Fuzzing-Reward-Chain: A Novel Blockchain for Incentivized Bug Discovery with Proof-of-Fuzzing Consensus

***Abstract*—传统的PoW（Proof of Work）共识机制存在着大量的计算资源浪费问题，大量非记账矿工的算力消耗最终被丢弃。同时，现有的模糊测试（Fuzzing）漏洞检测方法在面对复杂系统时，容易出现饱和（Saturation）效应，难以高效发现新的漏洞。本文提出一种创新性的解决方案，将模糊测试与PoW区块链相结合，利用PoW中被浪费的算力进行漏洞检测。具体而言，我们将模糊测试过程转化为PoW中的工作量证明，矿工通过对目标程序进行模糊测试来竞争区块打包权。这种机制不仅有效利用了PoW中的算力，还能够激励矿工更积极地参与漏洞检测，从而提高区块链系统的安全性。**

I. Introduction

a) *PoW*机制的算力浪费问题*:*

b) 模糊测试方法的饱和效应*:*

c) 本文提出的解决方案：将模糊测试与*PoW*结合*:*

II. Background

a) 技术背景*:*

III. Overview

A. 基于模糊测试的*PoF*共识机制

一、模糊测试工作量证明

模糊测试的配置:

- 明确目标程序（google，apple，microsoft漏洞奖励计划, 中心化分发）
- Fuzzing工具及策略（AFL)
- 期望的输出（任务信息、测试结果、路径覆盖率数据）

1. 将模糊测试过程转化为工作量证明

1) 执行路径的获取：每个矿工需要实际执行被测程序，输出本次测试的输入以及对应的执行路径（$payload$, $path$, is_crash）(已完成)。

2) 固定执行窗口：由于被测程序每次在执行时所耗费的时间不一致，而原有的hash计算一次的时间固定，所以为了对两者进行映射，采用了固定执行窗口，每次程序执行一个固定窗口后会将当前执行挂起，根据当前状态进行一次hash计算。完成验证后继续执行。(已完成)

3）为了确保矿工可信执行了挂起操作，将执行逻辑放入到SGX中，确保定时挂起，在给出的hash值时包含了该enclave的身份信息，私钥签名。（未完成）

4）矿工广播的计算结果：基于基本块的路径hash编码，根据1中输出的path（路径表示的基本块编号序列），对路径序列进行哈希计算，得到一个哈希值。每个矿工广播的路径hash值计算方式如下：输入：$payload, path, is\_crash, salt, prev\_block, transctions$, 其中$prev_block$为上一个区块的hash，$transctions$为一段时间内网络中发生的交易打包成一个区块,计算:

H = $sha256(prev\_block, transactions, payload, path, is\_crash, salt, timestamp)$

根据3的要求，最后广播值为$Sign(H, Sk_{enclave})$ (未完成)

5）获得记账权时：由于路径哈希值的分布是不均匀，引入了动态可变映射的hash区间，通过反馈的方式动态调整hash区间，确保在固定时间块内有人一定可以获得记账权，即所探索的路径的hash值会落入到区间内。其算法见 1：(未完成)

Algorithm 1 动态可变映射哈希区间调整算法

Require: 当前区块哈希值$H_{current}$，目标出块时间T_{target}，实际出块时间T_{actual}，调整参数α
Ensure: 下一个区块的目标哈希值集合S_{target}

$S_{target} \leftarrow \emptyset$
$H_{base} \leftarrow H_{current} + \Delta H$ // ΔH 为预设偏移量
$N \leftarrow$ 预设集合大小
if $T_{actual} > T_{target}$ **then**
  $R \leftarrow \alpha(T_{actual} - T_{target})$
  for $i = 0$ to $N - 1$ **do**
    $H_i \leftarrow H_{base} + i \cdot R$
    $S_{target} \leftarrow S_{target} \cup \{H_i\}$
  end for
else
  $R \leftarrow \frac{1}{\alpha}(T_{target} - T_{actual})$
  for $i = 0$ to $N - 1$ **do**
    $H_i \leftarrow H_{base} - i \cdot R$
    $S_{target} \leftarrow S_{target} \cup \{H_i\}$
  end for
end if
return S_{target}

确定好动态映射区间后，如果有需要可以采用下面的线性hash区间映射算法(2)，将求得的hash进行映射。

2. 定义模糊测试难度和奖励机制

发现并提交有效漏洞的矿工获得额外奖励。奖励金额可以根据漏洞的严重程度、影响范围、修复难度等因素进行评估。

模糊测试难度指标：（1）代码覆盖率：将达到特定代码覆盖率（例如行覆盖率、分支覆盖率、函数覆盖率）作为难度指标。覆盖率越高，难度越大，奖励也应相应提高。（2）执行时间/资源消耗：记录模糊测试过程中的执

Algorithm 2 线性哈希区间映射算法

Require: 哈希值H, 目标哈希区间最小值min, 目标哈希区间最大值max
Ensure: 区间值I

$range \leftarrow max - min$
$hash\_max \leftarrow 2^{hash\_bits} - 1$ // $hash\_bits$ 为哈希值位数
$I \leftarrow \frac{H}{hash\_max} \cdot range + min$
return I

行时间或资源消耗（例如CPU时间、内存占用）。执行时间越长、资源消耗越大，难度越大，奖励也应相应提高。（3）程序复杂度：根据目标程序的复杂度（例如代码行数、函数数量、依赖关系）来调整难度。程序越复杂，难度越大，奖励也应相应提高。

