SharkTeam

A Vulnerability Perspective Analysis of MoveLanguage Security —— ProposalAttack

Dec 12, 2022

SharkTeam, a leading blockchain security service team, offers smart contract audit services for developers. To satisfy the demands of different clients, thesmart contract audit services provide both manual auditing and automated auditing.

We implement almost 200 auditing contents that cover four aspects: high-level language layer, virtual machine layer, blockchain layer, and business logiclayer, ensuring that smart contracts are completely guaranteed and Safe.

In the previous “Top 10 Smart Contracts Security Threats” series, SharkTeam summarized and analyzed the top 10 vulnerabilities in the smart contract space based on historical smart contract security incidents. These vulnerabilities were usually found in Solidity smart contracts before, so will they be the same for Move smart contracts?

The SharkTeam [Move Language Security Analysis and Contract Audit Essentials] course series will take you step-by-step into the content, including permission vulnerabilities, re-entry vulnerabilities, logical checksum vulnerabilities, function malicious initialization, fallback attacks, manipulation of the prophecy machine, contract upgrade vulnerabilities, sandwich attacks, replay attacks, and proposal attacks. This chapter covers [proposal attack].

1. Introduction to Proposal Attack

The proposal attack targets decentralized autonomous organizations (DAOs). In DAO, participants will put forward a series of proposals on future protocol upgrades, fund management, etc. In order for a proposal to take effect, accounts holding governance tokens need to vote on it. DAO’s governance tokens represent the number of votes cast. Holders of governance tokens have DAO’s governance authority and can participate in a series of activities such as proposal initiation, voting, and execution. The more governance tokens you hold, the greater your authority, and even affect the degree of decentralization.

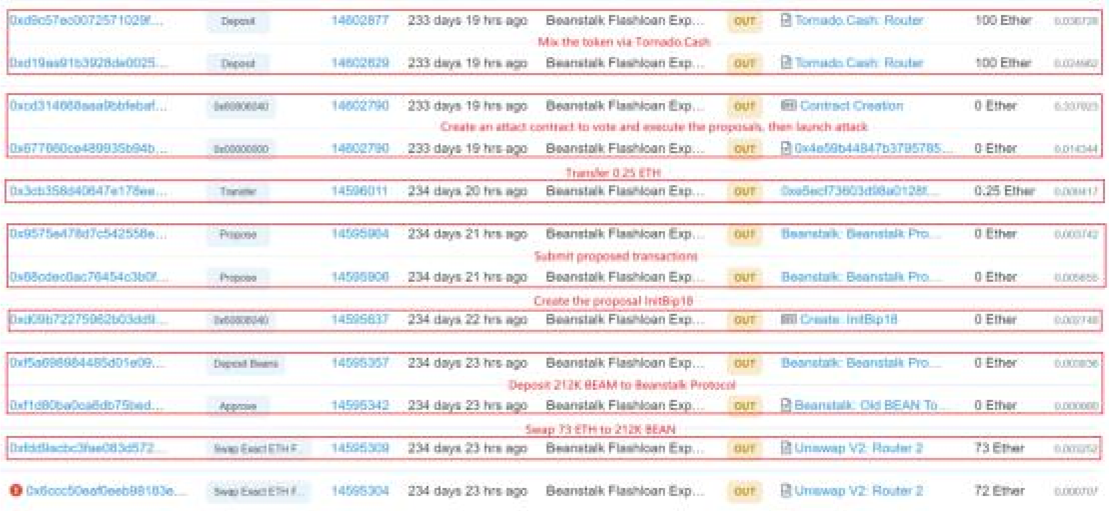
While proposal governance is good for building a decentralized future, it also has some drawbacks. Users with a small proportion of governance tokens have little influence on the decision-making of proposals. DAO’s governance is passive and negligent and their participation is low. Users with a high proportion of governance tokens will actively participate in governance and have little influence on proposal decisions. If it is large, it will even take the initiative to acquire the governance tokens held by passive people, which further leads to the centralization of voting rights in the DAO and turns to serve the interests of a few people. Users holding more governance tokens have excessive voting rights.

When a user’s voting power exceeds the voting threshold, the submission and execution of the proposal can be decided by a single user, which completely violates the intention of DAO. This constitutes a prerequisite for a proposal attack when a proposal can be decided by a single user, and at the same time that user can be the attacker who initiates the proposal attack.

