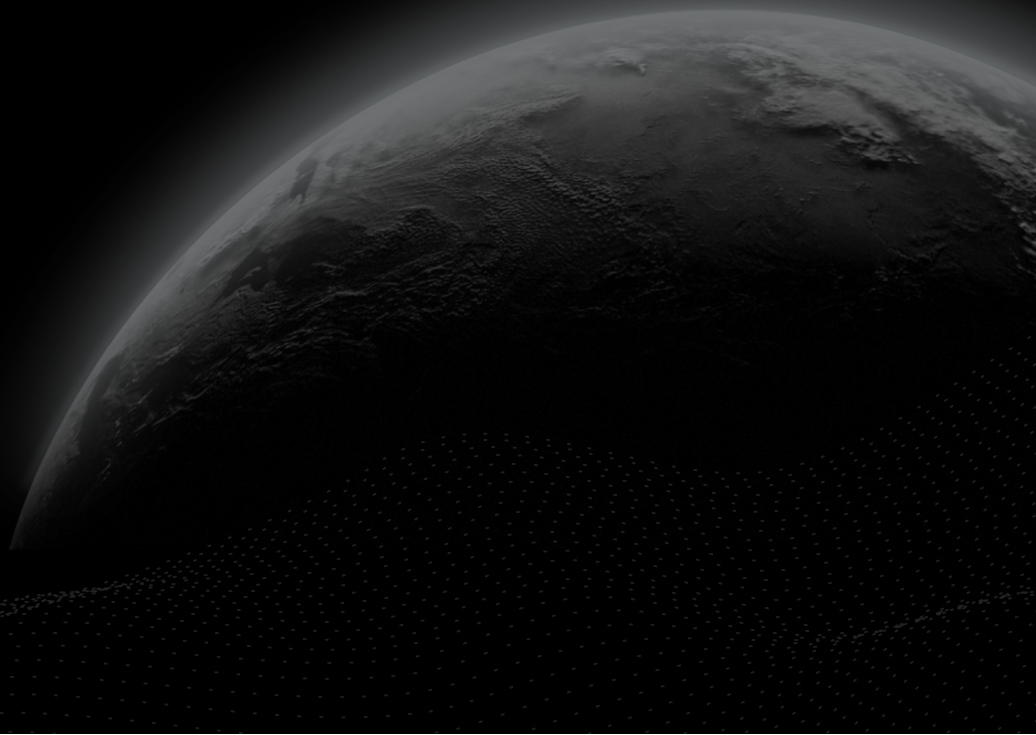




Security Assessment

Style Protocol

CertiK Assessed on Jun 3rd, 2024





CertiK Assessed on Jun 3rd, 2024

Style Protocol

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES DeFi	ECOSYSTEM Arbitrum (ARB) Ethereum (ETH)	METHODS Formal Verification, Manual Review, Static Analysis
LANGUAGE Solidity	TIMELINE Delivered on 06/03/2024	KEY COMPONENTS N/A
CODEBASE https://github.com/STYLE-Protocol/STYLE-Protocol-contracts View All in Codebase Page	COMMITTS 0391d9b1cbc103a3ac8ebd0c1dbd7610529786ae STYLE token: 0x9e91f79070926a191e41367d40ad582686f9e66d View All in Codebase Page	

Vulnerability Summary



0 Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
3 Major	3 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
1 Medium	1 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
3 Minor	3 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
2 Informational	2 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | STYLE PROTOCOL

I Summary

[Executive Summary](#)

[Vulnerability Summary](#)

[Codebase](#)

[Audit Scope](#)

[Approach & Methods](#)

I Findings

[STL-04 : Centralization Risks in STYLE-vesting-flattend.sol](#)

[STL-05 : Potential Phishing Attacks](#)

[STY-02 : Initial Token Distribution](#)

[STL-02 : Risk of Overflow/Underflow Due to Extensive Use of unchecked](#)

