

**HACKEN**

# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer:** Suberra

**Date:** February 3, 2023

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

## Document

<b>Name</b>	Smart Contract Code Review and Security Analysis Report for Suberra
<b>Approved By</b>	Evgeniy Bezuglyi   SC Audits Department Head at Hacken OU
<b>Type</b>	ERC5827; ERC1363
<b>Platform</b>	EVM
<b>Language</b>	Solidity
<b>Methodology</b>	<a href="#">Link</a>
<b>Website</b>	<a href="https://suberra.com">https://suberra.com</a>
<b>Changelog</b>	30.11.2022 - Initial Review 21.12.2022 - Second Review 03.02.2023 - Third Review

## Table of contents

<b>Introduction</b>	<b>4</b>
<b>Scope</b>	<b>4</b>
<b>Severity Definitions</b>	<b>8</b>
<b>Executive Summary</b>	<b>9</b>
<b>Checked Items</b>	<b>10</b>
<b>System Overview</b>	<b>13</b>
<b>Findings</b>	<b>14</b>
Critical	14
High	14
H01. Denial of Service Vulnerability	14
H02. Unverifiable Logic	14
H03. Data Consistency	14
Medium	15
M01. Inefficient Gas Model	15
M02. Inefficient Gas Model	15
M03. Unchecked Transfer	16
M04. Violated Checks-Effects-Interactions Pattern	16
M05. Best Practice Violation - Lock of Native Tokens	16
Low	17
L01. Floating Pragma	17
L02. Inconsistent Usage of External Libraries	17
L03. Redundant Imports	17
L04. Style Guide Violation	18
L05. Unfinished NatSpec	18
L06. State Variables that Could Be Declared as Constant	19
L07. Missing Zero Address Validation	19
L08. Comment Contradiction	20
L09. Comment Contradiction	20
L10. Comment Contradiction	20
L11. Comment Contradiction	20
L12. Unclear Use of the Virtual Specifier	21
L13. Functions that Can Be Declared External	21
L14. Redundant Use of Override Specifier	21
L15. Code Consistency	21
L16. Code Consistency	22
L17. Unindexed Events	22
L18. State Variable Default Visibility	22
<b>Disclaimers</b>	<b>24</b>

## Introduction

Hacken OÜ (Consultant) was contracted by Suberra (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

## Scope

The scope of the project is review and security analysis of smart contracts in the repository:

### Initial review scope

<b>Repository</b>	<a href="https://github.com/suberra/funnel-contracts">https://github.com/suberra/funnel-contracts</a>
<b>Commit</b>	f1f23607490f90e7f43dbf1392fb9e28d20ced8c
<b>Whitepaper</b>	Not provided
<b>Functional Requirements</b>	./README.md
<b>Technical Requirements</b>	./README.md
<b>Contracts Addresses</b>	Goerli Funnel (impl)0x3B1bbB0756250Bd981EEC8C02801d06ad5F86B75 Goerli FunnelFactory0xDd3e9D430D0681Eaa833DbD6B186E7f031f71837 Goerli USDC (funnel)0x3d5499808F8082d239a62B5c4876B6ffD23526d5
<b>Contracts</b>	File: ./src/Funnel.sol SHA3: 822a552042c90eba7fefcd35520451fafa2c199a3475bd8516056d335e6609a  File: ./src/FunnelFactory.sol SHA3: 32ed7a058b417030e05612b52d136ec7180e9d481dac27dd2da6d2ddadcb002a  File: ./src/interfaces/IERC5827.sol SHA3: 8f71695fec954be043eb4e40bcc9aa13bbf220b394639b399a8e56ff512ef84e  File: ./src/interfaces/IERC5827Payable.sol SHA3: b185e73251104612784e71f5446ecd9d4e337dac4a38ab016ad0385228be7e9c  File: ./src/interfaces/IERC5827Proxy.sol SHA3: 56db0f943c6e612c7536f9b3bea47f38a5713ad56656d3b5df3a5ebb9051ecc6  File: ./src/interfaces/IERC5827Spender.sol SHA3: 2480426d875e6cb47f975208703687d2cf1c6beb2bdba9f9ab1339fcf166abcf  File: ./src/interfaces/IFunnel.sol SHA3: 72fd64efe8ede88a71d0a047cb65d0bf8c4c342795b9952806ebe0ede88a0002  File: ./src/interfaces/IFunnelFactory.sol SHA3: a50543b85e3f6197695df0965b6be183e21a74f183bc763a2bfc67a48811934e  File: ./src/lib/EIP712.sol SHA3: e2b8d7e14a489df9eb6825eaf21dad0c2744e3748c1ea8d939cd91e3670db716  File: ./src/lib/MetaTxContext.sol SHA3: ab7101ff954b47ad6f2c863699edeefc4e6f5fbfe99f5cef4ec35a3e1690efdd  File: ./src/lib/NativeMetaTransaction.sol

