smart state

Web3 security easier than ever

MAGIC SQUARE
SQRLockup.sol
Smart contract audit report

January 28, 2024

Table of contents

Methodology

- Manual code analysis
- Best code practices
- ERC20/BEP20 compliance (if applicable)
- Locked ether
- Pool Asset Security (backdoors in the underlying ERC-20)
- FA2 compliance (if applicable)
- Logical bugs & code logic issues
- Error handling issues
- General Denial Of Service(DOS)
- Cryptographic errors
- Weak PRNG / Random number generators issues
- Protocol and header parsing errors
- Private data leaks
- Using components with known vulnerabilities
- Unchecked call return method
- Code with no effects
- Unused vars
- Use of deprecated functions
- Authorization issues
- Re-entrancy
- Arithmetic Overflows / Underflows
- Hidden Malicious Code
- External Contract Referencing
- Short Address/Parameter Attack
- Race Conditions / Front Running
- Uninitialized Storage Pointers
- Floating Points and Precision
- Signatures Replay

Summary

This audit encompasses the examination of the MAGIC SQUARE SQRLockup.sol smart contract. The purpose of the contract is to deposit/withdraw ERC-20 tokens of the project.

The user sends tokens to the contract via deposit method and those tokens that exceeded the limit are sent to a "cold" wallet in the same transaction.

After some time (arbitrary), the user can withdraw these tokens via withdrawSig if the user has a correct digital signature (which is generated by the off-chain service).

withdraw is a method similar in functionality to withdrawSig, called only by the contract owner.