[STL-08 : Check Effect Interaction Pattern \(Out-of-Order Events\)](#)

[STL-09 : Check Effect Interaction Pattern Violated \(Incrementing State\)](#)

[STL-10 : Missing Input Validation on `VestingWallet.initialize\(\)`](#)

[STL-11 : Unused Interface](#)

[STY-01 : Deprecate ERC777 Implementation](#)

I Optimizations

[STL-01 : `_owner` unused in `VestingWalletHolder`](#)

I Formal Verification

[Considered Functions And Scope](#)

[Verification Results](#)

I Appendix

I Disclaimer

CODEBASE | STYLE PROTOCOL

Repository

<https://github.com/STYLE-Protocol/STYLE-Protocol-contracts>



Commit

[0391d9b1cbc103a3ac8ebd0c1dbd7610529786ae](https://github.com/STYLE-Protocol/STYLE-Protocol-contracts/commit/0391d9b1cbc103a3ac8ebd0c1dbd7610529786ae)

[STYLE token: 0x9e91f79070926a191e41367d40ad582686f9e66d](https://github.com/STYLE-Protocol/STYLE-Protocol-contracts/commit/0391d9b1cbc103a3ac8ebd0c1dbd7610529786ae)

AUDIT SCOPE | STYLE PROTOCOL

2 files audited ● 2 files with Acknowledged findings

ID	Repo	File	SHA256 Checksum
● STY	STYLE-Protocol/STYLE-Protocol-contracts	 contracts/STYLE-Token.sol	27fa3d4b441efd8e35e5b2118846a04b333486d847c872c10a65a721245cd163
● STL	STYLE-Protocol/STYLE-Protocol-contracts	 contracts/STYLE-vesting-flattend.sol	3f4b42be4f490dc89ad7fca137c1f4bf5e6b48fc6c2168295f6ca3dece4e825

APPROACH & METHODS | STYLE PROTOCOL

This report has been prepared for Style Protocol to discover issues and vulnerabilities in the source code of the Style Protocol project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | STYLE PROTOCOL



9

Total Findings

0

Critical

3

Major

1

Medium

3

Minor

2

Informational

This report has been prepared to discover issues and vulnerabilities for Style Protocol. Through this audit, we have uncovered 9 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
STL-04	Centralization Risks In STYLE-Vesting-Flattend.Sol	Centralization	Major	● Acknowledged
STL-05	Potential Phishing Attacks	Design Issue	Major	● Acknowledged
STY-02	Initial Token Distribution	Centralization	Major	● Acknowledged
STL-02	Risk Of Overflow/Underflow Due To Extensive Use Of Unchecked	Incorrect Calculation	Medium	● Acknowledged
STL-08	Check Effect Interaction Pattern (Out-Of-Order Events)	Concurrency	Minor	● Acknowledged
STL-09	Check Effect Interaction Pattern Violated (Incrementing State)	Concurrency	Minor	● Acknowledged
STL-10	Missing Input Validation On <code>VestingWallet.initialize()</code>	Volatile Code	Minor	● Acknowledged
STL-11	Unused Interface	Coding Issue	Informational	● Acknowledged
STY-01	Deprecate ERC777 Implementation	Design Issue	Informational	● Acknowledged

