SharkTeam

Upgradable Contract Vulnerability—Analysis on the Hack of Web3 Music Platform Audius

July 26 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.

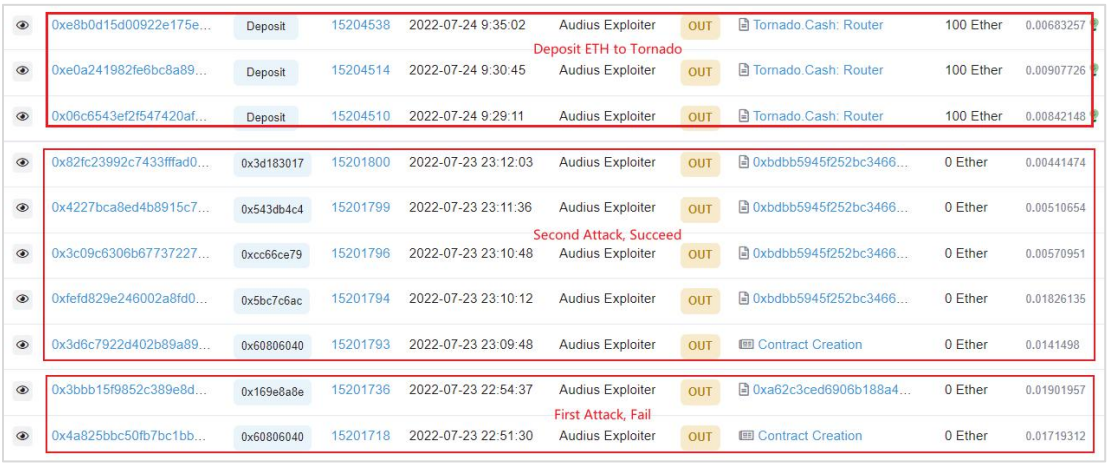
Upgradable Contract Vulnerability—Analysis on the Hack of Web3 Music Platform Audius

It was reported on July 24, 2022 that hackers transferred 18 million AUDIO tokens from music streaming protocol Audius, losing about $6 million.

SharkTeam analyzed the attack technology for the first time, And summarized the security precautions. He hoped that the following blockchain projects would learn from this and build a security defense line for the blockchain industry.

1. Incident analysis

The attacker launches an attack transaction as follows:



The attacker launched two attacks, the first one failed and the second one succeeded.

Attacker address: 0xa0c7bd318d69424603cbf91e9969870f21b8ab4c

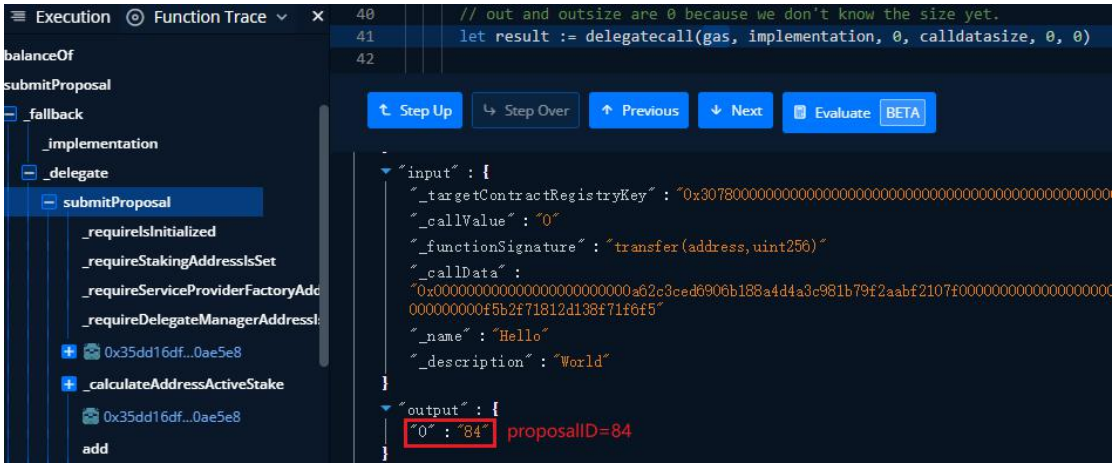
The first attack is as follows:

Attack contract: 0xa62c3ced6906b188a4d4a3c981b79f2aabf2107f

Attack transaction:

