SharkTeam

A Vulnerability Perspective Analysis of Move Language Security — — Function malicious initialization

Nov 25, 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 series of “Top 10 Smart Contract Security Threats”, SharkTeam summarized and analyzed the top 10 most harmful vulnerabilities in the field of smart contracts based on historical smart contract security incidents.

These vulnerabilities usually appeared in Solidity smart contracts before, so will the same harm exist for the emerging Move smart contracts?

SharkTeam [A Vulnerability Perspective Analysis of Move Language Security] series of courses will discuss and deepen with you. Lesson 3 [Function malicious initialization].

1. Solidity contract initialization

In the Solidity smart contract, there are two ways to realize the initialization of the contract:

(1) Constructor method, that is, use constructor to initialize the contract.

The constructor will be automatically executed after the contract is deployed without additional separate calls, saving a transaction fee.

Constructors are bound to contract deployment, and can only initialize the contract that is bound to be deployed, which has great advantages for the case of a single contract; for joint deployment and initialization of multiple contracts, the method of constructor lacks flexibility, even in Bugs or loopholes can occur in some special scenarios, such as proxy-based upgradable contracts. In the proxy-based upgradeable contract, the logic contract uses the constructor to initialize the state variables, and only the state variables in the logic contract can be initialized. It is invalid for the global state variables in the proxy contract, because the state variables accessed through the proxy contract are actually proxy contracts. state variables in .

(2) Customize the initialization function method, that is, customize the initialize function to initialize the contract.

To use the custom initialization function initialize to implement contract initialization, it is necessary to initiate a separate transaction to call the initialize function after the contract deployment is completed. The initialization function is the initial setting of the global state. It must only be called and executed by the specified account, and the initialization is performed once. However, the initialize function is not essentially different from ordinary functions and cannot be executed only once like a constructor, and it is executed when the contract is deployed. Therefore, the initialize function needs some additional restrictions to ensure its safety. If the initialize function can be called multiple times, there may be vulnerabilities and even threaten the security of digital assets, such as the Punk Protocol security incident.

2. Punk Protocol security incident

The decentralized annuity protocol Punk Protocol was hacked on August 10, 2021, causing a loss of $8.9 million, and the team later recovered $4.95 million. The reason for the attack is that there is a serious loophole in the investment strategy, and the CompoundModel code lacks the modifier of the initialization function, which can be initialized repeatedly.

```
23      function initialize(
24          address forge_,
25          address token_,
26          address cToken_,
27          address comp_,
28          address comptroller_,
29          address uRouterV2_ ) public {
30
31              addToken( token_ );
32              setForge( forge_ );
33              _cToken          = cToken_;
34              _comp            = comp_;
35              _comptroller     = comptroller_;
36              _uRouterV2       = uRouterV2_;
37
38      }
```

3.Move contract initialization

The projects in the Move ecosystem are still those types in the Solidity ecosystem, including tokens, NFT, DEX, etc. The economic model is the same, but the implementation language is Move. Therefore, the Move contract also needs an initialization function to initialize the global variables of the contract, especially resources. The Move contract does not have a constructor but requires a custom initialization function, similar to the custom initialization function in Solidity.

From a security point of view, the initialization function of the Solidity contract needs to have a limit on the caller and the number of calls. The initialization of the Move contract also requires these restrictions, such as creating a new AToken:

```
module Admin::AToken {
    use StarcoinFramework::Token;
    use StarcoinFramework::Account;

    struct AToken has copy, drop, store { }

    public(script) fun initialize(account: signer) {
        Token::register_token<AToken>(&account, 9);
        Account::do_accept_token<AToken>(&account);
    }

    public(script) fun mint(account: signer, amount: u128) {
       let token = Token::mint<AToken>(&account, amount);
       Account::deposit_to_self<AToken>(&account, token)
    }
    public(script) fun burn(account: signer, amount: u128){
       let burn_tokens = Account::withdraw<AToken>(&account, amount);
       Token::burn<AToken>(&account, burn_tokens);
    }
    public(script) fun transfer(from: signer, to: address, amount: u128){
       let transfer_tokens = Account::withdraw<AToken>(&from, amount);
       Account::deposit<AToken>(to, transfer_tokens);
    }
    //other functions
    //......
}
```

The intialize is an initialization function that needs to be called after the contract is deployed, because the visibility of this function is declared as public (script), so it can be called directly from the command line. In this initialization function, the register_token function in Token and the deposit_to_self function in Account are called successively.

(1) Token::register_token function

```
/// Register the type `TokenType` as a Token and got MintCapability and BurnCapability.
public fun register_token<TokenType: store>(
    account: &signer,
    precision: u8,
) {
    assert!(precision <= MAX_PRECISION, Errors::invalid_argument(EPRECISION_TOO_LARGE));
    let scaling_factor = Math::pow(10, (precision as u64));
    let token_address = token_address<TokenType>();
    assert!(Signer::address_of(account) == token_address, Errors::requires_address(ETOKEN_REGISTER));
    move_to(account, MintCapability<TokenType> {});
    move_to(account, BurnCapability<TokenType> {});
    move_to(
        account,
        TokenInfo<TokenType> {
            total_value: 0,
            scaling_factor,
            mint_events: Event::new_event_handle<MintEvent>(account),
            burn_events: Event::new_event_handle<BurnEvent>(account),
        },
    );
}
```

This function realizes the registration of AToken with a precision of 9, and grants the casting and destroying authority of AToken to account.

