[ASE 22] Effectively Generating Vulnerable Transaction Sequences in Smart Contracts with Reinforcement Learning-guided Fuzzing

Jianzhong Su, Hong-Ning Dai, Lingjun Zhao, Zibin Zheng, Xiapu Luo

挑战

- 现有的智能合约的模糊测试工具专注于满足生成复杂条件的特定输入，而忽略了交易序列的**顺序**。
- 现有的 *coverage-guided fuzzers* 容易生成有**高代码覆盖率、但是不易受到攻击**的交易序列 (non-vulnerable transaction sequences)
- 现有的 *vulnerability-guided fuzzers* 可能陷入局部最优解，仅生成**包含单个易受攻击的交易序列**，而忽略需要调用多个函数才能触发的漏洞。

方法

- 首先将智能合约模糊测试过程建模为马尔可夫决策过程 (Markov decision process)
- 在RFL的奖励机制中，兼顾漏洞性 + 代码覆盖率
- 通过DQN网络 + 经验回放训练生成触发合约漏洞的函数序列

技术细节

将模糊测试建模为 MDP

马尔可夫决策过程可以建模决定序列；

一个交易包含了需要调用的函数、函数的参数和其他元素。由于函数调研在交易中起到决定性作用，作者将交易序列的生成 转换为 函数调用序列的生成；

We define $s(SC, \bar{f})$ as the state of Agent, where s contains the features of the tested contract SC and previous function-call sequence $\bar{f} = \{f_1, f_2, ..., f_n\}$, which contains each function executed by the tested contract. At each step, Agent selects f by its policy π as the new action a, which is expressed as follows,

$$a \leftarrow \pi\big(s(SC, \bar{f})\big), \quad a \in A, \quad s \in S,$$

$$(s', r) \leftarrow P\big(s(SC, \bar{f}), a\big), \quad a \in A, \quad s, s' \in S,$$

其中 P(s, a) 是基于测试合约的转换函数，s 是状态，a 是行为，π是策略，r是奖励，SC是测试合约，f是前置函数调用序列。任务目标就是构建 Π*，使得 r 的总值最大；