	SHA3: 4f0a7b5375f2121c7a77898a64f2f597011be56d52eaf9ee1b2624db08c8fd49  File: ./src/lib/Nonces.sol SHA3: e896d1171c19f7e871d73bf232d485714150d1d532c30fd7ba5ac97157b154f7
--	--

## Second review scope

<b>Repository</b>	<a href="https://github.com/suberra/funnel-contracts">https://github.com/suberra/funnel-contracts</a>
<b>Commit</b>	1b5cab0693603edda7930698fef7911d638aaf72
<b>Whitepaper</b>	Not provided
<b>Functional Requirements</b>	./README.md <a href="https://eips.ethereum.org/EIPS/eip-5827">https://eips.ethereum.org/EIPS/eip-5827</a>
<b>Technical Requirements</b>	./README.md ./docs/index.md <a href="https://eips.ethereum.org/EIPS/eip-5827">https://eips.ethereum.org/EIPS/eip-5827</a>
<b>Contracts Addresses</b>	Not provided
<b>Contracts</b>	File: ./src/Funnel.sol SHA3: 1232c9e09815d9d5232e1520610b9c1faf63b43fc9618cc5f0010f04dc6086be  File: ./src/FunnelFactory.sol SHA3: b43d0c61eadec783f8fd244d140a1b86d6f8d1239cfdfac9830a50209117ce36  File: ./src/interfaces/IERC5827.sol SHA3: 025b5b094706d9d2650bcc88f2bd07925fe1ad7603797c2a15ca153ff5100bcd  File: ./src/interfaces/IERC5827Payable.sol SHA3: 46bcf7a92c58008690af034db49ae98c3973694ff49ef158c67b095072bbcab8  File: ./src/interfaces/IERC5827Proxy.sol SHA3: 60342184a67d7b9c16b60d077de39a1d706b1c00e8bc2ca510b369bf2855a91f  File: ./src/interfaces/IERC5827Spender.sol SHA3: fa991e940921cf3e9e1833078e544e43d2c2d33ea3302f402a1d6f31bc8f1d76  File: ./src/interfaces/IFunnel.sol SHA3: bd71395cc1773ed351160d0718e2b806f43d77b7cd4ddecdcf02c00a3b0f90d  File: ./src/interfaces/IFunnelErrors.sol SHA3: 5b5c7d98d0e06889eab7c1afccbfad904bcf61082ec7d7e2d02dd09fec11341  File: ./src/interfaces/IFunnelFactory.sol SHA3: 5dd5d2dc7ff465d7ff2994b1002cbbec68b896b29362ae7f926dbaa2bae7e496  File: ./src/lib/EIP712.sol SHA3: f8ef2179793144577b981b6851232a4897fe96e6e5e4cb867ed4c773dfe8d30c  File: ./src/lib/MathUtil.sol SHA3: d99606804be55d6b5b061399ea1608694d3dc65fc348f118c415352550cacbdd  File: ./src/lib/MetaTxContext.sol SHA3: e276e09c379a1c8e624495acb15825d9ab32877d57a05500014b04d3dff96cf7  File: ./src/lib/NativeMetaTransaction.sol SHA3: 8fa08ec245a7b6f25d0f0a02d103e27f28f002961b1a8bff3d4f1d06eae35f81  File: ./src/lib/Nonces.sol SHA3: 2f2448bbd1a46b8bcb14992195331b191cc584d33db2d7f3892eeecb9efa9db

