Competitive Security Assessment

TokenTable

Mar 14th, 2023

Secure3

secure3.io

Summary 3

Overview 4

Audit Scope 5

Code Assessment Findings 6

TTB-1:After `cancel()`, user can still `deposit()` and `claim()` 8

TTB-2:Cache the array length in for loop condition 11

TTB-3:Front-runable initializer 13

TTB-4:Miss 0 address check for `initialize` function 15

TTB-5:Miss array length check for user input parameters 16

TTB-6:Miss event for `setPermission()` 18

TTB-7:There is too little signature information in cancel() 20

TTB-8: `TokenTableUnlocker` does not support fee-on-transfer token 23

TTB-9:unrestricted presetIndex parameter allows the creation of actual outside of unlockingSchedulePresets 25

Disclaimer 26

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	TokenTable
Platform & Language	Solidity
Codebase	• https://github.com/EthSign/TokenTable-Unlock-Contract • audit commit - 85e2445c64f2b767b9aa649052fb9aa4bd7c2128 • final commit - 4dda9ea469765de973a8451ee87c9357749cf8f6
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

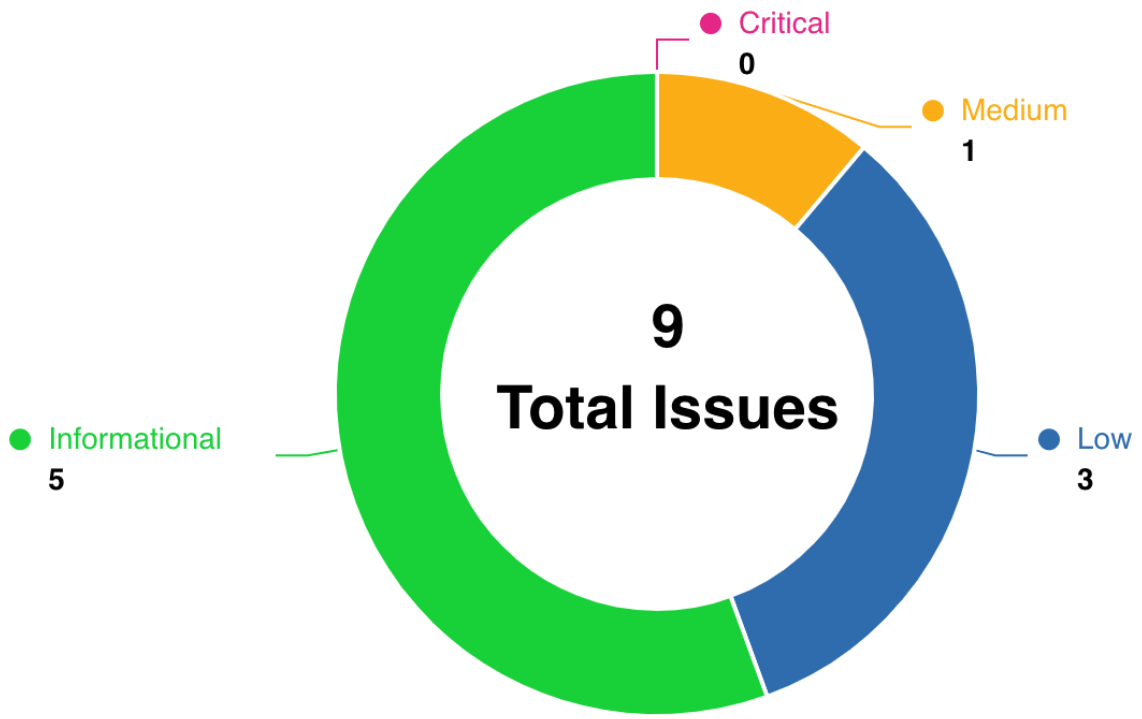
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	0	0	0	0	0	0
Medium	1	0	0	1	0	0
Low	3	0	1	2	0	0
Informational	5	0	1	3	0	1

