Smart Contract Security Audit

1. Executive Summary

On July 13, 2020, the SlowMist security team received the Anyswap team's security audit application for Anyswap system, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

SlowMist Smart Contract DApp project test method:

Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code module through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect wether there are vulnerabilities in programs suck as nodes, SDK, etc.

SlowMist Smart Contract DApp project risk level:

Critical vulnerabilities	Critical vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High-risk vulnerabilities	High-risk vulnerabilities will affect the normal operation of DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium-risk	Medium vulnerability will affect the operation of DeFi project. It is recommended

vulnerablities	to fix medium-risk vulnerabilities.
Low-risk vulnerabilities	Low-risk vulnerabilities may affect the operation of DeFi project in certain scenarios. It is suggested that the project party should evaluate and consider whether these vulnerabilities need to be fixed.
Weaknesses	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Enhancement Suggestions	There are better practices for coding or architecture.

2. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and in-house automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

- Reentrancy attack and other Race Conditions
- Replay attack
- Reordering attack
- Short address attack
- Denial of service attack
- Transaction Ordering Dependence attack

- Conditional Completion attack
- Authority Control attack
- Integer Overflow and Underflow attack
- TimeStamp Dependence attack
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Explicit visibility of functions state variables
- Logic Flaws
- Uninitialized Storage Pointers
- Floating Points and Numerical Precision
- tx.origin Authentication
- "False top-up" Vulnerability
- Scoping and Declarations

3. Project Background (Context)

3.1 Project Introduction

Anyswap is a decentralized, cross-chain trading protocol that supports real-time trading of mainstream cryptocurrencies such as BTC, ETH, USDT, FSN, automatic liquidity and token incentives

Project website: https://anyswap.exchange/

Project testnet: FUSION testnet

Audit code file:

ANY token contract: https://github.com/anyswap/anyswap-token

commit: 937c687c78b80d4d554877b7254ac6a2166fc3ae

ANY tokenvault contract: https://github.com/anyswap/ANYToken-locked

commit: 6e61beea7f95c25465291d7f79c0ce5e537f6dc5

Anyswap exchange contract: https://github.com/anyswap/anyswap-exchange

commit: 1c9cc0053b315535fc1c7f6cd6513888dd1a204d

Files provided by the project side:

HowItWorks.docx:

MD5: 09d557ae2eb3971e102787830bc08b33

ANY audit requirement description document.docx:

MD5: 0bd0c67d98b503d541e3de0760194104

3.2 Project Structure

anyswap-token:

```
.
├── AnyswapToken.sol
├── LICENSE
├── README.md
├── abi
│   └── AnyswapToken.json
├── bytecode
│   └── AnyswapToken.txt
├── contracts
│   ├── AnyswapToken.sol
│   └── Migrations.sol
├── migrations
```

```
|   ├── 1_initial_migration.js
|   └── 2_deploy_contracts.js
├── package-lock.json
├── package.json
└── truffle-config.js
```

ANYToken-locked:

```
.
├── Distribute.sol
├── README.md
├── abi
|   └── Distribute.json
├── contracts
|   ├── Distribute.sol
|   └── Migrations.sol
├── migrations
|   ├── 1_initial_migration.js
|   └── 2_deploy_contracts.js
├── package-lock.json
├── package.json
└── truffle-config.js
```

