

# Zhuo ZhangZhiqiang LinMarcelo MoralesXiangyu ZhangKaiyuan Zhang





August 9, 2023





#### What Happened to **S** Curve ?



#### What Happened to **S** Curve ?





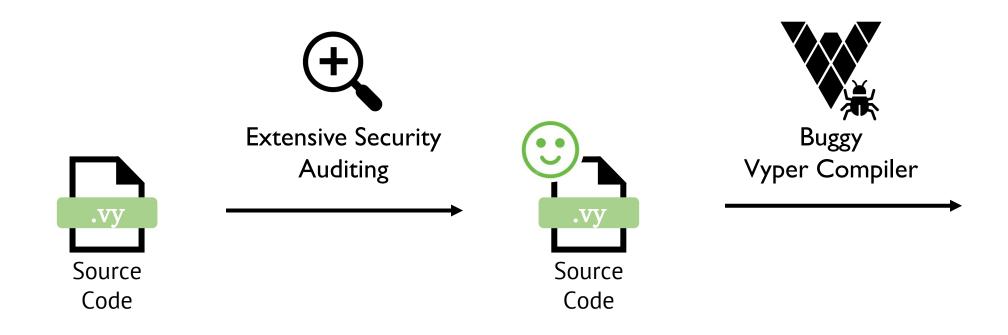
#### What Happened to **S** Curve ?





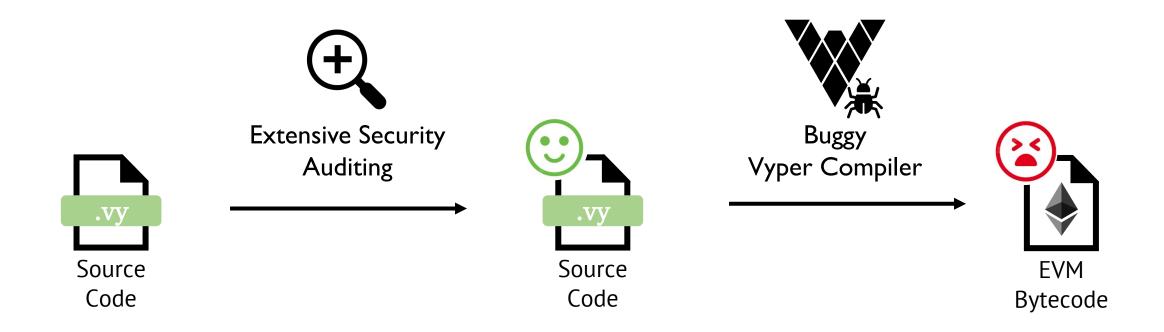
### What Happened to 🦻 Curve ?





