Security Research - Sablier

Turing Consulting

2023-11-30

Contents

1 Introduction

This audit report presents the comprehensive assessment conducted by Turing Consulting on the security research done on the smart contract codebase. As a leading provider of gas auditing services and security research, we specialize in analysing and optimizing smart contracts to enhance gas efficiency and reduce costs for users on the Ethereum network while keeping it on the utmost security.

Our auditing approach revolves around leveraging cutting-edge technologies and methodologies to secure the codebase. Throughout the audit process, we employ a combination of manual analysis and automated tool analysis, including the utilization of our proprietary tool developed specifically for gas optimization.

To achieve our objectives, we follow a comprehensive process that covers various stages of the audit:

- **Codebase Understanding:** Our team conducts a meticulous manual analysis of the smart contract codebase, thoroughly examining each line to gain a comprehensive understanding of its structure, logic, and functionality. This deep understanding allows us to identify potential areas for gas optimization.
- **Architecture Audit**: We conduct a detailed examination of the codebase architecture to evaluate its design. This audit helps us identify any structural improvements that can enhance the security.
- **Automated Tool Analysis**: As part of our audit process, we leverage our proprietary tool specifically developed for gas optimization. The tool offers advanced analysis capabilities, allowing us to conduct automated testing to identify specific areas for gas optimization within the codebase, we also run common static analysis tools for the EVM to check for security issues.
- **Solidity Audit:** Solidity is the primary programming language used for Ethereum smart contracts. We conduct a detailed review of the Solidity code to identify any areas where security can be improved

This report encompasses all the information gathered throughout the auditing process, providing a comprehensive overview of the security solutions analysed in the smart contract codebase. It includes specific recommendations for code, as well as the insights gained from our proprietary automated tool. These recommendations aim to enhance security, gas efficiency, reduce costs, and optimize the codebase while maintaining the highest level of security and functionality.

2 Audit details

All the changes described on the changelog of the version 1.10.0 up to this frozen commits:

2d19596e02e6c7ddab15ece1d140addee4d6f10c

0004fd2e61e032df3d895045ec414ecb212ddcc8

3 Audit findings

3.1 [LOW-01] - Use safeTransferFrom() instead of transferFrom()

3.1.1 Lines of code

SablierV2Batch.sol#L262

3.1.2 Bug description

SablierV2Batch's `_handleTransfer()` uses the standard `IERC20.sol`'s `transferFrom()` method.

The EIP20 standard dictates that `transfer()` and `transferFrom()` functions must return a boolean value indicating success. This parameter needs to be checked to ensure the token transfer was succesful. Despite this, some tokens (like USDT) don't correctly implement the EIP20 standard and their `transfer()`/`transferFrom()` functions return void instead of a success boolean. Calling these functions with the correct EIP20 function signatures will always revert.

Tokens that don't actually perform the transfer and return false are still counted as a correct transfer, and tokens that don't correctly implement the latest EIP20 spec, like USDT, will be unusable if such function is called on them as they revert the transaction because of the missing return value.

Because of this, SablierV2Batch's `_handleTransfer()` will revert with such special tokens, effectively DoS'ing functions calling `_handleTransfer()` internally.

3.1.3 Impact

Low, given that it won't result in value loss but it limits the protocol (and specifically SablierV2Batch's `createWithDurations()`, `createWithRange()`, `createWithDeltas()` and `createWithMilestones()` functions) usability for some assets.

3.1.4 Recommended Mitigation

It is recommended to use `safeTransferFrom()` instead of `transferFrom()` when transferring the assets to the batch contract in SablierV2Batch's `_handleTransfer()`, just like it is performed in all other protocol areas.

3.1.5 Remmediation

Fixed on PR 227

3.2 [LOW-02] - Lockup ERC-4906 redundant metadata update

3.2.1 Lines of code

SablierV2Lockup.sol#L325

3.2.2 Bug description

Upon calling the `withdrawMaxAndTransfer` function, the ERC-4906 compliant call responsible for emitting the `MetadataUpdate` event is executed twice. This duplication occurs due to the presence of the `updateMetadata` modifier within both the function itself and the internal function `_transfer`.

3.2.3 Impact

The duplication of the `MetadataUpdate` event emission through the `updateMetadata` modifier in the `withdrawMaxAndTransfer` function and its internal function `_transfer` can lead to misinterpretations by external applications such as frontend apps or subgraphs. This may result in an inaccurate count of metadata updates.

3.2.4 Recommended Mitigation

To address this issue, it is advised to remove the `updateMetadata` modifier from the `withdrawMaxAndTransfer` function, as the internal function `_transfer` already incorporates this modifier.

