Attack Vectors in DeFi Ecosystem

Chiachih Wu

杭州派盾信安科技有限公司
PeckShield.com

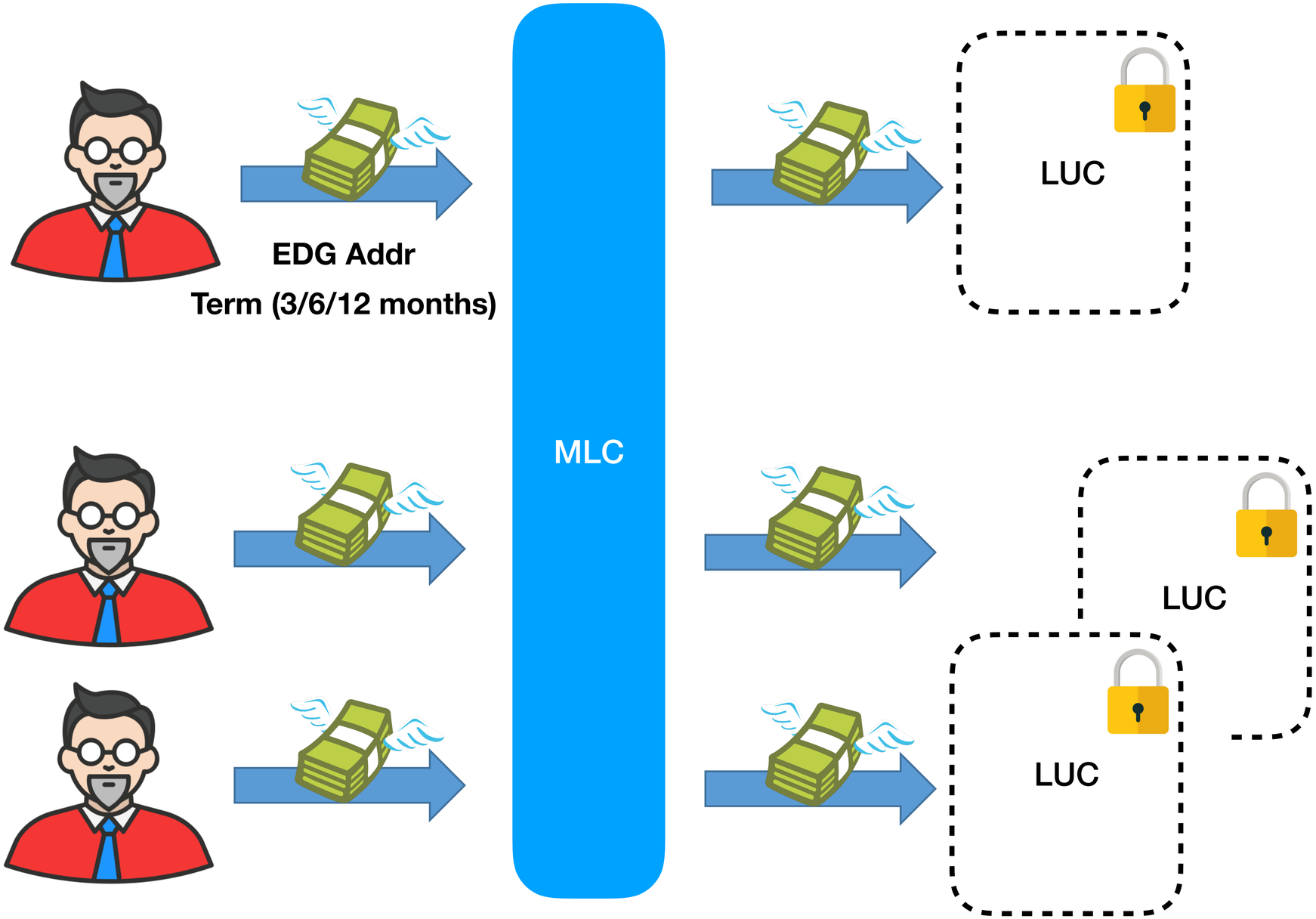
About Me

- @chiachih_wu
- Co-Founder & Research VP at PeckShield
- Auditing/Researching Smart Contracts/DApps/Blockchains
- Founder & Former Team Lead of 360 C0RE Team
- Hacking Android/Linux Since 2013
- NC State Computer Science PhD (Virtualization/System Security)