Audit Scope

File	Commit Hash
contracts/core/TokenTableUnlockerV1.sol	85e2445c64f2b767b9aa649052fb9aa4bd7c2128
contracts/abstract/ITokenTableUnlockerV1.sol	85e2445c64f2b767b9aa649052fb9aa4bd7c2128
contracts/presets/TTUEnumerableV1.sol	85e2445c64f2b767b9aa649052fb9aa4bd7c2128
contracts/presets/TTUERC20V1.sol	85e2445c64f2b767b9aa649052fb9aa4bd7c2128
contracts/core/EthSignCommonFramework.sol	85e2445c64f2b767b9aa649052fb9aa4bd7c2128
contracts/core/TTUSecureEnclave.sol	85e2445c64f2b767b9aa649052fb9aa4bd7c2128

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
TTB-1	After `cancel()`, user can still `deposit()` and `claim()`	Logical	Informational	Declined	8olidity
TTB-2	Cache the array length in for loop condition	Gas Optimization	Informational	Fixed	iczc
TTB-3	Front-runable initializer	Privilege Related	Informational	Acknowledged	8olidity
TTB-4	Miss 0 address check for `initialize` function	Logical	Low	Fixed	Hupixiong3

TTB-5	Miss array length check for user input parameters	Logical	Low	Acknowledged	Hupixiong3, 8olidity
TTB-6	Miss event for `setPermission()`	Logical	Informational	Fixed	comcat, 8olidity, co2kim
TTB-7	There is too little signature information in cancel()	Signature Forgery or Replay	Informational	Fixed	8olidity
TTB-8	`TokenTableUnlocker` does not support fee-on-transfer token	Logical	Low	Fixed	comcat
TTB-9	unrestricted presetIndex parameter allows the creation of actual outside of unlockingSchedulePresets	Logical	Medium	Fixed	iczc

TTB-1:After `cancel()`, user can still `deposit()` and `claim()`

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/contracts/core/TokenTableUnlockerV1.sol#L184-L190	Declined	8olidity

Code

```
184:    function cancel(
185:        address actualAddress,
186:        uint256 actualPresetIndex,
187:        address claimTo,
188:        bytes calldata claimToSig,
189:        address refundAddress
190:    )
```

Description

8olidity : The original design intention of `cancel()` should be to stop the project. In the code, the deposited funds minus the claimed funds are sent to refundAddress. And in the `TTUEnumerableV1` contract. It is judged that when the project has been canceled, it cannot be operated. But in the `tokentableUnlockerV1` contract, even if it has been canceled, the user can still `deposit()` and `claim()`

POC

```
it('to the same address', async () => {
  // Time jump to after end of first interval
  await time.setNextBlockTimestamp(
      linearStartTimestamp + unlockInterval + 1
  )
  await mine()
  // Cancel
  await expect(
      contract
          .connect(s1)
          .cancel(
              s2.address,
              presetIndex,
              s2.address,
              utils.formatBytes32String(''),
              s1.address
          )
  ).to.be.revertedWithCustomError(contract, 'NotPermissioned')
  await contract
      .connect(s0)
      .setPermission(CANCELABLE_PERMISSION, s1.address, true)
  const {amountClaimed, amountRefunded} = await contract
      .connect(s1)
      .callStatic.cancel(
          s2.address,
          presetIndex,
          s2.address,
          utils.formatBytes32String(''),
          s1.address
      )
  const cancelTx = await contract
      .connect(s1)
      .cancel(
          s2.address,
          presetIndex,
          s2.address,
          utils.formatBytes32String(''),
          s1.address
      )
  await expect(cancelTx)
      .to.emit(contract, 'ActualCancelled')
      .withArgs(
          actualId,
```

```
amountClaimed,
          amountRefunded,
          s2.address,
          s1.address
      )
  console.log("----------------------------------------");
  console.log(await mockERC20.balanceOf(contract.address));
  // deposit
  const depositTx2 = await contract.connect(s1).deposit(actualId, amountPerInterval)
  await expect(depositTx2)
  .to.emit(contract, 'TokensDeposited')
  .withArgs(actualId, amountPerInterval)
  console.log(await mockERC20.balanceOf(contract.address));
  console.log("----------------------------------------");

})
```

Comment from Secure3: We are not sure if this is by design or a miss on the condition checking, pass it to client for further review.