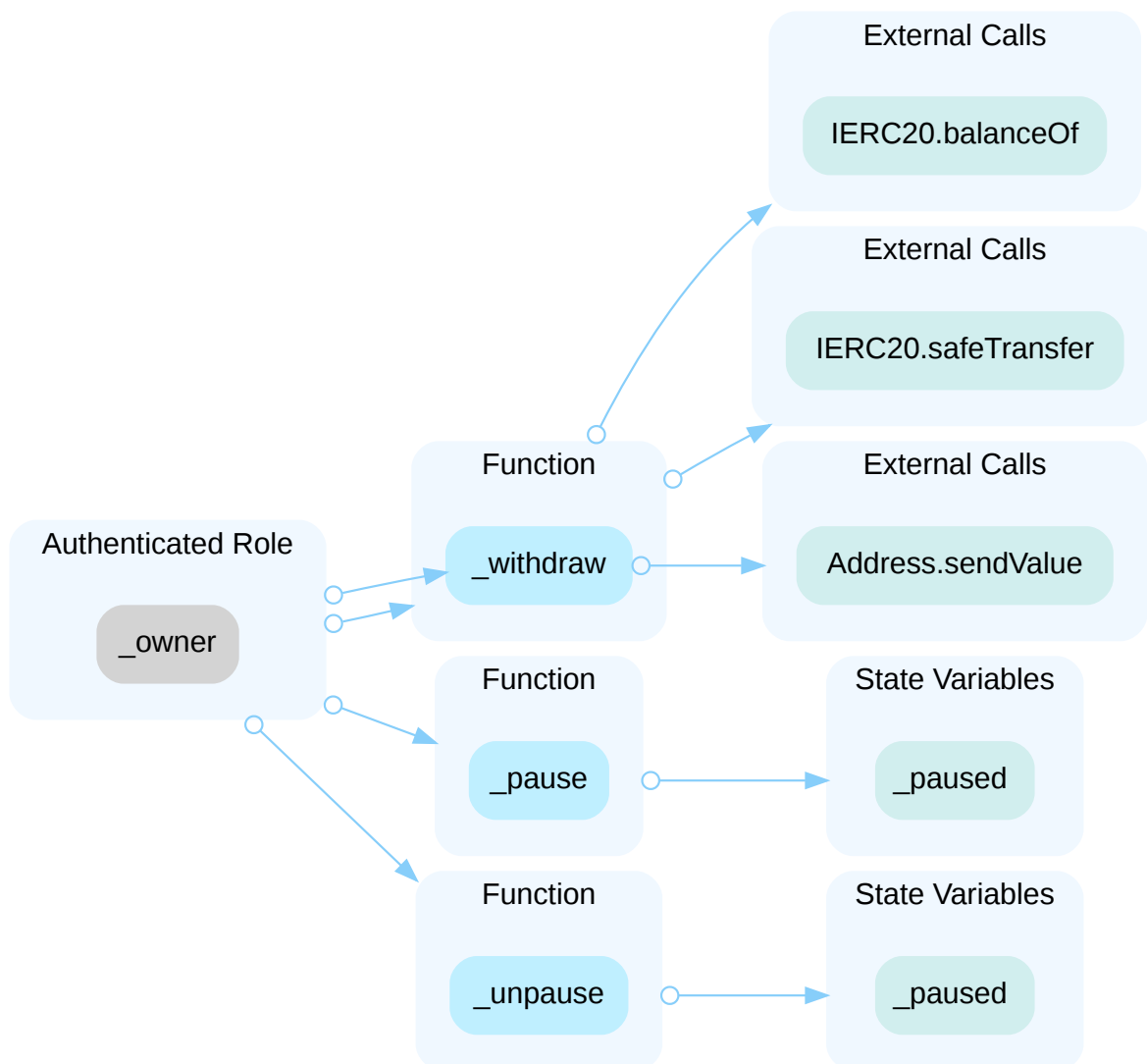
STL-04 | CENTRALIZATION RISKS IN STYLE-VESTING-FLATTEND.SOL

Category	Severity	Location	Status
Centralization	● Major	contracts/STYLE-vesting-flattend.sol: 1634, 1648, 1662, 1674	● Acknowledged

Description

In the contract `VestingWallet` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and:

- steal all the tokens from the wallet;
- pause contract and prevent users from retrieving their tokens;



Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (2/3, 3/5) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
OR
- Remove the risky functionality.

I Alleviation

[Style team, 2024/05/17]: In the initial setup it is easier and faster to have an individual issue smart contracts.