DeFi Security Incidents

- 2018–12 Compound **Smart Contract**
- 2019–01 HUSD **Business Logic**
- 2019–02 Nuo Network **Smart Contract**
- 2019–05 MakerDAO **Smart Contract**
- 2019–06 Synthetix Oracle **Oracle**
- 2019–07 Edgeware MLC DoS **Smart Contract**
- 2019–07 0x Signature Verification **Smart Contract**
- 2019–09 AirSwap Wrapper Authentication **Smart Contract**

Edgeware MLC DoS (1/5)



Edgeware MLC DoS (2/5)

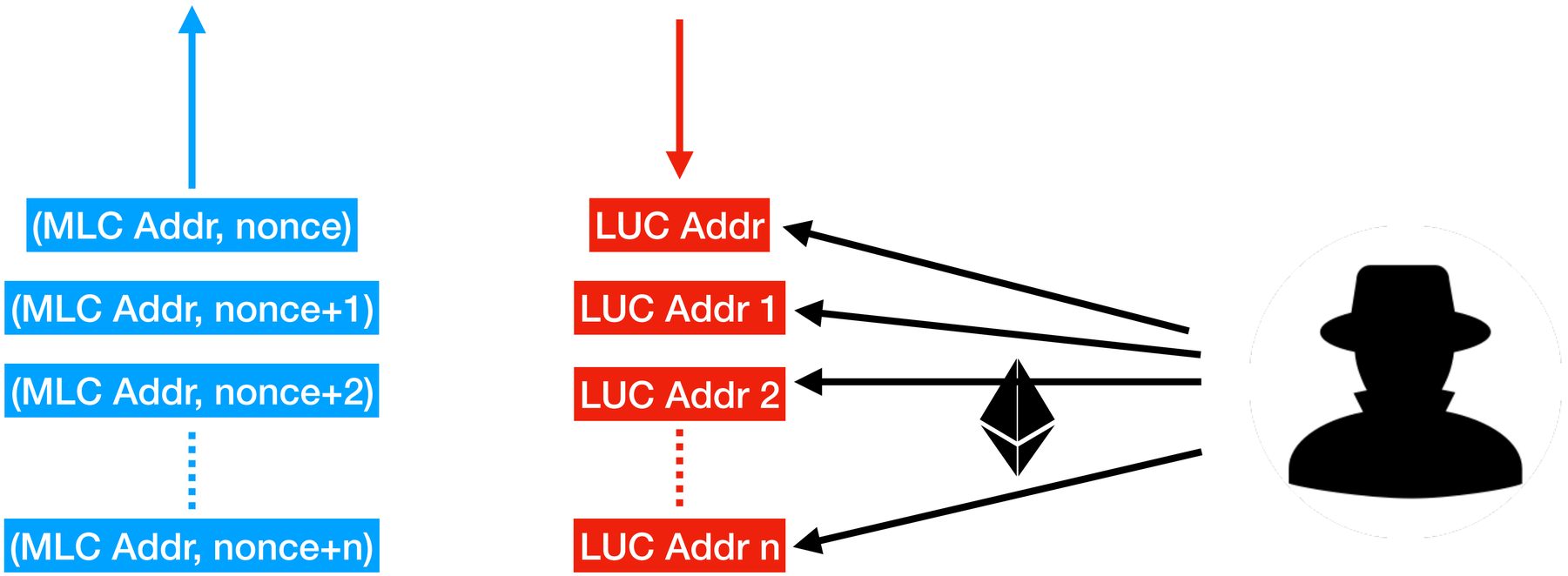
```
57      function lock(Term term, bytes calldata edgewareAddr, bool isValidator)
58          external
59          payable
60          didStart
61          didNotEnd
62      {
63          uint256 eth = msg.value;
64          address owner = msg.sender;
65          uint256 unlockTime = unlockTimeForTerm(term);
66          // Create ETH lock contract
67          Lock lockAddr = (new Lock).value(eth)(owner, unlockTime);
68          // ensure lock contract has all ETH, or fail
69          assert(address(lockAddr).balance == msg.value);
70          emit Locked(owner, eth, lockAddr, term, edgewareAddr, isValidator, now);
71      }
```

create LUC contract and deposit ETH

DoS … Why?

