Your Exploit is Mine: Instantly Synthesizing Counterattack Smart Contract

Zhuo Zhang  Zhiqiang Lin  Marcelo Morales
Xiangyu Zhang  Kaiyuan Zhang

August 9, 2023
What Happened to Curve?
What Happened to Curve?

Curve, one of the most popular decentralized exchanges (DEX), was hacked for around $61.7 million on July 30th, 2023.
What Happened to Curve?

Curve, one of the most popular decentralized exchanges (DEX), was hacked for around $61.7 million on July 30th, 2023.
What Happened to Curve?

Curve, one of the most popular decentralized exchanges (DEX), was hacked for around $61.7 million on July 30th, 2023.
What Happened to Curve?

Curve, one of the most popular decentralized exchanges (DEX), was hacked for around $61.7 million on July 30th, 2023.
What Happened to Curve?

Curve, one of the most popular decentralized exchanges (DEX), was hacked for around $61.7 million on July 30th, 2023.
What Happened to Curve?
What Happened to Curve?
What Happened to Curve?
Your Exploit is Mine: Instantly Synthesizing Counterattack Smart Contract

What Happened to Curve?

1. Curve.Fi
2. Attacking Transaction
3. Ethereum Mempool
What Happened to Curve?
What Happened to Curve?

1. Attacker
2. Ethereum Mempool
3. Higher Gas Tip
4. Synthesized Transactions
5. Automated Arbitrage Bot

Curve.Fi Attacker
Attacking Transaction
Ethereum Mempool
Higher Gas Tip
What Happened to Curve?

1. Curve.Fi Attacker
2. Attacking Transaction
3. Ethereum Mempool
4. Automated Arbitrage Bot
5. Higher Gas Tip
6. Succussed

Synthesized Transactions
What Happened to Curve?
What Happened to Curve?

Other Attributes:
- Txn Type: 2 (EIP-1559)
- Nonce: 1188
- Position In Block: 85

Input Data:
- moving funds to cold wallet for now, affected protocols can contact via etherscan chat.

Automated Arbitrage Bot → Synthesized Transactions

Ethereum Mempool

- Higher Gas Tip → Successed
- Failed

Your Exploit is Mine: Instantly Synthesizing Counterattack Smart Contract
Your Exploit is Mine: Instantly Synthesizing Counterattack Smart Contract

What Happened to 30 Curve.Fi Attacker

Automated Arbitrage Bot

Synthesized Transactions

Ethereum Mempool

Higher Gas Tip

Failed

Successed

Other Attributes:
- Txn Type: 2 (EIP-1559)
- Nonce: 1188
- Position In Block: 85

Input Data:
- Deployer from Curve. One tx you front-ran was a hack of CRV/ETH pool. Can refund?

moving funds to cold wallet for now, affected protocols can contact via etherscan chat.
Your Exploit is Mine: Instantly Synthesizing Counterattack Smart Contract

What Happened to Curve.Fi

Moving funds to cold wallet for now, affected protocols can contact via etherscan chat.

Deployer from Curve. One tx you front-ran was a hack of CRV/ETH pool. Can refund?

Yes, please send me a way to contact via etherscan chat.
Takeaways from Curve
Takeaways from \( \curve \) Curve

- Many vulnerabilities may remain hidden despite a large amount of auditing efforts having been put forth.
- Frontrunning attacking transactions provides another opportunity to protect user funds.
Takeaways from Curve

- Many vulnerabilities may remain hidden despite a large amount of auditing efforts having been put forth.
- Frontrunning attacking transactions provides another opportunity to protect user funds.
The Goal and the Timeline
The Goal and the Timeline

Pending Attacking Transaction
The Goal and the Timeline

Pending Attacking Transaction

Synthesized Transactions
The Goal and the Timeline

- **A** Pending Attacking Transaction
- **S** Synthesized Transactions
- **Attack failed**
- **Rescue user funds**
- **Curve**
The Goal and the Timeline

Pending Attacking Transaction

Attack failed

Curve

Rescue user funds

Synthesized Transactions
The Goal and the Timeline

2021: OfficerCia, a Twitter user proposed a similar idea.

