Lanturn: Measuring Cryptoeconomic Smart Contract Security through Learning

Kushal Babel
Cornell Tech, IC3
SBC'23

with Mojan Javaheripi, Mahimna Kelkar, Yan Ji, Farinaz Koushanfar, Ari Juels
Smart Contract Security

Inverse Finance exploited again for $1.2M in flash loan oracle attack

No user funds have been affected by the exploit, but Inverse Finance offered the attacker a bounty to return the stolen funds.

DeFi Lending Protocol Fortress Loses All Funds in Oracle Price Manipulation Attack

Solana-Based Decentralized Finance Platform Hit by $100 Million Exploit

Mango's MNGO token was down over 40% after suffering from the latest mass decentralised finance exploit.

Solana DeFi Protocol Nirvana Drained of Liquidity After Flash Loan Exploit

The price of the protocol's ANA token fell almost 80% following the attack.
Complex Interactions

• Smart-Contracts can interoperate very easily.

• Examples:
  1. Lending contract can use Decentralized Exchange (DEX) contract as price oracle.
  2. Multiple DEX contracts can be aggregated together.
  3. FlashLoans + DEX
  4. FlashLoans + Lending Contract ...
MEV, Informally...

MEV = Maximal Extractable Value (or Miner Extractable Value)

Ability of miners/validators/bots to extract value by reordering, inserting or censoring transactions

$EV = \text{Value extracted in a given transaction sequence}$

$MEV = \max(EV \text{ for any transaction sequence})$

We can use MEV as the measure of Cryptoeconomic Security, as it models the worst case adverserial advantage.
Talk Outline

• Previous approaches
• **Lanturn**
  • Overview
  • Optimization Module
  • Simulation Module
• Evaluation
• Limitations
• Conclusion
Prior Approaches: Heuristics Based

• Most works encode a specific attack strategy, essentially looking for patterns in transaction data and blockchain state

• Highly efficient at finding patterns such as arbitrage, sandwich attacks, but...

• These approaches do not generalizable to new contracts or new transaction types

• As a result, these approaches do not attempt to find the worst-case adversarial advantage

Prior Approaches: Clockwork Finance

- Leverage formal verification to find the optimal value of the inserted transactions and optimal order of transactions that maximize MEV.

- Formal guarantees on the optimality of obtained MEV.

- Not scalable for complex contracts such as contracts with loops (Curve Finance), or large number of transactions (>10)

Prior Approaches

Heuristics Based

Efficient but not generic

Formal Methods

Generic but not scalable

Lanturn

Generic and scalable learning-based tool to find MEV opportunities and understand the economic security risks of smart contracts and their composition
Lanturn Properties

- **Generalizability**: No encoding of strategies and contract-specific heuristics, black box execution of smart contract, but requires templates for insertions based on the interface.

- **Native Smart-Contract Execution**: Simulates smart contract bytecode, profit-yielding strategies are directly executable on-chain.

- **Scalability**: Scales with both contract complexity and number of transactions (>50 transactions, >10 insertions).

- **Adaptability to computation budget**: Optimization algorithm can be tuned to match the computation budget and amenable to parallelization.
Lanturn Applications

• **Developers and researchers**: Directly enables developers and researchers to understand the cryptoeconomics of smart contracts

• **Users**: Understand the value that can be extracted from their transactions

• **Strategic Agents**: Use Lanturn to extract value and discover new strategies. Real-time usage can be supported by parallelization across servers
Problem Overview

• Given a pool of user transactions and the current blockchain state, Lanturn automatically learns to maximize the validator’s EV.
Problem Overview

The strategies include:

- How to order the transactions in the block?
- What transactions can the validator insert to take advantage of the block?
Problem Overview

• The strategies include:
  • How to order the transactions in the block?
  • What transactions can the validator insert to take advantage of the block?