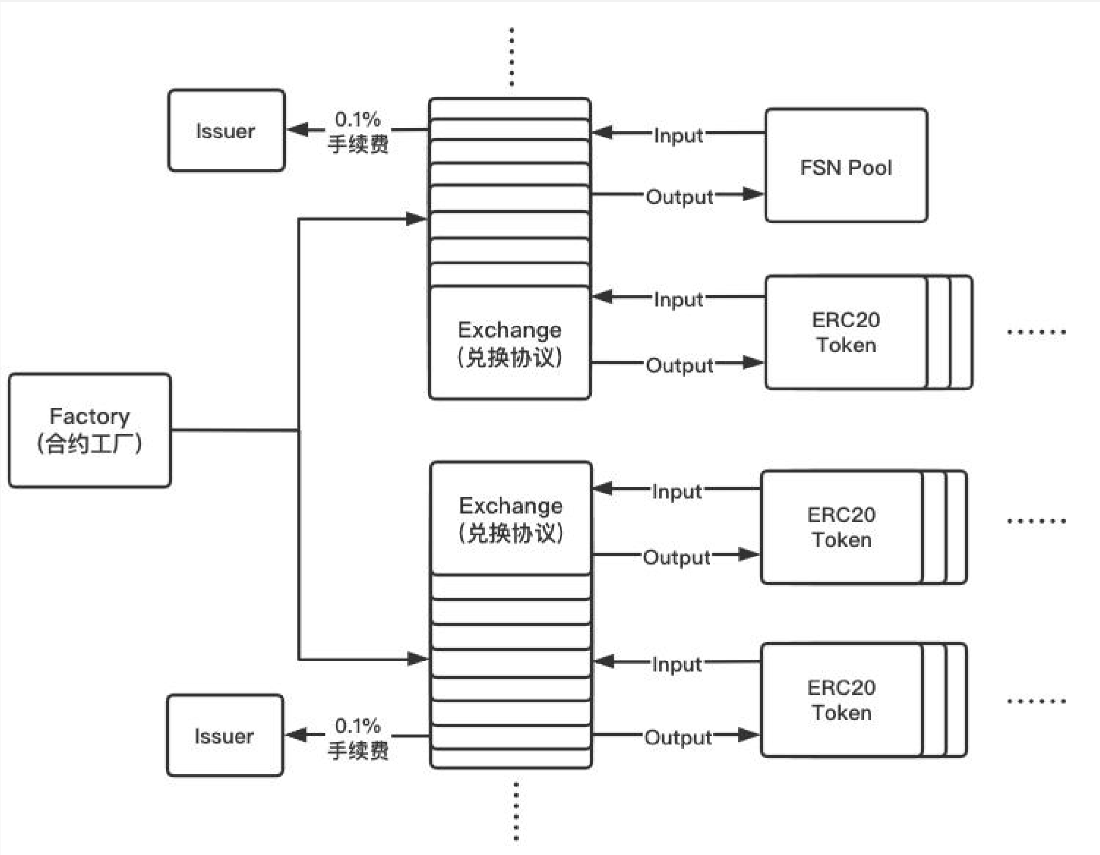
anyswap-exchange:

```
.
├── LICENSE.md
├── README.md
├── abi
|   ├── uniswap_exchange.json
|   └── uniswap_factory.json
├── bytecode
|   ├── exchange.txt
|   └── factory.txt
├── contracts
|   ├── test_contracts
|   |   └── ERC20.vy
|   ├── uniswap_exchange.vy
|   └── uniswap_factory.vy
├── requirements.txt
└── tests
    ├── __init__.py
    ├── conftest.py
    ├── constants.py
```

```
└── exchange
    ├── test_ERC20.py
    ├── test_eth_to_token.py
    ├── test_factory.py
    ├── test_liquidity_pool.py
    ├── test_token_to_eth.py
    ├── test_token_to_exchange.py
    └── test_token_to_token.py
```

3.3 Contract Structure

Anyswap DApp is mainly divided into two parts, namely the contract factory and the token exchange part. The uniswap_factory contract is responsible for creating an independent exchange contract for each ERC20 token. The uniswap_exchange contract is responsible for realizing the functions of providing a liquidity pool for token exchange, handling fee processing and custom capital pools. Each exchange contract is associated with an ERC20 Token and has a liquidity pool of FSN and the ERC20 Token for exchange between the FSN and the Token and between the Tokens. And part of the commission generated during the exchange process is deposited in the liquidity reserve, and the other part is sent to the issuer's wallet The overall structure of the contract is as follows:



4. Code Overview

4.1 Main File Hash

No	File Name	SHA-1 Hash
1	AnyswapToken.sol	5d3e5d61957cde64213aefa96c785eadc6b5d444
2	Distribute.sol	116b7f7b9867bb668c4f3d4aa44a1f22926fc923
3	uniswap_factory.vy	97d49145ec4fc6aa31099cb51c0c2f69b6e487b7
4	uniswap_exchange.vy	b3442cf7794d870511998ca3ba529d9ab42c4305