Pending Attacking Transaction

Attack failed

Synthesized Transactions

Rescue user funds

Curve
The Goal and the Timeline

2021: OfficerCia, a Twitter user proposed a similar idea.

2022: BlockSec, a DeFi security company, successfully prevented a real-world attack, rescuing around $3.8 million.
The Goal and the Timeline

2021: OfficerCia, a Twitter user proposed a similar idea.

2022: BlockSec, a DeFi security company, successfully prevented a real-world attack, rescuing around $3.8 million.

2023: Many well-known DeFi security companies have started to put their efforts into this arena: BlockSec, FuzzLand, Skylock, D23E.ch, Spotter, and more.
Our Solution: STING

1. Pending Attacking Transaction
2. Attack Information Identification
3. Counterattack Smart Contract Synthesis
4. Synthesized Transactions
5. Contract Execution and Validation
Running Example

Vulnerable Contract

```solidity
contract Victim {
    address operator;

    function setOperator(address _operator) {
        operator = _operator;
    }

    function emergencyExit(address to) {
        require(operator == msg.sender);
        to.transfer(address(this).balance);
    }
}
```

1 For illustrative purposes, the example is a crafted one, combining DAOmaker and TempleDao exploits.
Your Exploit is Mine: Instantly Synthesizing Counterattack Smart Contract

Running Example¹

Vulnerable Contract

```solidity
contract Victim {
    address operator;

    function setOperator(address _operator) {
        operator = _operator;
    }

    function emergencyExit(address to) {
        require(operator == msg.sender);
        to.transfer(address(this).balance);
    }
}
```

¹ For illustrative purposes, the example is a crafted one, combining DAOMaker and TempleDao exploits.
Running Example

Vulnerable Contract

```solidity
contract Victim {
    address operator;

    function setOperator(address _operator) {
        operator = _operator;
    }

    function emergencyExit(address to) {
        require(operator == msg.sender);
        to.transfer(address(this).balance);
    }
}
```

1 For illustrative purposes, the example is a crafted one, combining DAOMaker and TempleDao exploits.
Running Example

```
contract Victim {
    address operator;

    function setOperator(address _operator) {
        operator = _operator;
    }

    function emergencyExit(address to) {
        require(operator == msg.sender);
        to.transfer(address(this).balance);
    }
}
```

1 For illustrative purposes, the example is a crafted one, combining **DAOMaker** and **TempleDao** exploits.
Your Exploit is Mine: Instantly Synthesizing Counterattack Smart Contract

Running Example

Vulnerable Contract

```
contract Victim {
  address operator;

  function setOperator(address _operator) {
    operator = _operator;
  }

  function emergencyExit(address to) {
    require(operator == msg.sender);
    to.transfer(address(this).balance);
  }
}
```

1 For illustrative purposes, the example is a crafted one, combining DAOMaker and TempleDao exploits.
Running Example¹

Vulnerable Contract

```
contract Victim {
    address operator;

    function setOperator(address _operator) {
        operator = _operator;
    }

    function emergencyExit(address to) {
        require(operator == msg.sender);
        to.transfer(address(this).balance);
    }
}
```

¹ For illustrative purposes, the example is a crafted one, combining DAOMaker and TempleDao exploits.
Attack Information Identification

Goal: Pinpoint all attack-related malicious entities, including accounts and transactions.
Attack Information Identification

Goal: Pinpoint all attack-related malicious entities, including accounts and transactions.

Transactions and accounts are pinpointed in an iterative fashion.
Attack Information Identification

Goal: Pinpoint all attack-related malicious entities, including accounts and transactions.

Transactions and accounts are pinpointed in an iterative fashion.

- The attacking transaction is detected based on the profit.
Goal: Pinpoint all attack-related malicious entities, including accounts and transactions.

Transactions and accounts are pinpointed in an iterative fashion.

- The attacking transaction is detected based on the profit.
Attack Information Identification

Goal: Pinpoint all attack-related malicious entities, including accounts and transactions.

