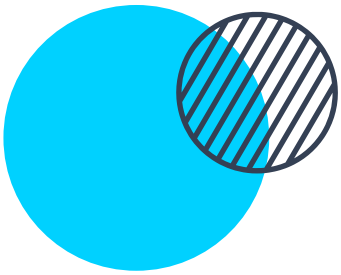
Veridise

Auditing Report

Hardening Blockchain Security with Formal Methods

FOR



onthis

Arbitrum Constructor Contract

Veridise Inc.

June 24, 2024

► **Prepared For:**

OnThis
https://www.onthis.xyz

► **Prepared By:**

Evgeniy Shishkin
Alberto Gonzalez

► **Contact Us:**

contact@veridise.com

► **Version History:**

Jun. 12, 2024	Initial Draft
Jun. 14, 2024	Official Report.
Jun. 24, 2024	Official Report V2

Contents

1 Executive Summary

From May 27, 2024 to May 28, 2024, OnThis engaged Veridise to review the security of their Arbitrum Constructor Contract. The review covered a set of smart contracts designed for launching token distribution events for early-time investors. Veridise conducted the assessment over 4 person-days, with 2 security analysts reviewing code over 2 days on commit `cf59b57`. The security assessment strategy included an extensive manual review of the code base.

Project summary. Arbitrum Constructor Contract is a no-code platform that helps Arbitrum projects launch their ERC20 tokens with minimal effort. Token launching consists of three phases:

1. **Initialization**. The project (token owner) decides on how many tokens they want to have in circulation called the *total supply*, and the type and duration of the Pre-sale Phase.
2. **Pre-sale Phase.** During this phase, early time investors and the project owner are able to invest their funds and receive a respective share of the Pre-Sale tokens.
3. **Circulation Phase.** After the pre-sale, the remaining tokens together with accumulated funds form a trading pair on Camelot DEX. From this point, anyone can sell and buy the tokens on the DEX for the market price.

More specifically, the whole process works as follows:

1. Token owner specifies the total number of tokens that will be minted. The token being created is non-mintable, hence the total supply will never change.
2. Token owner specify a pre-sale event type that they would like to run. This can be a time-based event, where the pre-sale has a specific duration, or a threshold-based event, where the pre-sale is only considered over when enough funds have been collected.
3. The token owner specifies the **snipe amount**, which is the amount of Ether they wish to invest on their side. This amount will be used to buy back tokens from the exchange after the pre-sale event has ended.
4. After the pre-sale has started, investors will deposit their funds into the token.
5. After the pre-sale event has ended, the tokens and any accumulated Ether funds will be distributed according to the following rules:
   - 1% of the Ether funds collected from investors go to the token owner.
   - 49% of the minted tokens are distributed among all investors, proportionally to their deposited amounts.
   - 2% of the minted tokens go to the token owner.
   - Accumulated Ether funds (minus the *snipe amount*) together with 49% of minted tokens form a trading pair on the Camelot DEX.
   - The *snipe amount* is used to purchase tokens from Camelot DEX by the token owner.
6. Note the following nuances:
   - All token transfers between addresses are a subject to 0.3% tax.
   - If an investor wants to get back their deposited funds during the pre-sale, 1% fee will be applied.

- The *snipe amount* may be arbitrarily changed by the token owner during the pre-sale.

Code assessment. The Arbitrum Constructor Contract developers provided the source code of the project contracts for review. The source code appears to be mostly original code written by the Arbitrum Constructor Contract developers. To facilitate the Veridise security analysts understanding of the code, the Arbitrum Constructor Contract developers shared the high-level description of the project, containing the most essential parts of the protocol.

The source code contained a test suite of 5 tests covering the most common usage scenarios.

Summary of issues detected. The code review uncovered 13 issues, 4 of which are assessed to be of high or critical severity by the Veridise security analysts. Specifically, two potential issues were identified where a motivated attacker could disrupt the token pre-sale event. The Veridise security analysts also identified 1 low severity issue, 2 warnings and 6 informational findings.