### Third review scope

<b>Repository</b>	<a href="https://github.com/suberra/funnel-contracts">https://github.com/suberra/funnel-contracts</a>
<b>Commit</b>	e7b6affb4b4c4189ee4ff5ad54f33fbe745bf1a0
<b>Whitepaper</b>	Not provided
<b>Functional Requirements</b>	./README.md <a href="https://eips.ethereum.org/EIPS/eip-5827">https://eips.ethereum.org/EIPS/eip-5827</a> <a href="https://github.com/suberra/funnel-contracts/tree/main/docs">https://github.com/suberra/funnel-contracts/tree/main/docs</a>
<b>Technical Requirements</b>	./README.md ./docs/index.md <a href="https://eips.ethereum.org/EIPS/eip-5827">https://eips.ethereum.org/EIPS/eip-5827</a> <a href="https://github.com/suberra/funnel-contracts/tree/main/docs">https://github.com/suberra/funnel-contracts/tree/main/docs</a>
<b>Contracts Addresses</b>	./README.md
<b>Contracts</b>	<p>File: ./src/Funnel.sol SHA3: 86f6f8eb32deff64d78a8c98e6fe3f728140e8ce4d82b66d6732a5a60267b522</p> <p>File: ./src/FunnelFactory.sol SHA3: 89ce342f1f9d9d4efd37e1d3c3997be0d242fd701c3694dd997c0b0ade7b3a5e</p> <p>File: ./src/interfaces/IERC5827.sol SHA3: 3a1b1543bb20663bd04358037ba60248722fdca16d59997ac215ef88fc2fdd4d</p> <p>File: ./src/interfaces/IERC5827Payable.sol SHA3: 46bcf7a92c58008690af034db49ae98c3973694ff49ef158c67b095072bbcab8</p> <p>File: ./src/interfaces/IERC5827Proxy.sol SHA3: 60342184a67d7b9c16b60d077de39a1d706b1c00e8bc2ca510b369bf2855a91f</p> <p>File: ./src/interfaces/IERC5827Spender.sol SHA3: fa991e940921cf3e9e1833078e544e43d2c2d33ea3302f402a1d6f31bc8f1d76</p> <p>File: ./src/interfaces/IFunnel.sol SHA3: 10a121b7278aeb7140654da6461ee5de7e182b421eed885b5b1722cb053f3df4</p> <p>File: ./src/interfaces/IFunnelErrors.sol SHA3: 5b5c7d98d0e06889eab7c1afccbfa1d904bcf61082ec7d7e2d02dd09fec11341</p> <p>File: ./src/interfaces/IFunnelFactory.sol SHA3: 5dd5d2dc7ff465d7ff2994b1002cbbec68b896b29362ae7f926dbaa2bae7e496</p> <p>File: ./src/lib/EIP712.sol SHA3: 7e5a9d8b86c218b4fcfdb35607930edd16606a83087ae790ff619fb0f52ac521</p> <p>File: ./src/lib/MathUtil.sol SHA3: d99606804be55d6b5b061399ea1608694d3dc65fc348f118c415352550cacbdd</p> <p>File: ./src/lib/MetaTxContext.sol SHA3: e276e09c379a1c8e624495acb15825d9ab32877d57a05500014b04d3dff96cf7</p> <p>File: ./src/lib/NativeMetaTransaction.sol SHA3: c401c31de361dd45412d338c14077713406f61ebab5f50441caebdd05dae374e</p> <p>File: ./src/lib/Nonces.sol SHA3: 2f2448bbd1a46b8bcb14992195331b191cc584d33db2d7f3892eeecb9efa9db</p>

## Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation by external or internal actors.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation by external or internal actors.
Medium	Medium vulnerabilities are usually limited to state manipulations but cannot lead to assets loss. Major deviations from best practices are also in this category.
Low	Low vulnerabilities are related to outdated and unused code or minor Gas optimization. These issues won't have a significant impact on code execution but affect the code quality

## Executive Summary

The score measurement details can be found in the corresponding section of the [scoring methodology](#).

### Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are detailed.
- Technical description is detailed.
- NatSpec is consistent.

### Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.
- The code follows official language style guides and best practices.
- Tests are provided and relevant.

### Test coverage

Test coverage of the project is **100%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative case coverage is present.
- Interactions by several users are tested thoroughly.