Transactions and accounts are pinpointed in an iterative fashion.

- The attacking transaction is detected based on the profit.
- Accounts are pinpointed based on historical behaviors.
Your Exploit is Mine: Instantly Synthesizing Counterattack Smart Contract

**Attack Information Identification**

Goal: Pinpoint all attack-related malicious entities, including accounts and transactions.

Transactions and accounts are pinpointed in an iterative fashion.

- The attacking transaction is detected based on the profit.
- Accounts are pinpointed based on historical behaviors.

![Diagram](image_url)
We compile these key heuristics, applied within our system, (10/11/2022). Conversely, benign entities typically have a tem's overall analysis and decision-making process. entities and can be instrumental in spotting exploit contracts. contributes to the differentiation between benign and malicious allows us to quickly distinguish malicious entities, such as of more than ten days. This simple but effective observation exploit contracts were deployed at most two days before the its deployment. Our evaluation (§ 4.1) also confirmed that all

Goal: Pinpoint all attack-related malicious entities, including accounts and transactions. Transactions and accounts are pinpointed in an iterative fashion.

• The attacking transaction is detected based on the profit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lifespan</td>
<td>Contracts deployed shortly before an attack are likely malicious.</td>
</tr>
<tr>
<td>Balance</td>
<td>Contracts whose initial assets exceed the attack profit are likely to be victims.</td>
</tr>
<tr>
<td>Fund Source</td>
<td>Contracts and Wallets funded from mixing servers (e.g., Tornado Cash) are likely malicious.</td>
</tr>
<tr>
<td>Activities</td>
<td>Contracts that frequently interact with users exhibiting diverse behaviors are likely benign.</td>
</tr>
<tr>
<td>Source Code</td>
<td>Contracts with unverified source code are likely malicious.</td>
</tr>
</tbody>
</table>
Attack Information Identification

Goal: Pinpoint all attack-related malicious entities, including accounts and transactions.

Transactions and accounts are pinpointed in an iterative fashion.

• The attacking transaction is detected based on the profit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lifespan</td>
<td>Contracts deployed shortly before an attack are likely malicious.</td>
</tr>
<tr>
<td>Balance</td>
<td>Contracts whose initial assets exceed the attack profit are likely to be victims.</td>
</tr>
<tr>
<td>Fund Source</td>
<td>Contracts and Wallets funded from mixing servers (e.g., Tornado Cash) are likely malicious.</td>
</tr>
<tr>
<td>Activities</td>
<td>Contracts that frequently interact with users exhibiting diverse behaviors are likely benign.</td>
</tr>
<tr>
<td>Source Code</td>
<td>Contracts with unverified source code are likely malicious.</td>
</tr>
</tbody>
</table>
Attack Information Identification

Goal: Pinpoint all attack-related malicious entities, including accounts and transactions.

Transactions and accounts are pinpointed in an iterative fashion.

- The attacking transaction is detected based on the profit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lifespan</td>
<td>Contracts deployed shortly before an attack are likely malicious.</td>
</tr>
<tr>
<td>Balance</td>
<td>Contracts whose initial assets exceed the attack profit are likely to be victims.</td>
</tr>
<tr>
<td>Fund Source</td>
<td>Contracts and Wallets funded from mixing servers (e.g., Tornado Cash) are likely malicious.</td>
</tr>
<tr>
<td>Activities</td>
<td>Contracts that frequently interact with users exhibiting diverse behaviors are likely benign.</td>
</tr>
<tr>
<td>Source Code</td>
<td>Contracts with unverified source code are likely malicious.</td>
</tr>
</tbody>
</table>
Attack Information Identification

Goal: Pinpoint all attack-related malicious entities, including accounts and transactions.

Transactions and accounts are pinpointed in an iterative fashion.

