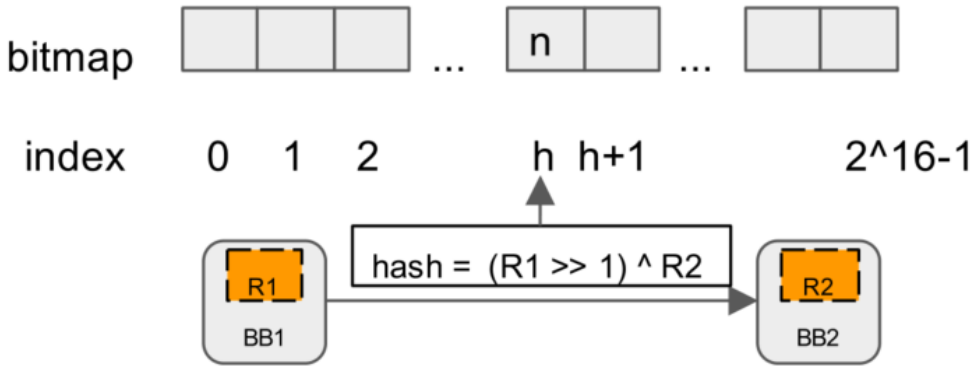



```
A -> B -> C -> D -> E (tuples: AB, BC, CD, DE)
A -> B -> D -> C -> E (tuples: AB, BD, DC, CE)
```

算法原理

- 程序跟踪
  - 代码覆盖监测（插桩）
    - 随机生成cur_location
    - 64Kb的shared_mem[] 的每个字节存储路径信息
    - 新路径产生时将计算哈希数
    - 移位操作使得让tuple具有定向性，将A→B和B→A、A→A和B→B区别开
  - 安全违规检查
    - 借助第三方软件：解释bug

```
cur_location = <COMPILE_TIME_RANDOM>;
shared_mem[cur_location ^ prev_location]
++;
prev_location = cur_location >> 1;
```



算法原理

- 突变策略：
  - 随机变异
    - 改变长度实现连续位翻转
    - 连续添加和减少较小的整数
    - 连续插入整数，0、1等
  - 测试样例拼接
- 存在问题
  - 测试用例生成的盲目性
    - 和程序运行信息无关，随机生成了大量无用的输入

算法原理

算法原理

T	提高测试样例触发漏洞的可能
I	初始测试用例
P	1.生成空执行集 2.随机生成候选集元素 3.计算测试样例和最领近的距离 4.选择距离最大的样例作为输入
O	自适应测试样例