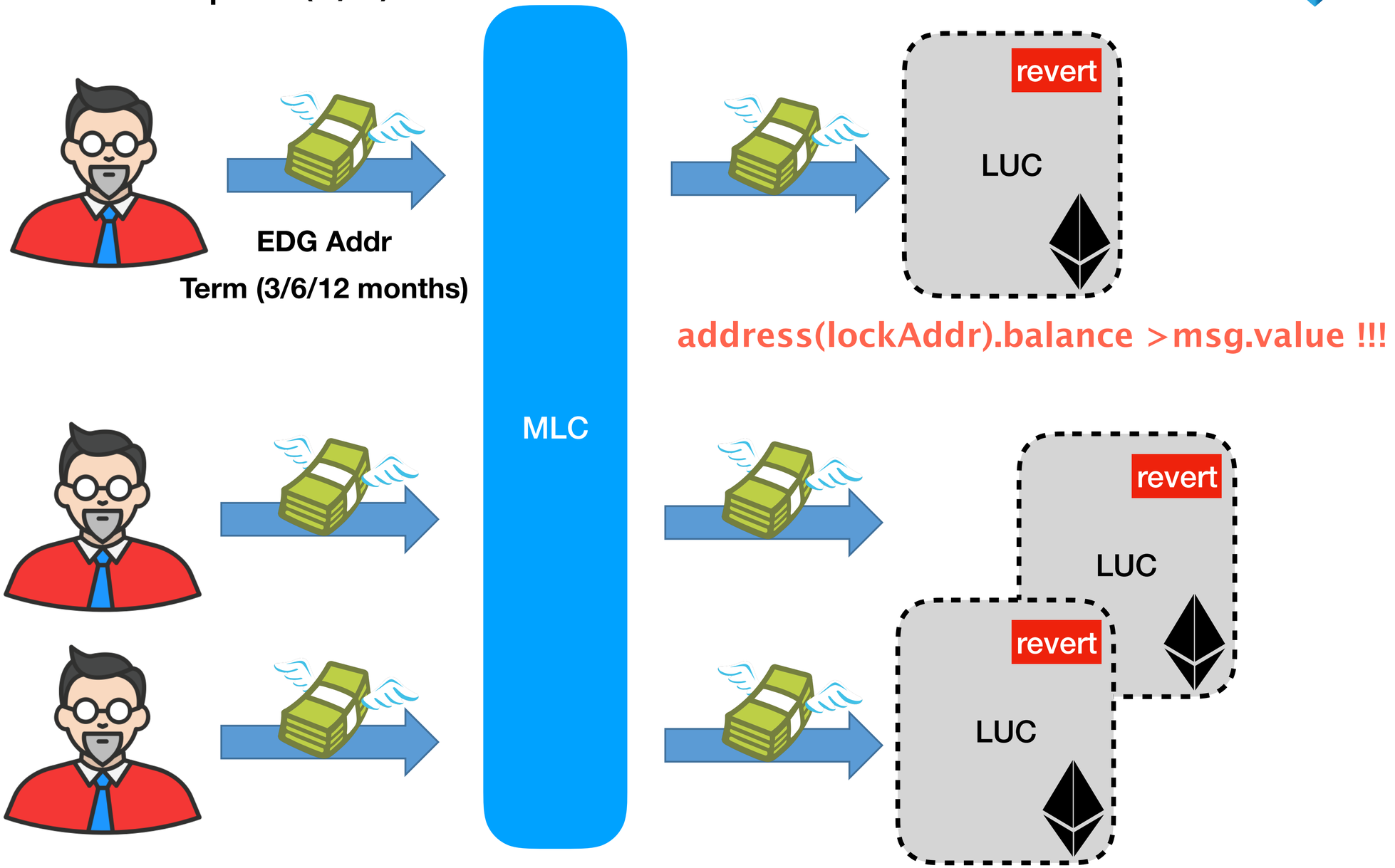
Edgeware MLC DoS: Exploit (3/5)

```
1 function nextAddress(address _origin, uint256 _nonce) internal view returns(address addr) {
2     if(_nonce == 0x00)      return address(keccak256(abi.encodePacked(uint8(0xd6), uint8(0x94), _origin, uint8(0x80))));
3     if(_nonce <= 0x7f)      return address(keccak256(abi.encodePacked(uint8(0xd6), uint8(0x94), _origin, uint8(_nonce))));
4     if(_nonce <= 0xff)      return address(keccak256(abi.encodePacked(uint8(0xd7), uint8(0x94), _origin, uint8(0x81), uint8(_nonce))));
5     if(_nonce <= 0xffff)    return address(keccak256(abi.encodePacked(uint8(0xd8), uint8(0x94), _origin, uint8(0x82), uint16(_nonce))));
6     if(_nonce <= 0xffffff)  return address(keccak256(abi.encodePacked(uint8(0xd9), uint8(0x94), _origin, uint8(0x83), uint24(_nonce))));
7     return address(keccak256(abi.encodePacked(uint8(0xda), uint8(0x94), _origin, uint8(0x84), uint32(_nonce))));
8 }
```