3.2.5 Remmediation

Fixed on PR 735

3.3 [I-01] - Consider implementing 2-step admin transfer

3.3.1 Bug description

The contracts under scope of this audit inherit from `Adminable` contract, which pemits the current owner to conduct a one-step transfer. Although this practice is widespread, it introduces a potential risk of losing control over the contract if the owner mistakenly transfers ownership to an incorrect address. Implementing a two-step ownership transfer would mitigate this risk. In a 2-step procedure, the current owner initiates the transfer by proposing a new owner. Subsequently, the designated account, acknowledged as the new owner, can claim the new ownership.

3.3.2 Recommended Mitigation

It is recommended to implement a two-step ownership transfer.

3.3.3 Remmediation

Acknowledged

3.4 [I-02] - Project's solidity version not supported on some chains

3.4.1 Bug description

Solidity version compatibility issues exist on certain chains due to the utilization of the PUSH0 instruction, which pushes the constant value 0 onto the stack. Notably, this opcode is not supported on various chains, including Arbitrum, and may pose challenges for projects compiled with Solidity versions equal to or greater than 0.8.20, where PUSH0 was introduced.

3.4.2 Impact

Since Sablier is supposed to be deployed across a huge variety of chains, its essential to make sure it will be supported in all of them. To prevent unexpected issues, consider using a Solidity version lower than 0.8.20 in the project configuration file `foundry.toml`.

3.4.3 Recommended Mitigation

It is advised to either retain Solidity version 0.8.19 in the configuration file or employ a script that dynamically adjusts between versions 0.8.19 and 0.8.21 based on the selected blockchain.

3.4.4 Remmediation

Acknowledged

3.5 [I-03] - Incomplete NatSpec comments

3.5.1 Bug description

It was identified that some contract functions have incomplete code documentation in their corresponding interfaces (i.e missing @return tags), which affects the understandability, auditability, and usability of the code.

3.5.2 Recommended Mitigation

Complete the interfaces documentation, adding the missing `@return` tag to the functions returning data in order to properly follow NatSpec and increase the codebase readability.

3.5.3 Remmediation

Acknowledged

3.6 [GAS-01] - Redundant return in `checkAndCalculateFees`

3.6.1 Lines of code

Helpers.sol#L30

3.6.2 Bug description

Helpers library's `checkAndCalculateFees()` allows Sablier to check and calculate the fee amounts when creating a stream. If the deposited amount is 0, `checkAndCalculateFees()` will directly return 0 as the `deposit`, `protocolFee`, and `brokerFee` amounts :

```
// Helpers.sol
function checkAndCalculateFees(
        uint128 totalAmount,
        UD60x18 protocolFee,
        UD60x18 brokerFee,
        UD60x18 maxFee
    )
        internal
        pure
        returns (Lockup.CreateAmounts memory amounts)
    {
        // When the total amount is zero, the fees are also zero.
        if (totalAmount == 0) {
            return Lockup.CreateAmounts(0, 0, 0);
        }
        ...
    }
```

Although this approach is correct, it will be more optimal in terms of gas to directly revert the transaction instead of returning 0 values, given that the transaction will always revert later if the `depositAmount` returned from fee calculation is 0:

- Creating a linear stream, `checkCreateWithRange()` will ensure the transaction reverts if `depositAmount` is 0:

```
// SablierV2LockupLinear.sol
function _createWithRange(LockupLinear.CreateWithRange memory params) internal returns
↪  (uint256 streamId) {
        ...

        // Checks: check the fees and calculate the fee amounts.
        Lockup.CreateAmounts memory createAmounts =
            Helpers.checkAndCalculateFees(params.totalAmount, protocolFee, params.broker.fee,
↪  MAX_FEE);

        // Checks: validate the user-provided parameters.
        Helpers.checkCreateWithRange(createAmounts.deposit, params.range);

        ...
    }

// Helpers.sol
function checkCreateWithRange(uint128 depositAmount, LockupLinear.Range memory range) internal
↪  view {
        // Checks: the deposit amount is not zero.
        if (depositAmount == 0) {
            revert Errors.SablierV2Lockup_DepositAmountZero();
        }
```

```
        ...
    }
```

- Creating a dynamic stream, `checkCreateWithMilestones()` will ensure the transaction reverts if `depositAmount` is 0:

```
// SablierV2LockupDynamic.sol
function _createWithMilestones(LockupDynamic.CreateWithMilestones memory params)
        internal
        returns (uint256 streamId)
    {
        ...

        // Checks: check the fees and calculate the fee amounts.
        Lockup.CreateAmounts memory createAmounts =
            Helpers.checkAndCalculateFees(params.totalAmount, protocolFee, params.broker.fee,
↪   MAX_FEE);

        // Checks: validate the user-provided parameters.
        Helpers.checkCreateWithMilestones(createAmounts.deposit, params.segments,
↪   MAX_SEGMENT_COUNT, params.startTime);

    }

// Helpers.sol
function checkCreateWithMilestones(
        uint128 depositAmount,
        LockupDynamic.Segment[] memory segments,
        uint256 maxSegmentCount,
        uint40 startTime
    )
        internal
        view
    {
        // Checks: the deposit amount is not zero.
        if (depositAmount == 0) {
            revert Errors.SablierV2Lockup_DepositAmountZero();
        }

            ...
    }
```

3.6.3 Recommended Mitigation

Directly revert the transaction in `Helpers.checkAndCalculateFees()` if `totalAmount` to create the stream is 0 instead of returning all `Lockup.CreateAmounts` amounts as 0.