$$\pi^* \leftarrow \arg\max_{\pi} \sum_{t=0}^{M} r_{t+1}^{\pi}.$$

RLF 系统

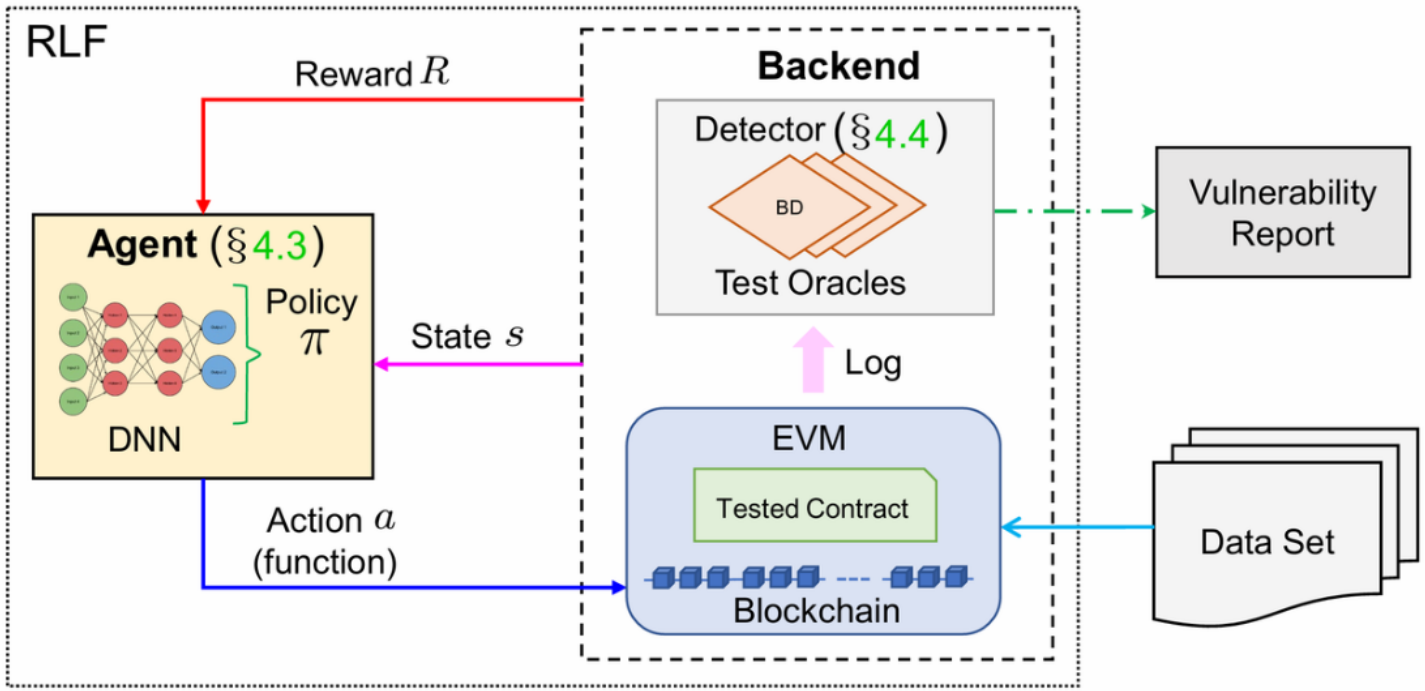

Figure 2: Overview of RLF. At each step, Agent selects an action to Backend, Backend returns the state and reward to Agent.

RLF 系统由 后端 Backend 和 Agent 两部分组成，其中 Backend 包含 以太坊虚拟机 (EVM) 和 Detector，负责执行合约并基于测试预言检测漏洞；Agent 利用 DNN 根据状态s 和奖励R 选择下一步调用的函数 (action/function)。两者形成闭环交互，从而不断生成更有效的函数调用序列以发现漏洞。