Currently, the decision-making on critical contract functions, including those that create vesting schedules is led by a BOD.

We plan to transition the control over critical contract functions, including those that create vesting schedules, to a DAO governed by our community leaders who can be voted and appointed.

This step will not only decentralize authority but also enhance user involvement and transparency in decision-making processes.

Regarding Function: Pause/UnPause:

This is a security measure put in place to allow adjusting of contracts in case of fraudulent attacks or lost wallets from Investors and other Parties that receive vesting.

We can revoke ongoing vestings and re-issue them as failsafe for investors or community funds that are vested and attacked.

STL-05 | POTENTIAL PHISHING ATTACKS

Category	Severity	Location	Status
Design Issue	● Major	contracts/STYLE-vesting-flattend.sol: 1414~1421, 1634~1635, 1648~1649, 1662~1663, 1674~1675, 1915~1916, 1940~1941	● Acknowledged

Description

The following elements can be exploited to drain tokens from wallets:

1. `VestingWallet.onlyOwner()` relies on `tx.origin`, which is susceptible to phishing attacks.
2. `VestingWalletHolder.withdraw()` has no access control, making it vulnerable.

It is also possible to use `VestingWalletHolder.switchPause()` to pause or unpause wallets through a phishing attack.

Scenario

Consider the following scenario:

Bob has created multiple `VestingWallet` contracts, where he is considered the owner of those wallets. As the owner, Bob has control over the `VestingWallet.withdraw()` function.

Alice, a malicious user, creates a contract that calls the `withdraw()` function for each `VestingWallet` that Bob has control over.

She tricks Bob into calling her malicious function, which then executes the `withdraw()` functions of all the wallets Bob owns.

Since ownership is based on `tx.origin`, all the `withdraw()` calls succeed, and the vested tokens are transferred to an address controlled by Alice.

Recommendation

We recommend replacing `tx.origin` in `onlyOwner` with `msg.sender` for more secure ownership verification and adding appropriate access controls to the `withdraw()` functions to ensure only authorized entities can withdraw tokens.

The appropriate access controls should also be applied to the `switchPause()` function.

Alleviation

[Style team, 2024/05/17]: A phishing scenario assumes the owner of the vesting contract will interact with a fake contract that will execute a transaction. However, this case will never happen as the Wallet that will interact with the vesting contract is

only one wallet (controlled by BOD and DAO) and it will never interact with any other contract.

In our security measures. Contracts are only created from the functional wallets of distribution based on our own tech. The wallet is not in use operationally.

Interaction is only with contracts that are related to each wallet. For example, the Vesting contract is deployed by the Vesting Allocation Wallet, and the Airdrop contract will be deployed by Ecosystem Airdrop Wallet. Each wallet will not interact with any other contract outside of our own Tech.

Additionally, the unused liquidity on Wallets is Vested.

STY-02 | INITIAL TOKEN DISTRIBUTION

Category	Severity	Location	Status
Centralization	● Major	contracts/STYLE-Token.sol: 11~13, 13	● Acknowledged

Description

All of the `STYLE` tokens are sent to the contract deployer and then some tokens are transferred to several hardcoded addresses. There is no restriction on the initial amount the deployer can mint, this is a centralization risk because the deployer can distribute tokens without obtaining the community's consensus. Any compromise to these addresses may allow a hacker to steal and sell tokens on the market, resulting in severe damage to the project.

Recommendation

It is recommended that the team be transparent regarding the initial token distribution process. The token distribution plan should be published in a public location that the community can access. The team should make efforts to restrict access to the private keys of the deployer account or EOAs. A multi-signature (2/3, 3/5) wallet can be used to prevent a single point of failure due to a private key compromise. Additionally, the team can lock up a portion of tokens, release them with a vesting schedule for long-term success, and deanonymize the project team with a third-party KYC provider to create greater accountability.

