



Cerebro : Context-Aware Adaptive Fuzzing for Effective Vulnerability Detection

论文相关信息

作者：

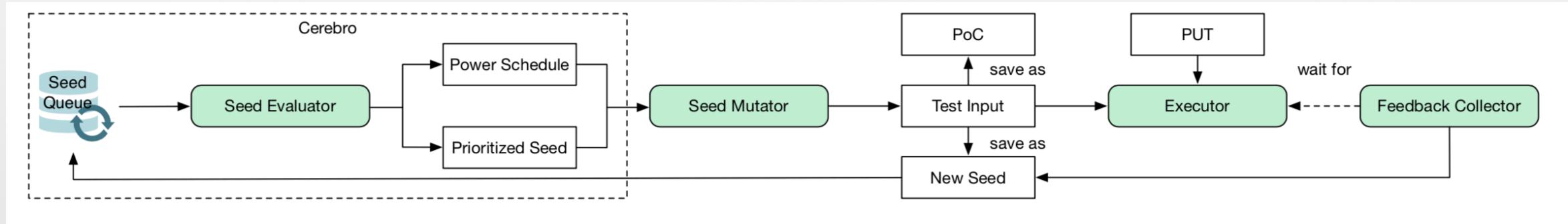
Yuekang Li:University of Science and Technology of China
Xiuheng Wu:Nanyang Technological University Singapore
Yinxing Xue:University of Science and Technology of China
Cen Zhang:Nanyang Technological University Singapore
Hongxu Chen: Nanyang Technological University Singapore
Xiaofei Xie:Nanyang Technological University Singapore
Haijun Wang Nanyang Technological University Singapore
Yang Liu:Nanyang Technological University Singapore

关键词：

Fuzz Testing; Software Vulnerability
模糊测试；软件漏洞

灰盒模糊测试的基本问题

- 种子优先级排序：如何选择下一个进行模糊测试的种子
- 功率调度：应该对当前种子进行多少工作



灰盒模糊测试的基本工作流程

AFL的功率调度基于种子的性能得分，该得分主要基于该种子的路径覆盖范围和执行时间来计算。而AFLFast将更多能量分配给可以覆盖稀有路径的种子。

灰盒模糊测试面对的问题和挑战

- 1.现有的测试器不跟踪上下文的代码 上下文分析可能需要占用大量的性能
如果执行轨迹周围的相邻代码被覆盖 这个种子是否应该被设定为更感兴趣的？
- 2.利用单一目标确定种子优先级或通过线性标量将多个目标混合到一个单一目标执行
种子的优先级排序

Cerebro为了解决以上两个问题提出了新的定义和新的模型及算法

Input Potential 输入潜能

Multi-Objective optimization 多目标优化模型

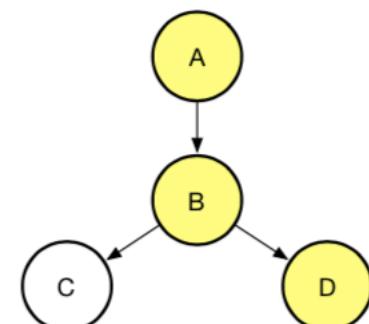
Input Potential



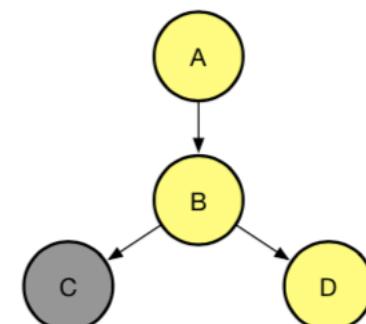
输入执行树上下文未覆盖的代码的复杂度

```
1 int mjs_ffi_call (...) { // Func A
2 ...
3     mjs_parse_ffi_signature (...); // Func B
4 ...
5 }
6 int mjs_parse_ffi_signature (...) {
7 ...
8     if (*tmp_e != '(') {
9         if (mjs->dlsym==NULL) {
10             mjsprepend_errorf(); // Func C
11             goto clean;
12         }
13         mjs->dlsym (...); // Func D
14 ...
15     }
16     else {
17         goto clean;
18     }
19 ...
20 clean:
21 ...
22 }
```