Agent

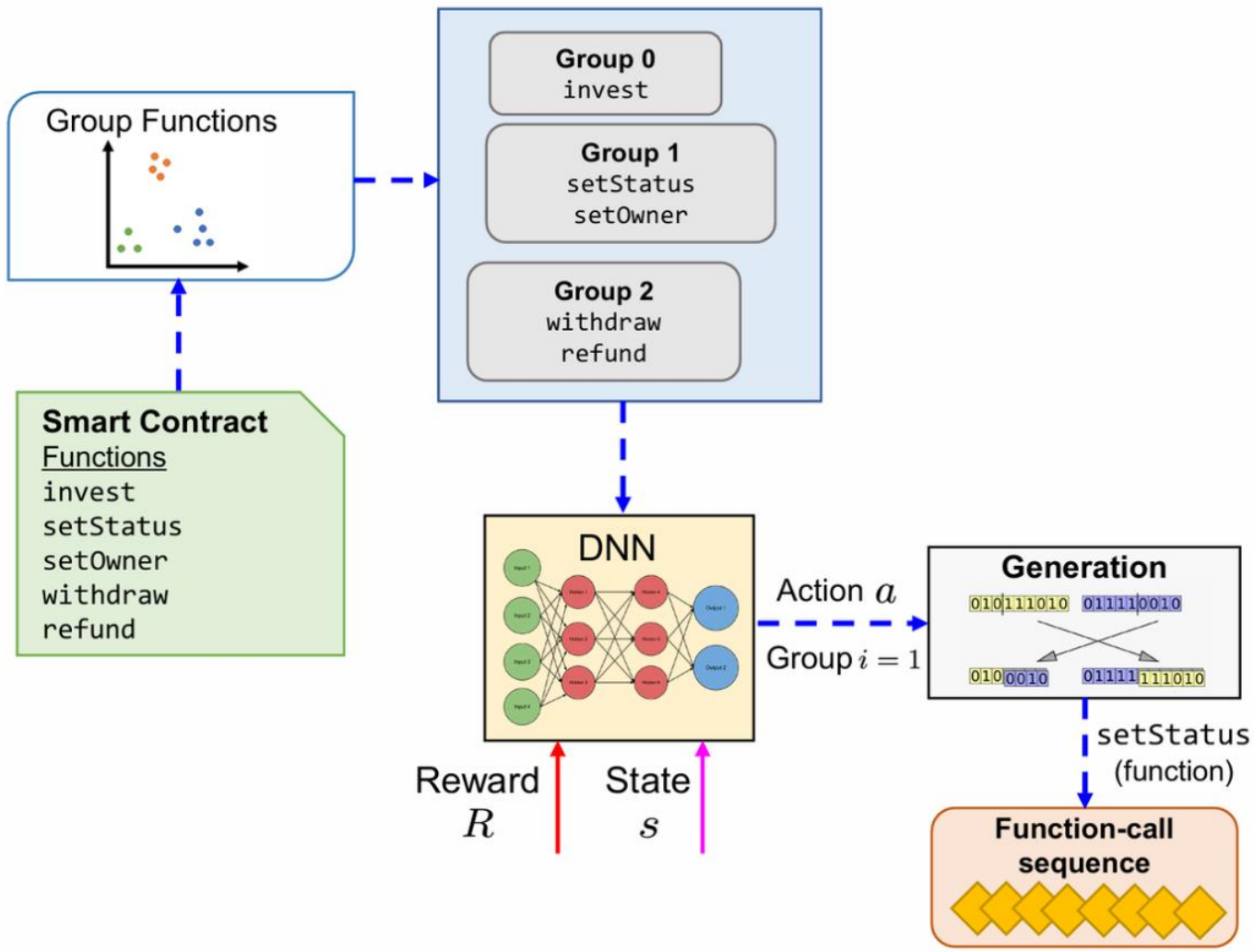

Figure 3: Working flow of Agent. At this step, DNN chooses Group 1 as Action and function setStatus is selected to execute tested contracts.

Action

将函数根据对应的**持久化状态变量的操作**，分类为若干组别以简化函数空间。具体而言，智能体不直接选择函数，而是先选定函数组作为action，再从该组随机抽取函数。

> 触发漏洞的函数序列，通常与智能合约的持久化状态变量 (Persistent Status) 相关；例如，某些漏洞的触发需要某些状态变量的满足作为前置条件；
>
> 智能合约有三类状态：balance, storage, existence
>
> 余额、存储和存在。余额指智能合约持有的以太币数量；存储指存储在区块链中的变量，这些变量在交易序列中持续存在；存在则表示智能合约尚未被销毁。要改变智能合约的状态，需调用对应的状态操作。

Table 1: Status operations in function.

Operation	Description
Payable	The function can receive ether from other addresses.
Call	The function can transfer ether or invoke other contracts.
Store	The function can change the storage.
Selfdestruct	The function can destroy the contract.

State

通过融合**合约级执行状态（SC）**与**函数级语义与执行特征（F）**，构建了一个连续状态空间，通过 (SC, F)描述Agent的状态，使 Agent 能同时感知整体测试进度与局部函数复杂性，从而更有效地指导函数调用序列的生成。

Table 2: State of Agent. f is the last call function in $\bar{f}$.

	Features	Description
SC	*Action*	Frequency of each action.
	Trace	Proportion of 8 key opcodes being executed cumulatively.
	Coverage	Instruction and block coverage of tested contract.
F	*Revert*	Fraction of ending with `revert` when executing f.
	Return	Fraction of ending with `return` when executing f.
	Assert	Fraction of ending with `assert` when executing f.
	Call	Frequency of f being executed in $\bar{f}$.
	Coverage	Instruction and block coverage of function f.
	Arguments	Number of arguments of f.
	Opcodes	Counts of 50 representative opcodes in function f.
	Name	Word embedding of f's name.

Agent 神经网络

Algorithm 1: Workflow of Training RLF.

input : Iteration steps T, Sample size M, Search rate ϵ, Episode E, Smart Contract SC

output : Network Q

$Q \leftarrow$ `InitializeNetwork`(); $s \leftarrow$ `getInitState`();

$A, G \leftarrow$ `groupFunctions`(SC);

for $k \leftarrow 1$ *to* T **do**

  $a \leftarrow$ `getRandomOrBestAction`(A,Q, ϵ);

  $f(x) \leftarrow$ `selectFunction`($G[a]$);

  $s', r \leftarrow$ `executeFunction`($f(x)$,SC);

  `saveExperience`(s, a, r, s');