• The attacking transaction is detected based on the profit.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lifespan</td>
<td>Contracts deployed shortly before an attack are likely malicious.</td>
</tr>
<tr>
<td>Balance</td>
<td>Contracts whose initial assets exceed the attack profit are likely to be victims.</td>
</tr>
<tr>
<td>Fund Source</td>
<td>Contracts and Wallets funded from mixing servers (e.g., Tornado Cash) are likely malicious.</td>
</tr>
<tr>
<td>Activities</td>
<td>Contracts that frequently interact with users exhibiting diverse behaviors are likely benign.</td>
</tr>
<tr>
<td>Source Code</td>
<td>Contracts with unverified source code are likely malicious.</td>
</tr>
</tbody>
</table>
Attack Information Identification

Goal: Pinpoint all attack-related malicious entities, including accounts and transactions.

Transactions and accounts are pinpointed in an iterative fashion.

- **The attacking transaction** is detected based on the profit.
- **Accounts** are pinpointed based on historical behaviors.
- **Transactions** are pinpointed by read-write dependency.

---

**Diagram:**
- **Attacker**
  - **Tx1:** Deploy
  - **Tx3:** Exploit
- **Exploit Contract**
  - **Tx3 (inner):** emergencyExit(AttackerAddr)
- **Victim Contract**
  - **Tx2:** setOperator(ExploitContractAddr)
Attack Information Identification

Goal: Pinpoint all attack-related malicious entities, including accounts and transactions.

Transactions and accounts are pinpointed in an iterative fashion.

- The attacking transaction is detected based on the profit.
- Accounts are pinpointed based on historical behaviors.
- Transactions are pinpointed by read-write dependency.

Attacker

Exploit Contract

Victim Contract

Tx1: Deploy

Tx3: Exploit

emergencyExit(AttackerAddr)

setOperator(ExploitContractAddr)

Tx2
### Attack Information Identification

**Vulnerable Contract**

```solidity
contract Victim {
    address operator;

    function setOperator(address _operator) {
        operator = _operator;
    }

    function emergencyExit(address to) {
        require(operator == msg.sender);
        to.transfer(address(this).balance);
    }
}
```

**Diagram**

1. **Tx1: Deploy**
   - **Attacker** deploys the **Exploit Contract**.

2. **Tx2**
   - **Operator** is set using `setOperator(ExploitContractAddr)`.

3. **Tx3**
   - **Exploit Contract** is exploited using `emergencyExit(AttackerAddr)`.

4. **Tx3 (inner)**
   - **Attacker** transfers funds using `to.transfer(address(this).balance)`.

---

**Transactions and Accounts Pinpointing**

Transactions and accounts are pinpointed in an iterative fashion.

**Account Pinpointing**

Accounts are pinpointed based on historical behaviors.

**Transaction Pinpointing**

Transactions are pinpointed by read-write dependency.

**Attacker Goal**

The goal is to pinpoint all attack-related malicious entities, including accounts and transactions.

- **Profit-based Detection**: The attacking transaction is detected based on the profit.
Attack Information Identification

Goal: Pinpoint all attack-related malicious entities, including accounts and transactions.

Transactions and accounts are pinpointed in an iterative fashion.

- The attacking transaction is detected based on the profit.
- Accounts are pinpointed based on historical behaviors.
- Transactions are pinpointed by read-write dependency.
Counterattack Smart Contract Synthesis

Goal: For each exploit contract, we aim to synthesize a counterattack contract that ensures the stolen funds are sent to accounts under our control.
Counterattack Smart Contract Synthesis

Goal: For each exploit contract, we aim to synthesize a counterattack contract that ensures the stolen funds are sent to accounts under our control.

Exploit Contract

```
contract Exploit {
    function hack(address toAddr) {
        if (msg.sender == AttackerAddr) {
            VICTIM.emergencyExit(toAddr + 1);
        }
    }
}
```
Counterattack Smart Contract Synthesis

Goal: For each exploit contract, we aim to synthesize a counterattack contract that ensures the stolen funds are sent to accounts under our control.