Recommendation

8olidity : It is recommended to set a variable after `cancel()` to mark that it has been cancelled. Unable to call deposit(), claim()

Client Response

cancel() is only used to cancel a single actual unlocking schedule for a single person in a project, it doesn't cancel the whole project, so the issue was more of a misunderstanding

TTB-2:Cache the array length in for loop condition

Category	Severity	Code Reference	Status	Contributor
Gas Optimization	Informational	• code/contracts/core/TokenTableUnlockerV1.sol#L150 • code/contracts/core/TokenTableUnlockerV1.sol#L172	Fixed	iczc

Code

```
150:        for (uint256 i = 0; i < to.length; i++) {

172:        for (uint256 i = 0; i < actualIds.length; i++) {
```

Description

iczc : In the following loop, the length of the array is fetched anew every time before the condition is checked. However, since the length of the array remains constant during execution, this results in unnecessary fetching.

```
    function depositBatch(
        bytes32[] calldata actualIds,
        uint256[] calldata amounts
    ) external override {
        // Will revert if lengths are different
        for (uint256 i = 0; i < actualIds.length; i++) {
            _deposit(actualIds[i], amounts[i]);
        }
    }
```

```
    function createActualBatch(
        address[] calldata to,
        uint256[] calldata presetIndex,
        uint256[] calldata linearStartTimestamp,
        uint256[] calldata totalAmount,
        uint256[] calldata amountDepositingNow,
        string[] calldata metadata
    ) external override hasPermission(FOUNDER_PERMISSION) {
        for (uint256 i = 0; i < to.length; i++) {
            _createActual(
                to[i],
                presetIndex[i],
                linearStartTimestamp[i],
                0, // compiler limitation, "stack too deep" error if we include amountSkipped[]
                totalAmount[i],
                amountDepositingNow[i],
                metadata[i]
            );
        }
    }
```

Recommendation

iczc : Use cache variable to reduce the number of array length fetch to just 1 for a reduction in gas.

```
unit256 length = actualIds.length;
for (uint256 i = 0; i < length; i++) {
    _deposit(actualIds[i], amounts[i]);
}
```

Client Response

Fixed

TTB-3:Front-runable initializer

Category	Severity	Code Reference	Status	Contributor
Privilege Related	Informational	• code/contracts/core/TokenTableUnlockerV1.sol#L71-L88	Acknowledged	8olidity

Code

```
71:     function initialize(
72:         uint256 chainId_,
73:         address forwarder,
74:         IERC20 projectToken
75:     ) public initializer {
76:         _initialize(chainId_, forwarder);
77:         _initializeSE(address(projectToken));
78:         _DOMAIN_SEPARATOR = keccak256(
79:             abi.encode(
80:                 _EIP712_DOMAIN_TYPE_HASH,
81:                 keccak256("TokenTableUnlocker"),
82:                 keccak256("1"),
83:                 chainId_,
84:                 address(this),
85:                 _SALT
86:             )
87:         );
88:     }
```

Description

8olidity : There is nothing preventing another account from calling the initializer before the contract owner. In the best case, the owner is forced to waste gas and re-deploy. In the worst case, the owner does not notice that his/her call reverts, and everyone starts using a contract under the control of an attacker

```
 function initialize(
    uint256 chainId_,
    address forwarder,
    IERC20 projectToken
) public initializer {//@audit
    _initialize(chainId_, forwarder);
    _initializeSE(address(projectToken));
    _DOMAIN_SEPARATOR = keccak256(
        abi.encode(
            _EIP712_DOMAIN_TYPE_HASH,
            keccak256("TokenTableUnlocker"),
            keccak256("1"),
            chainId_,
            address(this),
            _SALT
        )
    );
}
```

Recommendation

8olidity : It is recommended to call `initialize` function immediately after the the contract is deployed. Or use a deploy script to do programmatically for two steps.

Client Response

the initializer will be called at time of deployment in acccordance to the upgradeable pattern

TTB-4:Miss 0 address check for `initialize` function

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/core/TokenTableUnlockerV1.sol#L71-L78	Fixed	Hupixiong3

Code

```
71:     function initialize(
72:         uint256 chainId_,
73:         address forwarder,
74:         IERC20 projectToken
75:     ) public initializer {
76:         _initialize(chainId_, forwarder);
77:         _initializeSE(address(projectToken));
78:         _DOMAIN_SEPARATOR = keccak256(
```

Description

Hupixiong3 : The `initialize()` function lacks zero address check, and `forwarder` can set to 0 and `ERC2771Recipient` does not have a external setter function to update the `_trustedForwarder` (`_forwarder`). Same as `projectToken`, which can only be updated during `onlyInitializing` phase.