### Security score

As a result of the audit, the code contains no issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

### Summary

According to the assessment, the Customer's smart contract has the following score: **10**.



*Table. The distribution of issues during the audit*

Review date	Low	Medium	High	Critical
25 November 2022	13	3	3	0
21 December 2022	6	3	1	0
3 February 2023	0	0	0	0



## Checked Items

We have audited the Customers' smart contracts for commonly known and more specific vulnerabilities. Here are some items considered:

Item	Type	Description	Status
Default Visibility	<a href="#">SWC-100</a> <a href="#">SWC-108</a>	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	<a href="#">SWC-101</a>	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	<a href="#">SWC-102</a>	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	<a href="#">SWC-103</a>	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	<a href="#">SWC-104</a>	The return value of a message call should be checked.	Passed
Access Control & Authorization	<a href="#">CWE-284</a>	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	<a href="#">SWC-106</a>	The contract should not be self-destructible while it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	<a href="#">SWC-107</a>	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	<a href="#">SWC-110</a>	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	<a href="#">SWC-111</a>	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	<a href="#">SWC-112</a>	Delegatecalls should only be allowed to trusted addresses.	Not Relevant
DoS (Denial of Service)	<a href="#">SWC-113</a> <a href="#">SWC-128</a>	Execution of the code should never be blocked by a specific contract state unless required.	Passed
Race Conditions	<a href="#">SWC-114</a>	Race Conditions and Transactions Order Dependency should not be possible.	Passed

Authorization through tx.origin	<a href="#">SWC-115</a>	tx.origin should not be used for authorization.	Not Relevant
Block values as a proxy for time	<a href="#">SWC-116</a>	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	<a href="#">SWC-117</a> <a href="#">SWC-121</a> <a href="#">SWC-122</a> <a href="#">EIP-155</a> <a href="#">EIP-712</a>	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Passed
Shadowing State Variable	<a href="#">SWC-119</a>	State variables should not be shadowed.	Passed
Weak Sources of Randomness	<a href="#">SWC-120</a>	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant
Incorrect Inheritance Order	<a href="#">SWC-125</a>	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	<a href="#">EEA-Leve1-2</a> <a href="#">SWC-126</a>	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	<a href="#">SWC-131</a>	The code should not contain unused variables if this is not <a href="#">justified</a> by design.	Passed
EIP standards violation	<a href="#">EIP</a>	EIP standards should not be violated.	Passed
Assets integrity	Custom	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	Custom	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	Custom	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	Custom	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	Custom	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed

<b>Gas Limit and Loops</b>	<b>Custom</b>	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed
<b>Style guide violation</b>	<b>Custom</b>	Style guides and best practices should be followed.	Passed
<b>Requirements Compliance</b>	<b>Custom</b>	The code should be compliant with the requirements provided by the Customer.	Passed
<b>Environment Consistency</b>	<b>Custom</b>	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
<b>Secure Oracles Usage</b>	<b>Custom</b>	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
<b>Tests Coverage</b>	<b>Custom</b>	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed
<b>Stable Imports</b>	<b>Custom</b>	The code should not reference draft contracts, which may be changed in the future.	Passed

## System Overview

*Funnels* are contracts that enforce renewable token allowances on existing ERC20 tokens, they help rate-limit the amount of tokens that can be transferred in a given time period.

Each *Funnel* contract is a proxy for an ERC20 token, funneling a large unlimited allowance to a limited allowance that it regains over time. For example, USDC will have its own funnel contract proxy, while another token like WETH will have its own funnel contract.

The system contains the following contracts:

- *Funnel* – an ERC20 proxy token that implements the EIP-5827 (Auto-renewable) standard, with additions from the EIP-1363 (Payable token).
- *FunnelFactory* – a clone factory for *Funnel* contracts.

## Privileged roles

- None.

## Risks

- EIP-5827 is not recommended for general use or implementation as it is in “Draft” status.
- Funnel uses NativeMetaTransactions, which can be complicated for integration.
- Correct calculations of funneling allowances and rete limits depend heavily on the frontend Dapps.

## Findings

### Critical

No critical severity issues were found.