• For the latter question, the validator transactions are selected from a set of templates like:

\[ \text{swap } \alpha_0 \times \text{token}_0 \text{ with } \alpha_1 \times \text{token}_1 \]

In this example,
• The tokens are filled with those that user transactions have interacted with.
• The alpha values (template-variables) are found by Lanturn such that they maximize EV.
System Overview

• Learning is done through iterative interactions between our optimization module and simulator.
Deeper Look Inside

- High-level view of the internals of Lanturn components:
• Inputs to the optimizer (1):
  • User transactions to be ordered
  • Templates for validator-inserted transactions
Deeper Look Inside

- Output of the optimizer (3):
  - Maximal EV
  - Optimal transaction order
Optimization has two hierarchical learning loops:

1. The outer loop learns the optimal order of transactions (6).
2. The inner loop learns the optimal template-variables for the validator (7).
Lanturn Learning-based Optimizer

• Problem Formulation:

\[
\max_{\vec{x}} F(\vec{x})
\]

1. Objective function \((F)\): Validator’s EV in terms of Eth
2. Design Variables \((\vec{x})\): Order of transactions \((x_o)\), transaction amounts \((x_a)\)

Bi-level optimization:

\[
\max_{\vec{x}_o \in X_o} F(\vec{x}_o)
\]

where \(F(\vec{x}_o) = \max_{\vec{x}_a \in X_a} f(\vec{x}_o, \vec{x}_a)\)

Find the best transaction order

Find the best values for template-variables

3. Optimization Bounds \((X_o, X_a)\): Accepted range of values for each design variable, a.k.a, the search space
Goal: empirically search for optimal vector:

$$\bar{x}^* = \arg\max_{\bar{x} \in X} F(\bar{x})$$

What does optimality mean?  
- High EV for the validator
**Goal:** empirically search for optimal hyperparameter vector:

\[ \hat{x}^* = \arg\max_{\vec{x}} F(\vec{x}) \]

**What does empirical mean?**
- No analytical solution
- Relies on accurate simulations of EV
Goal: empirically search for optimal hyperparameter vector:

$$\vec{x}^* = \arg\max_{\vec{x}} F(\vec{x})$$

Search-space is not small, even for the below toy problem with only 4 transactions, the search space covers almost 30 billion possibilities.

How to search the large space?
- Fast convergence
- Computational efficiency
- Parallelizable

User Transactions:
- User1 swaps 100 usdc for eth
- User2 swaps 4 eth for usdc

Template Symbolic Transactions:
- Validator adds $a_0$ liquidity units in price range $a_1$ to $a_2$
- validator swaps $a_3$ usdc for eth

Template-variables:

- $a_0 : \text{uint128}$; $a_1, a_2 : \text{int24}; a_3 : \text{uint256}$

Constraints:

- $a_0 \in \{1, 10, 100\}; 1000 < a_1 < 1500; 2000 < a_2 < 2500; 400 < a_3 < 2000$
Search-space Geometry

- Visualization for finding transaction amounts for an example problem with validator executing a "Just-in-time" liquidity strategy.

- **Search goal:** Identify the peak region.

- **Challenges:** In general -
  - Non-monotonic
  - Non-convex
  - High correlations between variables
  - High-dimensional space

- horizontal plane: Transaction amounts
- vertical axis: objective function (EV)
Algorithm High Level Sketch

1. Initialize distribution $D$ to Uniform
2. $res = 0$
3. For $i = 1$ to $N$:
   4. $S = \text{Sample}(D)$
   5. $res = \max(res, \text{EV}(S))$
   6. $G = \text{Good}_\text{Samples}(S)$
   7. $D.\text{update}(G)$
8. return $res$
Simulation Module

1. Receive a concrete transaction sequence
1. Receive a concrete transaction sequence
2. Fork the blockchain state at height $H$
3. Give validator an initial capital (e.g. 10m Eth)
1. Receive a concrete transaction sequence
2. Fork the blockchain state at height $H$
3. Give validator an initial capital (e.g. 10m Eth)
4. Execute transaction sequence
1. Receive a concrete transaction sequence
2. Fork the blockchain state at height $H$
3. Give validator an initial capital (e.g. 10m Eth)
4. Execute transaction sequence
5. Convert any non-Eth tokens to Eth by executing DEX trades or using historical prices from CEX
1. Receive a concrete transaction sequence
2. Fork the blockchain state at height $H$
3. Give validator an initial capital (e.g. 10m Eth)
4. Execute transaction sequence
5. Convert any non-Eth tokens to Eth by executing DEX trades or using historical prices from CEX
6. Mine block $H+1$
7. Return the validator’s final Eth balance less initial balance.
Evaluation