### What Happened to 🦻 Curve ?





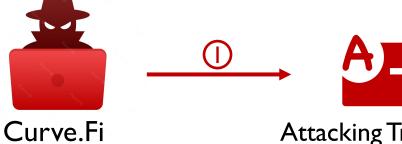




22



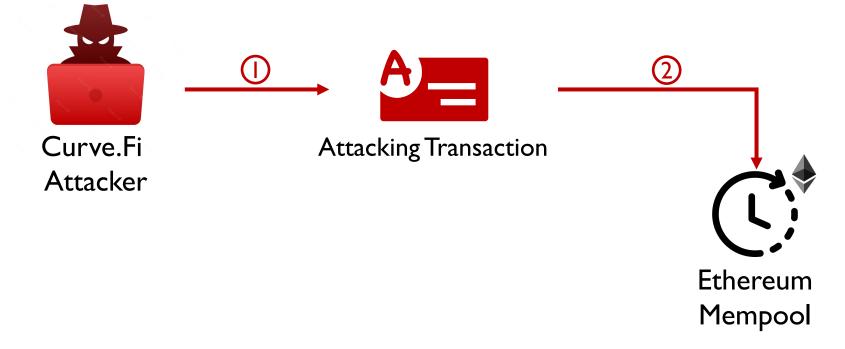




Attacker

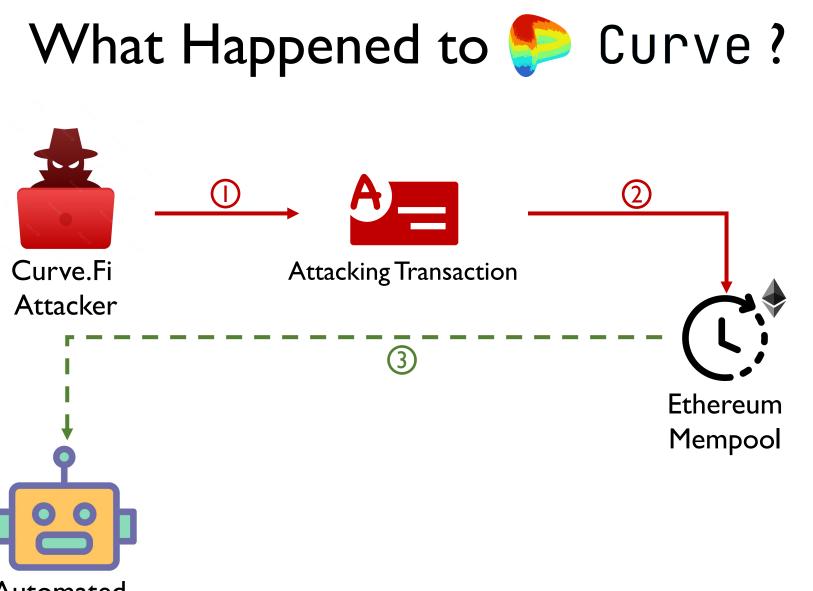
Attacking Transaction



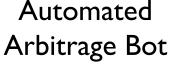




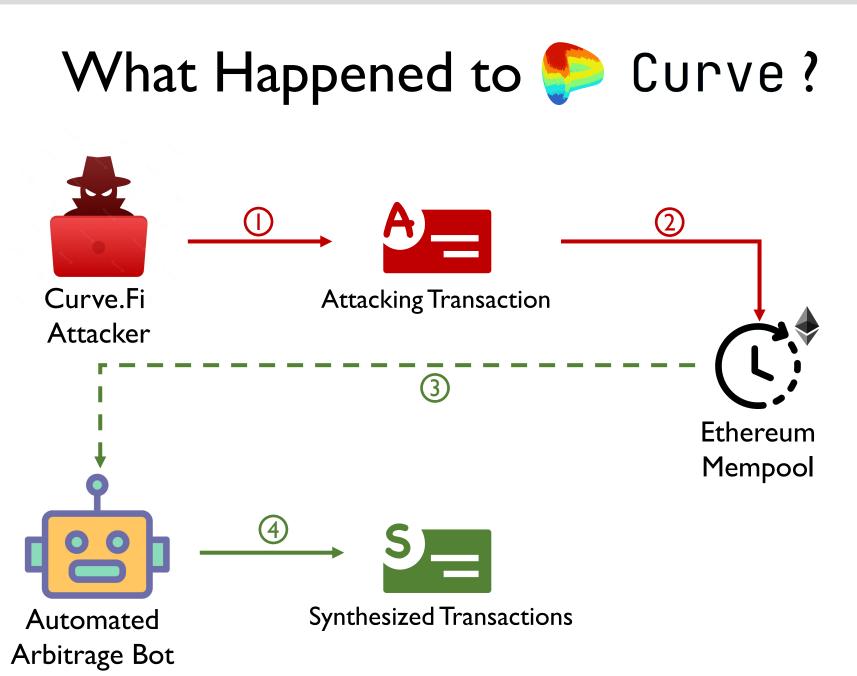
24



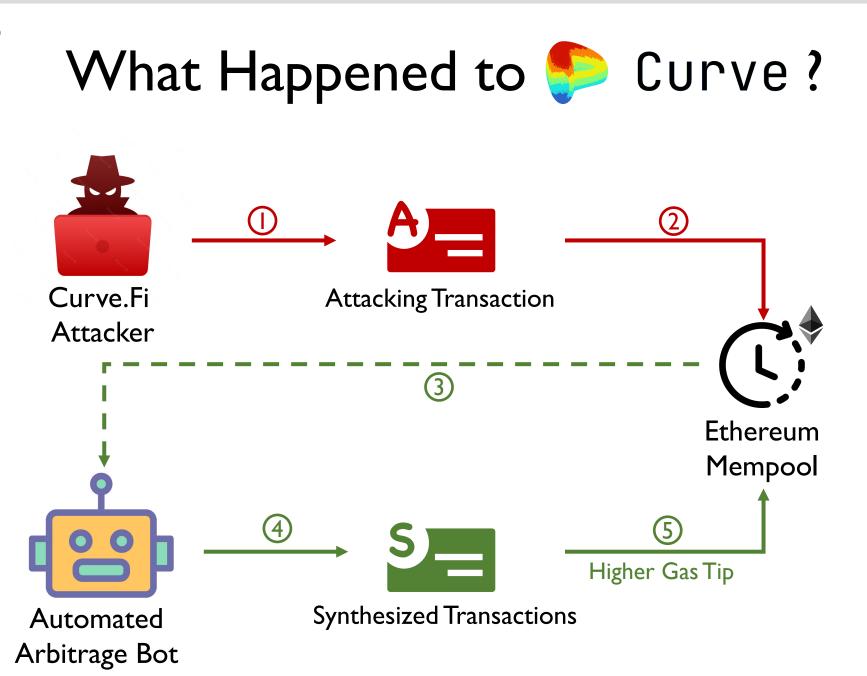
STJNG



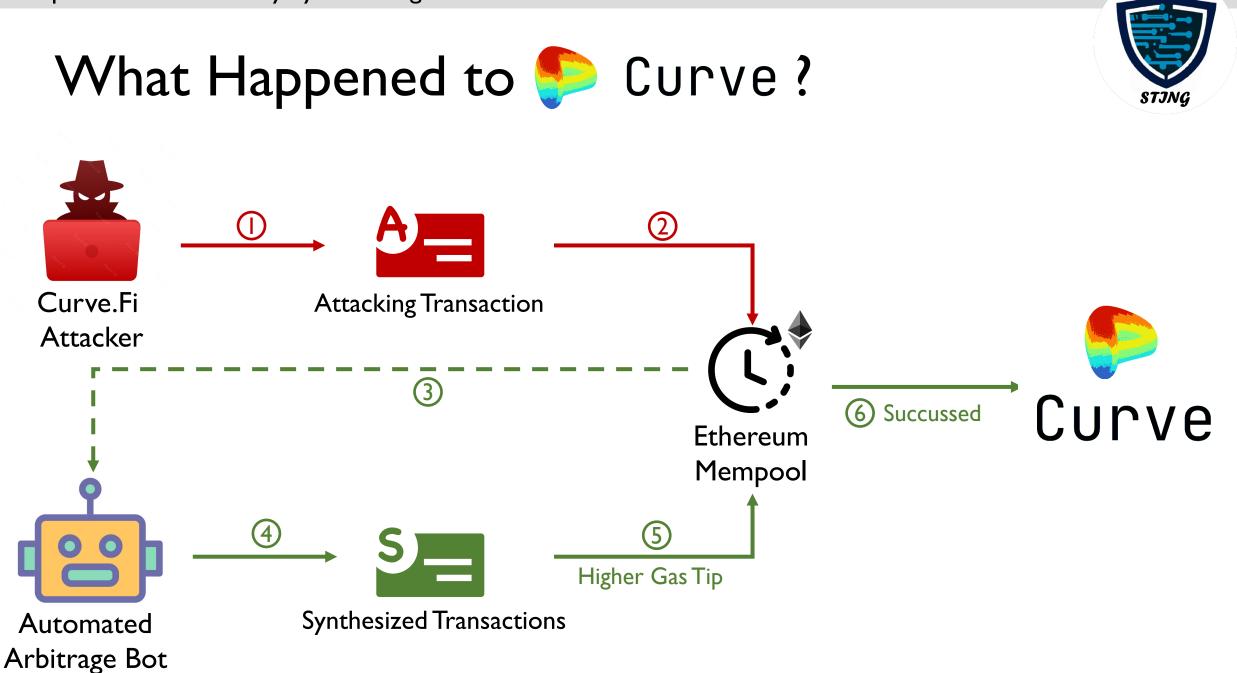
25

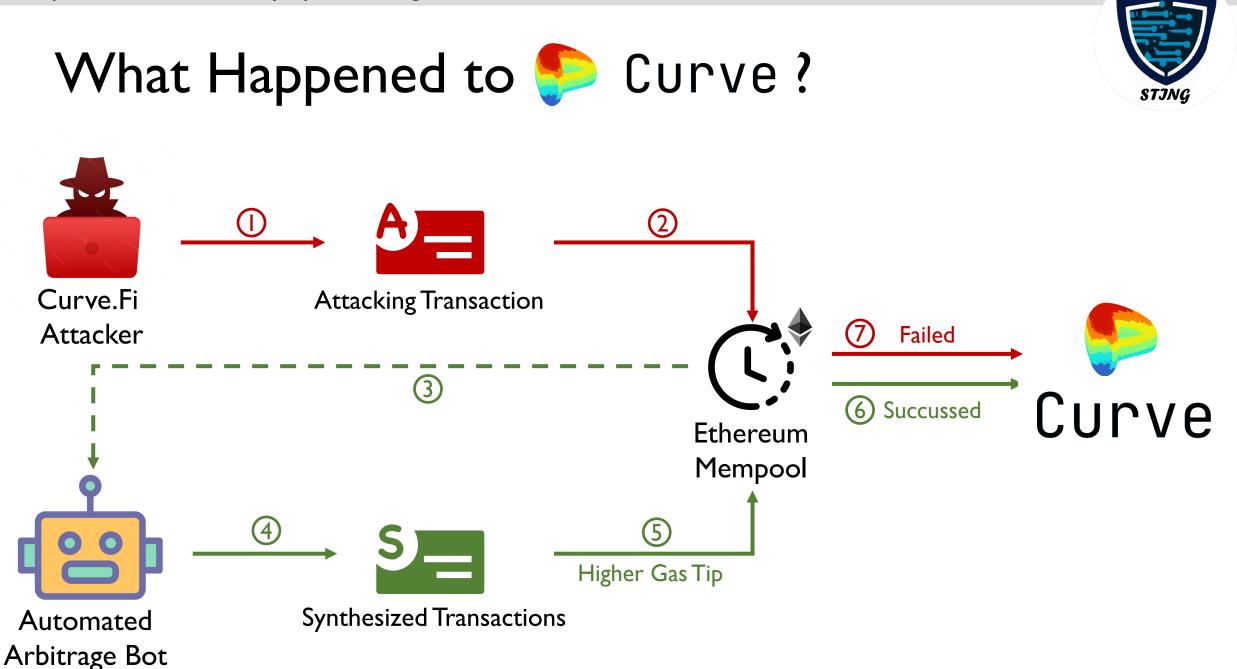


STJNG



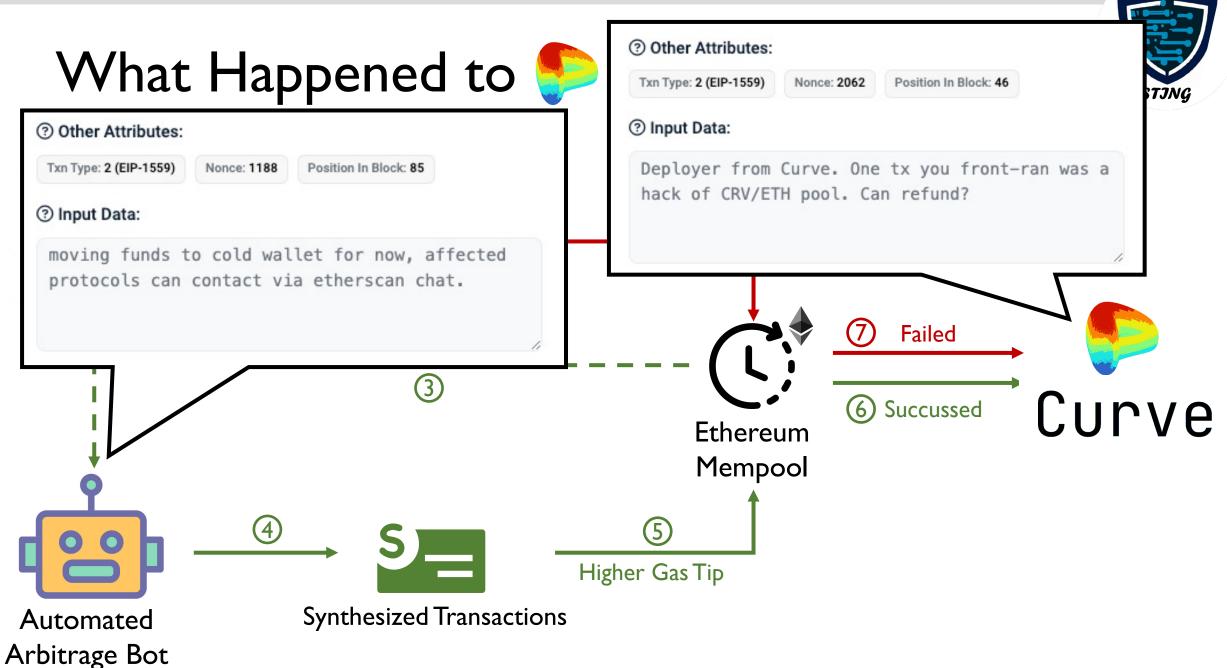


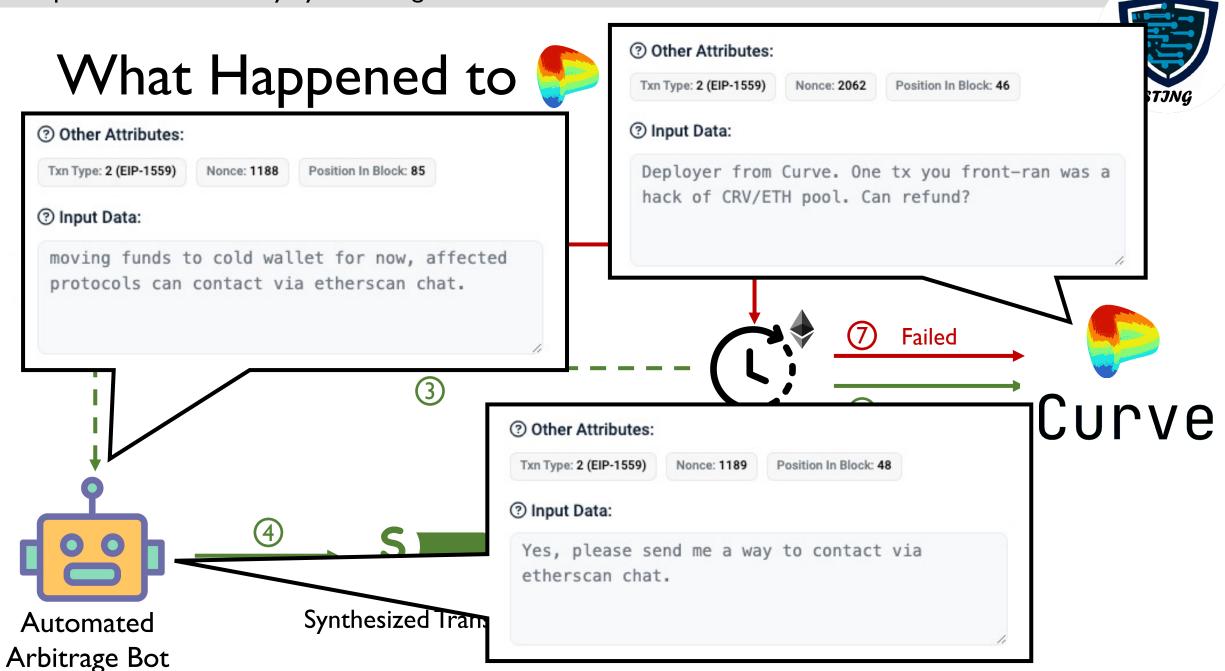




29 What Happened to 🤛 Curve? STJNG ⑦ Other Attributes: Txn Type: 2 (EIP-1559) Nonce: 1188 Position In Block: 85 ⑦ Input Data: 2 moving funds to cold wallet for now, affected protocols can contact via etherscan chat. (7) Failed 3 Curve 6 Succussed **Ethereum** Mempool (4)(5) Higher Gas Tip Synthesized Transactions Automated

Arbitrage Bot











33





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

Buggy Vyper Compiler





- 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.