In DAO, the attacker holds absolute voting rights for a long time or temporarily, and then initiates and executes illegal proposals, harming the interests of others and benefiting himself. This behavior is called proposal attack. For example, Beanstalk Farms and Fortress Loans in the Solidity ecosystem have both suffered proposal attacks.

2. Attack on Beanstalk Farms

Beanstalk Farms, an algorithmic stablecoin project on Ethereum, was hacked on April 17, 2022 and lost more than $80 million, including 24,830 ETH and 36 million BEAN. The complete attack process and transactions of this event are as follows:

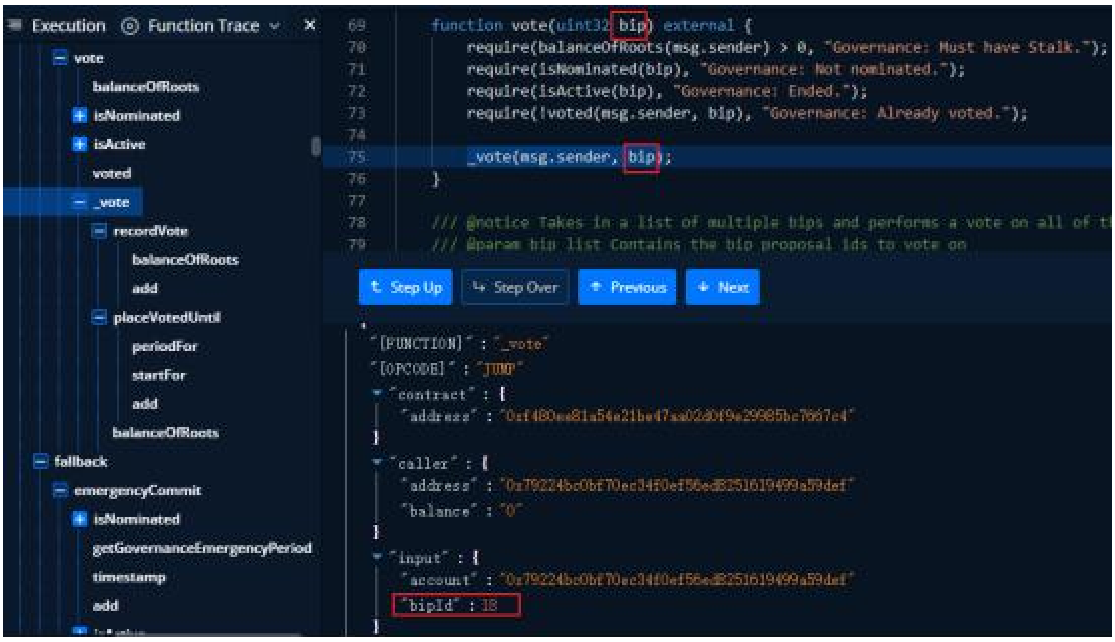


0xd9c57ec0072571029f...	Deposit	14602877	233 days 19 hrs ago	Beanstalk Flashloan Exp...	OUT	Tornado.Cash: Router	100 Ether	0.036728
				Mix the token via Tornado.Cash				
0xd19aa91b3928de0025...	Deposit	14602829	233 days 19 hrs ago	Beanstalk Flashloan Exp...	OUT	Tornado.Cash: Router	100 Ether	0.024662
0xcd314668aaa9bbfebaf...	0x60806040	14602790	233 days 19 hrs ago	Beanstalk Flashloan Exp...	OUT	Contract Creation	0 Ether	0.337023
				Create an attact contract to vote and execute the proposals, then launch attack				
0x677660ce489935b94b...	0x00000000	14602790	233 days 19 hrs ago	Beanstalk Flashloan Exp...	OUT	0x4e59b44847b3795785...	0 Ether	0.014344
				Transfer 0.25 ETH				
0x3cb358d40647e178ee...	Transfer	14596011	234 days 20 hrs ago	Beanstalk Flashloan Exp...	OUT	0xa5ecf73603d98a0128f...	0.25 Ether	0.008417
0x9575e478d7c542558e...	Propose	14595904	234 days 21 hrs ago	Beanstalk Flashloan Exp...	OUT	Beanstalk: Beanstalk Pro...	0 Ether	0.003742
				Submit proposed transactions				
0x68cdec0ac76454c3b0f...	Propose	14595906	234 days 21 hrs ago	Beanstalk Flashloan Exp...	OUT	Beanstalk: Beanstalk Pro...	0 Ether	0.005656
				Create the proposal InitBip18				
0xd09b72275962b03dd9...	0x60806040	14595837	234 days 22 hrs ago	Beanstalk Flashloan Exp...	OUT	Create: InitBip18	0 Ether	0.002746
0xf5a698984485d01e09...	Deposit Beans	14595357	234 days 23 hrs ago	Beanstalk Flashloan Exp...	OUT	Beanstalk: Beanstalk Pro...	0 Ether	0.003836
				Deposit 212K BEAM to Beanstalk Protocol				