Exploit Contract

```solidity
contract Exploit {
    function hack(address toAddr) {
        if (msg.sender == AttackerAddr) {
            VICTIM.emergencyExit(toAddr + 1);
        }
    }
}
```

Counterattack Contract

```solidity
contract Exploit {
    function hack(address toAddr) {
        if (true) {
            VICTIM.emergencyExit(OurAddress);
        }
    }
}
```
Counterattack Smart Contract Synthesis

Goal: For each exploit contract, we aim to synthesize a counterattack contract that ensures the stolen funds are sent to accounts under our control.

Exploit Contract

```solidity
contract Exploit {
    function hack(address toAddr) {
        if (msg.sender == AttackerAddr) {
            VICTIM.emergencyExit(toAddr + 1);
        }
    }
}
```

Counterattack Contract

```solidity
contract Exploit {
    function hack(address toAddr) {
        if (true) {
            victim.emergencyExit(OurAddress);
        }
    }
}
```
Counterattack Smart Contract Synthesis

Goal: For each exploit contract, we aim to synthesize a counterattack contract that ensures the stolen funds are sent to accounts under our control.

Exploit Contract

```solidity
class Exploit {
    function hack(address toAddr) {
        if (msg.sender == AttackerAddr) {
            VICTIM.emergencyExit(toAddr + 1);
        }
    }
}
```

Counterattack Contract

```solidity
class Exploit {
    function hack(address toAddr) {
        if (true) {
            // Counterattack code here
        }
    }
}
```
Counterattack Smart Contract Synthesis

Goal: For each exploit contract, we aim to synthesize a counterattack contract that ensures the stolen funds are sent to accounts under our control.

Exploit Contract

```solidity
contract Exploit {
    function hack(address toAddr) {
        if (msg.sender == AttackerAddr) {
            VICTIM.emergencyExit(toAddr + 1);
        }
    }
}
```

Counterattack Contract

```solidity
contract Exploit {
    function hack(address toAddr) {
        if (true) {
            VICTIM.emergencyExit(toAddr);
        }
    }
}
```
Counterattack Smart Contract Synthesis

Goal: For each exploit contract, we aim to synthesize a counterattack contract that ensures the stolen funds are sent to accounts under our control.

**Exploit Contract**

```solidity
contract Exploit {
    function hack(address toAddr) {
        if (msg.sender == AttackerAddr) {
            VICTIM.emergencyExit(toAddr + 1);
        }
    }
}
```

**Counterattack Contract**

```solidity
contract Exploit {
    function hack(address toAddr) {
        if (true) {
            VICTIM.emergencyExit(OurAddress);
        }
    }
}
```
Contract Execution and Validation

Goal: Ensure the success of the counterattack by locally deploying the synthesized contract, guaranteeing that it will result in a profit to our addresses.
Evaluation
Evaluation

Dataset:

We investigated a total of 86 attacks that occurred on the Ethereum mainnet prior to 2023, of which 24 are deemed out of scope.