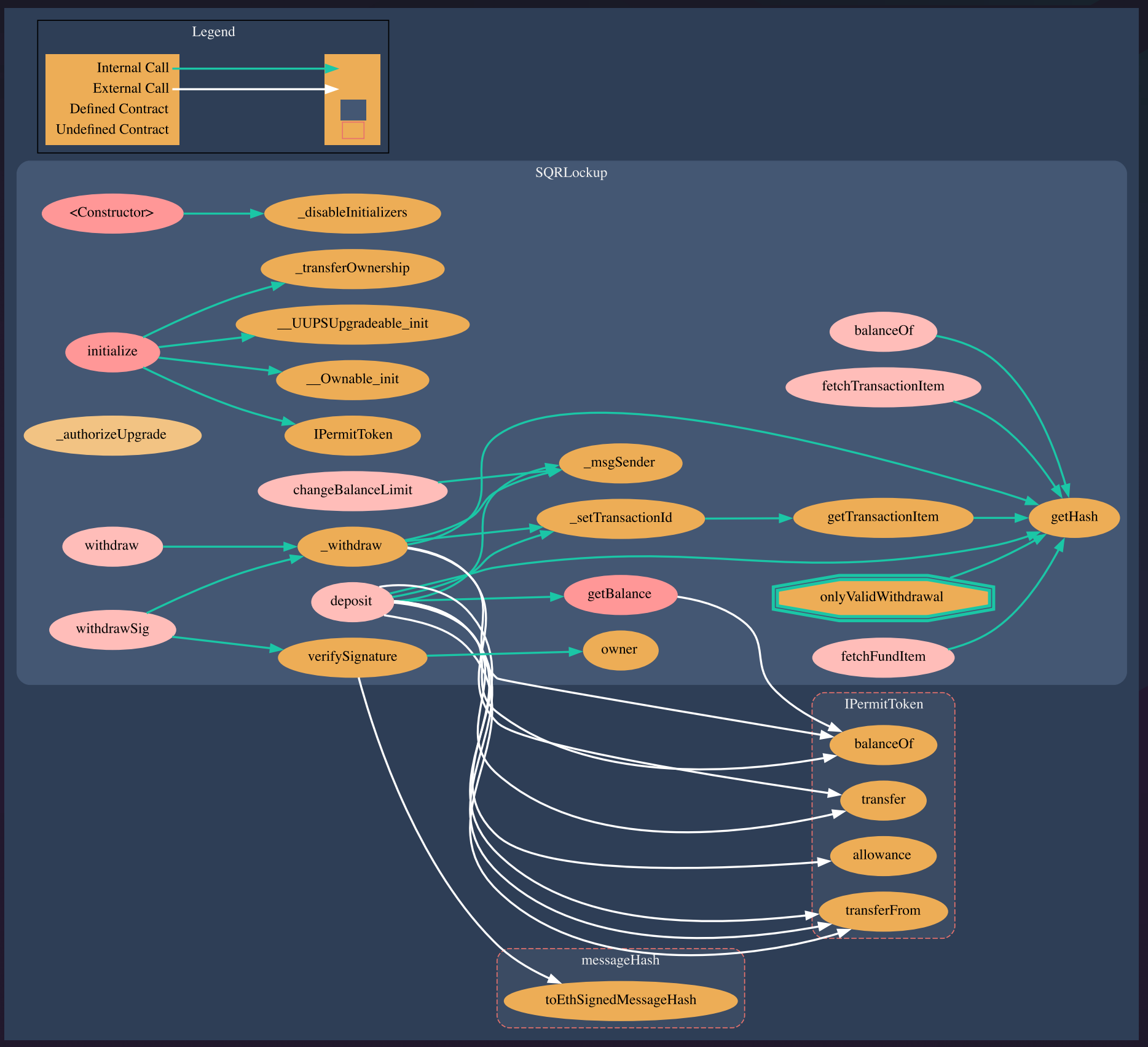
Disclaimer

Vulnerabilities found by type

Info	0
Warning	0
Warning	0
Total	0

1.1 Structure of contract:

SQRLockup.sol



pic.1.1 SQRLockup.sol

1.2 SQRLockup.sol contract methods analysis:

initialize(address _newOwner, address _sqrToken, address _coldWallet, uint256 _balanceLimit)
Vulnerabilities not detected

_authorizeUpgrade(address newImplementation)
Vulnerabilities not detected

changeBalanceLimit(uint256 _balanceLimit)
Vulnerabilities not detected

fetchFundItem(string memory userId)
Vulnerabilities not detected

getBalance()
Vulnerabilities not detected

balanceOf(string memory userId)
Vulnerabilities not detected

getHash(string memory value)
Vulnerabilities not detected

1.2 SQRLockup.sol contract methods analysis:

`fetchTransactionItem(`
`  string memory transactionId`
`  )`

Vulnerabilities not detected

`getTransactionItem(`
`  string memory transactionId`
`  )`

Vulnerabilities not detected

`_setTransactionId(uint256 amount,` `string memory transactionId)`
Vulnerabilities not detected

`deposit(`
`  string memory userId,`
`  string memory transactionId,`
`  uint256 amount`
`  )`

Vulnerabilities not detected
TOKEN FLOW: Tokens in, tokens out, public

`_withdraw(`
`  string memory userId,`
`  string memory transactionId,`
`  address to,`
`  uint256 amount,`
`  uint32 timestampLimit`
`  )`

Vulnerabilities not detected

1.2 SQRLockup.sol contract methods analysis:

```
withdraw(
    string memory userId,
    string memory transactionId,
    address to,
    uint256 amount,
    uint32 timestampLimit
  )
```

Vulnerabilities not detected

TOKEN FLOW	Tokens out, onlyOwner

```
verifySignature(
    string memory userId,
    string memory transactionId,
    address to,
    uint256 amount,
    uint32 timestampLimit,
    bytes memory signature
  )
```

Vulnerabilities not detected

```
withdrawSig(
    string memory userId,
    string memory transactionId,
    address to,
    uint256 amount,
    uint32 timestampLimit,
    bytes memory signature
  )
```

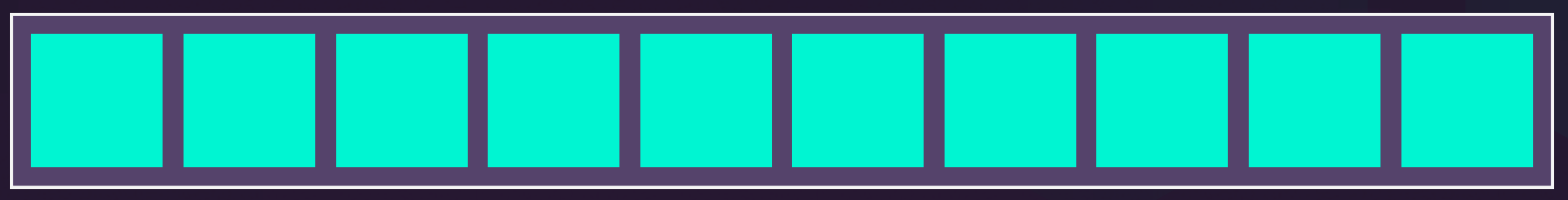
Vulnerabilities not detected

TOKEN FLOW	Tokens out, public

Verification checksums

Contract name	Bytecode hash(SHA-256)
SQRLockup.sol	8a7e547b7a011e15ab6783705558e91dc83694f0a4ea5c2643b36476d4180188

Project evaluation



10/10

Get in touch 👋

@smartstatetech

@smartstate

@SmartStateAudit

@smartstatetech

@smartstate.tech

View this report on Smartstate.tech

info@smartstate.tech

smartstate.tech