0xf1d80ba0ca6db75bed...	Approve	14595342	234 days 23 hrs ago	Beanstalk Flashloan Exp...	OUT	Beanstalk: Old BEAN To...	0 Ether	0.000000
				Swap 73 ETH to 212K BEAN				
0xfdd9acbc3fae083d572...	Swap Exact ETH F...	14595309	234 days 23 hrs ago	Beanstalk Flashloan Exp...	OUT	Uniswap V2: Router 2	73 Ether	0.003252
❗ 0x6ccc50eaf0eeb98183e...	Swap Exact ETH F...	14595304	234 days 23 hrs ago	Beanstalk Flashloan Exp...	OUT	Uniswap V2: Router 2	72 Ether	0.000707

The key process for the attacker to initiate the voting and execution of the proposal by attacking the contract is as follows:

(1) Through flash loans, adding liquidity and token exchange, the attacker obtained a large amount of governance tokens, totaling 58,924,887 BEAN3CRV-f

(2) Use all the BEAN3CRV-f obtained above to vote on the proposal, so that the proposal is passed and implemented.



```
vote
  balanceOfRoots
  isNominated
  isActive
  voted
  _vote
    recordVote
      balanceOfRoots
      add
    placeVotedUntil
      periodFor
      startFor
      add
    balanceOfRoots
fallback
  emergencyCommit
    isNominated
    getGovernanceEmergencyPeriod
    timestamp
    add
```

```
    function vote(uint32 bip) external {
        require(balanceOfRoots(msg.sender) > 0, "Governance: Must have Stalk.");
        require(isNominated(bip), "Governance: Not nominated.");
        require(isActive(bip), "Governance: Ended.");
        require(!voted(msg.sender, bip), "Governance: Already voted.");

        _vote(msg.sender, bip);
    }

    /// @notice Takes in a list of multiple bips and performs a vote on all of t
    /// @param bip list Contains the bip proposal ids to vote on
```

```
{
  "[FUNCTION]" : "_vote"
  "[OPCODE]" : "JUMP"
  "contract" : {
    "address" : "0xf480ee81a54e21be47aa02d0f9e29985bc7667c4"
  }
  "caller" : {
    "address" : "0x79224bc0bf70ec34f0ef56ed8251619499a59def"
    "balance" : "0"
  }
  "input" : {
    "account" : "0x79224bc0bf70ec34f0ef56ed8251619499a59def"
    "bipId" : 18
  }
```

After the implementation of the proposal, the attacker obtained 36,084,584 BEAN, 0.5407 UNI-V2, 874,663,982 NEAN3CRV-f and 60,562,844 BEANLUSD-f