P	测试用例多样性
C	测试用例大小长度一致
D	从候选集中选取合适的输入
L	2020 IEEE ISSRW

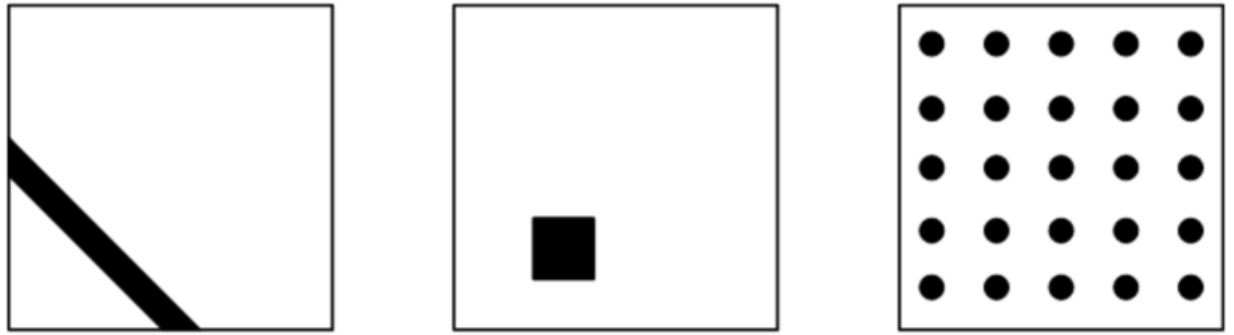
算法原理

- 算法流程
  - FSCS（Fixed Sized Candidate Set）
    - 固定大小候选集，属于自适应随机测试（Adaptive Random Testing，ART）
    - 应用流程
      ① 生成空执行集E
      ② 输入集中随机产生一个测试样例t
      ③ 将t加入E中
      ④ 随机生成k个候选输入组成候选集C
      ⑤ 比较候选输入和E中最相似的元素的距离
      ⑥ 选取距离最大的候选用例作为测试输入t
      ⑦ 若程序终止则执行③，否则退出

Algorithm 1 FSCS
1. Input an integer k, where $k > 1$.
2. Set $n = 0$ and $E = \{\}$.
3. Randomly generate a test case t from the input domain.
4. **While** (termination condition is not satisfied) **do**
5. Add t into E, and increment n by 1.
6. Randomly generate k candidates $c_1, c_2, \ldots, c_i, \ldots, c_k$ from the input domain, and construct $C=\{c_1, c_2, \ldots, c_i, \ldots, c_k\}$.
7. **For** each candidate c_i, where $i = 1, \ldots, k$. **do**
8. Calculate its distance d_i to its nearest neighbor in E.
9. **End for**
10. Find $c_b \in C$ such that $\forall i = 1, \ldots, k, d_b \geq d_i$.
11. Set $t = c_b$.
12. **End while**
13. **Exit.**

算法原理

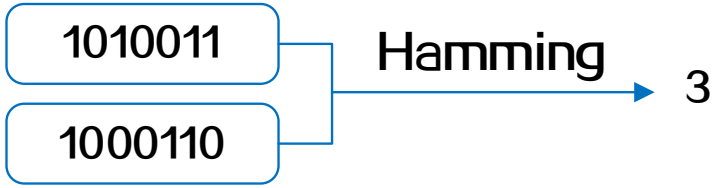
- 故障区域（Failure areas）
  - 指导致非预期输出的输入在整个输入域的分布情况
  - 类型
    - 带状（strip）
    - 块状（block）
    - 点状（point）



- AFL-ART的思路
  - 对于不能触发程序bug的测试用例，与其相似的也不太可能触发bug
  - 选择最不相同的测试用例

算法原理

- 测试用例选取
  - 距离计算策略：汉明距离，更适用于数值文件
  - 执行集里最相似的元素
    - 计算每一个候选用例和执行集中所有元素的汉明距离
    - 距离最小的元素视为最相似的
  - 候选用例选取
    - 选取距离最大候选的加入到执行集中



$$\forall c \in C, \forall e \in E, \min dist(c', e) \geq \min dist(c, e)$$

Distance / Set E / Set C	Et01	Et02	Et03	the shortest distance
Ct01	236	127	398	127
Ct02	45	453	158	45
Ct03	78	65	265	65
Ct04	382	501	214	214
Ct05	482	217	69	69
Ct06	418	203	539	203
Ct07	65	202	436	65
Ct08	101	214	576	101
Ct09	482	213	89	89
Ct10	208	159	66	66
The longest distance in the shortest distance				214

算法原理

- AFL-ART
  - 对AFL的改进
    - ① 选取/构造种子
    - ② 由种子生成输入集
    - ③ 生成空集E，又输入集随机生成C
    - ④ 随机选取第一个测试用例t并且执行
    - ⑤ 依据上一个输入应用FSCS算法生成下一个输入
    - ⑥ 测试输入
    - ⑦ 执行②，直至工具停止运行

