Clockwork Finance: Automated Analysis of Economic Security in Smart Contracts

To Appear in IEEE S&P’23

Kushal Babel
Philip Daian
Mahimna Kelkar
Ari Juels

(First three authors contributed equally)
DeFi
Smart Contracts, Like Clockwork!

• Smart contracts execute in sequential and atomic transactions
• Execution is deterministic
• Most blockchains have transparent execution
• Therefore: Easy interoperability among smart contracts and novel financial instruments
Money Legos

Source: https://medium.com/totle/building-with-money-legos-ab63a58ae764
Unintended Behaviour

Swap 1,000 USD into ETH
Unintended Behaviour

Swap 1,000 USD into ETH
Unintended Behaviour

Swap 1,000 USD into ETH

ETH Price
Unintended Behaviour

Swap $1,000 USD into ETH

Swap ETH into USD

Swap $X USD into ETH

Swap 1,000 USD into ETH

Swap $Y ETH into USD

Sandwich
Unintended Behaviour

MEV = Miner Extractable Value (or Maximal Extractable Value) - Ability to extract value by reordering, inserting or censoring transactions

Swap X **USD** into **ETH**

Swap **1,000 USD** into **ETH**

Swap **Y ETH** into **USD**

Sandwich
Contract Composition

Source: https://medium.com/totle/building-with-money-legos-ab63a58ae764
Contract Composition

- Flashloans + DEX

Source: https://medium.com/totle/building-with-money-legos-ab63a58ae764
Contract Composition

- Flashloans + DEX
- Lending contracts using DEX to price the debt

Source: https://medium.com/totle/building-with-money-legos-ab63a58ae764
Contract Composition

- Flashloans + DEX
- Lending contracts using DEX to price the debt
- Flashloans + Governance Contract

Source: https://medium.com/totle/building-with-money-legos-ab63a58ae764
Contract Composition

- Flashloans + DEX
- Lending contracts using DEX to price the debt
- Flashloans + Governance Contract
- DEX + DEX + DEX ...

Source: https://medium.com/totle/building-with-money-legos-ab63a58ae764
Unintended Behaviour

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.

By Sheurya Mahra  Jul 26, 2022 at 4:41 a.m. PDT  Updated Jul 26, 2022 at 8:06 a.m. PDT

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

No user funds have been affected by the exploit, but Inverse Finance has incurred debt and offered the attacker a bounty to

BAYC ApeCoin Suffers $800k Flash Loan “Attack” During Airdrop

Posted on Mar 30, 2022  BLOG
MEV...An Industry

$674,300,932
Total Extracted MEV

$6,930,451
Last 30 days Extracted MEV

$113k
Last 24h Extracted MEV

https://explore.flashbots.net
Existing Techniques for Security

- Human Auditing
- Fuzz Testing
- Static Analysis (eg. Slither)
- Formal Verification of functional correctness

Focus on Bug Hunting, Functional Correctness and Secret Leaks
This Work - Clockwork Finance

Directly reason about economic properties of smart contracts (and their interactions) by leveraging existing formal verification techniques

Unlike Traditional Finance, Smart Contracts execution is deterministic, sequential, transparent and atomic — allowing for formal verification of the behaviour of DeFi applications
Benefits to the ecosystem

Developers - Prove bounds on the value exposed by their contracts and interaction of their contracts with other contracts

Users - Find bounds on the value extractable from their transactions

Consensus Researchers - Rigorously study the impact of MEV on consensus
Outline

- Definitional tools
  - Defining (M)EV
  - Defining Secure Composition
- Practical Instantiation into Clockwork Finance Framework (CFF)
  - Design
  - Use for proofs
  - Use for finding attacks
Outline

• Definitional tools
  • Defining (M)EV
  • Defining Secure Composition
• Practical Instantiation into Clockwork Finance Framework (CFF)
  • Design
  • Use for proofs
  • Use for finding attacks
Miner Extractable Value (MEV)
Extractable Value (EV)
Extractable Value (EV)