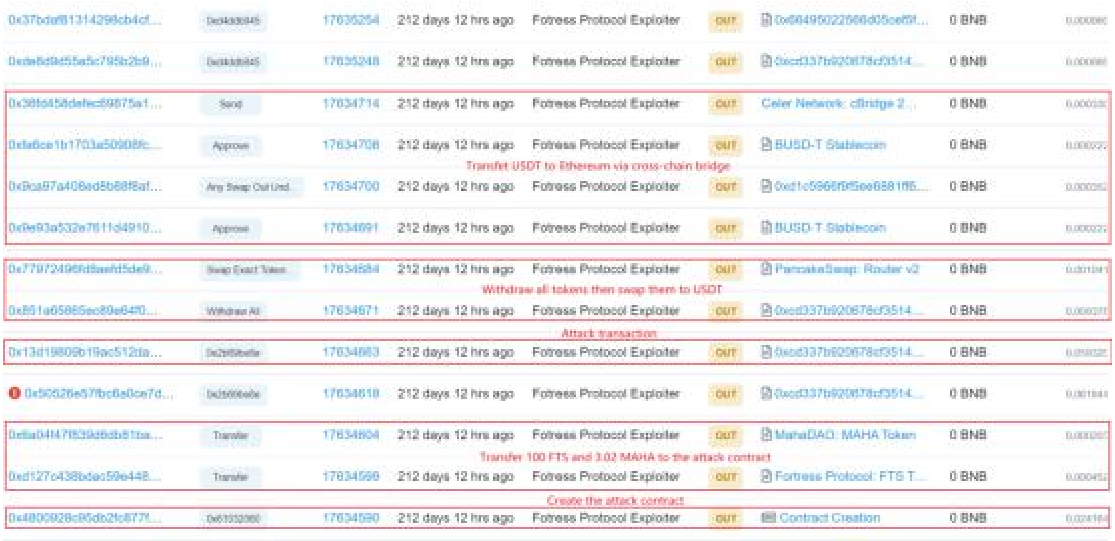
(3) Remove the liquidity to obtain the tokens in the trading pair, then return the amount of the flash loan and the handling fee, and donate 250k USDC to Ukraine Crypto Donation.

(4) Convert the remaining Tokens to WETH to withdraw the resulting 24,830 WETH and transfer it to the attacker's address to complete the attack.

In this proposal attack, the attacker obtained a large number of governance tokens through flash loans, and stole absolute control in the DAO, that is, the proposal can be passed and executed without the need for other people to vote. This makes the adoption and execution of the illegal proposal InitBip18 submitted by it can be decided by the attacker's own vote. In the end, the illegal proposal InitBip18 was successfully implemented, allowing the attacker to obtain a large amount of illegal income.

3. Fortress Loans attack incident

Binance Smart Chain's Fortress Loans was hacked on May 9, 2022. The attack caused the project party to lose 1048.1 ETH and 400,000 DAI. The event attack process and its transactions are as follows:



0x37bdaf81314298cb4cf...	0x04dd0845	17635254	212 days 12 hrs ago	Fotress Protocol Exploiter	OUT	0x6849502256d05cef9f...	0 BNB	0.000080
0xde8d9d55a5c795b2b9...	0x04dd0845	17635248	212 days 12 hrs ago	Fotress Protocol Exploiter	OUT	0xcd337b920678cf3514...	0 BNB	0.000080
0x38fd458defec69875a1...	Send	17634714	212 days 12 hrs ago	Fotress Protocol Exploiter	OUT	Celer Network: cBridge 2...	0 BNB	0.000330
0xfa6ce1b1703a50908fc...	Approve	17634708	212 days 12 hrs ago	Fotress Protocol Exploiter	OUT	BUSD-T Stablecoin	0 BNB	0.000222

Transfet USDT to Ethereum via cross-chain bridge

0x8ca97a408ed8b88f8af...	Any Swap Out Und...	17634700	212 days 12 hrs ago	Fotress Protocol Exploiter	OUT	0xd1c5966f5ee6881f6...	0 BNB	0.000250
0x9e93a532e7611d4910...	Approve	17634691	212 days 12 hrs ago	Fotress Protocol Exploiter	OUT	BUSD-T Stablecoin	0 BNB	0.000222
0x7797249fd8aefd5de9...	Swap Exact Token...	17634684	212 days 12 hrs ago	Fotress Protocol Exploiter	OUT	PancakeSwap: Router v2	0 BNB	0.001041

Withdraw all tokens then swap them to USDT

0x851a65865ec89e6470...	Withdraw All	17634671	212 days 12 hrs ago	Fotress Protocol Exploiter	OUT	0xcd337b920678cf3514...	0 BNB	0.000271