<table>
<thead>
<tr>
<th>Attack</th>
<th>Date</th>
<th>Loss</th>
<th>Root Cause</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wintermute</td>
<td>09/20/22</td>
<td>160.0M</td>
<td>Key compromised or rugged</td>
</tr>
<tr>
<td>SudoRare</td>
<td>08/23/22</td>
<td>800.0K</td>
<td>Key compromised or rugged</td>
</tr>
<tr>
<td>Curve Finance</td>
<td>08/09/22</td>
<td>575.0K</td>
<td>Off-chain component compromise</td>
</tr>
<tr>
<td>Harmony Bridge</td>
<td>06/24/22</td>
<td>100.0M</td>
<td>Key compromised or rugged</td>
</tr>
<tr>
<td>Ronin Network</td>
<td>03/29/22</td>
<td>624.0M</td>
<td>Key compromised or rugged</td>
</tr>
<tr>
<td>BuildFinance</td>
<td>02/14/22</td>
<td>470.0K</td>
<td>Key compromised or rugged</td>
</tr>
<tr>
<td>Dego Finance</td>
<td>02/10/22</td>
<td>10.0M</td>
<td>Key compromised or rugged</td>
</tr>
<tr>
<td>Meter</td>
<td>02/06/22</td>
<td>7.7M</td>
<td>No fund lost on the mainnet</td>
</tr>
<tr>
<td>Qubit Finance</td>
<td>01/28/22</td>
<td>80.0M</td>
<td>No fund lost on the mainnet</td>
</tr>
<tr>
<td>Crypto.com</td>
<td>01/18/22</td>
<td>33.7M</td>
<td>Key compromised or rugged</td>
</tr>
<tr>
<td>LCX</td>
<td>01/08/22</td>
<td>7.9M</td>
<td>Key compromised or rugged</td>
</tr>
<tr>
<td>Vulcan Forged</td>
<td>12/13/21</td>
<td>140.0M</td>
<td>Key compromised or rugged</td>
</tr>
<tr>
<td>Bitmart</td>
<td>12/04/21</td>
<td>196.0M</td>
<td>Key compromised or rugged</td>
</tr>
<tr>
<td>Badger</td>
<td>12/02/21</td>
<td>120.0M</td>
<td>Off-chain component compromise</td>
</tr>
<tr>
<td>AnubisDAO</td>
<td>10/29/21</td>
<td>60.0M</td>
<td>Key compromised or rugged</td>
</tr>
<tr>
<td>JayPegs Automart</td>
<td>09/17/21</td>
<td>3.1M</td>
<td>Key compromised or rugged</td>
</tr>
<tr>
<td>DAO Maker</td>
<td>08/12/21</td>
<td>7.0M</td>
<td>Key compromised or rugged</td>
</tr>
<tr>
<td>Thorchain</td>
<td>07/22/21</td>
<td>8.0M</td>
<td>Off-chain component compromise</td>
</tr>
<tr>
<td>Thorchain</td>
<td>07/15/21</td>
<td>5.0M</td>
<td>Off-chain component compromise</td>
</tr>
<tr>
<td>Bondly</td>
<td>07/15/21</td>
<td>5.9M</td>
<td>Key compromised or rugged</td>
</tr>
<tr>
<td>Anyswap</td>
<td>07/10/21</td>
<td>7.9M</td>
<td>Key compromised or rugged</td>
</tr>
<tr>
<td>Chainswap</td>
<td>07/02/21</td>
<td>800.0K</td>
<td>No fund lost on the mainnet</td>
</tr>
<tr>
<td>Roll</td>
<td>03/14/21</td>
<td>5.7M</td>
<td>Key compromised or rugged</td>
</tr>
<tr>
<td>Paid Network</td>
<td>03/05/21</td>
<td>3.0M</td>
<td>Key compromised or rugged</td>
</tr>
</tbody>
</table>
Evaluation

Dataset:

We investigated a total of 86 attacks that occurred on the Ethereum mainnet prior to 2023, of which 24 are deemed out of scope.

Effectiveness:

We successfully synthesized 54 counterattacks out of 62 attacks.
Evaluation

Dataset:
We investigated a total of 86 attacks that occurred on the Ethereum mainnet prior to 2023, of which 24 are deemed out of scope.

Effectiveness:
We successfully synthesized 54 counterattacks out of 62 attacks.

Efficiency:
The median runtime overhead is 0.29 seconds, while the worst-case value rises to 8.51 seconds (only two cases exceed 1.00 second).
Limitations

- **Adaptive Evasion**: Multiple adaptive evasion techniques, such as code obfuscation, may exist against STING, enabling attacks to circumvent our defense mechanism.

- **Private Transactions**: Private transactions provide a mechanism for blockchain users to execute transactions that remain hidden until being confirmed.

- **Blind Spots**: STING does not provide comprehensive protection against all DeFi attacks.

- **Performance Issue**: The execution overhead of STING is not optimal for MEV bots to initiate front-running transactions, with a worst-case duration of 8.51 seconds.
  - Our prototype implementation is not optimal: **3.3** seconds (for our archive node) vs. **0.74** seconds (for Reth) in the 8.51-second worst-case scenario.
Related Works


Thank You

Zhuo Zhang, zhan3299@purdue.edu