Player $P$

State $S$

$3$-EV =

$$EV(P, B, s) = \max_{(B_1, \ldots, B_k) \in B} \left\{ \sum_{a \in A_P} \text{balance}_k(a)[0] - \text{balance}_0(a)[0] \right\}$$
Extractable Value (EV)
Extractable Value (EV)

Use MEV as the measure of economic security

Miner is the most powerful out of all permissionless players - MEV subsumes all other attacks
Secure Composition

State S

Valid Block
Secure Composition
Secure Composition

State S → Valid Block

State S → Valid Block
Secure Composition

State $S$  Valid Block

State $S' = S + C$  Valid Block

$C$ does NOT compose with $S$

$C$ does compose with $S$
Outline

• Closer look at an example

• Definitional tools
  • Defining EV
  • Defining Secure Composition

• Practical Instantiation into Clockwork Finance Framework (CFF)
  • Design
  • Use for proofs
  • Use for finding attacks
Formal Verification

Verify Property

Proof
Formal Verification

Program

State S

Verify Property

Proof

Counterexample
Clockwork Finance Framework (CFF)
Clockwork Finance Framework (CFF)

(Symbolic) Transactions = tx1, tx2, tx3

Verify Property: MEV < $\delta$
Clockwork Finance Framework (CFF)

(Symbolic) Transactions = tx1, tx2, tx3

Swap $X$ Eth for $Y$ USD
$X \geq 0$, $Y \geq 0$

Verify Property: $\text{MEV} < \delta$
Clockwork Finance Framework (CFF)

(Symbolic) Transactions = tx1, tx2, tx3

Swap $X$ Eth for $Y$ USD

$X \geq 0$, $Y \geq 0$

Verify Property: \( \text{MEV} < \delta \)
Clockwork Finance Framework (CFF)

(Symbolic) Transactions = tx1, tx2, tx3

Swap $X$ Eth for $Y$ USD
$X \geq 0, Y \geq 0$

Verify Property: MEV < $\delta$

Counterexample
• Human Readable Formal Specification

• KEVM - Formal Ethereum Semantics in K

CFF Design
CFF Design

CFF Language Model

Input - (Symbolic) State and Transactions → K Deductive Verifier
CFF Design

CFF Language Model

Codified CF Definitions

Input - (Symbolic) State and Transactions

K Deductive Verifier
CFF Design

CFF Language Model

Codified CF Definitions

Smart Contracts Code

Input - (Symbolic) State and Transactions

K Deductive Verifier
CFF Design

CFF Language Model

- Codified CF Definitions
- Smart Contracts Code
- EVM Semantics

Input - (Symbolic) State and Transactions

K Deductive Verifier
CFF Design

CFF Language Model

Codified CF Definitions
Smart Contracts Code
EVM Semantics

Input - (Symbolic) State and Transactions

K Deductive Verifier

Composability Proof
Counterexample Strategies
CFF Design

CFF Language Model

- Codified CF Definitions
- CFF Models
- EVM Semantics

Smart Contracts Code

Input - (Symbolic) State and Transactions

K Deductive Verifier

- Composability Proof
- Counterexample Strategies
CFF Models

(Symbolic) Transactions = tx1, tx2, tx3

Swap $X$ Eth for $Y$ USD
$X \geq 0, Y \geq 0$

Verify Property: $\text{MEV} < \delta$

$F(X,Y)$
(Symbolic) Transactions = tx1, tx2, tx3

Swap $\mathbf{X}$ Eth for $\mathbf{Y}$ USD  
$\mathbf{X} \geq 0$, $\mathbf{Y} \geq 0$

Verify Property: MEV < $\delta$

CFF models are over approximations of the smart contract code.
CFF Models

(Symbolic) Transactions = tx1, tx2, tx3

Swap $X$ Eth for $Y$ USD
$X \geq 0$, $Y \geq 0$

Verify Property: MEV < $\delta$

CFF models are over approximations of the smart contract code.
CFF Models

(Symbolic) Transactions = tx1, tx2, tx3

Swap \( x \) Eth for \( y \) USD
\( x \geq 0, y \geq 0 \)

Verify Property: MEV < \( \delta \)

CFF models are over approximations of the smart contract code.