Attack transaction

0x13d19809b19ac512da...	0x2e85befe	17634663	212 days 12 hrs ago	Fotress Protocol Exploiter	OUT	0xcd337b920678cf3514...	0 BNB	0.059532
● 0x50526e57fbc6a0ce7d...	0x2e85befe	17634618	212 days 12 hrs ago	Fotress Protocol Exploiter	OUT	0xcd337b920678cf3514...	0 BNB	0.001844
0x8a04f47f839d6db81ba...	Transfer	17634604	212 days 12 hrs ago	Fotress Protocol Exploiter	OUT	MahaDAO: MAHA Token	0 BNB	0.000207

Transfer 100 FTS and 3.02 MAHA to the attack contract

0xd127c438bdac59e448...	Transfer	17634599	212 days 12 hrs ago	Fotress Protocol Exploiter	OUT	Fortress Protocol: FTS T...	0 BNB	0.000452

Create the attack contract

0x4800928c95db2fc877f...	0x61032060	17634590	212 days 12 hrs ago	Fotress Protocol Exploiter	OUT	Contract Creation	0 BNB	0.024164

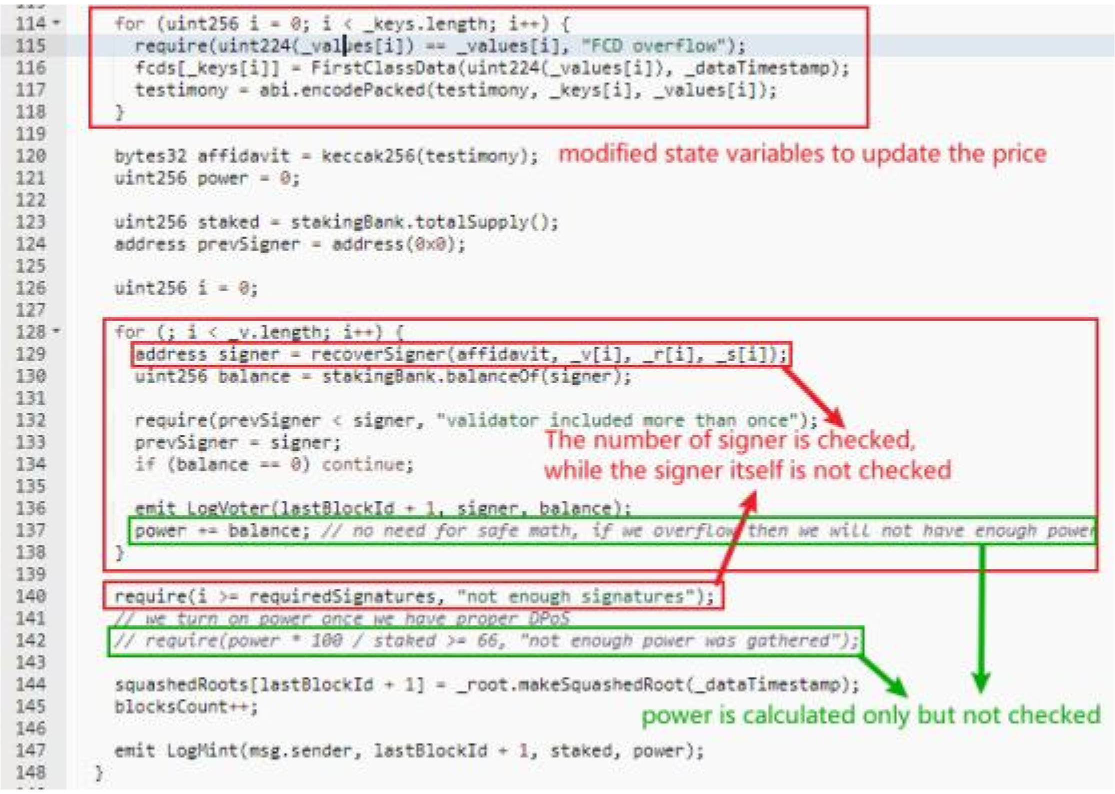
The key attack process is as follows:

txHash: 0x13d19809b19ac512da6d110764caee75e2157ea62cb70937c8d9471afcb061bf

(1) The attacker contract calls the Fortress governance contract to execute the proposal with Id=11. The content of the proposal with Id=11 is to set the mortgage factor of fToken to 700000000000000000.

(2) After modifying the mortgage factor, the attack contract calls the submit function of the Chain contract, modifying the state variables in it further affects the price calculation of the price oracle.