Alleviation

[CertiK, 2024/06/04]: During deployment of the `STYLE` token, the total supply of 920 000 000 tokens (100%) has been minted and distributed as follow:

	Address	Amount	Percentage	Type
Seed Round	<u>0xdc5554F864905bdbD183Fb8324dDD1Ed72E8685E</u>	92,000,000	10.00%	EOA
Private Round	<u>0x779bAaFd5afB91Edcdf64146E57df001dE8EF7Ae</u>	59,800,000	6.50%	EOA
Public Round	<u>0x44EC9A93759a283a9019406eA219636225c63F7c</u>	80,040,000	8.70%	EOA
KOL Round	<u>0xBA5505Df24055879d9b31d3C82dDCaC18b84242f</u>	26,680,000	2.90%	EOA

	Address	Amount	Percentage	Type
Team and Advisors	<u>0x82f73083dd932c207549d18cb439D70870B0CB6d</u>	110,400,000	12.00%	EOA
Treasury	<u>0x1638C018FBc136de0265927DeA1c76802f8454b4</u>	128,800,000	14.00%	EOA
Ecosystem	<u>0x36b758a181640C8caCc8039CAa58f464EF302b16</u>	92,000,000	10.00%	EOA
Alpha Claim	<u>0x396F6a18c98BFf86D3850F646F5794F5a18B9b26</u>	92,000,000	10.00%	EOA
Strategic Claim	<u>0x3FB950AD30CA169B498AF20BBDEc29cE16aFD387</u>	92,000,000	10.00%	EOA
Marketing	<u>0xa234a5B2458E5C24C30D401a4a36Df516B68CeB5</u>	66,240,000	7.20%	EOA
Liquidity	<u>0x90f782bDEF80eF43Ce8dDCe6a583838426231882</u>	80,040,000	8.70%	EOA

STL-02 | RISK OF OVERFLOW/UNDERFLOW DUE TO EXTENSIVE USE OF UNCHECKED

Category	Severity	Location	Status
Incorrect Calculation	● Medium	contracts/STYLE-vesting-flattend.sol: 1531~1532, 1549~1550, 1571~1572, 1606~1607, 1815~1816, 1853~1854, 1876~1877, 1895~1896, 1920~1921, 1944~1945	● Acknowledged

Description

The contract makes frequent use of `unchecked` blocks to bypass overflow/underflow checks. However, many of these usages rely on hidden assumptions about the values involved, making the code fragile and prone to potential issues.

Scenario

Here are a few examples:

- In `VestingWallet._release()`, if tokens have been withdrawn and then revested, `_vestedAmount(token, uint48(block.timestamp)) - _erc20Released[token]` could underflow;
- In `VestingWallet._vestedAmount()`, `start_ + duration_` could overflow.
- In `VestingWallet._vestedAmount()`, `IERC20(token).balanceOf(address(this)) + _erc20Released[token]` could overflow.

Recommendation

We recommend:

- Avoid using unchecked blocks unless necessary and their behavior is documented and thoroughly tested;
- Adding explicit checks to ensure values do not underflow or overflow;
- If unchecked blocks are required for performance reasons, document the assumptions that prevent underflow/overflow issues.

Alleviation

[Style team, 2024/05/17]: We had unchecked blocks made primarily to optimize contract performance.

Given the gas-intensive nature of blockchain transactions, especially on networks like Ethereum, it is sometimes necessary to find a balance between absolute security (through rigorous checks for overflows and underflows) and efficient gas usage, which directly impacts transaction costs for users.

STL-08 | CHECK EFFECT INTERACTION PATTERN (OUT-OF-ORDER EVENTS)

Category	Severity	Location	Status
Concurrency	● Minor	contracts/STYLE-vesting-flattend.sol: 1535, 1537, 1553, 1555, 1636, 1638, 1650, 1652, 1860, 1864	● Acknowledged

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

This finding is considered minor because the reentrancy only causes out-of-order events.