4.2 Main function visibility analysis

Contract Name	Function Name	Visibility
Distribute	Implementation	——
	setBeneficiary	Public
	setStartBlock	Public
	setStableHeight	Public
	setBlocksPerCycle	Public
	setReleasedPerBlock	Public
	revoke	Public
	release	Public
	releasableAmount	Public
AnyswapToken	Implementation	——
	destroy	Public
	burn	Public
	burnFrom	Public
	totalSupply	Public
	balanceOf	Public
	allowance	Public

	approve	Public
	transfer	Public
	transferFrom	Public
uniswap_factory	Implementation	——
	initializeFactory	Public
	createExchange	Public
	getExchange	Public
	getToken	Public
	getTokenWithId	Public
	Implementation	——
	setup	Public
	addLiquidity	Public
	removeLiquidity	Public
	getInputPrice	Private
	getOutputPrice	Private
	ethToTokenInput	Private
	default	Public
	ethToTokenSwapInput	Public
	ethToTokenTransferInput	Public

uniswap_exchange	ethToTokenOutput	Private
	ethToTokenSwapOutput	Public
	ethToTokenTransferOutput	Public
	tokenToEthInput	Private
	tokenToEthSwapInput	Public
	tokenToEthTransferInput	Public
	tokenToEthOutput	Private
	tokenToEthSwapOutput	Public
	tokenToEthTransferOutput	Public
	tokenToTokenInput	Private
	tokenToTokenSwapInput	Public
	tokenToTokenTransferInput	Public
	tokenToTokenOutput	Private
	tokenToTokenSwapOutput	Public
	tokenToTokenTransferOutout	Public
	tokenToExchangeSwapInput	Public
	tokenToExchangeTransferInput	Public
	tokenToExchangeSwapOutput	Public
	tokenToExchangeTransferOutput	Public

	transfer	Public
	transferFrom	Public
	approve	Public

4.3 Code Audit

4.3.1 The risk of ERC777 contract reentrancy attack

The user's tokens were not deducted before the transfer operation was performed during the token exchange. As the standard defined by ERC777 protocol tries to call the tokensToSend function of the token sender every time a transfer operation occurs, when the exchange contract calls the token contract transferFrom function, the token contract will call the tokensToSend function of the token sender. This function is a function controllable by the token sender. Through this function, the token exchange function of the exchange contract can be called for the second time, causing reentrancy risk.