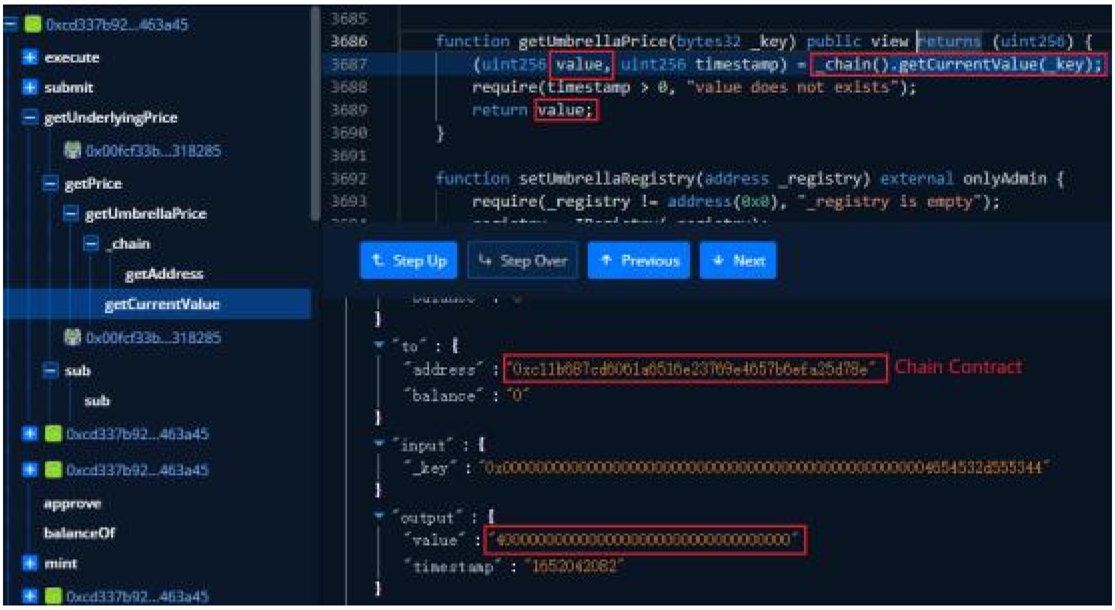
The submit function is as follows:



```
    for (uint256 i = 0; i < _keys.length; i++) {
      require(uint224(_values[i]) == _values[i], "FCD overflow");
      fcds[_keys[i]] = FirstClassData(uint224(_values[i]), _dataTimestamp);
      testimony = abi.encodePacked(testimony, _keys[i], _values[i]);
    }

    bytes32 affidavit = keccak256(testimony);
    uint256 power = 0;

    uint256 staked = stakingBank.totalSupply();
    address prevSigner = address(0x0);

    uint256 i = 0;

    for (; i < _v.length; i++) {
      address signer = recoverSigner(affidavit, _v[i], _r[i], _s[i]);
      uint256 balance = stakingBank.balanceOf(signer);

      require(prevSigner < signer, "validator included more than once");
      prevSigner = signer;
      if (balance == 0) continue;

      emit LogVoter(lastBlockId + 1, signer, balance);
      power += balance; // no need for safe math, if we overflow then we will not have enough powe
    }

    require(i >= requiredSignatures, "not enough signatures");
    // we turn on power once we have proper DPoS
    // require(power * 100 / staked >= 66, "not enough power was gathered");

    squashedRoots[lastBlockId + 1] = _root.makeSquashedRoot(_dataTimestamp);
    blocksCount++;

    emit LogMint(msg.sender, lastBlockId + 1, staked, power);
  }
```

modified state variables to update the price
The number of signer is checked,
while the signer itself is not checked
power is calculated only but not checked