Deposit ETH into LUC Addrs before They're Created

Edgeware MLC DoS: Exploit (4/5)



Edgeware MLC DoS: Patch (5/5)

```
--- MLC_v1.sol  2019-08-09 12:22:10.000000000 +0800
+++ MLC_v2.sol  2019-08-09 12:22:29.000000000 +0800
@@ -1,5 +1,5 @@
 /**
- *Submitted for verification at Etherscan.io on 2019-05-14
+ *Submitted for verification at Etherscan.io on 2019-06-19
 */

 pragma solidity ^0.5.0;
@@ -65,8 +65,8 @@ contract Lockdrop {
         uint256 unlockTime = unlockTimeForTerm(term);
         // Create ETH lock contract
         Lock lockAddr = (new Lock).value(eth)(owner, unlockTime);
-        // ensure lock contract has all ETH, or fail
-        assert(address(lockAddr).balance == msg.value);
+        // ensure lock contract has at least all the ETH, or fail
+        assert(address(lockAddr).balance >= msg.value);
         emit Locked(owner, eth, lockAddr, term, edgewareAddr, isValidator, now);
     }

```

itchyDAO in MakerDAO Voting Contract (1/4)

PeckShield

```
69      function etch(address[] memory yays)
70          public
71          note
72          returns (bytes32 slate)
73      {
74          require( yays.length <= MAX_YAYS );
75          requireByteOrderedSet(yays);
76
77          bytes32 hash = keccak256(abi.encodePacked(yays));
78          slates[hash] = yays;
79          emit Etch(hash);
80          return hash;
81      }
82
83      function vote(address[] memory yays) public returns (bytes32)
84          // note  both sub-calls note
85      {
86          bytes32 slate = etch(yays);
87          vote(slate);
88          return slate;
89      }
90
91      function vote(bytes32 slate)
92          public
93          note
94      {
95          uint weight = deposits[msg.sender];
96          subWeight(weight, votes[msg.sender]);
97          votes[msg.sender] = slate;
98          addWeight(weight, votes[msg.sender]);
99      }
```

1

2

3

itchyDAO in MakerDAO Voting Contract (2/4)

PeckShield

```
69      function etch(address[] memory yays)
70          public
71          note
72          returns (bytes32 slate)
73      {
74          require( yays.length <= MAX_YAYS );
75          requireByteOrderedSet(yays);
76
77  3       bytes32 hash = keccak256(abi.encodePacked(yays));
78          slates[hash] = yays;
79          emit Etch(hash);
80          return hash;
81      }
82
83      function vote(address[] memory yays) public returns (bytes32)
84          // note  both sub-calls note
85      {
86          bytes32 slate = etch(yays);
87          vote(slate);
88          return slate;
89      }
90
91      function vote(bytes32 slate)                              2
92          public
93          note
94      {
95          uint weight = deposits[msg.sender];
96          subWeight(weight, votes[msg.sender]);
97          votes[msg.sender] = slate;
98          addWeight(weight, votes[msg.sender]);
99      }
```