Note the assert statement marked in the code. The assertion is a verification of the account that initiated the call. It must be the address of the AToken, that is, the address of the contract deployer.

This is similar to the caller permission check (such as onlyOwner check) of the initialization function in Solidity. Without this statement, any account can initialize AToken to obtain minting and destroying permissions, which will cause the highest authority of AToken to leak, and the token will become worthless.

On the other hand, it is guaranteed that the initialization function can only be called once, and the uniqueness of resources in the Move language can guarantee this.

```
spec register_token {
    include RegisterTokenAbortsIf<TokenType>;
    include RegisterTokenEnsures<TokenType>;
}
spec schema RegisterTokenAbortsIf<TokenType> {
    precision: u8;
    account: signer;
    aborts_if precision > MAX_PRECISION;
    aborts_if Signer::address_of(account) != SPEC_TOKEN_TEST_ADDRESS();
    aborts_if exists<MintCapability<TokenType>>(Signer::address_of(account));
    aborts_if exists<BurnCapability<TokenType>>(Signer::address_of(account));
    aborts_if exists<TokenInfo<TokenType>>(Signer::address_of(account));
}
spec schema RegisterTokenEnsures<TokenType> {
    account: signer;
    ensures exists<MintCapability<TokenType>>(Signer::address_of(account));
    ensures exists<BurnCapability<TokenType>>(Signer::address_of(account));
    ensures exists<TokenInfo<TokenType>>(Signer::address_of(account));
}
```

In the specification of the register_token function, there are checks on the minting ability, destroying ability, and token information 3 resources, that is, to ensure that these three resources do not exist before the function is executed, and these three resources will exist after the function is executed. The uniqueness of the resource ensures that the function can only be executed once. This is safer than Solidity. After all, Solidity contracts require additional restrictions, such as using the Initializer in Openzeppelin.

(2) Account::deposit_to_self function

```
/// Add a balance of `Token` type to the sending account.
public fun do_accept_token<TokenType: store>(account: &signer) acquires Account {
    move_to(account, Balance<TokenType>{ token: Token::zero<TokenType>() });
    let token_code = Token::token_code<TokenType>();
    // Load the sender's account
    let sender_account_ref = borrow_global_mut<Account>(Signer::address_of(account));
    // Log a sent event
    Event::emit_event<AcceptTokenEvent>(
        &mut sender_account_ref.accept_token_events,
        AcceptTokenEvent {
            token_code:  token_code,
        },
    );
}
spec do_accept_token {
    aborts_if exists<Balance<TokenType>>(Signer::address_of(account));
    aborts_if !exists<Account>(Signer::address_of(account));
}
```

This function adds AToken to the Balance of the account account, and its amount is 0. The necessary conditions are defined in the specification, including that there is no AToken in the balance of the account account. This ensures that the function can only be executed once.

There is no verification of account permissions here, because any account should be allowed to add AToken to its Balance.

4. Summary

Calling the above two functions completes the initialization of AToken. From the entire initialization process, we found that the initialization function of Move also requires:

(1) It can only be called by the designated authorized account. For example, the token address here is also the deployment address of the token contract

(2) The initialization function can only be called once. The initialization function in Move is generally an initialization structure for resources. The uniqueness of resources and the use of Move specifications can ensure this more conveniently and safely than Solidity. This further highlights the security features of the Move language.

5. supplement

The contract deployment and initialization process is as follows:

(1) Use the mpm release command to compile the module

```
:~/codes/                                    /a-token$ mpm release
Packaging Modules:
        0xd7d7                                c7643::AToken
Release done: release/atoken.v0.0.1.blob, package hash: 0x5807b1682c2b1079d4113afac1fc411f1cc53fc001873a44a5972ddb7afebac5
```

(2) Unlock the Admin account and use the dev deploy command to deploy the contract

```
starcoin% account unlock
{
  "ok": {
    "address": "0xd7d7                          7643",
    "is_default": true,
    "is_locked": false,
    "is_readonly": false,
    "public_key": "0x2f56                                                                    0ae1",
    "receipt_identifier": "stc1p6lt6v5ssuvce6wh9enkk48rkgv9ew5xt"
  }
}
starcoin% dev deploy codes/                                  /a-token/release/atoken.v0.0.1.blob -b
Use package address (0xd7d7                          7643) as transaction sender
txn 0xf73968b8c8e9f9e6f0feef8bd84b853bb70ace01fd2d68321e7cb4908df46d64 submitted.
{
  "ok": {
    "dry_run_output": {
      "events": [],
      "explained_status": "Executed",
      "gas_used": "7800",
      "status": "Executed",
      "write_set": [
```

(3) Use the account execute-function command to call the initialize function to initialize the contract

```
starcoin% account execute-function --function 0xd7d7                                    7643::AToken::initialize -b
txn 0x56ace3e141cf77f887f7c0982ab18698e4d8b65d5effd90933961049e4108aca submitted.
{
  "ok": {
    "dry_run_output": {
      "events": [
        {
          "data": "0xd7d7                          :76430641546f6b656e0641546f6b656e",
          "event_key": "0x0200000000000000d7d7                          7643",
          "event_seq_number": "2",
          "type_tag": "0x00000000000000000000000000000001::Account::AcceptTokenEvent"
        }
      ],
      "explained_status": "Executed",
      "gas_used": "111385",
      "status": "Executed",
      "write_set": [
```

(4) Admin account details after initialization

Added AToken to balances and its balance is 0.

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