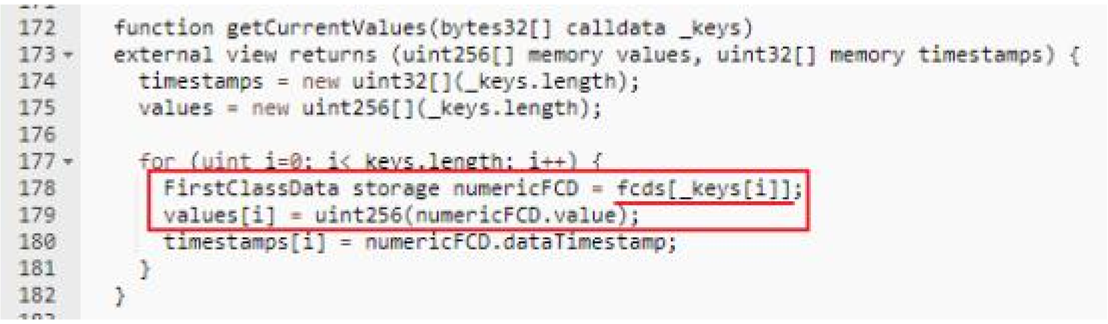
The reason why the state variable fcds can be successfully modified here is that the verification of the signer itself and the verification of the power are missing in the submit function. The function to read the price is as follows:



```
    function getUmbrellaPrice(bytes32 _key) public view returns (uint256) {
        (uint256 value, uint256 timestamp) = _chain().getCurrentValue(_key);
        require(timestamp > 0, "value does not exists");
        return value;
    }

    function setUmbrellaRegistry(address _registry) external onlyAdmin {
        require(_registry != address(0x0), "_registry is empty");
```

```
}
"to" : {
  "address" : "0xc11b087cd6061a6516e23709e4657b6efa25d78e"
  "balance" : "0"
}
"input" : {
  "_key" : "0x000000000000000000000000000000000000000000000000004554482d555344"
}
"output" : {
  "value" : "4000000000000000000000000000000000"
  "timestamp" : "1652042082"
}
```

Chain Contract



```
function getCurrentValues(bytes32[] calldata _keys)
external view returns (uint256[] memory values, uint32[] memory timestamps) {
  timestamps = new uint32[](_keys.length);
  values = new uint256[](_keys.length);

  for (uint i=0; i< keys.length; i++) {
    FirstClassData storage numericFCD = fcds[_keys[i]];
    values[i] = uint256(numericFCD.value);
    timestamps[i] = numericFCD.dataTimestamp;
  }
}
```

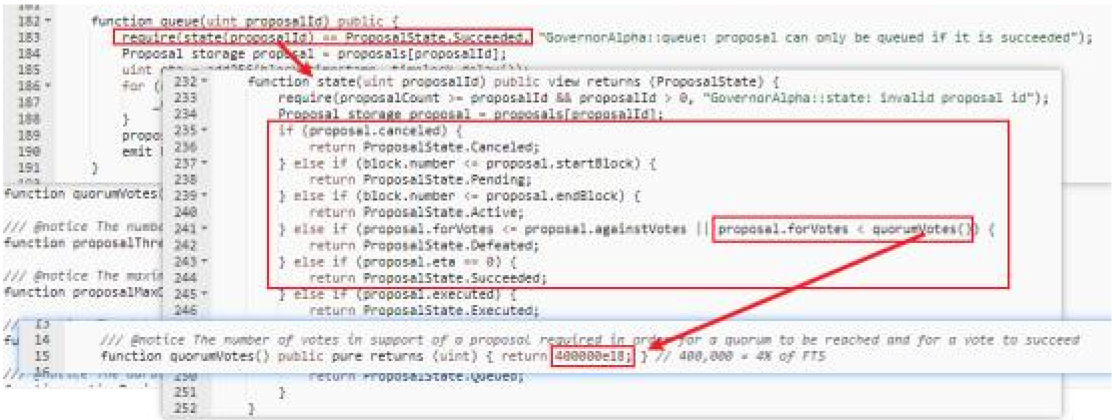
Because the state variable fcds is modified by calling the submit function, the price in the price oracle is finally modified.

(3) After completing the above modifications, the attacker borrowed a large number of other Tokens from the lending contract, and then converted them all into USDT.