### High

#### H01. Denial of Service Vulnerability

The internal function `_remainingAllowance()` will revert with overflow in situations where the `approveRenewable()` or `permitRenewable()` functions are used to approve a `max uint256 value` with a `recoveryRate > 0`.

The overflow can occur with different edge cases:

```
approveRenewable(, type(uint256).max - type(uint192).max + 1,  
type(uint192).max);  
approveRenewable(, type(uint256).max - type(uint64).max + 1,  
type(uint64).max)
```

The `_remainingAllowance()` function is used by `allowance()` and `transferFrom()`. These functions will be unusable after such approval.

**Path:**

`./src/Funnel.sol: allowance(), _remainingAllowance(), transferFrom()`

**Recommendation:** Rewrite the logic to prevent overflows.

**Status:** Fixed

(revised commit: 1b5cab0693603edda7930698fef7911d638aaf72)

#### H02. Unverifiable Logic

The Funnel contract uses the functionality of the external `solmate` contracts, which are out of the scope of this audit and whose description states that it is an experimental software.

Therefore, their security cannot be guaranteed, and their usage may lead to unexpected behavior.

**Path:**

`./src/Funnel.sol : ERC20, SafeTransferLib`

**Recommendation:** Interact only with trusted contracts, validate results after calling outer contracts.

**Status:** Fixed

(revised commit: 1b5cab0693603edda7930698fef7911d638aaf72)

#### H03. Data Consistency

The approvals performed in the Funnel contract are not connected with the approvals done in the `_baseToken` tokens.

The `EIP-5827` should check if it has enough allowance in `_baseToken` in functions `allowance()`, `transferFrom()`, and `transfer()`.

In situations where the allowance in `_baseToken` is less than the allowance calculated by Funnel, there will be data inconsistency and denial of service in transfer functions.

**Path:**

`./src/Funnel.sol : allowance(), transferFrom(), transfer()`

**Recommendation:** Consider checking the allowance from `_baseToken` and compare it with `_remainingAllowance`. React to the result in a friendly user manner.

**Status:** Mitigated (SHA2 Labs: “Intended behaviour. Checking ERC20 allowance is out of scope of the Funnel as it is solely responsible for the additional check on the renewable allowance. Also, the extra gas cost does not justify the additional check when the Funnel is meant to be used after allowance is delegated to the funnel. Finally, the ERC20 allowance is already checked and thrown by the underlying ERC20 token.”)

## ■ ■ Medium

### M01. Inefficient Gas Model

The Funnel smart contract imports and uses the `ERC20` contract directly for the `_baseToken` storage variable.

It is best practice to use interfaces when interacting with external contracts.

Importing and using an `ERC20` smart contract directly may lead to higher deployment Gas expenses when deploying new funnels.

**Path:**

`./src/Funnel.sol`

**Recommendation:** Consider using the `IERC20` interface for the `_baseToken` variable in the Funnel contract.

**Status:** Fixed

(revised commit: 1b5cab0693603edda7930698fef7911d638aaf72)

### M02. Inefficient Gas Model

The FunnelFactory smart contract imports the `Funnel` contract directly to use it in the `initialization` process.

It is best practice to use interfaces when interacting with external contracts.

Importing contracts directly increases the bytecode size of the deployed smart contract.

**Path:**

`./src/FunnelFactory.sol`

[www.hacken.io](http://www.hacken.io)

**Recommendation:** Consider using the `IFunnel` interface, with an additional declaration of the `initialize()` function, in the `FunnelFactory` contract.

**Status:** Fixed

(revised commit: e7b6affb4b4c4189ee4ff5ad54f33fbe745bf1a0)

### M03. Unchecked Transfer

An unchecked `transferFrom()` function is used in the `transfer()` function.

Tokens that do not follow the ERC20 standard (such as USDT) may return false in the case of a transfer failure, or they may not return any value at all.

This may lead to denial of service vulnerabilities when interacting with non-standard ERC20 tokens.

**Path:**

./src/Funnel.sol: transfer()

**Recommendation:** Use the `SafeERC20` library to interact with tokens safely.

**Status:** Fixed

(revised commit: 1b5cab0693603edda7930698fef7911d638aaf72)

### M04. Violated Checks-Effects-Interactions Pattern

During the function execution, some state variables are updated after the external calls.