Listing 1: code snippets from *mjs*



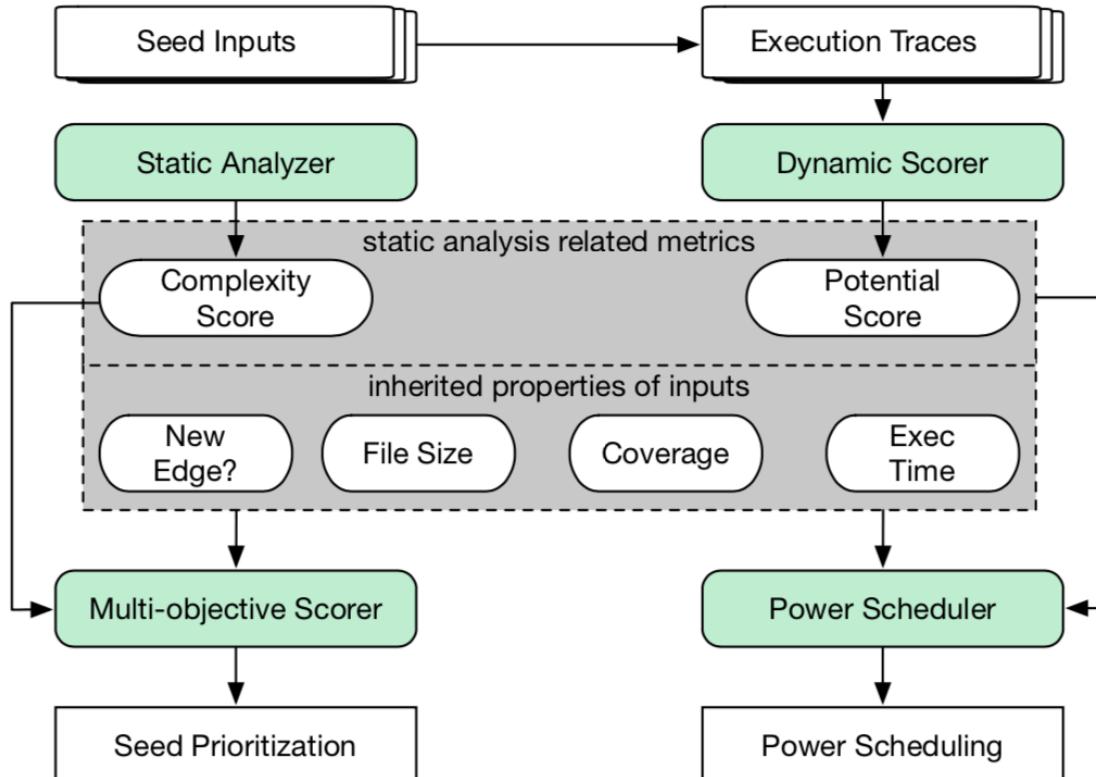
(a) state 1



(b) state 2

Cerebro的基本工作流程和主要组成部分

系统的输入是一组初始种子集



输出是种子优先级和功率调度的决策

Figure 3: Overview of CEREBRO *

*Legend: green rounded rectangles denote the major components of the system; squashed rectangles in the grey boxes denote scores or metrics associated with a seed.