0x3bbb15f9852c389e8d77399fe88b49b042d0f22aad4a33c979fbabc60a34b24f

Proposition 84 submitted in transaction

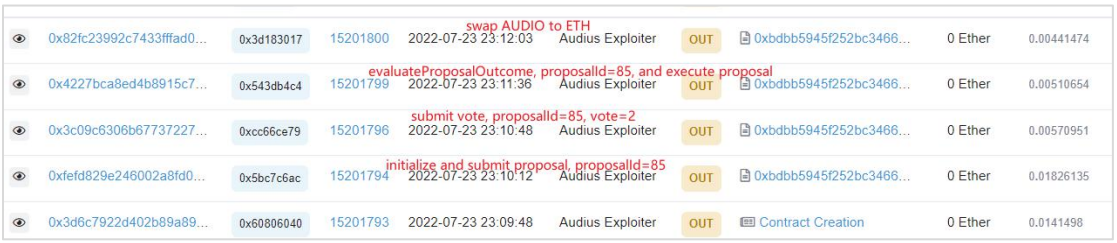


There was no vote on the proposal, so it wasn't executed, but the transaction modified its state by calling the Governance contract's initialization function and evaluating proposals 82 and 83:

Next, we illustrate the entire attack process with the transaction initiated by the second attack.

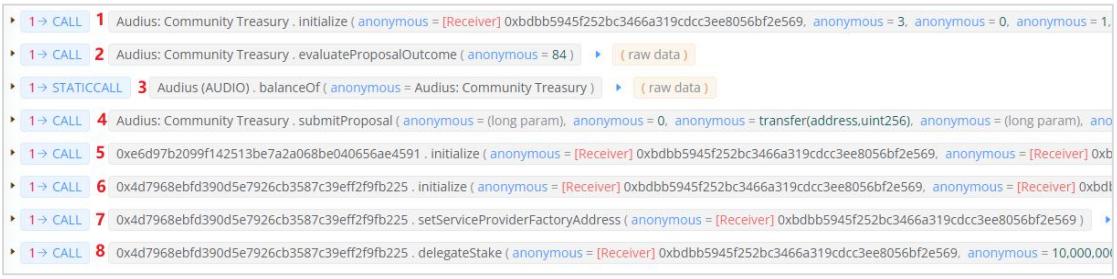
Attack contract address: 0xbdbb5945f252bc3466a319cdcc3ee8056bf2e569, abbreviated as 0xbdbb

After the attacker created the attack contract, he launched an attack through the attack contract, including 4 transactions:



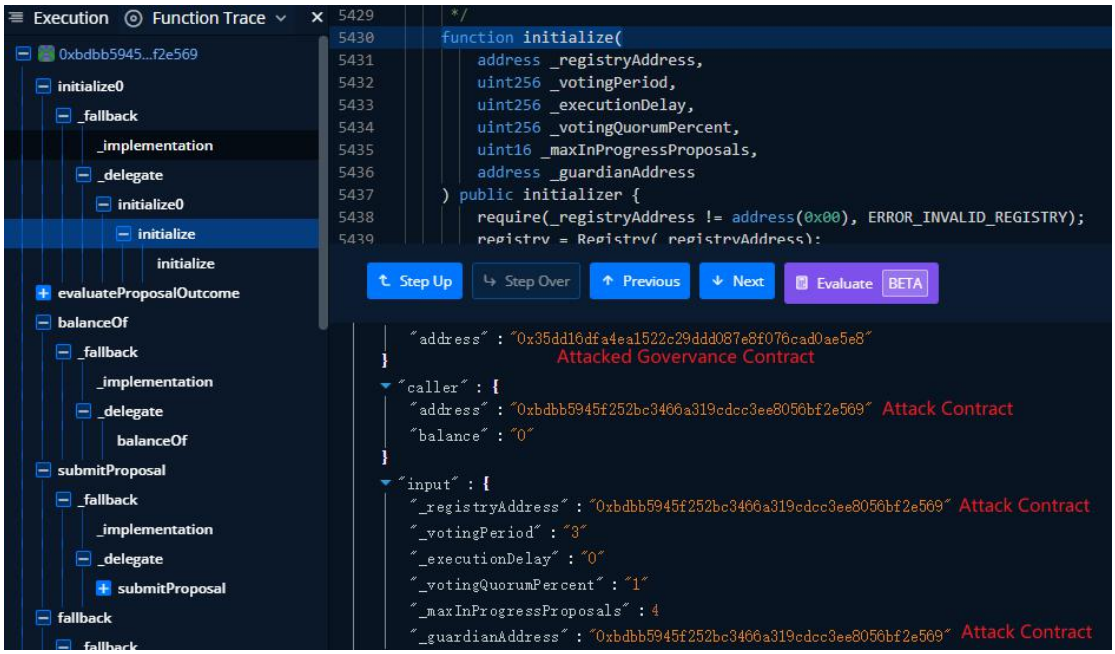
Transaction 1 txHash:

0xfefd829e246002a8fd061eede7501bccb6e244a9aacea0ebceaecef5d877a984



The transaction consists of 7 steps, as follows:

1. Initialize the Governance contract through the Audius proxy contract



The initialization function is as follows:

```
5434    function initialize(
5435        address _registryAddress,
5436        uint256 _votingPeriod,
5437        uint256 _executionDelay,
5438        uint256 _votingQuorumPercent,
5439        uint16 _maxInProgressProposals,
5440        address _guardianAddress
5441    ) public initializer {
5442        require(_registryAddress != address(0x00), ERROR_INVALID_REGISTRY);
5443        registry = Registry(_registryAddress);
5444
5445        require(_votingPeriod > 0, ERROR_INVALID_VOTING_PERIOD);
5446        votingPeriod = _votingPeriod;
5447
5448        // executionDelay does not have to be non-zero
5449        executionDelay = _executionDelay;
5450
5451        require(
5452            _maxInProgressProposals > 0,
5453            "Governance: Requires non-zero _maxInProgressProposals"
5454        );
5455        maxInProgressProposals = _maxInProgressProposals;
5456
5457        require(
5458            _votingQuorumPercent > 0 && _votingQuorumPercent <= 100,
5459            ERROR_INVALID_VOTING_QUORUM
5460        );
5461        votingQuorumPercent = _votingQuorumPercent;
5462
5463        require(
5464            _guardianAddress != address(0x00),
5465            "Governance: Requires non-zero _guardianAddress"
5466        );
5467        guardianAddress = _guardianAddress;  //Guardian address becomes the only party
5468
5469        InitializableV2.initialize();
5470    }
```

This initialization function initializes the voting system. The variable description and parameter settings are as follows:

(1) registryAddress: The registration agent contract address, set as the attack

contract address;

(2) votingPeriod: The block period for the open voting of governance proposals is set to 3, that is, voting is possible within 3 blocks, and the fourth and subsequent blocks cannot be voted anymore;

(3) executionDelay: The number of blocks that must be passed after the voting period is over before the proposal can be evaluated/executed. Set to 1, that is, 1 block can be executed after the voting period ends, or 1 block after the voting period ends. to evaluate/execute the proposal;

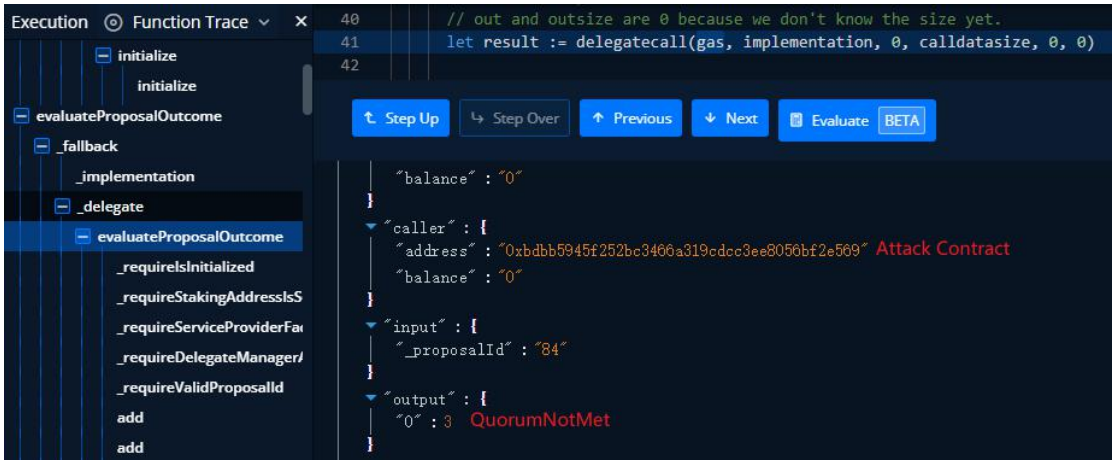
(4) votingQuorumPercent: The minimum percentage of total shares required to vote for the proposal to be valid, set to 1;

(5) maxInProgressProposals: the maximum number of proposals that may be in the InProgress state at one time, set to 4;