1. pre-calculate a slate with one victim proposal and one “salt” proposal

do nothing due to slates[slate] is empty

```
110     function addWeight(uint weight, bytes32 slate)
111         internal                                  yays.length = 0
112     {
113         address[] storage yays = slates[slate];
114         for( uint i = 0; i < yays.length; i++) {
115             approvals[yays[i]] = add(approvals[yays[i]],
116         }
117     }
118
119     function subWeight(uint weight, bytes32 slate)
120         internal                                  yays.length = 0
121     {
122         address[] storage yays = slates[slate];
123         for( uint i = 0; i < yays.length; i++) {
124             approvals[yays[i]] = sub(approvals[yays[i]],
125         }
126     }
```

itchyDAO in MakerDAO Voting Contract (3/4)

PeckShield

```
49      function lock(uint wad)
50          public
51          note
52      {
53          GOV.pull(msg.sender, wad);
54          IOU.mint(msg.sender, wad);
55          deposits[msg.sender] = add(deposits[msg.sender], wad);
56          addWeight(wad, votes[msg.sender]);
57      }
58
59      function free(uint wad)
60          public
61          note
62      {
63          deposits[msg.sender] = sub(deposits[msg.sender], wad);
64          subWeight(wad, votes[msg.sender]);
65          IOU.burn(msg.sender, wad);
66          GOV.push(msg.sender, wad);
67      }
```

4

The victim proposal is subWeight()'ed w/o any addWeight()

itchyDAO in MakerDAO Voting Contract (4/4)

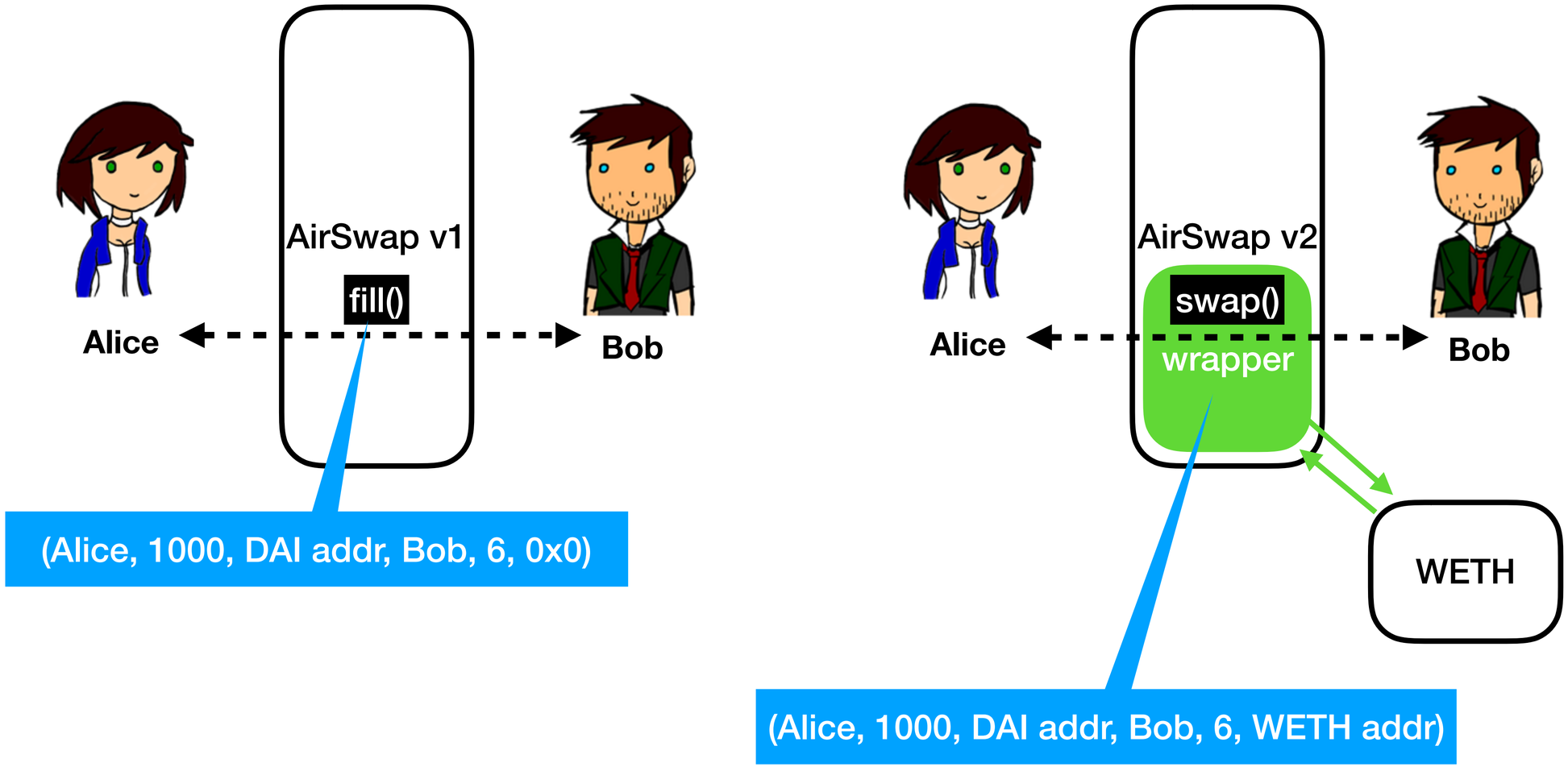
PeckShield

```
110     function addWeight(uint weight, bytes32 slate)
111         internal
112     {
113         address[] storage yays = slates[slate];
114         for( uint i = 0; i < yays.length; i++) {
115             approvals[yays[i]] = add(approvals[yays[i]], weight);
116         }
117     }
118
119     function subWeight(uint weight, bytes32 slate)
120         internal
121     {
122         address[] storage yays = slates[slate];
123         for( uint i = 0; i < yays.length; i++) {
124             approvals[yays[i]] = sub(approvals[yays[i]], weight);
125         }
126     }
```