The creation, voting and execution process of the proposal with Id=11 in the attack is as follows:

<a> May 3rd, create a proposal;

<b> On May 6, after the proposal passed 2 votes, the queue function was called to add it to the execution queue.



```
function queue(uint proposalId) public {
    require(state(proposalId) == ProposalState.Succeeded, "GovernorAlpha::queue: proposal can only be queued if it is succeeded");
    Proposal storage proposal = proposals[proposalId];
```

```
function state(uint proposalId) public view returns (ProposalState) {
    require(proposalCount >= proposalId && proposalId > 0, "GovernorAlpha::state: invalid proposal id");
    Proposal storage proposal = proposals[proposalId];
    if (proposal.canceled) {
        return ProposalState.Canceled;
    } else if (block.number <= proposal.startBlock) {
        return ProposalState.Pending;
    } else if (block.number <= proposal.endBlock) {
        return ProposalState.Active;
    } else if (proposal.forVotes <= proposal.againstVotes || proposal.forVotes < quorumVotes()) {
        return ProposalState.Defeated;
    } else if (proposal.eta == 0) {
        return ProposalState.Succeeded;
    } else if (proposal.executed) {
        return ProposalState.Executed;
        return ProposalState.Queued;
    }
}
```

```
/// @notice The number of votes in support of a proposal required in order for a quorum to be reached and for a vote to succeed
function quorumVotes() public pure returns (uint) { return 400000e18; } // 400,000 = 4% of FTS
```

The number of votes supported here only needs to be no less than 400,000 FTS, and the votes can be added to the execution queue for execution. The total number of votes for the two votes is 296,193 + 119,774 = 415,917 FTS > 400,000 FTS, and eta is always 0, so the status of the proposal should be Suceeed and can be added to the execution queue.

In addition, the voting FTS was obtained from the Ethereum account through the cross-chain protocol Celer Network by the attacker's account (on April 19th). Due to the low price of FTS, the attacker actually exchanged more than 400,000 FTS (actually 400,413 FTS) with only 9 ETH, completing the entire attack process.

<c> On May 8, vote to implement the proposal to implement the proposal attack.

In this proposal attack, the price of DAO's governance tokens was extremely low, and the attacker exchanged only 9 ETH for governance tokens exceeding the DAO voting threshold (400,000). This allows the proposal initiated by the attacker to pass and then be executed only by the attacker himself voting.

4. Proposal attack analysis in Move

Proposal attacks occur in DAO, and all projects that apply DAO may have proposal attacks, regardless of the development language. Therefore, in the Move ecosystem, projects using DAO also need to beware of proposal attacks.

Through two events in the Solidity ecosystem, we found that a necessary prerequisite for launching a proposal attack is to obtain a large number of voting rights. Attackers can obtain governance tokens exceeding the voting threshold through loans, flash loans, token exchanges, etc., or obtain votes by bribing other users who hold a large number of governance tokens.

Projects that are vulnerable to proposal attacks are more prone to centralization of their governance tokens:

(1) Obtain governance tokens exceeding the voting threshold through flash loans;

(2) Governance tokens are cheap, and attackers can obtain governance tokens that exceed the voting threshold by paying a small amount of value;

(3) Governance tokens are concentrated in a small number of users. Only a very small number of users (such as 2 users) need to participate to obtain votes exceeding the voting threshold. Attackers can bribe other users to obtain votes exceeding the voting threshold.

Projects that apply DAO should avoid the above situations as much as possible, and ensure that only a majority of participants vote to pass the proposal, so as to avoid proposal attacks.

Token's decentralized governance, that is, DAO is an indispensable part of the blockchain, and it is also the development trend of blockchain projects and token management. For example, Starcoin has a built-in implementation of the DAO module in its standard library, through which various parameters on the chain can be voted and governed. For various other projects, such as decentralized exchanges, etc., if DAO is used to implement token governance, it is necessary to consider how to avoid proposal attacks.

About Us

Our vision is to improve security globally. We believe that by building this security barrier, we can significantly improve lives around the world.SharkTeam composes of members with many years of cyber security experiences and blockchain, team members are based in Suzhou, Beijing, Nanjing and Silicon Valley, proficient in the underlying theories of blockchain and smart contracts, and we provide comprehensive services including threat modeling, smart contract auditing, emergency response, etc. SharkTeam has established strategic and long-term cooperations with key players in many areas of the blockchain ecosystem, such as Huobi Global, OKX, polygon, Polkadot, imToken, ChainIDE, etc

SharkTeam

In Math, We Trust!

https://sharkteam.org

https://t.me/sharkteamorg

https://twitter.com/sharkteamorg