External call(s)

```
1553 IERC20(token).safeTransfer(_beneficiary, releasable);
```

- This function call executes the following external call(s).
- In `SafeERC20._callOptionalReturn`,
 - `returndata = address(token).functionCall(data)`
- In `Address.functionCallWithValue`,
 - `(success, returndata) = target.call{value: value}(data)`

Events emitted after the call(s)

```
1555 emit ERC20Release(token, releasable);
```

External call(s)

```
1636 Address.sendValue(payable(to), amount);
```

- This function call executes the following external call(s).
- In `Address.sendValue`,

- `(success, None) = recipient.call{value: amount}("")`

Events emitted after the call(s)

```
1638      emit EtherWithdraw(amount);
```

External call(s)

```
1650      IERC20(token).safeTransfer(to, amount);
```

- This function call executes the following external call(s).
- In `SafeERC20._callOptionalReturn`,
 - `returndata = address(token).functionCall(data)`
- In `Address.functionCallWithValue`,
 - `(success, returndata) = target.call{value: value}(data)`

Events emitted after the call(s)

```
1652      emit ERC20Withdraw(token, amount);
```

External call(s)

```
1860      IVestingWallet(clone).initialize(beneficiary, start, duration);
```

Events emitted after the call(s)

```
1864      emit CreateVesting(clone);
```

Recommendation

We recommend using the [Checks-Effects-Interactions Pattern](#) to avoid the risk of calling unknown contracts. Here placing the external call at the end of the function would resolve this finding.

Alleviation

[Style team, 2024/05/17]: The potential reentrancy in our contract does not compromise the integrity of transactions but may cause out-of-order events.

Given this limited impact and the controlled nature of the contract interactions, we assess the risk as manageable within our operational framework and each transaction that could potentially involve a reentrancy attack is subject to oversight and approval by our Decentralized Autonomous Organization (DAO).

STL-09 | CHECK EFFECT INTERACTION PATTERN VIOLATED (INCREMENTING STATE)

Category	Severity	Location	Status
Concurrency	● Minor	contracts/STYLE-vesting-flattend.sol: 1860, 1861, 1862	● Acknowledged

Description

A reentrancy attack can occur when the contract creates a function that makes an external call to another untrusted contract before resolving any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the external call resolved the effects.

This finding is considered minor because the state variable is only incremented or decremented. So, the effect of out-of-order increments may be unobservable after transaction. However, the reentrancy vulnerability may still cause other issues in the middle of transaction.

External call(s)

```
1860         IVestingWallet(clone).initialize(beneficiary, start, duration);
```

State variables written after the call(s)

```
1861         _contracts[msg.sender][_vestingcounts[msg.sender]] = clone;
```

```
1862         ++_vestingcounts[msg.sender];
```

Recommendation

We recommend using the [Checks-Effects-Interactions Pattern](#):

here the following code snippet:

```
1857         address payable clone = payable(  
1858             Clones.clone(_vestingWalletImplementation)  
1859         );  
1860         IVestingWallet(clone).initialize(beneficiary, start, duration);  
1861         _contracts[msg.sender][_vestingcounts[msg.sender]] = clone;  
1862         ++_vestingcounts[msg.sender];  
1863  
1864         emit CreateVesting(clone);
```

should be rewritten

```
address payable clone = payable(
    Clones.clone(_vestingWalletImplementation)
);

_contracts[msg.sender][_vestingcounts[msg.sender]] = clone;
++_vestingcounts[msg.sender];
emit CreateVesting(clone);

IVestingWallet(clone).initialize(beneficiary, start, duration);
```

I Alleviation

[Style team, 2024/05/17]: The potential reentrancy in our contract does not compromise the integrity of transactions and can only cause out-of-order events.

STL-10 | MISSING INPUT VALIDATION ON `VestingWallet.initialize()`

Category	Severity	Location	Status
Volatile Code	● Minor	contracts/STYLE-vesting-flattend.sol: 1461~1462	● Acknowledged

Description

In `VestingWallet.initialize()`, no check is made on the `_beneficiary` address.