四个主要组成部分：

- 静态分析器
- 动态评分器
- 多目标评估器
- 功率调度器

静态分析器和动态评分器



静态分析器扫描整个源代码 为每个函数计算复杂性得分

动态评分器的目的是评估当前没有被fuzzer覆盖但是可能通过突变覆盖的代码的复杂性

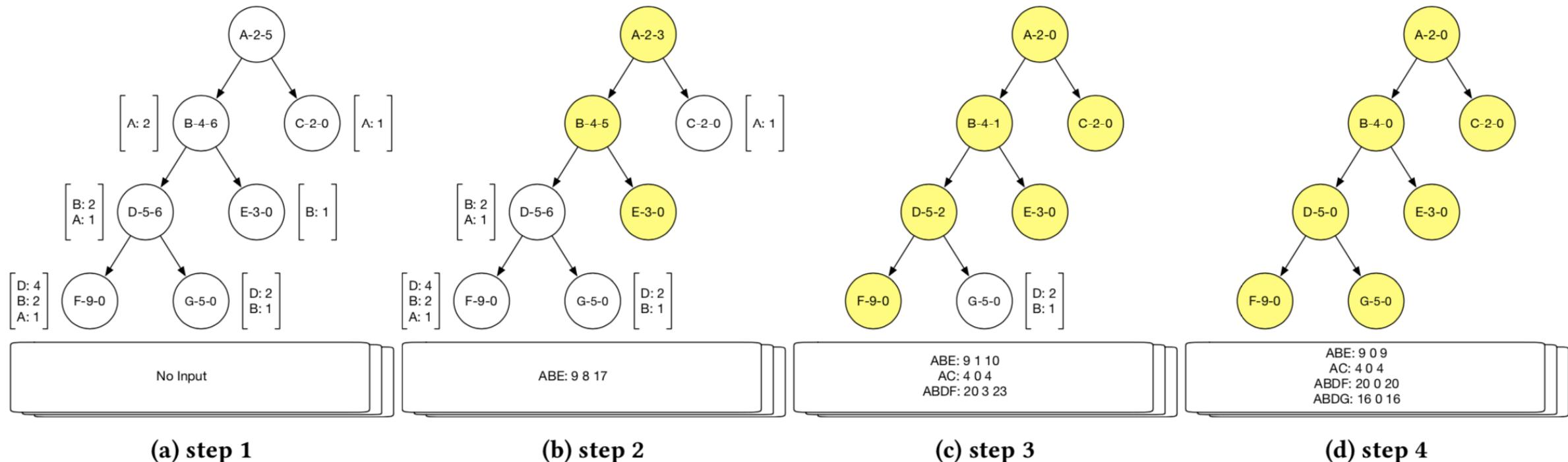


Figure 4: A step-by-step demonstration of the dynamic scoring algorithm*

多目标评估器和功率调度器

多目标评估器：

通过各种度量对种子进行优先级排序

- 文件大小
- 执行时间
- 覆盖变数
- 是否带来新的覆盖率
- 其执行跟踪的静态复杂性得分

功率调度器：

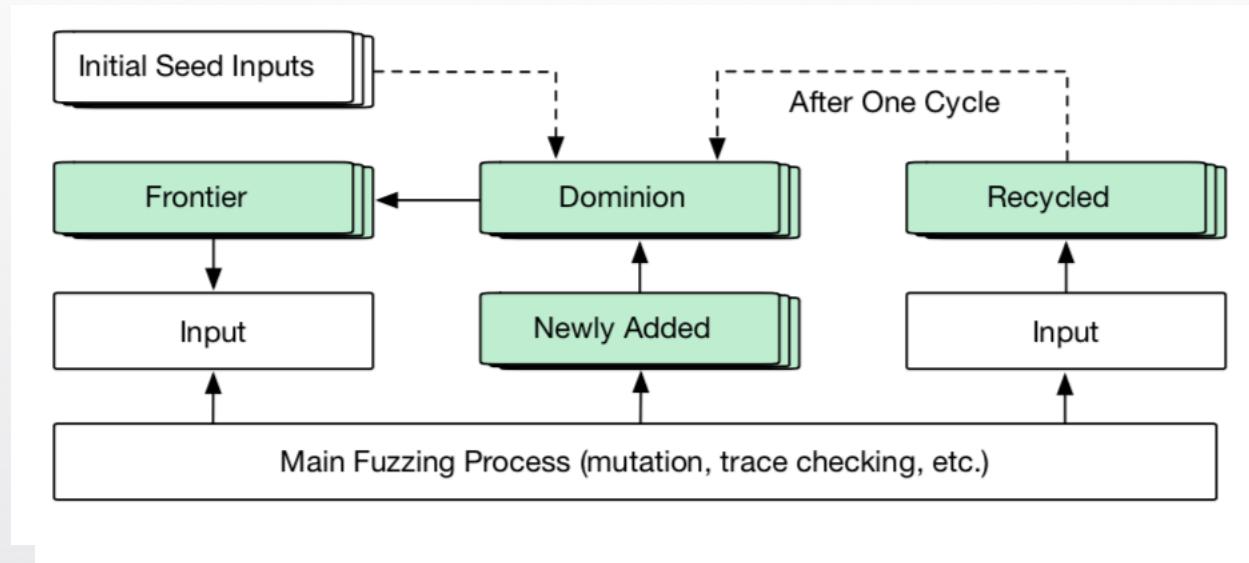
AFL等传统的fuzzer是根据种子的质量分配power能量

$$energy = allocate_energy(q_i)$$

但是cerebro中， 功率调度器设定能量还和 动态潜力得分和固有复杂性

$$energy' = energy \cdot s_f \cdot \frac{i_s}{a_s} \quad (11)$$

种子队列的结构



实施细节



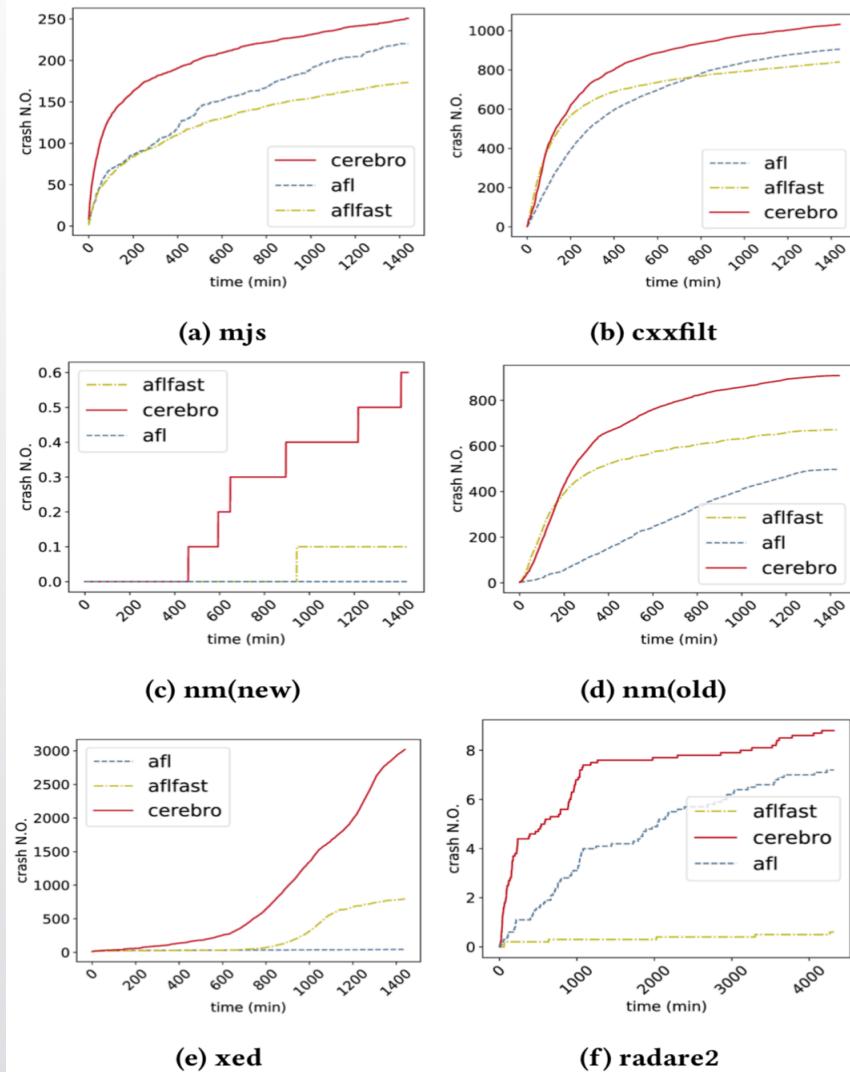
Cerebro的主要代码不是基于afl的，而是使用RUST语言编写的。

- afl-fuzz的源码紧凑，很难修改
- Rust的语言特性使得我们可以使用一些抽象的东西，使fuzzer更加模块化，易于拓展。

```
"func_map": {  
    "133": {  
        "file": "mjs/mjs.c",  
        "bonus_map": {  
            "132": 12,  
            "57": 3,  
            "102": 1,  
            "182": 1,  
            "98": 6,  