Code location: File uniswap_exchange.vy line 219, 220, 257, 258, 294, 295, 340, 341

```
@private
def tokenToEthInput(tokens_sold: uint256, min_eth: uint256(wei), deadline: timestamp, buyer: address, recipient: address) ->
uint256(wei):
    assert deadline >= block.timestamp and (tokens_sold > 0 and min_eth > 0)
    tokens_fee: uint256 = (tokens_sold + 999) / 1000
    tokens_sold2: uint256 = tokens_sold - tokens_fee
    token_reserve: uint256 = self.token.balanceOf(self)
    eth_bought: uint256 = self.getInputPrice(tokens_sold2, token_reserve, as_unitless_number(self.balance))
    wei_bought: uint256(wei) = as_wei_value(eth_bought, 'wei')
    assert wei_bought >= min_eth
    send(recipient, wei_bought)
```

```
    assert self.token.transferFrom(buyer, self.issuer, tokens_fee)
    assert self.token.transferFrom(buyer, self, tokens_sold2)
    log.EthPurchase(buyer, tokens_sold, wei_bought)
    return wei_bought
```

```
@private
def tokenToEthOutput(eth_bought: uint256(wei), max_tokens: uint256, deadline: timestamp, buyer: address, recipient: address) -> uint256:
    assert deadline >= block.timestamp and eth_bought > 0
    token_reserve: uint256 = self.token.balanceOf(self)
    tokens_sold: uint256 = self.getOutputPrice(as_unitless_number(eth_bought), token_reserve, as_unitless_number(self.balance))
    tokens_fee: uint256 = (tokens_sold + 999) / 1000
    tokens_sold2: uint256 = tokens_sold + tokens_fee
    # tokens sold is always > 0
    assert max_tokens >= tokens_sold2
    send(recipient, eth_bought)
    assert self.token.transferFrom(buyer, self.issuer, tokens_fee)
    assert self.token.transferFrom(buyer, self, tokens_sold)
    log.EthPurchase(buyer, tokens_sold2, eth_bought)
    return tokens_sold2
```

```
@private
def tokenToTokenInput(tokens_sold: uint256, min_tokens_bought: uint256, min_eth_bought: uint256(wei), deadline: timestamp, buyer: address, recipient: address, exchange_addr: address) -> uint256:
    assert (deadline >= block.timestamp and tokens_sold > 0) and (min_tokens_bought > 0 and min_eth_bought > 0)
    assert exchange_addr != self and exchange_addr != ZERO_ADDRESS
    tokens_fee: uint256 = (tokens_sold + 999) / 1000
    tokens_sold2: uint256 = tokens_sold - tokens_fee
    token_reserve: uint256 = self.token.balanceOf(self)
    eth_bought: uint256 = self.getInputPrice(tokens_sold2, token_reserve, as_unitless_number(self.balance))
    wei_bought: uint256(wei) = as_wei_value(eth_bought, 'wei')
    assert wei_bought >= min_eth_bought
    assert self.token.transferFrom(buyer, self.issuer, tokens_fee)
    assert self.token.transferFrom(buyer, self, tokens_sold2)
    tokens_bought: uint256 = Exchange(exchange_addr).ethToTokenTransferInput(min_tokens_bought, deadline, recipient, value=wei_bought)
    log.EthPurchase(buyer, tokens_sold, wei_bought)
    return tokens_bought
```

```
@private
```

```
def tokenToTokenOutput(tokens_bought: uint256, max_tokens_sold: uint256, max_eth_sold: uint256(wei), deadline: timestamp, buyer: address, recipient: address, exchange_addr: address) -> uint256:
    assert deadline >= block.timestamp and (tokens_bought > 0 and max_eth_sold > 0)
    assert exchange_addr != self and exchange_addr != ZERO_ADDRESS
    eth_bought: uint256(wei) = Exchange(exchange_addr).getEthToTokenOutputPrice(tokens_bought)
    eth_bought2:uint256(wei) = eth_bought * 1000 / 998 + 1
    token_reserve: uint256 = self.token.balanceOf(self)
    tokens_sold: uint256 = self.getOutputPrice(as_unitless_number(eth_bought2), token_reserve, as_unitless_number(self.balance))
    tokens_fee: uint256 = (tokens_sold + 999) / 1000
    tokens_sold2: uint256 = tokens_sold + tokens_fee
    # tokens sold is always > 0
    assert max_tokens_sold >= tokens_sold2 and max_eth_sold >= eth_bought2
    assert self.token.transferFrom(buyer, self.issuer, tokens_fee)
    assert self.token.transferFrom(buyer, self, tokens_sold)
    eth_sold: uint256(wei) = Exchange(exchange_addr).ethToTokenTransferOutput(tokens_bought, deadline, recipient, value=eth_bought2)
    log.EthPurchase(buyer, tokens_sold2, eth_bought2)
    return tokens_sold2
```

Solution: Add an prevent reentrancy mechanism to all token exchange functions, such as OpenZeppelin's ReentrancyGuard.sol, and when performing token exchange, the user's tokens will be deducted before the transfer operation.

Fix status: After confirming with the project party, all connected tokens will be rechecked. After the review is passed, the administrator will update to the front-end. The project party will check whether there is such a risk in the token contract entered during the contract audit.

4.3.2 The risk that the liquidity pool cannot be removed

`assert self.token.transferFrom(msg.sender, self, token_amount)` is used in the addLiquidity function, and `assert self.token.transfer(msg.sender, token_amount)` is used in the removeLiquidity function. The two are inconsistent it may cause the risk that liquidity cannot be removed. For example: The return value of a contract's transferFrom function conforms to the return value specification for ERC20 tokens defined in the EIP 20 standard, but the return value of the transfer function does not conform to the return value specification for ERC20 tokens defined in the EIP20 standard. This may cause the addLiquidity operation to succeed, but the removeLiquidity operation cannot succeed.