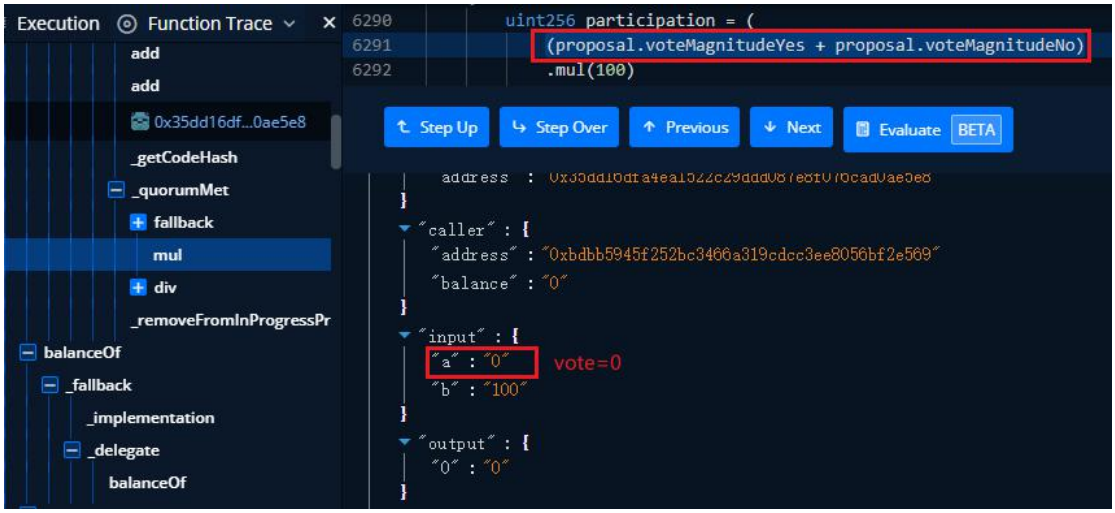
(6) guardianAddress: The account address with special governance authority, which is set as the attack contract.

The parameters are set this way so that proposals can be executed without complex multi-party voting.

2. Evaluate/enforce Proposition 84, the proposal submitted by the first attack, evaluated as QuorumNotMet

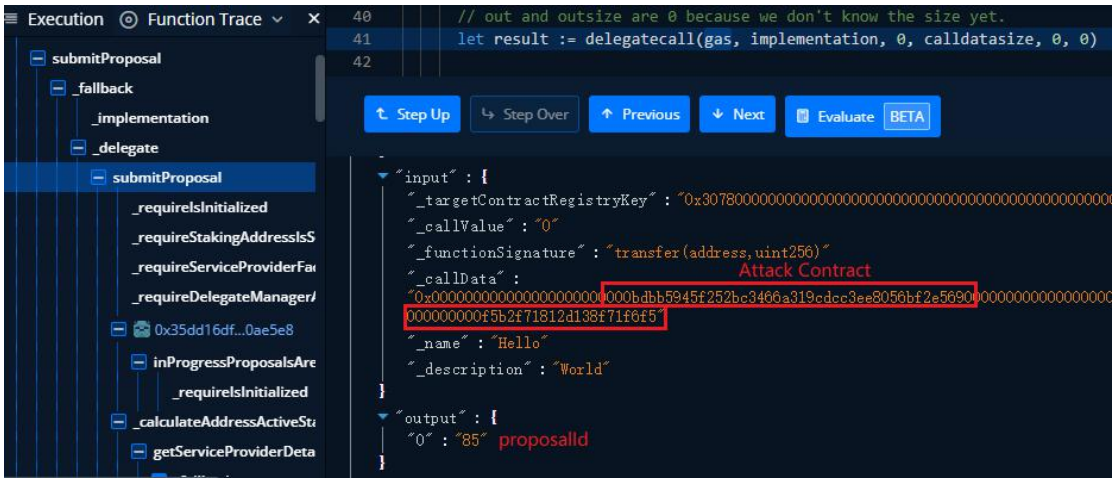


The reason is that there are no votes.



3. Query the AUDIO token balance of the Governance proxy contract

4. Submit Proposition 85, the same as Proposition 84. The function of this proposal is to transfer the AUDIO in the Governance proxy contract to the attack contract, and the amount cannot exceed the AUDIO balance of the Governance proxy contract.