A voter of the victim proposal can't free() her tokens afterward.

```
18 contract DSMath {
19     function add(uint x, uint y) internal pure returns (uint z) {
20         require((z = x + y) >= x, "ds-math-add-overflow");
21     }
22     function sub(uint x, uint y) internal pure returns (uint z) {
23         require((z = x - y) <= x, "ds-math-sub-underflow");
24     }
```

AirSwap Wrapper Authentication (1/5)



AirSwap Wrapper Authentication (2/5)

PeckShield

Wrapper

```
* @notice Send an Order
* @dev Taker must authorize this contract on the swapContract
* @dev Taker must approve this contract on the wethContract
* @param _order Types.Order
    */
function swap(Types.Order calldata _order) external payable {
    // Ensure message sender is taker wallet.
    require(_order.taker.wallet == msg.sender,
      "SENDER_MUST_BE_TAKER");
    ...
    // Perform the swap.
    swapContract.swap(_order);
    ...
}
```

Swap

```
// Validate the taker side of the trade.
address finalTakerWallet;

if (_order.taker.wallet == address(0)) {
  /**
    * Taker is not specified. The sender of the transaction becomes
    * the taker of the _order.
    */
  finalTakerWallet = msg.sender;

} else {
  /**
    * Taker is specified. If the sender is not the specified taker,
    * determine whether the sender has been authorized by the taker.
    */
  if (msg.sender != _order.taker.wallet) {
    require(isAuthorized(_order.taker.wallet, msg.sender),
      "SENDER_UNAUTHORIZED");
  }
  // The specified taker is all clear.
  finalTakerWallet = _order.taker.wallet;
}
```

Taker needs to authorize the wrapper contract

AirSwap Wrapper Authentication (3/5)

Swap

```
// Validate the maker side of the trade.
if (_order.signature.v == 0) {
  /**
    * Signature is not provided. The maker may have authorized the sender
    * to swap on its behalf, which does not require a signature.
    */
  require(isAuthorized(_order.maker.wallet, msg.sender),
    "SIGNER_UNAUTHORIZED");
}
```