            "134": 12,  
            "91": 1,  
            "93": 1,  
            "92": 3,  
            "95": 3,  
            "94": 1  
        },  
        "score": 51,  
        "func": "mjs_disasm_single",  
        "dyn_score": 0  
    },  
    "132": {  
        "file": "mjs/mjs.c",  
        "bonus_map": {},  
        "score": 4,  
        "func": "mjs_disasm",  
        "dyn_score": 12  
    },  
}
```

Mjs的评分文件代码段，使用了json格式，
其中包含了所有计算出的复杂度得分和初始潜质得分。

Cerebro发现的crashes与afl/afl-fast对比



CEREBRO; AF stands for AFLFAST; A stands for AFL.

Project	Average Crash #			\hat{A}_{12}	
	C	AF	A	AF	A
mjs	250.7	173.3	223.8	0.97	0.85
cxxfilt	1031.7	840.4	905.3	1.00	0.92
nm(old)	908.5	671.0	497.5	0.91	1.00
nm(new)	0.6	0.1	0.0	0.56	0.60
radare2	8.8	0.6	7.2	1.00	0.66
xed	3019.0	797.0	44.7	0.90	1.00

总结



- 本文提出了输入潜能和用于灰盒模糊测试种子评估的MOO模型，解决了灰盒模糊测试的两个普遍问题：种子优先级划分和功率调度
- Cerebro是对fuzzer的优化，改进了漏洞挖掘的效率，优于AFL/AFL-fast