35

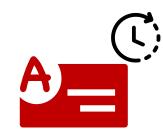
#### The Goal and the Timeline



36

#### The Goal and the Timeline



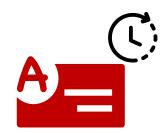


Pending Attacking Transaction

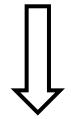
37

#### The Goal and the Timeline





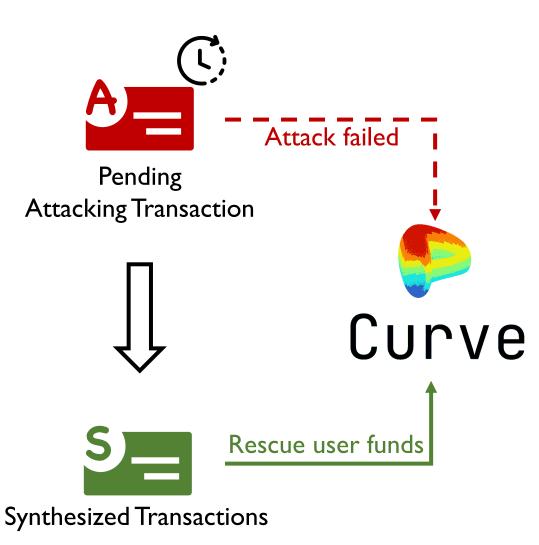
Pending Attacking Transaction





38

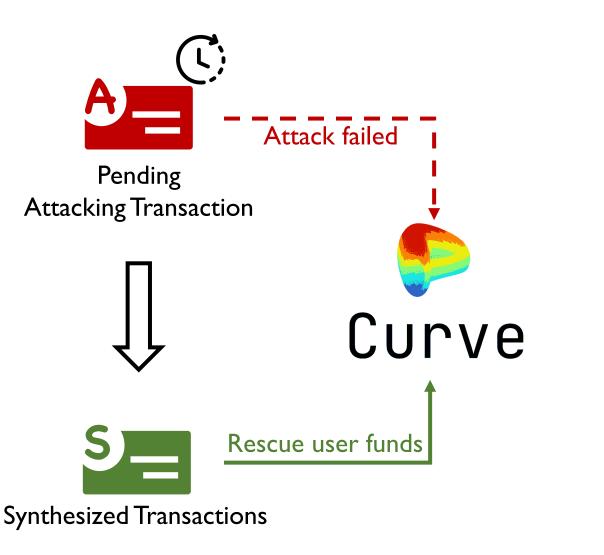
#### The Goal and the Timeline





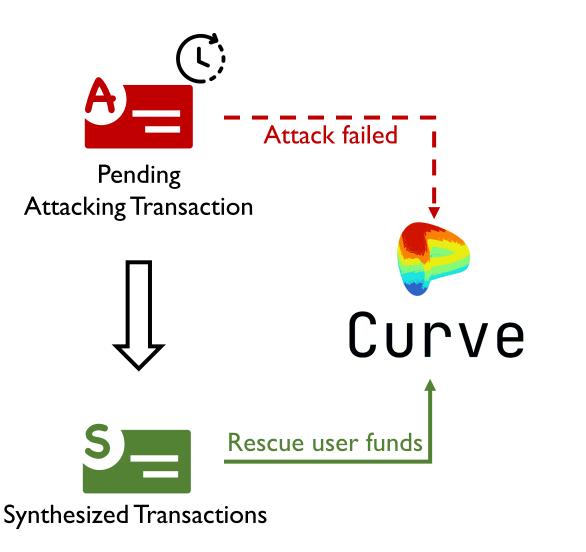
39

#### The Goal and the Timeline





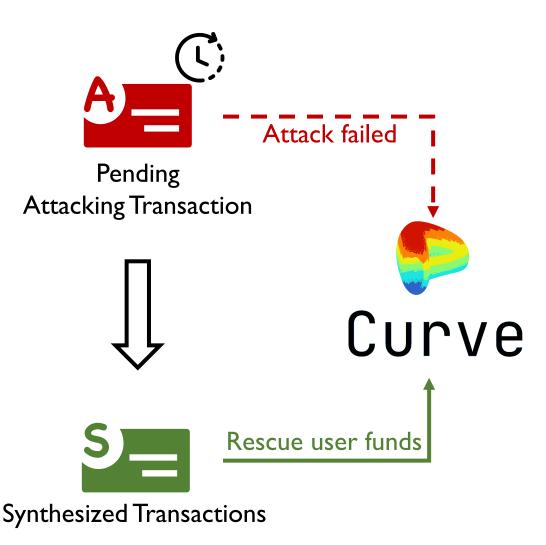
#### The Goal and the Timeline





*OfficerCia*, a Twitter user proposed a similar idea.

#### The Goal and the Timeline



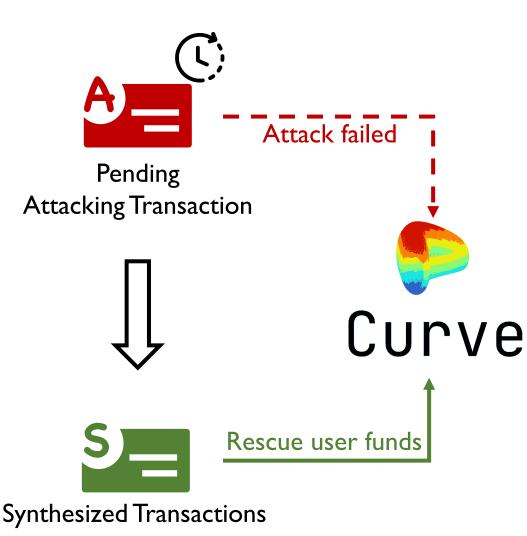


*OfficerCia*, a Twitter user proposed a similar idea.

2021

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

### The Goal and the Timeline





*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

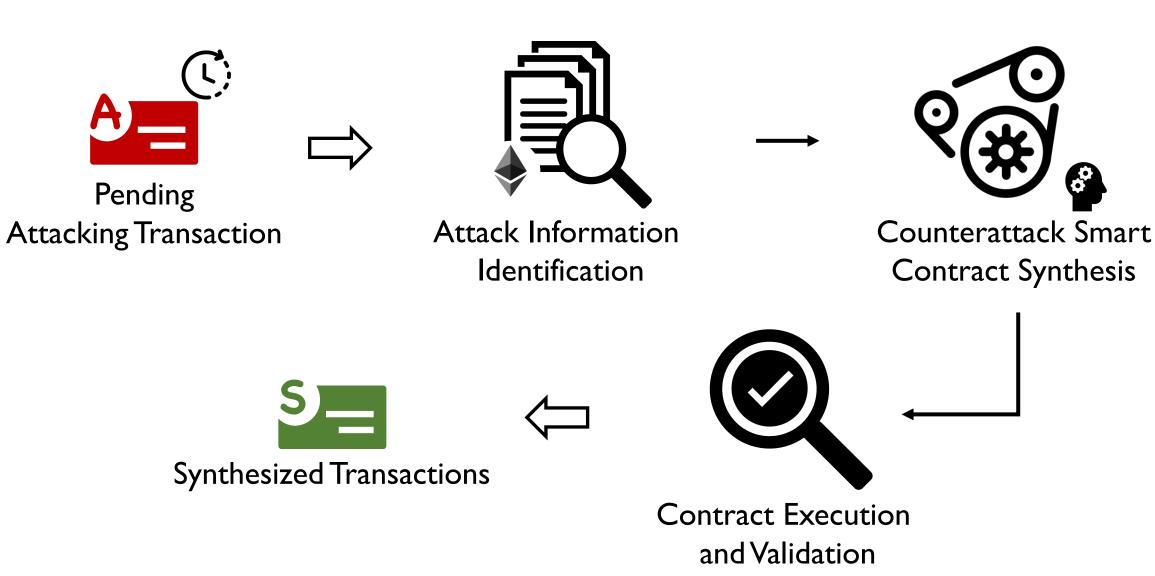
2021

Spotter 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





**44** 

## Running Example<sup>1</sup>

#### Vulnerable Contract

```
contract Victim {
01
02
     address operator;
03
04
     function setOperator(address operator) {
05
       operator = _operator;
06
07
     function emergencyExit(address to) {
08
09
        require(operator == msg.sender);
       to.transfer(address(this).balance);
10
11
12
   }
```