  $s \leftarrow s'$;

  if k *MOD* $E == 0$ **then**

    $\{(s_i, a_i, r_i, s_i')\} \leftarrow$ `loadExperience`(M);

    $Q \leftarrow$ `train`($\{(s_i, a_i, r_i, s_i')\}$, Q);

  end

end

1. 在智能合约SC上，初始化神经网络 Q、状态s、函数组 A, G； 总共训练 T 步。
2. 每一步中，
   a. 有 ϵ 概率选择随机动作，反之根据智能体选择动作，得到动作 action
   b. 从函数组 G[a] 中 随机选择函数，得到函数 f(x)
   c. 在合约 SC 上执行函数 f(x)，得到下一状态 s' 和 即时反馈 r
   d. 保存经验 (s, a, r, s') （为了通过经验回放更新 Q网络）

d. 保存经验 (s, a, r, s') （为了通过经验回放更新 Q网络）

> DQN (深度 Q-learning 网络）中，**经验池（Experience Replay Buffer）是一个存储智能体交互经验的核心组件**。它通过一个循环队列实现，将每一步的交互经验存储到池中，在满的时候会覆盖旧的经验。
>
> 它像一个智能的"记忆仓库"，持续保存每一步的交互数据——包括当前状态(s)、采取的动作(a)、获得的奖励(r)、转移后的新状态(s')以及回合是否终止的标志。当经验积累到一定数量后，训练时**不再按时间顺序使用最近的经验，而是从这个池中随机抽取小批量历史经验**来更新Q网络。这种设计能**有效打破连续状态间的强相关性**，避免神经网络因学习高度相关的序列数据而产生训练不稳定或发散；同时**显著提高数据利用效率**，使每条经验可被多次重复学习，还能**混合不同时期、不同质量的经验**，让智能体获得更全面的状态-动作值估计，从而稳定训练过程并提升最终性能。
>
> 没有经验池的问题：
> 状态1 → 状态2 → 状态3 → ... （强相关）
> ↓ ↓ ↓
> 学习 学习 学习 ← 连续相关数据不好
>
> 有经验池的解决方案：
> 从经验池随机采样：
> 状态3 ↰
> 状态1 ↰ ← 不相关的数据
> 状态5 ↰
> ↓
> 学习

e. 每 E 步进行一次经验回放训练，通过经验池中的历史数据更新Q网络。

实验评估

实验设置

在训练阶段，我们将ϵ设为较大值，使智能体能尽可能探索更多状态和动作；而在测试阶段，我们将ϵ降至0.15，使智能体能够利用训练好的神经网络生成的结果。

数据集: D1 来源于标注数据集 [31]，包含 Ether-Leaking 和 Suicidal Contract 两类依赖交易序列触发的漏洞。为保证工具可比性，仅选取所有工具均可分析的 85 个漏洞合约，其中涉及 108 个 Ether-Leaking 函数和 46 个 Suicidal Contract 函数，部分合约为人工构造或修改。 D2 为真实世界数据集，来自以太坊主网 Solidity 0.4.25 合约（采集自 XBlock），去重后随机抽取 1,206 个合约。

Table 5: Datasets Used.

ID	Source	Used For	Avg. LoC	Num. of Contracts
D1	VeriSmart [2]	RQ1, RQ2, RQ3	330	85
D2	XBlock [6]	RQ3	343	1206

函数在动作中的划分

Table 6: The number of functions in each group (action) that are classified by status operations. The function group with ● (○) means the functions contain (not contain) the status operations. The function group with □ means the functions contain either status operations or no status operations.

	Payable	*Call*	*Store*	*Selfdestruct*	**Number**
Group 1	●	●	□	○	51
Group 2	○	●	□	○	242
Group 3	●	○	□	○	97
Group 4	○	○	●	○	659
Group 5	□	□	□	●	55
Pure/View Functions	○	○	○	○	929