This may lead to reentrancies, race conditions, and denial of service vulnerabilities during implementation of new functionality.

**Path:**

./src/FunnelFactory.sol: deployFunnelForToken()

**Recommendation:** Common best practices should be followed, functions should be implemented according to the Check-Effect-Interaction pattern.

**Status:** Fixed

(revised commit: e7b6affb4b4c4189ee4ff5ad54f33fbe745bf1a0)

### M05. Best Practice Violation - Lock of Native Tokens

It is considered following best practices to avoid unclear situations and prevent common attack vectors.

The contract accepts native tokens in the `executeMetaTransaction()` payable function, but there are no mechanisms for withdrawals.

This may lead to native coins being locked in the contract.

**Path:**

./src/NativeMetaTransaction.sol : executeMetaTransaction()

**Recommendation:** Remove payable mutability modifier.

**Status:** Fixed

(revised commit: e7b6affb4b4c4189ee4ff5ad54f33fbe745bf1a0)

## ■ Low

### L01. Floating Pragma

Locking the pragma helps to ensure that contracts are not accidentally deployed using an outdated compiler version that might introduce bugs that affect the contract system negatively.

**Paths:**

```
./src/Funnel.sol  
./src/FunnelFactory.sol  
./src/interfaces/IERC5827.sol  
./src/interfaces/IERC5827Payable.sol  
./src/interfaces/IERC5827Proxy.sol  
./src/interfaces/IERC5827Spender.sol  
./src/interfaces/IFunnel.sol  
./src/interfaces/IFunnelFactory.sol  
./src/lib/EIP712.sol  
./src/lib/MetaTxContext.sol  
./src/lib/NativeMetaTransaction.sol  
./src/lib/Nonces.sol
```

**Recommendation:** Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

**Status:** Fixed

(revised commit: 1b5cab0693603edda7930698fef7911d638aaf72)

### L02. Inconsistent Usage of External Libraries

All contracts use OpenZeppelin external libraries heavily.

However, the Funnel contract imports the solmate *ERC20* and *SafeTransferLib* libraries. This is inconsistent with overall external library usage.

**Path:**

```
./src/Funnel.sol
```

**Recommendation:** Consider using only one external dependency - the *IERC20* and *SafeERC20* from OpenZeppelin.

**Status:** Fixed

(revised commit: 1b5cab0693603edda7930698fef7911d638aaf72)

### L03. Redundant Imports

The use of unnecessary imports will increase the Gas consumption of the code. Thus, they should be removed from the code.

**Paths:**

```
./src/Funnel.sol : IERC20Metadata, IERC1271, Nonces, EIP712
```



```
./src/FunnelFactory.sol : IERC5827  
./src/interfaces/IFunnel.sol : IERC1363, IERC165
```

**Recommendation:** Consider removing redundant code.

**Status:** Fixed

(revised commit: 1b5cab0693603edda7930698fef7911d638aaf72)

#### L04. Style Guide Violation

The project should follow official code style guidelines.  
Inside each contract, library, or interface, use the following order:

- Type declarations
- State variables
- Events
- Modifiers
- Functions

**Path:**

```
./src/Funnel.sol
```

Functions should be grouped according to their visibility and ordered:

- constructor
- receive function (if exists)
- fallback function (if exists)
- external
- public
- internal
- private

Within a grouping, place the view and pure functions at the end.  
Some contracts are not formatted correctly.

**Paths:**

```
./src/Funnel.sol  
./src/FunnelFactory.sol
```

**Status:** Fixed

(revised commit: e7b6affb4b4c4189ee4ff5ad54f33fbe745bf1a0)

#### L05. Unfinished NatSpec

It is recommended that the code should be kept clean and properly documented with NatSpec. There are multiple functions, structs, and public storage variables that are missing proper NatSpec documentation.

There is no consistency in the multiline comment format. The `///` and `/** */` comments are used alternately.

In the IERC5827 and IERC5827Proxy interfaces, multiline comment is used incorrectly with `/* */`.

In the IERC5827Spender interface, @title is misused for description. Use @notice/@dev for explanations.