45

## Running Example<sup>1</sup>

#### Vulnerable Contract

```
contract Victim {
01
02
     address operator;
03
     function setOperator(address _operator) {
04
05
       operator = _operator;
06
07
08
     function emergencyExit(address to) {
09
        require(operator == msg.sender);
       to.transfer(address(this).balance);
10
11
12
   }
```



46

07

08

09

10

11

12

### Running Example<sup>1</sup>

#### Vulnerable Contract

```
01 contract Victim {
02 address operator;
```

```
03
04 function setOperator(address _operator) {
05 operator = _operator;
06 }
```

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



```
Attacker
```





47

11

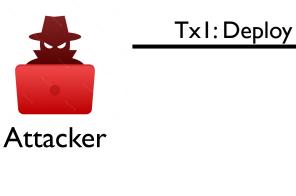
12

### Running Example<sup>1</sup>

#### **Vulnerable Contract**

```
01 contract Victim {
02 address operator;
03
04 function setOperator(address _operator) {
05 operator = _operator;
```

06 }
07
08 function emergencyExit(address to) {
09 require(operator == msg.sender);
10 to.transfer(address(this).balance);





```
Exploit Contract
```





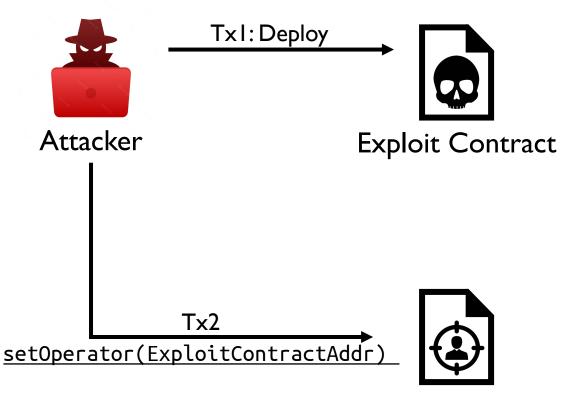
**48** 

12

#### Running Example<sup>1</sup>

#### **Vulnerable Contract**

```
contract Victim {
01
02
     address operator;
03
04
     function setOperator(address operator) {
       operator = _operator;
05
06
07
08
     function emergencyExit(address to) {
09
        require(operator == msg.sender);
       to.transfer(address(this).balance);
10
11
```



Victim Contract

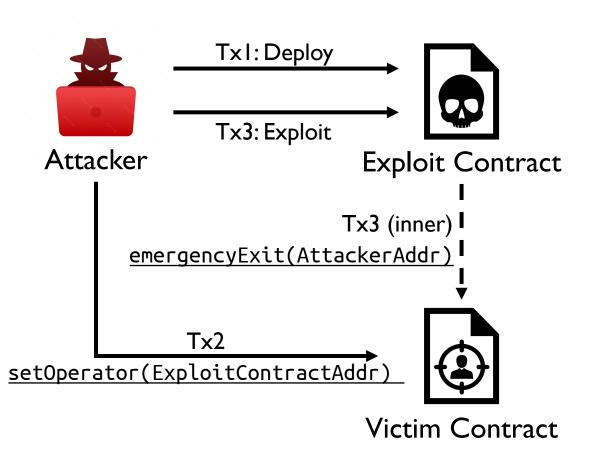


**49** 

## Running Example<sup>1</sup>

#### **Vulnerable Contract**

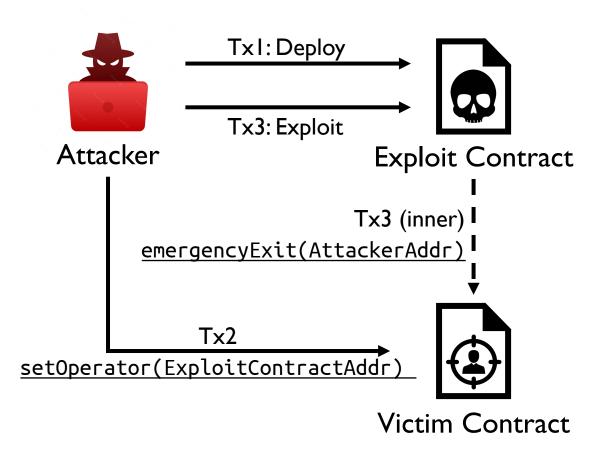
```
contract Victim {
01
02
     address operator;
03
04
     function setOperator(address _operator) {
       operator = _operator;
05
06
07
08
     function emergencyExit(address to) {
09
        require(operator == msg.sender);
        to.transfer(address(this).balance);
10
11
12
```



50

## Attack Information Identification

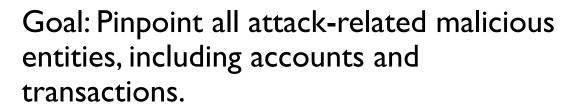
Goal: Pinpoint all attack-related malicious entities, including accounts and transactions.

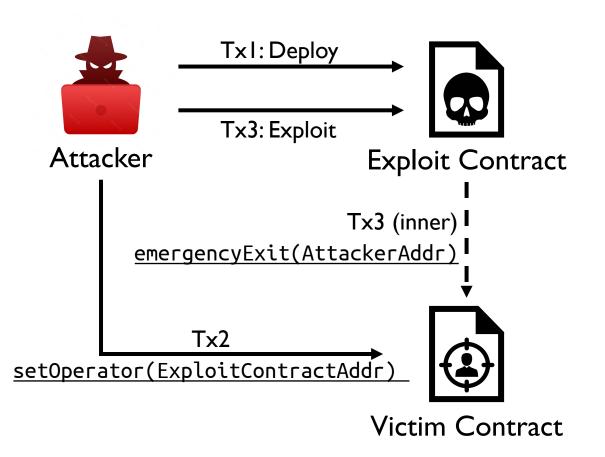




51

### Attack Information Identification







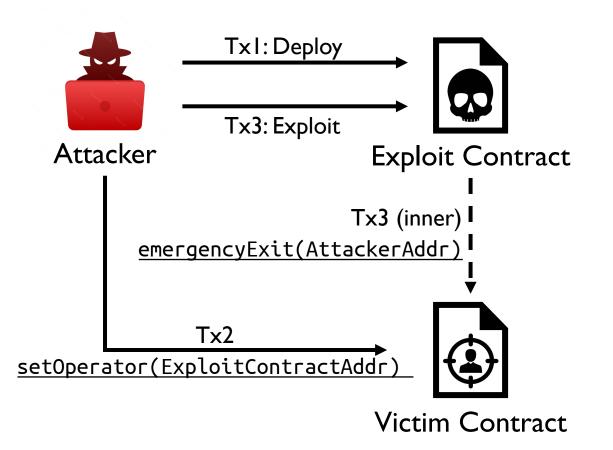
52

## Attack Information Identification

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

Transactions and accounts are pinpointed in an iterative fashion.