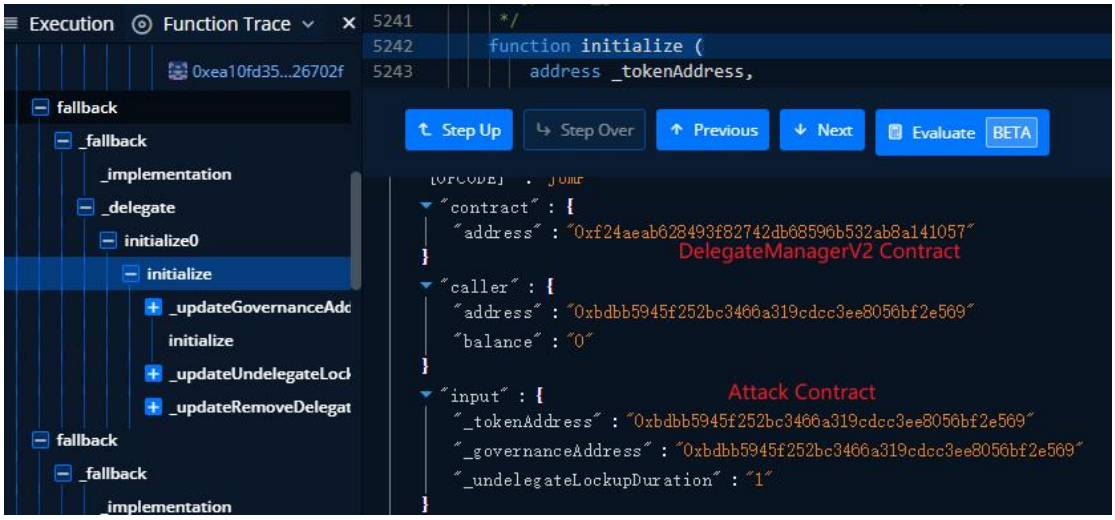


5. Initialize the Staking contract through the proxy contract


```
1480    function initialize(
1481        address _tokenAddress,
1482        address _governanceAddress
1483    ) public initializer
1484    {
1485        _updateGovernanceAddress(_governanceAddress);
1486
1487        audiusToken = ERC20Mintable(_tokenAddress);
1488
1489        fundingRoundBlockDiff = 46523;
1490        fundingAmount = 1342465753420000000000000; // 1342465.75342 AUDS
1491        roundNumber = 0;
1492
1493        currentRound = Round({
1494            fundedBlock: 0,
1495            fundedAmount: 0,
1496            totalClaimedInRound: 0
1497        });
1498
1499        // Community pool funding amount and address initialized to zero
1500        recurringCommunityFundingAmount = 0;
1501        communityPoolAddress = address(0x0);
1502
1503        InitializableV2.initialize();
1504    }
```

Set both the governance token address and the proxy contract address as the attack contract.

6. Initialize the DelegateManagerV2 contract through the proxy contract



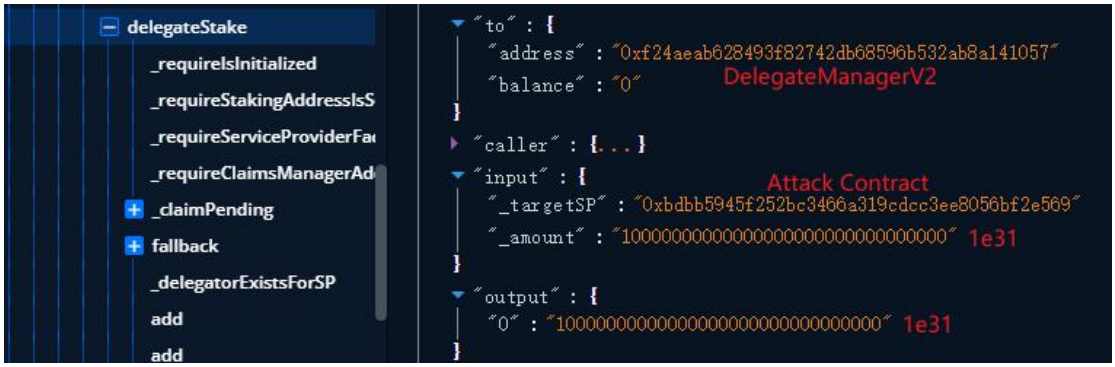
```
5246    function initialize (
5247        address _tokenAddress,
5248        address _governanceAddress,
5249        uint256 _undelegateLockupDuration
5250    ) public initializer
5251    {
5252        _updateGovernanceAddress(_governanceAddress);
5253        audiusToken = ERC20Mintable(_tokenAddress);
5254        maxDelegators = 175;
5255        // Default minimum delegation amount set to 100AUD
5256        minDelegationAmount = 100 * 10**uint256(18);
5257        InitializableV2.initialize();
5258
5259        _updateUndelegateLockupDuration(_undelegateLockupDuration);
5260
5261        // 1 week = 168hrs * 60 min/hr * 60 sec/min / ~13 sec/block = 46523 blocks
5262        _updateRemoveDelegatorLockupDuration(46523);
5263
5264        // 24hr * 60min/hr * 60sec/min / ~13 sec/block = 6646 blocks
5265        removeDelegatorEvalDuration = 6646;
5266    }
```

Set both the governance token address and the proxy governance address as the attack contract.