3.6.4 Remmediation

Acknowledged

3.7 [GAS-02] - Redundant zero initialization

3.7.1 Bug description

Solidity does not recognize null as a value, so `uint256` variables are initialized to zero. Setting a `uint256` variable to zero is redundant and can waste gas.

There are several places where variables are initialized to zero:

- `SablierV2LockupDynamic.sol`:
  - SablierV2LockupDynamic.sol#L343
  - SablierV2LockupDynamic.sol#L566
- `SablierV2NFTDescriptor.sol`:
  - SablierV2NFTDescriptor.sol#L141
- `SablierV2Lockup.sol`:
  - SablierV2Lockup.sol#L174
  - SablierV2Lockup.sol#L346
- `Helpers.sol`:
  - Helpers.sol#L180
- `SVGElements.sol`:
  - SVGElements.sol#L242
- `SablierV2Batch.sol`:
  - SablierV2Batch.sol#L43
  - SablierV2Batch.sol#L97
  - SablierV2Batch.sol#L155
  - SablierV2Batch.sol#L209
- `SablierV2ProxyTarget.sol`:
  - SablierV2ProxyTarget.sol#L65
  - SablierV2ProxyTarget.sol#L196

- SablierV2ProxyTarget.sol#L252
- SablierV2ProxyTarget.sol#L424
- SablierV2ProxyTarget.sol#L480
- SablierV2ProxyTarget.sol#L646
- SablierV2ProxyTarget.sol#L675

3.7.2 Recommended Mitigation

Remove redundant zero initializations.

3.7.3 Remmediation

Acknowledged

3.8 [GAS-03] - State variable only set in the constructor should be declared `immutable`

3.8.1 Lines of code

SablierV2MerkleStreamerLL.sol#L35

3.8.2 Bug description

`SablierV2MerkleStreamerLL.sol` allows setting a value for the storage variable `streamDurations` in the constructor. Given that there is no function that allows `streamDurations` to be changed after the contract deployment, it is recommended to make `streamDurations` immutable to avoid unnecessarily writing to storage.

3.8.3 Recommended Mitigation

Make the `streamDurations` variable immutable.

3.8.4 Remmediation

Acknowledged

3.9 [GAS-04] - Direct comparisons are recommended instead of greater-than/less-than comparisons

3.9.1 Bug description

Greater-than and less-than comparisons depend on stack order. On the other hand, direct comparisons (i.e == and `!=`) do not depend on stack order. Hence, using == and `!=` is more optimal in situations where such comparisons don't alter the code's purpose.

3.9.2 Recommended Mitigation

It is recommended to replace greater-than/less-than comparisons with direct comparisons where applicable. This is especially encouraged in all the for loops present in the codebase.

3.9.3 Remmediation

Acknowledged

3.10 [GAS-05] - Stream status can be checked more efficiently

3.10.1 Lines of code

SablierV2Lockup.sol#L100

SablierV2Lockup.sol#L88

3.10.2 Bug description

Some functions allow Sablier to check the current status of a stream. Concretely, the `isCold()` and `isWarm()` functions perform several checks that can be optimized.

Because structs are actually `uint8` types, the implementation of such functions can be performed in a way that several checks can be reduced to one.

3.10.3 Recommended Mitigation

Perform the current replacements to optimize `isCold()` and `isWarm()` functions:

- `isCold()`:

```
// SablierV2Lockup.sol
function isCold(uint256 streamId) external view override notNull(streamId) returns (bool
↪   result) {
    Lockup.Status status = _statusOf(streamId);
- result = status == Lockup.Status.SETTLED || status == Lockup.Status.CANCELED || status ==
↪   Lockup.Status.DEPLETED;
+ result = status > Lockup.Status.STREAMING;
}
```

- `isWarm()`:

```
function isWarm(uint256 streamId) external view override notNull(streamId) returns (bool
↪   result) {
    Lockup.Status status = _statusOf(streamId);
-   result = status == Lockup.Status.PENDING || status == Lockup.Status.STREAMING;
+ result = status < Lockup.Status.SETTLED;
}
```

This technique can be applied in several other places in the codebase. However, such places check the status and throw a specific error considering the stream state, as we can see in renounce(). The Sablier team should evaluate if these optimizations should be performed in functions where specific errors are thrown.

3.10.4 Remmediation

Acknowledged