• <u>The attacking transaction</u> is detected based on the profit.





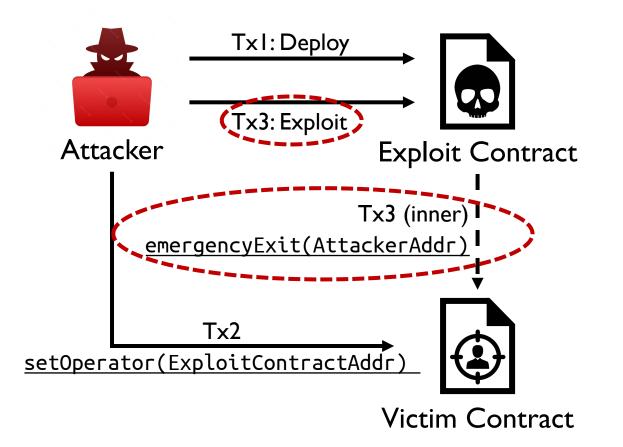
53

## Attack Information Identification

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

Transactions and accounts are pinpointed in an iterative fashion.

• <u>The attacking transaction</u> is detected based on the profit.



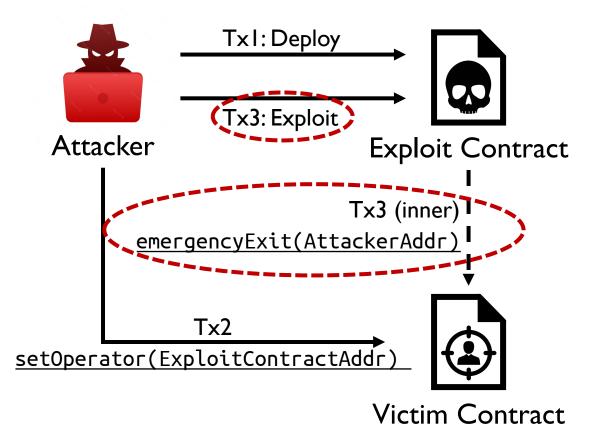


54

## Attack Information Identification

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

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



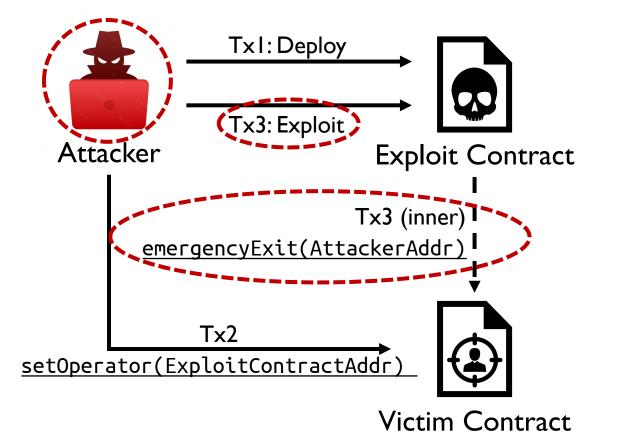


55

## Attack Information Identification

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

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





Your		<pre>Deploy UniswapPair() =&gt; [TempleFraxPool]     [Temple].transferFrom([Temple DAO Wallet], [TempleFraxPool], 37500000 ether)</pre>	ntract	
56		[Frax].transferFrom([Temple DAO Wallet], [TempleFraxPool], 9375000 ether) [TempleFraxPool].transfer([Temple DAO Wallet], 18750000 ether)		
	06-02-2022 · [ Tx <sub>2</sub>	<pre>[Temple DAO Wallet] deploy StaxLPToken() =&gt; [StaxLPToken]</pre>	cation	STJNG
	06-03-2022 [ Tx <sub>3</sub>	<pre>[Temple DAO Wallet] deploy StaxLPStaking([StaxLPToken], [TempleFraxPool]) =&gt; [StaxLPStak</pre>	ing]	
		Tephe DAO Wallet I all attack-related malicious	TxI: Deploy	
	<sup>06-13</sup> entit <sup>Tx</sup> ₅ trans	Tes, in alid ing Paccountsoon () [TempleFraxPool].transferFrom([Temple DAO Wallet], [StaxLPStaking], 18750000 ether) Sactor Das J.mint([Temple DAO Wallet], 18750000 ether)	(Tx3: Exploit)	
		actions and accounts are pinpointed in an <b>A</b> ive fashion.	Attacker	Exploit Contract
	• <u>The</u>	<u>e attacking transaction</u> is detected based on profit.	<u>emergencyExit(Atta</u>	Tx3 (inner) <u>ckerAddr)</u>
	Name	Description		
	Lifespan Balance	Contracts deployed shortly before an attack are likely malicious. Contracts whose initial assets exceed the attack profit are likely t be victims.	<ul> <li>Tx2</li> <li>ator(ExploitContractAction</li> </ul>	
	Fund Sour			Victim Contract
	Activities	Contracts that frequently interact with users exhibiting diverse behaviors are likely benign.		
	Source Co	ode Contracts with unverified source code are likely malicious.		

Your	Ix <sub>1</sub>	Deploy UniswapPair() => [TempleFraxPool] [Temple].transferFrom([Temple DAO Wallet], [TempleFraxPool], 37500000 ether)	ntract	
57		<pre>[Frax].transferFrom([Temple DAO Wallet], [TempleFraxPool], 9375000 ether) [TempleFraxPool].transfer([Temple DAO Wallet], 18750000 ether)</pre>		
	06-02-2022 Tx <sub>2</sub>	<pre>[Temple DAO Wallet] deploy StaxLPToken() =&gt; [StaxLPToken]</pre>	ation	STJNG
	06-03-2022 Tx <sub>3</sub>	<pre>[Temple DAO Wallet] deploy StaxLPStaking([StaxLPToken], [TempleFraxPool]) =&gt; [StaxLPStakin</pre>	9]	
	-	al: Pinpoint all attack-related malicious		
	06-13 <b>ent</b>	Ties, in aluding Pace (Temple DAO Wallet], [StaxLPStaking], 18750000 eth [TempleFraxPool].transferFrom([Temple DAO Wallet], [StaxLPStaking], 18750000 eth Saction Saction ([Temple DAO Wallet], 18750000 ether)	Tx1: Deploy (Tx3: Exploit)	
		I I	tacker Exploit Cont	ract
	Itera	tive fashion.		
	_	<u>ne attacking transaction</u> is detected based on e profit.	Tx3 (inner) <u>emergencyExit(AttackerAddr)</u>	•
	Nan	e Description		
	Lifespa	n Contracts deployed shortly before an attack are likely malicious.	Tx2	
	Balance	Contracts whose initial assets exceed the attack profit are likely to be victims.	ator(ExploitContractAddr)	
	Fund Se	ource Contracts and Wallets funded from mixing servers (e.g., Tornado Cash) are likely malicious.	Victim Conti	ract
	Activiti	es Contracts that frequently interact with users exhibiting diverse behaviors are likely benign.		
	Source			