模糊测试难度计算公式：

$$D = w_1 \cdot C + w_2 \cdot R + w_3 \cdot P$$

其中：
D：模糊测试难度
C：代码覆盖率因子
R：资源消耗因子
P：程序复杂度因子
$w_1 \Delta w_2 \Delta w_3$：各个因子对应的权重
各因子计算方法：

$$C = \frac{LC + BC + FC}{3}$$

其中：
LC：行覆盖率
BC：分支覆盖率
FC：函数覆盖率

$$R = \frac{T}{T_{max}} + \frac{M}{M_{max}}$$

其中：
T：模糊测试执行时间
T_{max}：最大允许执行时间
M：模糊测试内存占用M_{max}：最大允许内存占用

$$P = \frac{L}{L_{max}} + \frac{F}{F_{max}} + \frac{D_{num}}{D_{max}}$$

其中：
L：代码行数
L_{max}：总代码行数
F：函数数量
F_{max}：总函数数量
D_{num}：依赖库数量
D_{max}：总依赖库数量

奖励机制：（1）基本奖励：成功打包区块的矿工获得基本奖励，类似于传统PoW机制。基本奖励可以根据区块高度、难度等因素进行调整。$Reward = k * D$.（2）漏洞奖励：发现并提交有效漏洞的矿工获得额外奖励。奖励金额可以根据漏洞的严重程度、影响范围、修复难度等因素进行评估(见二)

二、漏洞验证与奖励分配

1. 设计漏洞验证机制，确保漏洞的真实性和有效性 (未完成)

提交执行路径的矿工给出payload，其他矿工通过该payload进行验证。

- 验证测试结果：检查矿工提交的测试结果是否正确，是否存在漏洞，是否是enclave签名的
- 验证任务信息：确认矿工执行的是指定的目标程序和测试用例

2. 根据漏洞的严重程度和贡献度，合理分配奖励 (未完成)

根据漏洞获得的CVE的分级，确定其严重程度S_{bug}, 根据不同矿工在本次记账是贡献的执行的路径占比计算器贡献度C_{bug}，（引入博弈论设计奖励的标准）奖励机制的目标是达到纳什均衡，即每个矿工在给定其他矿工策略的情况下，都无法通过改变自己的策略来获得更高的收益。这样可以促使矿工选择最优策略，从而提高整个系统的效率和安全性。除了追求纳什均衡，引入惩罚机制可以进一步改变矿工的收益矩阵，使得作假行为的预期收益降低，从而促使矿工选择诚实参与的策略。

（考虑是否引入声誉系统，记录矿工的历史行为。声誉高的矿工可以获得更高的奖励或更多的特权，而声誉低的矿工则可能受到限制或惩罚。）

- 基本奖励：每个成功打包区块的矿工获得基本奖励B。
- 漏洞奖励：发现漏洞的矿工获得漏洞奖励V，其中V与漏洞的严重程度S 成正比，即$V = k_1 \cdot S$。
- 惩罚：若矿工被证实提供虚假信息，将受到惩罚P，其中P 与作假行为的严重程度F 成正比，即$P = k_3 \cdot F$。

Algorithm 3 基于博弈论的模糊测试奖励与惩罚机制（还需细化）

Require: 矿工m，区块高度h，漏洞严重程度S_{bug}，贡献度C_{bug}，作假行为严重程度F
Ensure: 奖励或惩罚金额R

$B \leftarrow$ 基本奖励(h)
$V \leftarrow k_1 \cdot S_{bug}$
$C_r \leftarrow k_2 \cdot C_{bug}$
$P \leftarrow k_3 \cdot F$
if m 发现漏洞 **then**
    $R \leftarrow B + V + C_r$
else if m 被证实作假 **then**
    $R \leftarrow -P$
else
    $R \leftarrow B + C_r$
end if
return R

B. Example

- 矿工从区块链获取当前模糊测试任务（目标程序、AFL配置、覆盖率目标）。
- 矿工使用AFL对目标程序进行模糊测试，记录测试用例、执行结果、覆盖率信息。
- 矿工将测试结果和覆盖率数据打包成工作量证明，提交到区块链。

- 其他节点验证工作量证明，检查测试结果、覆盖率数据和任务信息。
- 如果验证通过，矿工获得区块奖励；如果发现有效漏洞，矿工还可获得额外奖励。

IV. Design

A. 区块链架构设计

结合PoW和模糊测试，新的区块链架构的设计，包括哪些核心的模块（修改或新增），对每个模块进行详细介绍

V. Implementation

先给出总体实现的概况，然后给出在具体实现时的一些难点是如何解决的

VI. Evaluation

A. 实验环境搭建

a) 选择目标程序和模糊测试工具*:*

b) 搭建区块链测试网络*:*

B. 性能分析*(Macro)*

a) 评估漏洞检测效率和准确性*:*

b) 分析系统吞吐量、延迟等性能指标*:*

C. 性能分析（*Micro)*

a) 评估新增或修改部分中不同的技术在性能上取得的收益*:*

D. 安全性分析

a) 评估系统抵御攻击的能力*:*

b) 分析奖励机制的公平性和安全性*:*

VII. Related work

补充与本论文研究内容相关的研究工作，主要围绕区块链的算力浪费

VIII. Conclusion

IX. Acknowledgment

References