7. Use the proxy contract to make the service provider factory contract in the DelegateManagerV2 contract an attack contract


```
5931    /**
5932     * @notice Set the ServiceProviderFactory address
5933     * @dev Only callable by Governance address
5934     * @param _spFactory - address for new ServiceProviderFactory contract
5935     */
5936    function setServiceProviderFactoryAddress(address _spFactory) external {
5937        _requireIsInitialized();
5938
5939        require(msg.sender == governanceAddress, ERROR_ONLY_GOVERNANCE);
5940        serviceProviderFactoryAddress = _spFactory;
5941        emit ServiceProviderFactoryAddressUpdated(_spFactory);
5942    }
```

8. Authorize the pledged proxy rights to the attack contract through the proxy contract, and the number of authorized tokens is 1e31



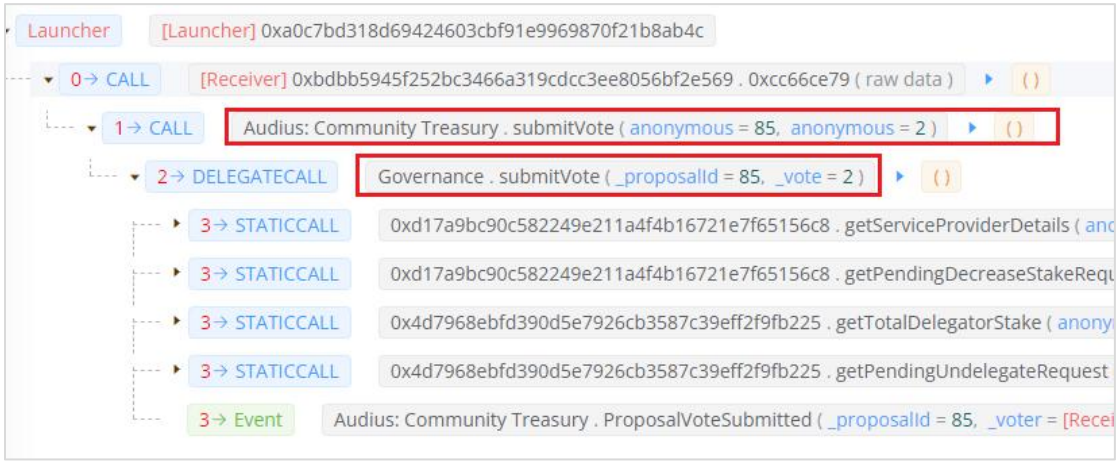

```
5268    /**
5269     * @notice Allow a delegator to delegate stake to a service provider
5270     * @param _targetSP - address of service provider to delegate to
5271     * @param _amount - amount in wei to delegate
5272     * @return Updated total amount delegated to the service provider by delegator
5273     */
5274    function delegateStake(
5275        address _targetSP,
5276        uint256 _amount
5277    ) external returns (uint256)
5278    {
5279        _requireIsInitialized();
5280        _requireStakingAddressIsSet();
5281        _requireServiceProviderFactoryAddressIsSet();
5282        _requireClaimsManagerAddressIsSet();
```

Through the above steps, the attacker tampered with and obtained the highest authority of the governance system.

The attacker then voted within 3 blocks after submitting proposal 85, the second transaction.