Your 58		<pre>Deploy UniswapPair() =&gt; [TempleFraxPool]     [Temple].transferFrom([Temple DAO Wallet], [TempleFraxPool], 37500000 ether)     [Frax].transferFrom([Temple DAO Wallet], [TempleFraxPool], 9375000 ether)     [TempleFraxPool].transfer([Temple DAO Wallet], 18750000 ether) [Temple DAO Wallet] deploy StaxLPToken() =&gt; [StaxLPToken]</pre>	ntract	
	Tx <sub>3</sub>	[Temple DAO Wallet] deploy StaxLPStaking([StaxLPToken], [TempleFraxPool]) => [StaxLPStak [Temple DAO Wallet] . Pinpoint all attack-related malicious	ing]	STJNG
	<sup>06-13</sup> entit <sup>Tx₅</sup> trans	TempleFraxPool].transferFrom([Temple DAO Wallet], [StaxLPStaking], 18750000 eth Cacata Pools].mint([Temple DAO Wallet], 18750000 ether)	Tx1: Deploy (Tx3: Exploit)	Exploit Contract
	iterati • <u>The</u>	ive fashion. <u>e attacking transaction</u> is detected based on profit.		x3 (inner)
	Name	Description		
	Lifespan Balance	Contracts deployed shortly before an attack are likely malicious. Contracts whose initial assets exceed the attack profit are likely to be victims.	• <u>ator(ExploitContractAd</u>	
	Fund Sour	rce Contracts and Wallets funded from mixing servers (e.g., Tornado Cash) are likely malicious.		Victim Contract
	Activities			
	Source Co	ode Contracts with unverified source code are likely malicious.		

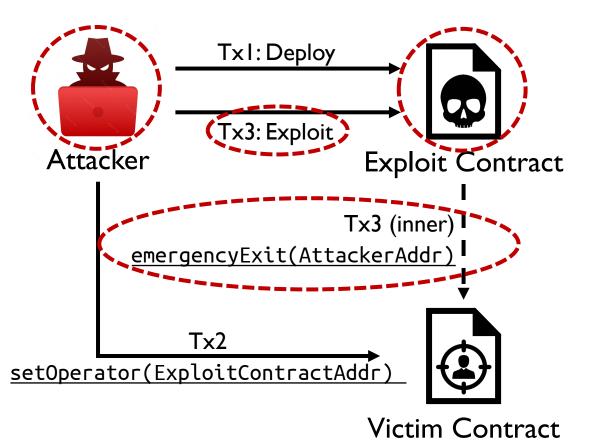
Your 59	06-02-2022 • Tx <sub>2</sub> 06-03-2022 •	<pre>Deploy UniswapPair() =&gt; [TempleFraxPool]       [Temple].transferFrom([Temple DAO Wallet], [TempleFraxPool], 37500000 ether)       [Frax].transferFrom([Temple DAO Wallet], [TempleFraxPool], 9375000 ether)       [TempleFraxPool].transfer([Temple DAO Wallet], 18750000 ether) [Temple DAO Wallet] deploy StaxLPToken() =&gt; [StaxLPToken] [Temple DAO Wallet] deploy StaxLPStaking([StaxLPToken], [TempleFraxPool]) =&gt; [StaxLPStaking([StaxLPToken], [TempleFraxPool])] =&gt; [StaxLPStaking([StaxLPToken]]] =&gt; [StaxLPStaking([StaxLPStaking([StaxLPToken]]] =&gt; [StaxLPStaking([StaxLPStaki</pre>	ntract Cation	STJNG
	<sup>06-13</sup> entit <sup>Tx</sup> ₅ trans Trans	Tempe DAO Wallet] [StaxLPToken].transfer@wer([StaxLPStaking]) .Pinpoint all attack-related malicious ies; including Pacicounts of DAO Wallet], [StaxLPStaking], 18750000 ether) . [TempleFraxPool].transferFrom([Temple DAO Wallet], [StaxLPStaking], 18750000 ether) actions and accounts are pinpointed in an ive fashion.	Tx1: Deploy Tx3: Exploit	Exploit Contract
	• <u>The</u>	<u>e attacking transaction</u> is detected based on profit.	T <u>emergencyExit(Attac</u>	x3 (inner) <u>kerAddr)</u>
		-	— Tx2	
	Lifespan Balance	Contracts deployed shortly before an attack are likely malicious. Contracts whose initial assets exceed the attack profit are likely to be victims.		
	Fund Sou	rce Contracts and Wallets funded from mixing servers (e.g., Tornado Cash) are likely malicious.		Victim Contract
	Activities			
	Source Co			

60

## Attack Information Identification

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

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



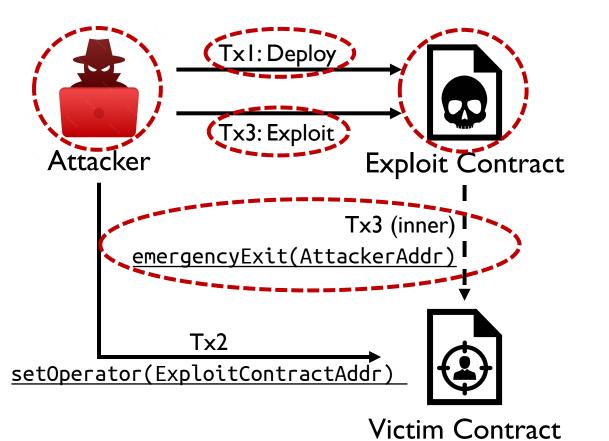


61

### Attack Information Identification

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

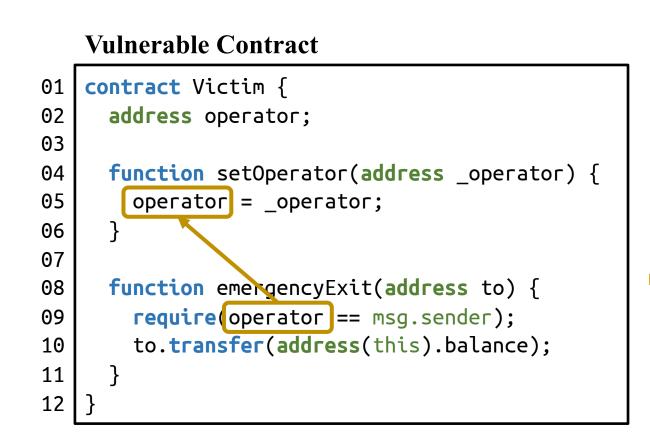
- <u>The attacking transaction</u> is detected based on the profit.
- <u>Accounts</u> are pinpointed based on historical behaviors.
- <u>Transactions</u> are pinpointed by read-write dependency.