**Paths:**

```
./src/Funnel.sol : Funnel, _baseToken, RenewableAllowance,
rAllowance, INITIAL_CHAIN_ID, INITIAL_DOMAIN_SEPARATOR,
PERMIT_RENEWABLE_TYPEHASH, PERMIT_TYPEHASH, initialize(),
computeDomainSeparator(), DOMAIN_SEPARATOR(), permit(),
permitRenewable(), approve(), approveRenewable(), _approve(),
_remainingAllowance(), _checkOnApprovalReceived(), baseToken(),
supportsInterface(), balanceOf(), totalSupply(), transfer(),
fallback(), _fallback()
./src/FunnelFactory.sol : FunnelFactory, deployments,
funnelImplementation, constructor(), deployFunnelForToken(),
getFunnelForToken(), isFunnel()
./src/interfaces/IERC5827.sol : IERC5827
./src/interfaces/IERC5827Payable.sol : IERC5827Payable
./src/interfaces/IERC5827Proxy.sol : IERC5827Proxy
./src/interfaces/IFunnel.sol : IFunnel, RecoveryRateExceeded()
./src/interfaces/IFunnelFactory.sol : IFunnelFactory,
FunnelNotDeployed(), FunnelAlreadyDeployed(), InvalidToken(),
DeployedFunnel()
./src/lib/EIP712.sol : EIP712
./src/lib/MetaTxContext.sol : MetaTxContext, _msgSender()
./src/lib/NativeMetaTransaction.sol : NativeMetaTransaction,
MetaTransactionExecuted(), executeMetaTransaction(), _verifyMetaTx()
./src/lib/Nonces.sol : Nonces, _nonces
```

**Recommendation:** NatSpec documentation best practices should be followed. For reference:

<https://docs.soliditylang.org/en/v0.8.17/natspec-format.html#documentation-example>

<https://dev.to/perelynsama/natspec-the-right-way-to-comment-ethereum-smart-contracts-1b0c>

**Status:** Fixed

(revised commit: 1b5cab0693603edda7930698fef7911d638aaf72)

## L06. State Variables that Could Be Declared as Constant

There are variables in the contract that can be declared as constants to save Gas.

**Path:**

```
./src/Funnel.sol : PERMIT_RENEWABLE_TYPEHASH, PERMIT_TYPEHASH
```

**Recommendation:** Variables that do not change should be declared as constants.

**Status:** Fixed

(revised commit: 1b5cab0693603edda7930698fef7911d638aaf72)

## L07. Missing Zero Address Validation

Address parameters are used without checking against the possibility of being 0x0.

This can lead to unwanted external calls to 0x0.

**Paths:**

./src/Funnel.sol : initialize()  
./src/FunnelFactory.sol : constructor(), deployFunnelForToken()

**Recommendation:** Implement zero address validations.

**Status:** Fixed

(revised commit: 1b5cab0693603edda7930698fef7911d638aaf72)

**L08. Comment Contradiction**

The comment in the `executeMetaTransaction()` function contradicts the code: “Append userAddress and relayer address ...”. In the code, only the userAddress is appended, which is correct.

**Path:**

./src/NativeMetaTransaction.sol : executeMetaTransaction()

**Recommendation:** Remove the contradiction about the relayer address.

**Status:** Fixed

(revised commit: 1b5cab0693603edda7930698fef7911d638aaf72)

**L09. Comment Contradiction**

Spelling error in the `name()` function NatSpec description: “*fallback*”

**Path:**

./src/Funnel.sol : name()

**Recommendation:** Spelling should be fixed.

**Status:** Fixed

(revised commit: 1b5cab0693603edda7930698fef7911d638aaf72)

**L10. Comment Contradiction**

The comment in the `_fallback()` function contradicts the code: “*delegatecall* ...”. In the code, the `staticcall` is used, which is correct.

**Path:**

./src/Funnel.sol : initialize()

**Recommendation:** Remove the contradiction in the comment.

**Status:** Fixed

(revised commit: e7b6affb4b4c4189ee4ff5ad54f33fbe745bf1a0)

**L11. Comment Contradiction**

The comment on *Line 42* of the IERC5827 interface is misplaced.