Transaction 2 txHash:

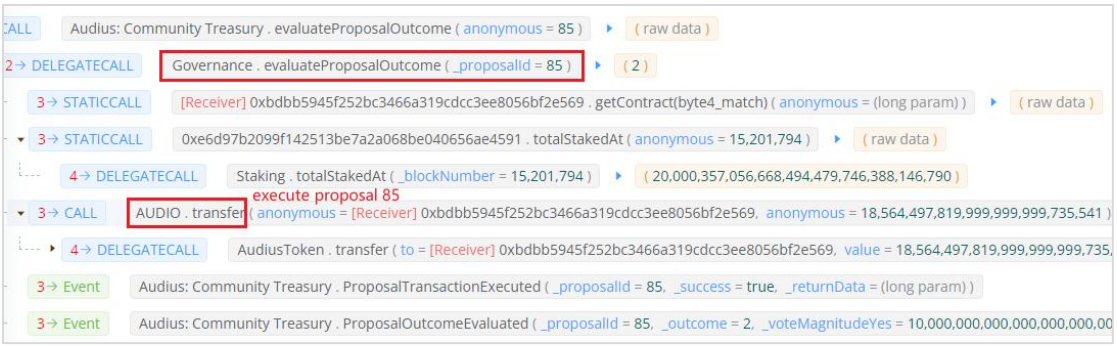
0x3c09c6306b67737227edc24c663462d870e7c2bf39e9ab66877a980c900dd5d5



After voting is complete, the attacker evaluates/executes proposal 85 after a voting period of 3 blocks and a waiting period of 1 block, the 3rd transaction:

Transaction 3 txHash:

0x4227bca8ed4b8915c7eec0e14ad3748a88c4371d4176e716e8007249b9980dc9

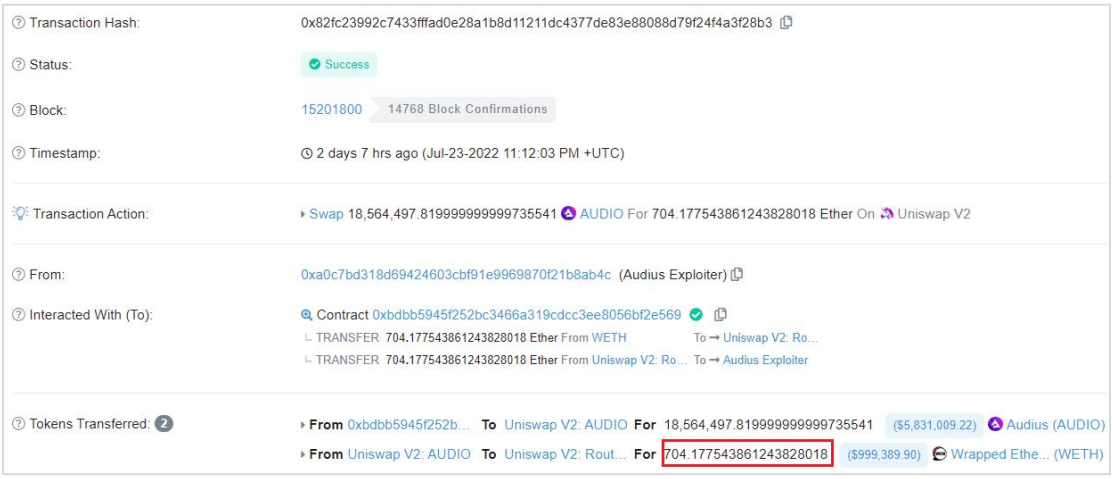


Execute proposal 85, transfer 185.6 million AUDIO to the attack contract.

After the attack contract received 18 million AUDIO, it exchanged it for 704 ETH, which is the fourth transaction:

Transaction 4 txHash:

0x82fc23992c7433fffad0e28a1b8d11211dc4377de83e88088d79f24f4a3f28b3



Finally, the attacker deposited the exchanged ETH into the Tornash platform:

0x85fcad53a171af3edaa...	Deposit	15204590	2022-07-24 9:45:12	Audius Exploiter	OUT	Tornado.Cash: Router	1 Ether	0.00558106
0xe08270c27c76bc3e66...	Deposit	15204590	2022-07-24 9:45:12	Audius Exploiter	OUT	Tornado.Cash: Router	1 Ether	0.00555022
0x6af311f2b190e2818a9...	Deposit	15204579	2022-07-24 9:43:02	Audius Exploiter	OUT	Tornado.Cash: Router	100 Ether	0.00580533
0x46e3b6c0bbd7a14cfb...	Deposit	15204564	2022-07-24 9:41:05	Audius Exploiter	OUT	Tornado.Cash: Router	100 Ether	0.00618974
0xee716d362376a706aa...	Deposit	15204552	2022-07-24 9:39:14	Audius Exploiter	OUT	Tornado.Cash: Router	100 Ether	0.00816117
0xd07f36a1cbbae24e7b...	Deposit	15204543	2022-07-24 9:36:16	Audius Exploiter	OUT	Tornado.Cash: Router	100 Ether	0.00746226
0xe8b0d15d00922e175e...	Deposit	15204538	2022-07-24 9:35:02	Audius Exploiter	OUT	Tornado.Cash: Router	100 Ether	0.00683257
0xe0a241982fe6bc8a89...	Deposit	15204514	2022-07-24 9:30:45	Audius Exploiter	OUT	Tornado.Cash: Router	100 Ether	0.00907726
0x06c6543ef2f547420af...	Deposit	15204510	2022-07-24 9:29:11	Audius Exploiter	OUT	Tornado.Cash: Router	100 Ether	0.00842148