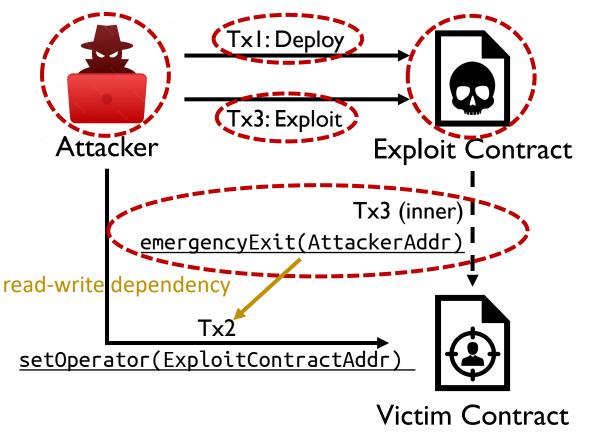


62

## Attack Information Identification





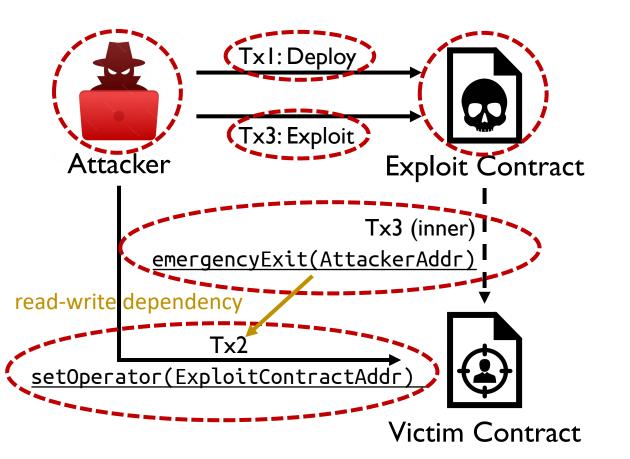


63

## Attack Information Identification

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

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





64

### 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.



Attacker



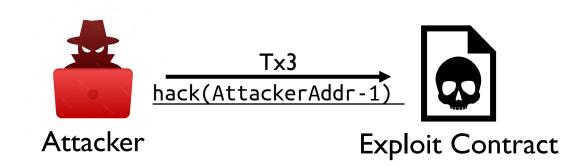
**Exploit Contract** 

65

## 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** 

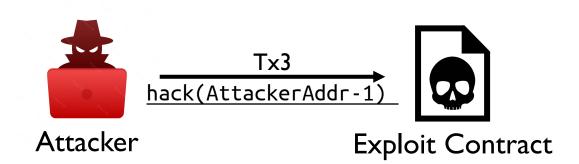
```
01 contract Exploit {
02 function hack(address toAddr) {
03 if (msg.sender == AttackerAddr) {
04 VICTIM.emergencyExit(toAddr + 1);
05 }
06 }
07 }
```

66

## Counterattack Smart Contract Synthesis

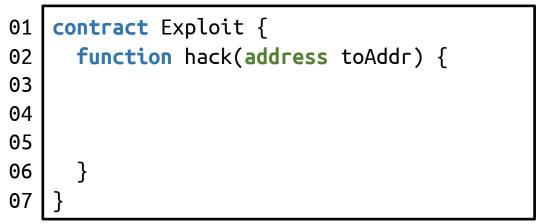


Goal: For each exploit contract, we aim to synthesize a counterattack contract that ensures the stolen funds are sent to accounts under our control.



#### **Exploit Contract**

01	<pre>contract Exploit {</pre>
02	<pre>function hack(address toAddr) {</pre>
03	<pre>if (msg.sender == AttackerAddr) {</pre>
04	<pre>VICTIM.emergencyExit(toAddr + 1);</pre>
05	}
06	}
07	}

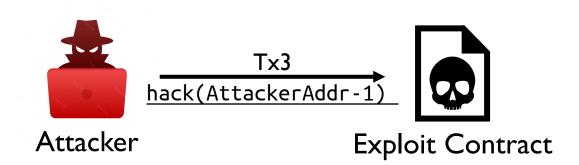


67

## 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**

01	<pre>contract Exploit {</pre>
02	<pre>function hack(address toAddr) {</pre>
03	<pre>if (msg.sender == AttackerAddr) {</pre>
04	<pre>VICTIM.emergencyExit(toAddr + 1);</pre>
05	}
06	}
07	}

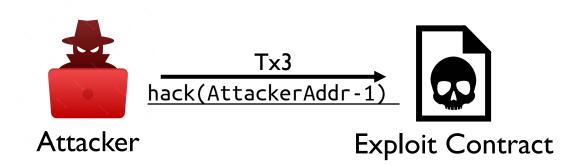


68

### 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**

01	<pre>contract Exploit {</pre>
02	<pre>function hack(address toAddr) {</pre>
03	<pre>if (msg.sender == AttackerAddr) {</pre>
04	<pre>VICTIM.emergencyExit(toAddr + 1);</pre>
05	}
06	}
07	}

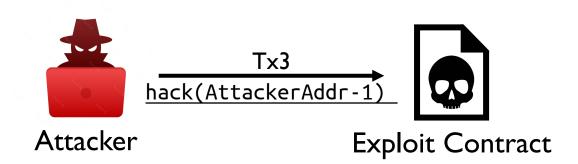


69

### 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**

01	<pre>contract Exploit {</pre>
02	<pre>function hack(address toAddr) {</pre>
03	<pre>if (msg.sender == AttackerAddr) {</pre>
04	<pre>VICTIM.emergencyExit(toAddr + 1);</pre>
05	}
06	}
07	}

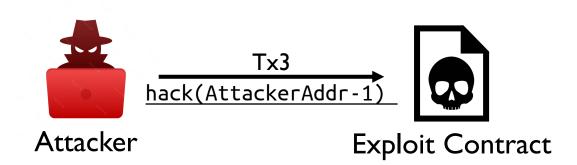


70

### 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**

01	<pre>contract Exploit {</pre>
02	<pre>function hack(address toAddr) {</pre>
03	<pre>if (msg.sender == AttackerAddr) {</pre>
04	<pre>VICTIM.emergencyExit(toAddr + 1);</pre>
05	}
06	}
07	}



71

## 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

72



#### 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.

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

#### 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



#### <u>Dataset</u>:

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 **8.5** seconds (only two cases exceed **1.00** second).

## Limitations



- <u>Adaptive Evasion</u>: Multiple adaptive evasion techniques, such as code obfuscation, may exist against STING, enabling attacks to circumvent our defense mechanism.
- <u>Private Transactions</u>: Private transactions provide a mechanism for blockchain users to execute transactions that remain hidden until being confirmed.
- <u>Blind Spots</u>: STING does not provide comprehensive protection against all DeFi attacks.
- <u>Performance Issue</u>: 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**



Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In 2022 IEEE Symposium on Security and Privacy (SP), pages 198–214. IEEE, 2022.

Kaihua Qin, Liyi Zhou, Benjamin Livshits, and Arthur Gervais. Attacking the defi ecosystem with flash loans for fun and profit. In Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part I, pages 3–32. Springer, 2021.

Liyi Zhou, Kaihua Qin, Antoine Cully, Benjamin Livshits, and Arthur Gervais. On the just-intime discovery of profit-generating transactions in defi protocols. In 2021 IEEE Symposium on Security and Privacy (SP), pages 919–936. IEEE, 2021.

Dabao Wang, Siwei Wu, Ziling Lin, Lei Wu, Xingliang Yuan, Yajin Zhou, Haoyu Wang, and Kui Ren. Towards a first step to understand flash loan and its applications in defi ecosystem. In Proceedings of the Ninth International Workshop on Security in Blockchain and Cloud Computing, pages 23–28, 2021.



# Thank You

#### Zhuo Zhang, zhan3299@purdue.edu





August 9, 2023