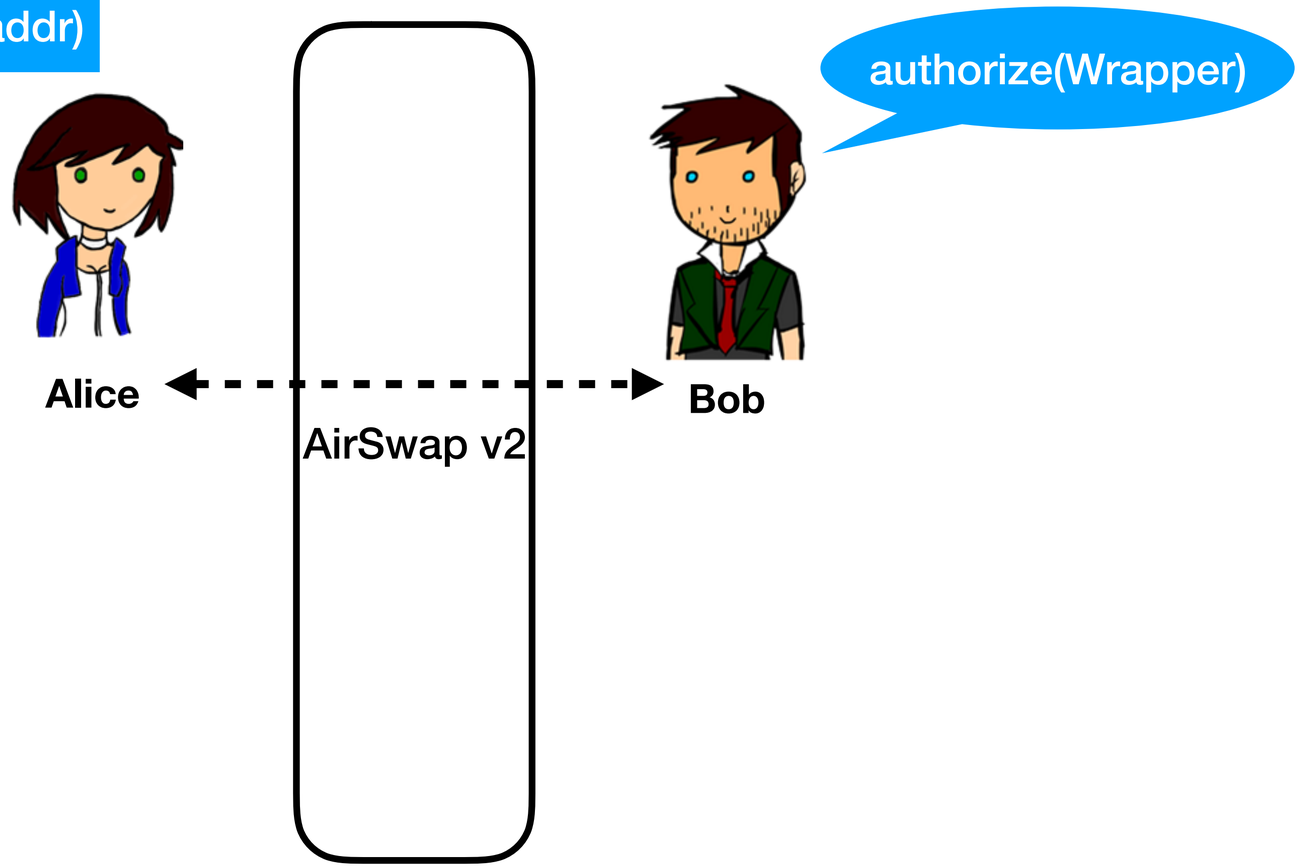
Swap

```
// Validate the taker side of the trade.
address finalTakerWallet;

der.taker.wallet == address(0)) {

aker is not specified. The sender of the transaction becomes
he taker of the _order.

TakerWallet = msg.sender;

{

  * Taker is specified. If the sender is not the specified taker,
  * determine whether the sender has been authorized by the taker.
  */
if (msg.sender != _order.taker.wallet) {
  require(isAuthorized(_order.taker.wallet, msg.sender),
    "SENDER_UNAUTHORIZED");
}
// The specified taker is all clear.
finalTakerWallet = _order.taker.wallet;
}
```