Scenario

If `_beneficiary == address(0)` then the vested `ETH` could be lost.

If `_beneficiary == address(this)` then all the vested tokens might get stuck in the contract.

Recommendation

We recommend adding checks to prevent any issue with the `_beneficiary` address.

Alleviation

[Style team, 2024/05/17]: We have strict operational controls in place that ensure the accuracy and validity of `_beneficiary` addresses before they are inputted into the system.

These controls are part of our standard security procedures, which involve multiple checks and balances during the initialization phase.

STL-11 | UNUSED INTERFACE

Category	Severity	Location	Status
Coding Issue	● Informational	contracts/STYLE-vesting-flattend.sol: 1127	● Acknowledged

Description

The smart contract contains one or more interface definitions that are not used, which can lead to unnecessary complexity and reduced maintainability.

```
1127 interface IERC20Permit {
```

- `IERC20Permit` is declared but never used.

Recommendation

It is advised to ensure that all necessary interfaces are used, remove redundant interfaces.

Alleviation

[Style team, 2024/05/17]: It does not compromise the integrity of the contract.

STY-01 | DEPRECATE ERC777 IMPLEMENTATION

Category	Severity	Location	Status
Design Issue	● Informational	contracts/STYLE-Token.sol: 7-8	● Acknowledged

Description

The ERC777 standard is not recommended due to concerns regarding over-engineering, potential security vulnerabilities, and unnecessary complexity.

Recommendation

We recommend using ERC20 standard as explained [here](#).

Alleviation

[Style team, 2024/05/17]: While ERC777 may introduce additional complexity, it also aligns with broader trends in the blockchain ecosystem toward interoperability and standardization. By adopting ERC777, we position our contracts to seamlessly integrate with future DEXs and other platforms that may leverage its capabilities such as operators.

OPTIMIZATIONS | STYLE PROTOCOL

ID	Title	Category	Severity	Status
<u>STL-01</u>	<code>_owner</code> Unused In <code>VestingWalletHolder</code>	Volatile Code	Optimization	● Acknowledged

STL-01 | `_owner` UNUSED IN `VestingWalletHolder`

Category	Severity	Location	Status
Volatile Code	● Optimization	contracts/STYLE-vesting-flattend.sol: 1793~1794, 1798~1804, 1845~1846	● Acknowledged

Description

In the contract `VestingWalletHolder`, a state variable `_owner` is defined:

```
1793     address private immutable _owner;
```

it is set during the deployment

```
1843     constructor() {
1844         _vestingWalletImplementation = payable(address(new VestingWallet()));
1845         _owner = msg.sender;
1846     }
```

And it is never used even if an `onlyOwner` modifier is defined:

```
1798     modifier onlyOwner() {
1799         if (_owner == msg.sender) {
1800             _;
1801         } else {
1802             revert ImproperOwner();
1803         }
1804     }
```

Recommendation

We recommend either:

- removing the code snippets above if they are not meant to be used;
- using the `onlyOwner` in the relevant functions and adding a mechanism to transfer or renounce ownership to allow a centralization risk mitigation strategy, if ownership is meant to be used.

Alleviation

[Style team, 2024/05/17]: It does not compromise the integrity of the contract. It was added to limit the contract to only one owner who can create vesting.

FORMAL VERIFICATION | STYLE PROTOCOL

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied formal verification to prove that important functions in the smart contracts adhere to their expected behaviors.

Considered Functions And Scope

In the following, we provide a description of the properties that have been used in this audit. They are grouped according to the type of contract they apply to.