• UniswapV3, UniswapV2, Sushiswap, Aave

• Dataset
  • Filter blocks that have more than 500 ETH trade on Sushiswap or UniswapV2, 1000 ETH on UniswapV3 or Liquidation event on Aave
  • CEX prices: Freely available historical minute-level price data (Binance)
  • Baseline: Flashbots data for MEV bribes paid to the validators per-bundle (pre-MEV-Boost), through gas fees and direct transfers
  • Template Transactions: Symbolic transactions for swapping in either direction, liquidity provision and removal, liquidation on Aave
AMM Trading – UniswapV2

- Frontrunning
- Sandwiching
- New Strategies
### Lazarus Strategy

Bring a dead transaction back to life

<table>
<thead>
<tr>
<th>Transaction Hash:</th>
<th>0x367e86b305d5a722b39a332ddfc203a4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status:</td>
<td>![Fail]</td>
</tr>
<tr>
<td>Block:</td>
<td>![14954940] 3073023 Block Confirmations</td>
</tr>
</tbody>
</table>

#### From:
0xbc1c16b50Ecf01bD1e4f6C2fE21887A67aC2eC33

#### To:
0x793FF66B5435A6De9D6d7e70C4C68522E8cD24E1 (MEV Bot: 0x793...4F)

Warning! Error encountered during contract execution [Reverted] 😞

Reorder, make it succeed, but then frontrun it!
Gas-leeching Strategy

• MEV Bots usually have safeguards in their transactions
• But some don’t!
• Many transactions, which pay a high gas fees, run into an infinite loop when reordered, and end up consuming all the gas in the block --- transaction fees goes to the validator!

Transactions

For 0x0000000000003b3cc22af3ae1eac0440bcee416b40 MEV Bot: 0x000...B40

A total of 378,509 transactions found
(Showing the last 100k records)
AMM Trading – UniswapV3

Frontrunning
Sandwiching
Lazarus, Gas-Leeching

MEV (in ETH)

Block number
UniswapV3 Liquidity Provision

Just-in-time (JIT) Liquidity
Multiple AMM Composition

![Graph showing MEV (in ETH) over block number for UniswapV2 and UniswapV3. The graph plots two lines: one for UniswapV2 and another for UniswapV2 + UniswapV3.](image-url)
AMM+Lending Contract Composition

Sushiswap + UniswapV3 + Aave

MEV (in ETH)

Block number

Lanturn
Flashbots
Execution Time

- Single Server
- AMD Ryzen Threadripper 3960X, 48 CPU threads, 128 GB RAM and SSD storage
- 44 parallel simulations
Limitations

1. Mild regularity conditions needed for learning
2. Manual specification of templates
Conclusion

• Formulation of cryptoeconomic smart-contract security as a learning task.
• The approach can generalize to any smart-contract by treating smart-contract bytecode as black box simulation environment, output executable as is on-chain.
• **Lanturn** can scale to MEV strategies spanning over large (>50) number of transactions efficiently, and without modelling (complex) smart-contract behavior.
• **Lanturn** uncovers significant MEV by discovering not only well-known strategies but new strategies as well.

Github: [www.github.com/lanturn-defi/lanturn](www.github.com/lanturn-defi/lanturn)
Mutations – Gaussian+Uniform Distribution
Mutations - Permutations

Original Transactions

- user 1, trans. 1
- user 1, trans. 2
- user 2, trans. 1
- miner, trans. 1
- miner, trans. 2

Intermediate Representation

- \( u_1, u_1, m_1, u_2, m_2 \)
- \( u_1, u_1, u_2, m_1, m_2 \)
- \( u_1, u_1, u_2, m_2, m_1 \)
- \( u_1, u_2, u_1, m_1, m_2 \)

Search Space

- 1 hop
- 2 hops