Code location: File uniswap_exchange.vy line 62, 96

```
@public
@payable
def addLiquidity(min_liquidity: uint256, max_tokens: uint256, deadline: timestamp) -> uint256:
    assert deadline > block.timestamp and (max_tokens > 0 and msg.value > 0)
    total_liquidity: uint256 = self.totalSupply
    if total_liquidity > 0:
        assert min_liquidity > 0
        eth_reserve: uint256(wei) = self.balance - msg.value
        token_reserve: uint256 = self.token.balanceOf(self)
        token_amount: uint256 = msg.value * token_reserve / eth_reserve + 1
        liquidity_minted: uint256 = msg.value * total_liquidity / eth_reserve
        assert max_tokens >= token_amount and liquidity_minted >= min_liquidity
        self.balances[msg.sender] += liquidity_minted
        self.totalSupply = total_liquidity + liquidity_minted
        assert self.token.transferFrom(msg.sender, self, token_amount)
        log.AddLiquidity(msg.sender, msg.value, token_amount)
        log.Transfer(ZERO_ADDRESS, msg.sender, liquidity_minted)
        return liquidity_minted
    else:
        assert (self.factory != ZERO_ADDRESS and self.token != ZERO_ADDRESS) and msg.value >= 1000000000
        assert self.factory.getExchange(self.token) == self
        token_amount: uint256 = max_tokens
        initial_liquidity: uint256 = as_unitless_number(self.balance)
        self.totalSupply = initial_liquidity
        self.balances[msg.sender] = initial_liquidity
        assert self.token.transferFrom(msg.sender, self, token_amount)
        log.AddLiquidity(msg.sender, msg.value, token_amount)
        log.Transfer(ZERO_ADDRESS, msg.sender, initial_liquidity)
        return initial_liquidity
```

```
@public
```

```
def removeLiquidity(amount: uint256, min_eth: uint256(wei), min_tokens: uint256, deadline: timestamp) -> (uint256(wei), uint256):
    assert (amount > 0 and deadline > block.timestamp) and (min_eth > 0 and min_tokens > 0)
    total_liquidity: uint256 = self.totalSupply
    assert total_liquidity > 0
    token_reserve: uint256 = self.token.balanceOf(self)
    eth_amount: uint256(wei) = amount * self.balance / total_liquidity
    token_amount: uint256 = amount * token_reserve / total_liquidity
    assert eth_amount >= min_eth and token_amount >= min_tokens
    self.balances[msg.sender] -= amount
    self.totalSupply = total_liquidity - amount
    send(msg.sender, eth_amount)
    assert self.token.transfer(msg.sender, token_amount)
    log.RemoveLiquidity(msg.sender, eth_amount, token_amount)
    log.Transfer(msg.sender, ZERO_ADDRESS, amount)
    return eth_amount, token_amount
```

Solution: It is recommended that the addLiquidity function and the removeLiquidity function uniformly use the transferFrom function for transfer operations.

Fix status: After confirming with the project party, all connected tokens will be rechecked. After the review is passed, the administrator will update to the front-end. The project parties will check whether the transferFrom function of the connected token and the return value of the transfer function meet the EIP standard during the contract audit to avoid this risk.

4.3.3 Use block.timestamp to get the current time

Because miners can change the block time, the time obtained by using block.timestamp in the contract may not be accurate.