Verification of Standard Pausable Properties

We verified *partial* properties of the public interfaces of those token contracts that implement the Pausable interface. This involves:

- function `paused` that returns the if the contract is paused,
- function `pause` that pauses the contract, and
- function `unpause` that unpauses the contract.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
pausable-paused-succeed-normal	<code>paused</code> Function Always Succeeds
pausable-pause-correct	Function <code>pause</code> Always Pauses
pausable-unpause-correct	Function <code>unpause</code> Always Unpauses

Verification Results

For the following contracts, formal verification established that each of the properties that were in scope of this audit (see scope) are valid:

Detailed Results For Contract VestingWallet (contracts/STYLE-vesting-flattend.sol) In Commit 0391d9b1cbc103a3ac8ebd0c1dbd7610529786ae

Verification of Standard Pausable Properties

Detailed Results for Function `paused`

Property Name	Final Result	Remarks
pausable-paused-succeed-normal	● True	

Detailed Results for Function `pause`

Property Name	Final Result	Remarks
pausable-pause-correct	● True	

Detailed Results for Function `unpause`

Property Name	Final Result	Remarks
pausable-unpause-correct	● True	

APPENDIX | STYLE PROTOCOL

Finding Categories

Categories	Description
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Incorrect Calculation	Incorrect Calculation findings are about issues in numeric computation such as rounding errors, overflows, out-of-bounds and any computation that is not intended.
Concurrency	Concurrency findings are about issues that cause unexpected or unsafe interleaving of code executions.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Details on Formal Verification

Some Solidity smart contracts from this project have been formally verified. Each such contract was compiled into a mathematical model that reflects all its possible behaviors with respect to the property. The model takes into account the semantics of the Solidity instructions found in the contract. All verification results that we report are based on that model.

The following assumptions and simplifications apply to our model:

- Certain low-level calls and inline assembly are not supported and may lead to a contract not being formally verified.
- We model the semantics of the Solidity source code and not the semantics of the EVM bytecode in a compiled contract.

Formalism for property specifications

All properties are expressed in a behavioral interface specification language that CertiK has developed for Solidity, which allows us to specify the behavior of each function in terms of the contract state and its parameters and return values, as well as contract properties that are maintained by every observable state transition. Observable state transitions occur when the contract's external interface is invoked and the invocation does not revert, and when the contract's Ether balance is changed by the EVM due to another contract's "self-destruct" invocation. The specification language has the usual Boolean connectives, as well as the operator `\old` (used to denote the state of a variable before a state transition), and several types of specification clause:

Apart from the Boolean connectives and the modal operators "always" (written `[]`) and "eventually" (written `<>`), we use the following predicates to reason about the validity of atomic propositions. They are evaluated on the contract's state whenever a discrete time step occurs:

- `requires [cond]` - the condition `cond`, which refers to a function's parameters, return values, and contract state variables, must hold when a function is invoked in order for it to exhibit a specified behavior.
- `ensures [cond]` - the condition `cond`, which refers to a function's parameters, return values, and both `\old` and current contract state variables, is guaranteed to hold when a function returns if the corresponding requires condition held when it was invoked.
- `invariant [cond]` - the condition `cond`, which refers only to contract state variables, is guaranteed to hold at every observable contract state.
- `constraint [cond]` - the condition `cond`, which refers to both `\old` and current contract state variables, is guaranteed to hold at every observable contract state except for the initial state after construction (because there is no previous state); constraints are used to restrict how contract state can change over time.

Description of the Analyzed Standard Pausable Properties Properties

Properties related to function `paused`

`pausable-paused-succeed-normal`

The `paused` function must always succeed, assuming that its execution does not run out of gas.

Specification:

```
reverts_only_when false;
```

Properties related to function `pause`

`pausable-pause-correct`

All non-reverting invocations of `pause()` must pause the contract.

Specification:

```
ensures this.paused();
```

Properties related to function `unpause`

pausable-unpause-correct

All non-reverting invocations of `unpause()` must unpause the contract.

Specification:

```
ensures !this.paused();
```

DISCLAIMER | CERTIK

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 Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. 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 CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. 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.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.