Anyone can impersonate a previous taker to “make” an order with signature.v = 0

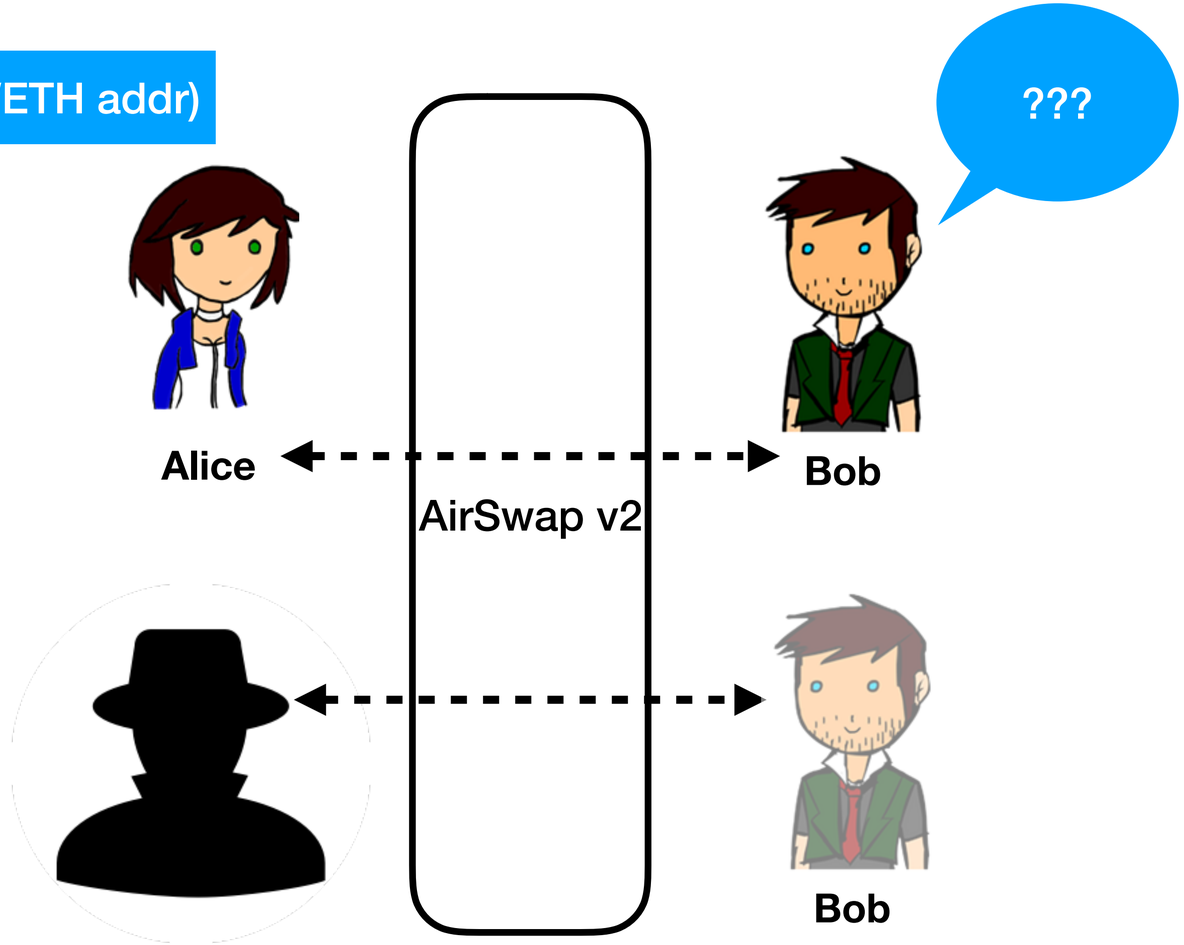
AirSwap Wrapper Authentication (4/5)

(Alice, 1000, DAI addr, Bob, 6, WETH addr)



AirSwap Wrapper Authentication (5/5)

(Alice, 1000, DAI addr, Bob, 6, WETH addr)



(Bob, 1000, DAI addr, BlackHat, 0, WETH addr)

Attack Vectors that You Should to Put More Eyes On

- Smart Contract
  - Underlying EVM behaviors
    - 0x, Edgeware
  - Corner cases that cause Denial–of–Service
    - Edgeware, MakerDAO
  - Excessive authorization that allow bad actors to trigger hidden logic
    - AirSwap
- Oracle: Synthetix
- Business Logic: HUSD

Thank You!

https://peckshield.com

contact@peckshield.com