Recommendation

Hupixiong3 : Add 0 address check

Client Response

added `setTrustedForwarder` function with `onlyOwner` to be able to reset it with

TTB-5:Miss array length check for user input parameters

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/core/TokenTableUnlockerV1.sol#L142-L149 • code/contracts/core/TokenTableUnlockerV1.sol#L167-L170	Acknowledged	Hupixiong3, 8olidity

Code

```
142:    function createActualBatch(
143:        address[] calldata to,
144:        uint256[] calldata presetIndex,
145:        uint256[] calldata linearStartTimestamp,
146:        uint256[] calldata totalAmount,
147:        uint256[] calldata amountDepositingNow,
148:        string[] calldata metadata
149:    ) external override hasPermission(FOUNDER_PERMISSION) {

167:    function depositBatch(
168:        bytes32[] calldata actualIds,
169:        uint256[] calldata amounts
170:    ) external override {
```

Description

Hupixiong3 : The user input array arguments for the function `createActualBatch()` and `depositBatch()` need to be at the same length and used as one-to-one pairs. However, there is a lack of array length consistency check and calls may fail or the variable from the shorter array will be assigned to default value 0 due to inconsistent array lengths of the arrays.

8olidity : In `createActualBatch()` and `depositBatch()`, multiple arrays are passed in, but it is not judged whether the lengths of each array are the same. If the length of one array is smaller than the other arrays, the operation will revert.

Recommendation

Hupixiong3 : Added an array length consistency check.

Consider below fix in the `depositBatch()` function

```
if(actualIds.length ! = amounts.length) revert DifferentArrayLength();
```

8olidity :

```
    function depositBatch(
        bytes32[] calldata actualIds,
        uint256[] calldata amounts
    ) external override {
        require(actuallds.length == amounts.length);
        for (uint256 i = 0; i < actualIds.length; i++) {
            _deposit(actualIds[i], amounts[i]);
        }
    }
```

Client Response

Mismatched array lengths will result in a revert due to out of range index (as far as I can see) and shouldn't cause any damage. Besides, createActualBatch has way too many parameters to compare and imo it's a waste of gas

TTB-6:Miss event for `setPermission()`

Category	Severity	Code Reference	Status	Contributor
Logical	Informational	• code/contracts/core/TokenTableUnlockerV1.sol#L230-L236	Fixed	comcat, 8olidity, co2kim

Code

```
230:    function setPermission(
231:        bytes32 permissionId,
232:        address addr,
233:        bool isPermitted
234:    ) external override onlyOwner {
235:        permissions[permissionId][addr] = isPermitted;
236:    }
```

Description

comcat : the function `setPermission` is a critical state chaging function while it lacks the corresponding event emission.

8olidity : For key operations in the contract, such as setting the permissions of addr in the `setPermission()` function, it is recommended to add events.

```
    function setPermission(
        bytes32 permissionId,
        address addr,
        bool isPermitted
    ) external override onlyOwner {
        permissions[permissionId][addr] = isPermitted;
    }
```

co2kim : The `setPermission()` function changes critical contract state variable, but it does not have event emitted, this is not the best practice.

Recommendation

comcat : add the corresponding events.

```
function setPermission(bytes32 permissionId, address addr, bool isPermitted)
        external
        override
        onlyOwner
    {
        permissions[permissionId][addr] = isPermitted;
        emit SetPermission(permissionId,addr,isPermitted);
    }
```

8olidity : Add an event for use with `setPermission()`
co2kim : Emit an event in the `setPermission()` function for better monitoring.

Client Response

Fixed

TTB-7:There is too little signature information in cancel()

Category	Severity	Code Reference	Status	Contributor
Signature Forgery or Replay	Informational	• code/contracts/core/TokenTableUnlockerV1.sol#L201-L213	Fixed	8olidity

Code

```
201:         if (claimTo != actualAddress) {
202:             if (
203:                 !SignatureChecker.isValidSignatureNow(
204:                     actualAddress,
205:                     ECDSA.toTypedDataHash(
206:                         _DOMAIN_SEPARATOR,
207:                         getCancelClaimToSigHash(actualPresetIndex, claimTo)
208:                     ),
209:                     claimToSig
210:                 )
211:             ) revert InvalidClaimSig();
212:             claimToAddress = claimTo;
213:         }
```

Description

8olidity : In the `cancel()` function, when `claimTo != actualaddress`, the signature `claimToSig` will be checked against the signer `actualAddress`. The signature does not contain the `msg.sender` and no nonce information.