Code Location: File uniswap_exchange.vy line 51, 86, 130, 173, 211, 249, 286, 330

```
@public
@payable
def addLiquidity(min_liquidity: uint256, max_tokens: uint256, deadline: timestamp) -> uint256:
```

```
    assert deadline > block.timestamp and (max_tokens > 0 and msg.value > 0)
```

```
@public
def removeLiquidity(amount: uint256, min_eth: uint256(wei), min_tokens: uint256, deadline: timestamp) -> (uint256(wei), uint256):
    assert (amount > 0 and deadline > block.timestamp) and (min_eth > 0 and min_tokens > 0)
```

```
@private
def ethToTokenInput(eth_sold: uint256(wei), min_tokens: uint256, deadline: timestamp, buyer: address, recipient: address) -> uint256:
    assert deadline >= block.timestamp and (eth_sold > 0 and min_tokens > 0)
```

```
@private
def ethToTokenOutput(tokens_bought: uint256, max_eth: uint256(wei), deadline: timestamp, buyer: address, recipient: address) -> uint256(wei):
    assert deadline >= block.timestamp and (tokens_bought > 0 and max_eth > 0)
```

```
@private
def tokenToEthInput(tokens_sold: uint256, min_eth: uint256(wei), deadline: timestamp, buyer: address, recipient: address) -> uint256(wei):
    assert deadline >= block.timestamp and (tokens_sold > 0 and min_eth > 0)
```

```
@private
def tokenToEthOutput(eth_bought: uint256(wei), max_tokens: uint256, deadline: timestamp, buyer: address, recipient: address) -> uint256:
    assert deadline >= block.timestamp and eth_bought > 0
```

```
@private
def tokenToTokenInput(tokens_sold: uint256, min_tokens_bought: uint256, min_eth_bought: uint256(wei), deadline: timestamp, buyer: address, recipient: address, exchange_addr: address) -> uint256:
    assert (deadline >= block.timestamp and tokens_sold > 0) and (min_tokens_bought > 0 and min_eth_bought > 0)
```

```
@private
```

```
def tokenToTokenOutput(tokens_bought: uint256, max_tokens_sold: uint256, max_eth_sold: uint256(wei), deadline: timestamp, buyer: address, recipient: address, exchange_addr: address) -> uint256:
    assert deadline >= block.timestamp and (tokens_bought > 0 and max_eth_sold > 0)
```

Solution: The project can use oracles for time acquisition.

Fix status: After confirming with the project party, the block time that miners can modify is limited to a block period of 15 seconds. Exceeding this time is considered evil and the whole network will not accept it. The accuracy of the time limit in the trading contract is minute level, the default is 15 minutes, and the error of 1/60 is acceptable.

4.3.4 Part of the code is redundant

Because `Exchange(exchange).setup(token)` needs to call the setup function of the exchange contract. Therefore, if the `exchangeTemplate` passed in the zero address, the setup function will not be successfully called, so the transaction contract will not be successfully created.

Code location: File uniswap_factory.vy line 15, 21

```
@public
def initializeFactory(template: address):
    assert self.exchangeTemplate == ZERO_ADDRESS
    assert template != ZERO_ADDRESS
    self.exchangeTemplate = template
```

```
@public
def createExchange(token: address) -> address:
    assert token != ZERO_ADDRESS
    assert self.exchangeTemplate != ZERO_ADDRESS
    assert self.token_to_exchange[token] == ZERO_ADDRESS
    exchange: address = create_with_code_of(self.exchangeTemplate)
    Exchange(exchange).setup(token)
    self.token_to_exchange[token] = exchange
    self.exchange_to_token[exchange] = token
    token_id: uint256 = self.tokenCount + 1
```

```
    self.tokenCount = token_id
    self.id_to_token[token_id] = token
    log.NewExchange(token, exchange)
    return exchange
```

Solution: Remove unnecessary code.

Fix status: After confirming with the project party, the zero address check logic has no impact on the business, and the code will not be modified.

5. Audit Result

5.1 Medium-risk Vulnerability

- The risk of reentrancy

5.2 Low-risk Vulnerability

- Remove liquidity pool design defects
- Block time obtain design defects

5.3 Enhancement Suggestions

- Part of the code is redundant

5.4 Conclusion

Audit Result : Passed

Audit Number : 0X002007200001

Audit Date : July 20, 2020

Audit Team : SlowMist Security Team

Summary conclusion: The are 4 security issues found during the audit. After communication and feedback, with the Anyswap team, confirms that the risks found in the audit process are within the tolerable range.

6. Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility base on these.
For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance this report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

慢雾科技
slow mist

Official Website
www.slowmist.com

E-mail
team@slowmist.com

Twitter
@SlowMist_Team

WeChat Official Account