**Path:**

./src/interfaces/IERC5827.sol

**Recommendation:** Correct the misplacement.

**Status:** Fixed

(revised commit: e7b6affb4b4c4189ee4ff5ad54f33fbe745bf1a0)

## L12. Unclear Use of the Virtual Specifier

There are functions in the contracts that are declared with the `virtual` specifier. These functions are not expected to be overridden, so the use of the `virtual` specifier is redundant.

**Path:**

```
./src/Funnel.sol      :      computeDomainSeparator(),      permit(),  
permitRenewable(),      _checkOnTransferReceived(),  
_checkOnApprovalReceived(), supportsInterface(), _fallback()
```

**Recommendation:** Consider removing redundant code.

**Status:** Fixed

(revised commit: e7b6affb4b4c4189ee4ff5ad54f33fbe745bf1a0)

## L13. Functions that Can Be Declared External

In order to save Gas, `public` functions that are never called in the contract should be declared as `external`.

**Paths:**

```
./src/Funnel.sol      :      initialize(),      permit(),      permitRenewable(),  
approve(),      approveRenewable(),      allowance(),      renewableAllowance(),  
supportsInterface()  
./src/FunnelFactory.sol : deployFunnelForToken(), isFunnel()
```

**Recommendation:** Use the `external` attribute for functions that are never called from the contract.

**Status:** Fixed

(revised commit: 1b5cab0693603edda7930698fef7911d638aaf72)

## L14. Redundant Use of Override Specifier

The `approve()` function does not need the override specifier in its declaration.

Since the 0.8.8 version, a function that overrides only a single interface function does not require the override specifier.

**Path:**

```
./src/Funnel.sol : approve()
```

**Recommendation:** Consider removing redundant code.

**Status:** Fixed

(revised commit: 1b5cab0693603edda7930698fef7911d638aaf72)

## L15. Code Consistency

It is best practice to write code uniformly.

There is no consistency in how reverts are handled or in the messages for those reverts in the Funnel contract.

In one case, custom errors are used; in another, revert with a message; and in another, requires.

**Path:**

```
./src/Funnel.sol      :      permit(),      permitRenewable(),  
transferFromAndCall(),      _checkOnTransferReceived(),  
approveRenewableAndCall(), _checkOnApprovalReceived()
```

**Recommendation:** Be consistent with the approach to reverting and the messages sent when reverting.

**Status:** Fixed

(revised commit: 1b5cab0693603edda7930698fef7911d638aaf72)

## L16. Code Consistency

In the FunnelFactory contract, there is no consistency in using single-line if statements. Sometimes single-line is used, and sometimes curly braces are in use. Line 30 vs. Line 34.

**Path:**

```
./src/FunnelFactory.sol      :      deployFunnelForToken(),  
getFunnelForToken(), isFunnel()
```

**Recommendation:** Be consistent with style, formatting, and patterns.

**Status:** Fixed

(revised commit: 1b5cab0693603edda7930698fef7911d638aaf72)

## L17. Unindexed Events

Having indexed event parameters makes it easier to search for these events using indexed event parameters as filters.

**Path:**

```
./src/lib/NativeMetaTransaction.sol : MetaTransactionExecuted()
```

**Recommendation:** The “indexed” keyword should be used for the event parameters.

**Status:** Fixed

(revised commit: e7b6affb4b4c4189ee4ff5ad54f33fbe745bf1a0)

## L18. State Variable Default Visibility

There is no visibility set on `rAllowance` mapping in the Funnel contract and on `deployments` mapping in the FunnelFactory contract.

Explicitly labeling the visibility makes it easier to catch incorrect assumptions about who can access the variable.

By default, the variables are marked as public, and the compiler automatically generates view functions that can be unnecessary in this case.

**Paths:**

./src/Funnel.sol : rAllowance  
./src/FunnelFactory.sol : deployments

**Recommendation:** Variables can be specified as being public, internal, or private. Explicitly define the visibility for all state variables.

**Status:** Fixed

(revised commit: e7b6affb4b4c4189ee4ff5ad54f33fbe745bf1a0)

## Disclaimers

### Hacken Disclaimer

The smart contracts given for audit have been analyzed based on the best industry practices at the time of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only – we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

### Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, Consultant cannot guarantee the explicit security of the audited smart contracts.