POC

The s3 account re-uses the claimToSig from the s2 account to call `cancel` and suceeded

```
it('to a different address', async () => {
        const claimTo = s0.address
        // Time jump to after end of first interval
        await time.setNextBlockTimestamp(
            linearStartTimestamp + unlockInterval + 1
        )
        await mine()
        // Cancel
        await contract
            .connect(s0)
            .setPermission(CANCELABLE_PERMISSION, s1.address, true)
        const message = {
            contract: contract.address,
            presetIndex: presetIndex,
            claimTo: claimTo
        }
        const claimToSig = await s2._signTypedData(
            EIP712_CONSTANTS.DOMAIN_DATA,
            EIP712_CONSTANTS.STRUCT_TYPES,
            message
        )
        const {amountClaimed, amountRefunded} = await contract
            .connect(s1)
            .callStatic.cancel(
                s2.address,
                presetIndex,
                claimTo,
                claimToSig,
                s1.address
            )
        await contract
        .connect(s0)
        .setPermission(CANCELABLE_PERMISSION, s3.address, true)
        const cancelTx = await contract
            .connect(s3)
            .cancel(
                s2.address,
                presetIndex,
                claimTo,
                claimToSig,
                s1.address
            )
        await expect(cancelTx)
```

```
  .to.emit(contract, 'ActualCancelled')
              .withArgs(
                  actualId,
                  amountClaimed,
                  amountRefunded,
                  claimTo,
                  s1.address
              )
          })
  })
```

Recommendation

8olidity : It is recommended that the signature(`claimToSig`) to include `msg.sender` and a nonce to avoid signature theft and replay.

Client Response

Fixed

TTB-8: `TokenTableUnlocker` does not support fee-on-transfer token

Category	Severity	Code Reference	Status	Contributor
Logical	Low	• code/contracts/core/TokenTableUnlockerV1.sol#L407-L411	Fixed	comcat

Code

```
407:         IERC20(getProjectToken()).safeTransferFrom( // If project token isn't set, this will
revert
408:             _msgSender(),
409:             address(this),
410:             amountDepositingNow
411:         );
```

Description

comcat : inside the `_createActual` function, it specifies the totalAmount, however, when transfer the token from funder to the contract, it only uses the

```
IERC20(getProjectToken()).safeTransferFrom( // If project token isn't set, this will revert
        _msgSender(), address(this), amountDepositingNow);
```

and it ignores the fee on transfer tokens, which means that the actual token received on the contract is less than the `amountDepositingNow`. however, inside the `_claim` logic, it just transfer tokens out according to the amount stored in the contract, not the real balance of the contract.

Recommendation

comcat : use snapchat method to measure the actual deposit amount:

```
uint balanceBefore = IERC20(getProjectToken()).balanceOf(address(this));
IERC20(getProjectToken()).safeTransferFrom( // If project token isn't set, this will revert
        _msgSender(), address(this), amountDepositingNow);
uint balanceAfter = IERC20(getProjectToken()).balanceOf(address(this));;
uint actualDepositingAmount = balanceAfter - balanceBefore;
```

Client Response

Fixed

TTB-9:unrestricted presetIndex parameter allows the creation of actual outside of unlockingSchedulePresets

Category	Severity	Code Reference	Status	Contributor
Logical	Medium	• code/contracts/core/TokenTableUnlockerV1.sol#L363	Fixed	iczc

Code

```
363:    function _createActual(
```

Description

iczc : The `presetIndex` parameter in `createActual()` and `createActualBatch()` functions is currently unrestricted, meaning that users can input any number even if it is outside the range of the unlockingSchedulePresets array. This could potentially result in the creation of an actual under incorrect corresponding preset. And the `claim()` and the `cancel()` function suffer from a similar issue.

Recommendation

iczc : It is recommended to limit the values of `presetIndex` to ensure that only referencing existing presets can be created, by checking `presetIndex < unlockingSchedulePresets.length` in the `_createActual()` function.

Client Response

Fixed

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3's prior written consent in each instance.

This report is not an "endorsement" or "disapproval" of any particular project or team. This report is not an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3's position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.