False Positive
CFF Models

(Symbolic) Transactions = tx1, tx2, tx3

Swap Eth for USD
\(X \geq 0, Y \geq 0\)

Verify Property: MEV < \(\delta\)

CFF models are over approximations of the smart contract code.

False Positive
But No False Negatives

\[ F(X, Y) \]
CFF Models

```python
def ethToTokenInput(eth_sold: uint256(wei), min_tokens: uint256, deadline: timestamp, token_reserve: uint256 = self.token.balanceOf(self), tokens_bought: uint256 = self.getInputPrice(as_unitless_number(eth_sold), as_unit) assert tokens_bought >= min_tokens assert self.token.transfer(recipient, tokens_bought) log.TokenPurchase(buyer, eth_sold, tokens_bought)
```


Becomes easier if the contract has been verified formally for functional correctness

Open sourced CFF models for UniswapV1, UniswapV2, MakerDAO, FlashLoans, Airdrops
CFF Models

def ethToTokenInput(eth_sold: uint256(wei), min_tokens: uint256, deadline: timestamp,
assert deadline >= block.timestamp and (eth_sold > 0 and min_tokens > 0)
token_reserve: uint256 = self.token.balanceOf(self)
tokens_bought: uint256 = self.getInputPrice(as_unitless_number(eth_sold), as_unit
assert tokens_bought >= min_tokens
assert self.token.transfer(recipient, tokens_bought)
log.TokenPurchase(buyer, eth_sold, tokens_bought)


Becomes easier if the contract has been verified formally for functional correctness

Open sourced CFF models for UniswapV1, UniswapV2, MakerDAO, FlashLoans, Airdrops
CFF Models

```python
def ethToTokenInput(eth_sold: uint256(wei), min_tokens: uint256, deadline: timestamp,
assert deadline > block.timestamp and (eth_sold > 0 and min_tokens > 0)
token_reserve: uint256 = self.token.balanceOf(self)
tokens_bought: uint256 = self.getInputPrice(as_unitless_number(eth_sold), as_unit
assert tokens_bought >= min_tokens
assert self.token.transfer(recipient, tokens_bought)
log.TokenPurchase(buyer, eth_sold, tokens_bought)
```


Becomes easier if the contract has been verified formally for functional correctness

Open sourced CFF models for UniswapV1, UniswapV2, MakerDAO, FlashLoans, Airdrops
CFF Design

CFF Language Model

- Codified CF Definitions
- CFF Models
- EVM Semantics

Smart Contracts Code

K Deductive Verifier

Input - (Symbolic) State and Transactions

- Composability Proof
- Counterexample Strategies

Validated by Simulation on Archive Node
More Scaling Optimisations

1. General Optimisations
   1. Transactions for a sender need to be serialised using “nonces”. Many invalid orderings are equivalent
   2. Reorderings across different non interacting contracts are equivalent
   3. Randomised reorderings lead to better convergence in practice.

2. Contract Specific Optimisations
   1. Uniswap-like AMMs are path independent
CFF Evaluation - AMM

7-day moving average of MEV per block in a sample of 1000 random blocks in each month
CFF Evaluation - Maker + Uniswap

Uniswap price used as oracle in Maker

CFF Model for Maker abstracts out liquidation auction
CFF Evaluation

Many More in the paper…
Governance, Flashloans, Airdrops
Conclusion

- Initiated the formal study of economic behaviour of smart contracts through the lens of MEV
  - Definitions for MEV and Secure Composition
  - Clockwork Finance Framework (CFF) : Practical Proof System based on Formal Verification

- Developers can use CFF to generate proofs of bounds on the MEV exposed by their contracts, and users can use CFF to analyse the MEV extractable from their transactions.

Paper: [https://cs.cornell.edu/~babel/cff.pdf](https://cs.cornell.edu/~babel/cff.pdf)

Github: [https://github.com/defi-formal/cff](https://github.com/defi-formal/cff)

Contact: babel@cs.cornell.edu
Appendix
Execution and proving times

[Graph showing execution and proving times with various transaction counts and CPU/Wall Time.