RQ1: Rewards 的 影响

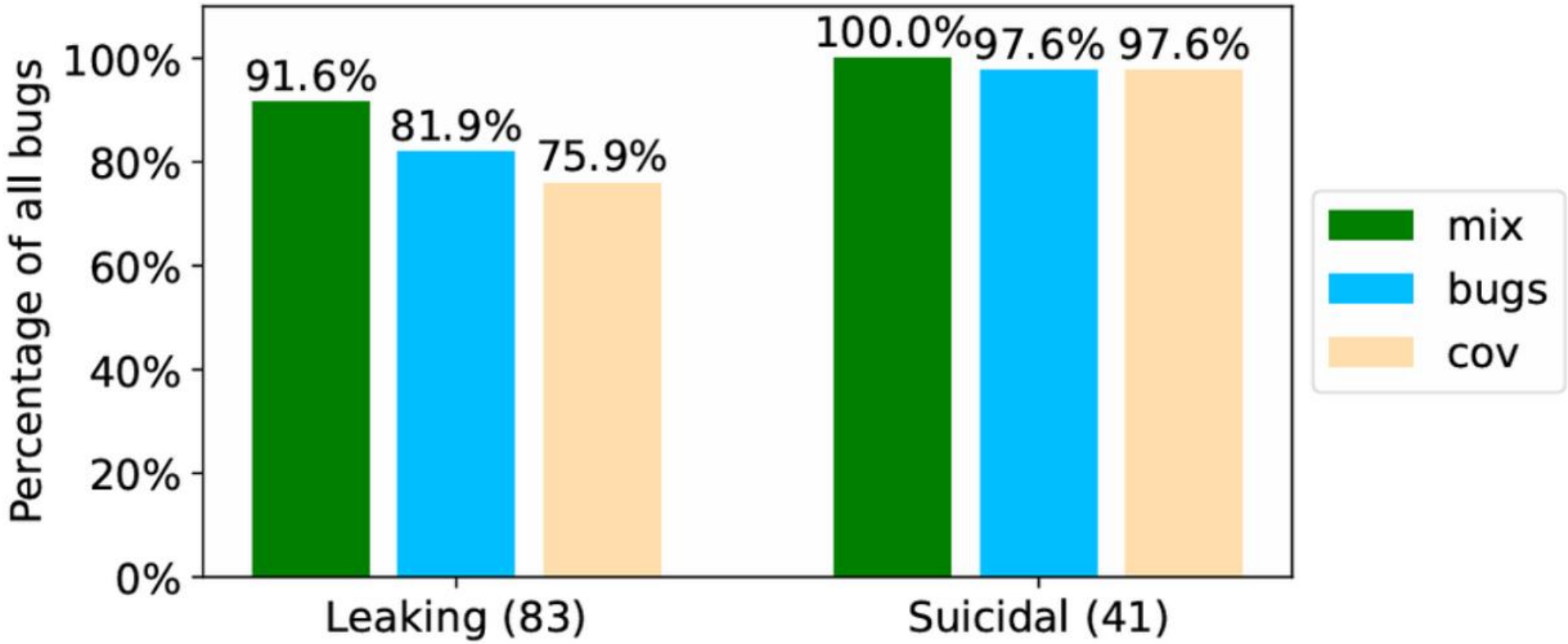

Figure 4: Detected vulnerabilities by fuzzers with different rewards. The number of all vulnerabilities is computed as the union of all vulnerabilities detected by each fuzzer.