From the above attack process, we found that the fundamental reason why the attacker can attack successfully is that the initialization function is called multiple times through the proxy contract, and the initialization function should only be called once. Take the initialization function in the Governance contract as an example, the code is as follows:

```
5434    function initialize(
5435        address _registryAddress,
5436        uint256 _votingPeriod,
5437        uint256 _executionDelay,
5438        uint256 _votingQuorumPercent,
5439        uint16 _maxInProgressProposals,
5440        address _guardianAddress
5441    ) public initializer {
5442        require(_registryAddress != address(0x00), ERROR_INVALID_REGISTRY);
5443        registry = Registry(_registryAddress);
5444
5445        require(_votingPeriod > 0, ERROR_INVALID_VOTING_PERIOD);
5446        votingPeriod = _votingPeriod;
5447
```

The initializer in Openzeppelin is used here. The code is as follows:

```
412 ▸   /*[...]*/
415     bool private initialized;
416
417 ▸   /*[...]*/
420     bool private initializing;
421
422 ▸   /*[...]*/
425 ▾   modifier initializer() {
426       require(initializing || isConstructor() || !initialized, "Contract
427
428       bool isTopLevelCall = !initializing;
429 ▾     if (isTopLevelCall) {
430         initializing = true;
431         initialized = true;
432       }
433
434       _;
435
436 ▾     if (isTopLevelCall) {
437         initializing = false;
438       }
439     }
```

Obviously, the initializer doesn't do anything because the proxy call. The two bool-type state variables initialized and initializing defined in the implementation contract occupy the first 16 bytes in the storage slot slot0 respectively.

The first 8 bytes are initialized and the second 8 bytes are initializing. Since the proxy contract itself defines a state variable proxyAdmin of address type, its value is 0x80ab62886eacfebca74511823d4699eb88fd097e, which also occupies the storage slot slot0, the first 8 bytes are 0, the second 8 bytes are 0x80ab6288, and the third 8 words Section is 0x6eacfebca7451182 and the 4th 8 bytes is 0x3d4699eb88fd097e.

```
212 ▾ contract AudiusAdminUpgradeabilityProxy is UpgradeabilityProxy {
213       address private proxyAdmin;
214       string private constant ERROR_ONLY_ADMIN = (
215           "AudiusAdminUpgradeabilityProxy: Caller must be current proxy admin"
216       );
```

Therefore, the initialized and initializing in the implementation contract and the proxyAdmin in the proxy contract occupy the storage slot slot0 at the same time, thus causing a storage conflict.

The data distribution in slot 0 is as follows:

When the initialization function is executed to the initializer, initialized is read from the first 8 bytes of the storage slot slot0, and its value is 0, that is, false; initializing is read from the second 8 bytes of the storage slot slot0, and its value is 0x80ab6288 > 0, which is true.

```
412 ▸ /*[...]*/
415   bool private initialized;  false
416
417 ▸ /*[...]*/
420   bool private initializing;  true
421
422 ▸ /*[...]*/
425 ▾ modifier initializer() {                                   true
426     require(initializing || isConstructor() || !initialized, "Contract
427
428     bool isTopLevelCall = !initializing;  false
429 ▾   if (isTopLevelCall) {
430         initializing = true;
431         initialized = true;                skip the code
432     }
433
434     _;
435
436 ▾   if (isTopLevelCall) {
437         initializing = false;              skip the code
438     }
439   }
```

So the initializer does nothing at all.

2. Safety advice

The main reason of this attack is that during the implementation of the upgradeable contract, the storage of the proxy contract and the implementation contract conflicted, causing the initializer to completely lose its function. In response to this vulnerability, we recommend that when using an upgradeable contract, you should first be familiar with the proxy mode, and plan the contract data and logic to avoid problems such as data conflict and loss. In addition, selecting multiple smart contract audit teams to conduct

multiple rounds of audits is also an important guarantee for improving contract security.

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