Algorithm 2 FuzzTest (*Prog, Seeds*)
1. Construct queue *Queue* = Seeds
2. Construct Set *E* = {} to store executed test cases
3. Construct Set *C* = {} to store candidate test cases, and candidate test cases are randomly generated from *Queue*.
4. Randomly generate a test case *t* from *Queue*
5. Invoke RunAndSave (*Prog, t, Queue*) to test *t*
6. **While** Termination is not triggered **do**
7. Execute FSCS-ART algorithms to select a test case *c*
8. Mutate *c* and generate many new test cases *newTs*
9. **For** *newt* in *newTs* **do**
10. Invoke RunAndSave (*Prog, t, Queue*) to test *newt*
11. **End For**
12. **End while**
13. **Exit.**

算法原理

- 实验结果
  - 测试C/C++程序：SQLite、tcpdump
  - 工具运行参数

Test tool	Running time	Total paths	Own finds	Level	Stability
AFL	1h	1906	1229	2	99.21%
AFL-ART	1h	1878	1859	3	99.98%

  - 代码覆盖率比较

TABLE IV. CODE COVERAGE INFORMATION FOR AFL

Structure	Coverage statistics		
	Hit	*Total*	*Coverage*
Line	11150	31586	35.3%
Function	890	1719	51.8%
Branch	5550	21171	26.2%

TABLE V. CODE COVERAGE INFORMATION FOR AFL-ART

Structure	Coverage statistics		
	Hit	*Total*	*Coverage*
Line	12234	31586	38.7%
Function	919	1719	53.5%
Branch	6296	21171	29.7%

算法原理

- 实验结果
  - 运行速度

Fig. 3. AFL execution speed

Fig. 4. AFL-ART execution speed

  - 测试tcpdump
    - 测试时间：20min
    - 结果比较

TABLE VI. TEST RESULTS FOR TCPDUMP

Test tool	Running time	Num. of crashes
AFL	20min	0
AFL-ART	20min	20

算法原理

- 实验结论
  - AFL和AFL-ART都具有较高的稳定性
  - AFL-ART相比AFL代码覆盖率更高
  - AFL-ART的后期能维持较高的运行速度
  - AFL-ART相比AFL可以发现更多的漏洞

优劣分析

优劣分析

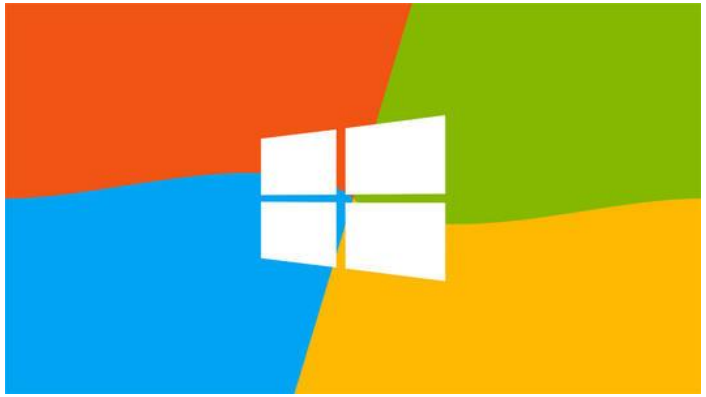
- 优势：
  - AFL初始输入较为简单
  - AFL实现了较高的代码覆盖率
  - AFL-ART提高了测试的质量
    - 代码覆盖率提升
    - 发现更多的程序漏洞

- 劣势：
  - 没有提出非数值输入之间距离的计算策略
  - FSCS的计算开销较大

应用总结

- 操作系统支持
  - Linux
  - Windows

- 贡献
  - 几年间发现上千个真实软件漏洞



参考文献

- [1] LI J, ZHAO B, ZHANG C. Fuzzing: a survey[J]. Cybersecurity, 2018,1(1): 6.
- [2] J. C, J. C, D. G, et al. An improved fuzzing approach based on adaptive random testing: 2020 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW) [C], 2020.
- [3] https://lcamtuf.coredump.cx/afl/
- [4] https://lcamtuf.coredump.cx/afl/technical_details.txt

谢谢！

大成若缺，其用不弊。大盈若冲，其用不穷。大直若屈。大巧若拙。大辩若讷。静胜躁，寒胜热。清静为天下正。