RQ2: RLF 的效率

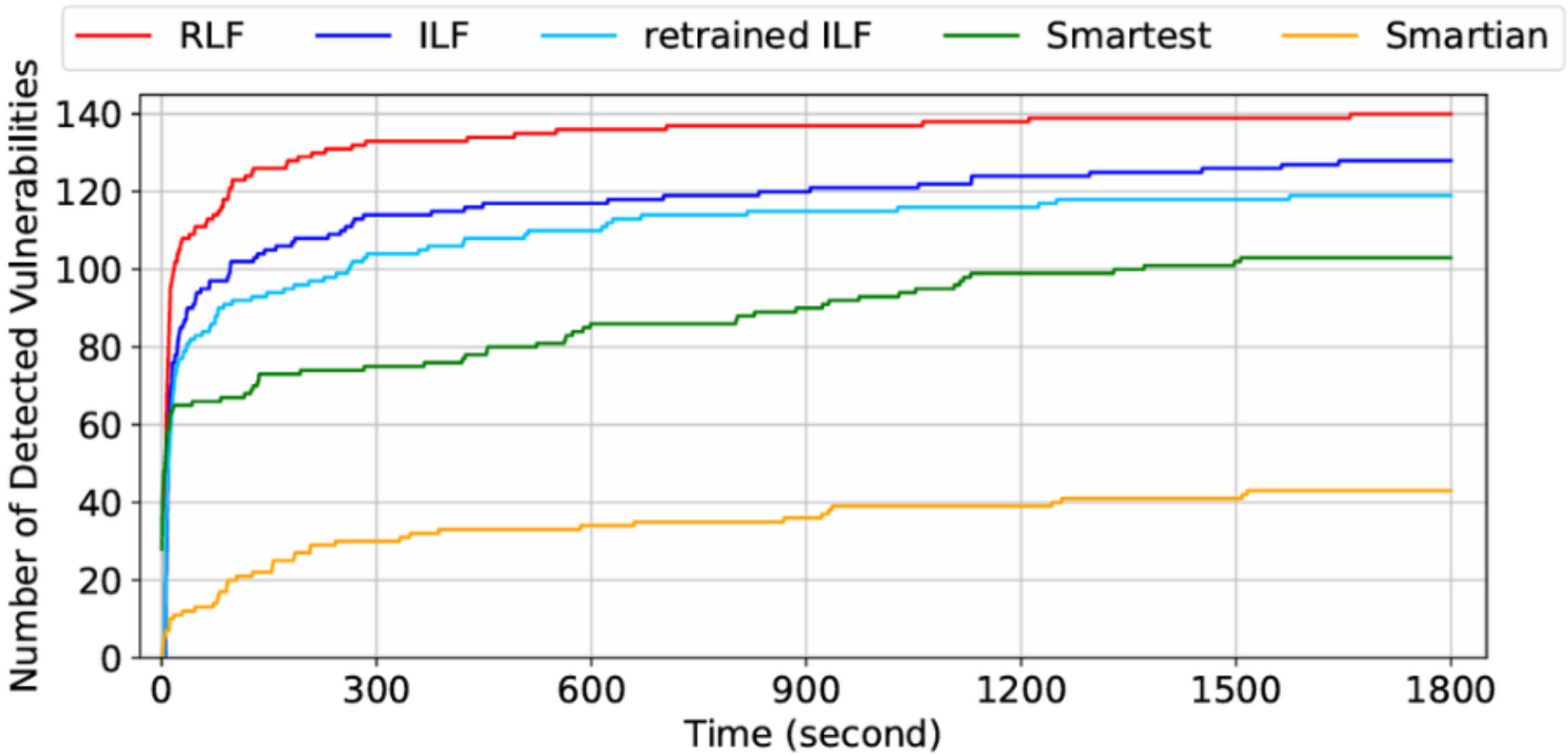

Figure 5: True vulnerabilities being found versus time (second) by different tools in D1.

RQ3: RLF 报告bug的能力

Table 7: Vulnerabilities reported by different tools in D1. The number in ( ) is the number of functions with corresponding vulnerabilities. TP is the number of True Positives and # is the number of reported vulnerabilities.

	Tools	EL (108)		SC (46)		ALL (154)	
		TP	#	TP	#	TP	#
Symbolic Executors	Mythril [25]	9	12	26	26	35	38
	SmarTest [31]	62	63	41	41	103	104
Fuzzers	Smartian [11]	21	21	22	22	43	43
	$ILF_{retrained}$ [15]	78	78	41	41	119	119
	ILF [15]	86	88	43	43	129	131
	our RLF	98	100	43	43	141	143

总结

提出一种基于强化学习的漏洞引导模糊测试器（RLF），用于有效生成智能合约中的易受攻击交易序列。具体而言，首先将智能合约模糊测试过程建模为马尔可夫决策过程（MDP），以此构建强化学习框架。随后设计兼顾漏洞利用率与代码覆盖率的新型奖励机制，引导RLF生成针对性交易序列以揭示漏洞——尤其针对涉及多函数的复杂漏洞。最后，为RLF设计神经网络以自动学习历史序列，从而实现高效生成漏洞交易序列。在有限时间内，该RLF检测到的漏洞数量超越其他尖端工具。