Graph on the right showing CFF Parallelism using Multiple Threads with different transaction counts.]
Directions for Future Work

• MEV Definitions for Leaderless Protocols
• Arbitrary Symbolic Transaction Insertions
• Scaling the Backend
Under the Hood- Sushiswap + Uniswap

```
claim <k>

On UniswapV2 697323163401596485410334513241460920685086001293 swaps for ETH by providing
  1300000000000000000000000000000 COMP and 0 ETH with change 0 fee 1767957155464 ;
On Sushiswap Miner swaps for ETH by providing Alpha:Int COMP and 0 ETH with change 0 fee 0 ;
On UniswapV2 Miner swaps for Alpha COMP by providing ETH fee 0 ;

=> .K
</k>

<S> (Sushiswap in COMP) |-> 107495485843438764484770 (Sushiswap in ETH) |-> 49835502094518088853633
  (UniswapV2 in COMP) |-> 5945498629669852264883 (UniswapV2 in ETH) |-> 2615599823603823616442 =>
  ?S:Map </S>

<B> .List => ?_ </B>

requires (Alpha > Int 0 ) andBool (Alpha < Int 10000000000000000000000000000000 ) //10**22
ensures ( {?S[Miner in COMP]}:Int <= Int 0 ) andBool ( {?S[Miner in ETH]}:Int <= Int 0 )
```
Under the Hood- Sushiswap + Uniswap

1. \( \text{claim } \langle k \rangle \)
2. \( \text{On } \text{UniswapV2 } 697323163401596485410334513241460920685086001293 \) swaps \textit{for} ETH by providing
3. \( \leftrightarrow 130000000000000000000000 \) COMP and 0 ETH with change 0 fee 1767957155464;
4. \( \text{On Sushiswap Miner swaps for ETH by providing Alpha: Int COMP and 0 ETH with change 0 fee 0}; \)
5. \( \text{On UniswapV2 Miner swaps for Alpha COMP by providing ETH fee 0}; \)
6. \( \Rightarrow .K \)
7. \( \langle k \rangle \)
8. \( \langle S \rangle \) (Sushiswap in COMP) \( \Rightarrow 107495485843438764484770 \) (Sushiswap in ETH) \( \Rightarrow 49835502094518088853633 \)
9. \( \leftrightarrow \) (UniswapV2 in COMP) \( \Rightarrow 5945498629669852264883 \) (UniswapV2 in ETH) \( \Rightarrow 2615599823603823616442 \) \( \Rightarrow \)
10. \( \langle B \rangle . \text{List } = \Rightarrow \_ \langle /B \rangle \)
11. requires (Alpha > Int 0) andBool (Alpha < Int 1000000000000000000000000) //10**22
12. ensures (\{?S[Miner in COMP]\}:Int <=0 ) andBool (\{?S[Miner in ETH]\}:Int <=0 )
Under the Hood- Sushiswap + Uniswap

```plaintext
claim <k>
On UniswapV2 697323163401596485410334513241460920685086001293 swaps for ETH by providing
  1300000000000000000000000 COMP and 0 ETH with change 0 fee 1767957155464 ;
On Sushiswap Miner swaps for ETH by providing Alpha: Int COMP and 0 ETH with change 0 fee 0 ;
On UniswapV2 Miner swaps for Alpha COMP by providing ETH fee 0 ;
=> K
</k>
S (Sushiswap in COMP) |-> 107495485843438764484770 (Sushiswap in ETH) |-> 49835502094518088853633
  (UniswapV2 in COMP) |-> 5945498629669852364883 (UniswapV2 in ETH) |-> 2615599823603823616442 =>
  ?S:Map </S>
B .List => _ </B>
requires (Alpha > Int 0) and Bool (Alpha < Int 1000000000000000000000000) //10**22
ensures {{ ?S[Miner in COMP] : Int <= Int 0 } and Bool {{ ?S[Miner in ETH] }} : Int <= Int 0 }
```

![Graph](image-url)