Recommendations. After reviewing the protocol, the security analysts had a few suggestions to improve the Arbitrum Constructor Contract.

- It is suggested to improve the documentation by providing a better explanation of the tokenomics used. For example, it may not be perfectly clear why participating in the pre-sale event is more beneficial than buying tokens on the decentralized exchange (DEX) after the pre-sale has finished. How does the *snipe amount* influence the DEX price of the token afterwards?
- Provide a designated test that users can use to simulate different pre-sale distribution scenarios. This will allow them to see exactly what outcomes they will receive.

Disclaimer. We hope that this report is informative but provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the system is secure in all dimensions. In no event shall Veridise or any of its employees be liable for any claim, damages or other liability, whether in an action of contract, tort or otherwise, arising from, out of or in connection with the results reported here.

2 Project Dashboard

Table 2.1: Application Summary.

Name	Version	Type	Platform
Arbitrum Constructor Contract	`cf59b57`	Solidity	Arbitrum

Table 2.2: Engagement Summary.

Dates	Method	Consultants Engaged	Level of Effort
May 27 - May 28, 2024	Manual	2	4 person-days

Table 2.3: Vulnerability Summary.

Name	Number	Fixed	Acknowledged
Critical-Severity Issues	0	0	0
High-Severity Issues	4	4	4
Medium-Severity Issues	0	0	0
Low-Severity Issues	1	1	1
Warning-Severity Issues	2	2	2
Informational-Severity Issues	6	6	6
TOTAL	13	13	13

Table 2.4: Category Breakdown.

Name	Number
Maintainability	5
Logic Error	2
Denial of Service	2
Usability Issue	2
Flashloan	1
Missing/Incorrect Events	1

3 Audit Goals and Scope

3.1 Audit Goals

The engagement was scoped to provide a security assessment of Arbitrum Constructor Contract program code. In our audit, we sought to answer questions such as:

- Is there any way for a malicious actor to disrupt or interfere with the pre-sale event?
- Is the amount of tokens nominated to investors after the pre-sale is over calculated correctly?
- Is there any way to prevent investors from receiving their nominated tokens after the pre-sale period has ended?
- Are access modifiers in place and correct?
- Are functions input validation strong enough?
- Are there any common Solidity implementation flaws, such as reentrancy, out-of-gas attacks, or others?
- Does the program implementation reflects the higher-level business intent?

3.2 Audit Methodology & Scope

Audit Methodology. To address the questions raised, a thorough review of the source code has been carried out. When in doubt about the code's intent, auditors refer to the logic described in the project documentation.

Scope. The scope of this audit is limited to the *contracts* folder of the source code provided by the Arbitrum Constructor Contract developers, which contains the smart contract implementation of the Arbitrum Constructor Contract.

3.3 Classification of Vulnerabilities

When Veridise auditors discover a possible security vulnerability, they must estimate its severity by weighing its potential impact against the likelihood that a problem will arise. Table 3.1 shows how our auditors weigh this information to estimate the severity of a given issue.

Table 3.1: Severity Breakdown.

	Somewhat Bad	Bad	Very Bad	Protocol Breaking
Not Likely	Info	Warning	Low	Medium
Likely	Warning	Low	Medium	High
Very Likely	Low	Medium	High	Critical

In this case, we judge the likelihood of a vulnerability as follows in Table 3.2:

In addition, we judge the impact of a vulnerability as follows in Table 3.3:

Table 3.2: Likelihood Breakdown

Not Likely	A small set of users must make a specific mistake
Likely	Requires a complex series of steps by almost any user(s) - OR - Requires a small set of users to perform an action
Very Likely	Can be easily performed by almost anyone

Table 3.3: Impact Breakdown

Somewhat Bad	Inconveniences a small number of users and can be fixed by the user
Bad	Affects a large number of people and can be fixed by the user - OR - Affects a very small number of people and requires aid to fix
Very Bad	Affects a large number of people and requires aid to fix - OR - Disrupts the intended behavior of the protocol for a small group of users through no fault of their own
Protocol Breaking	Disrupts the intended behavior of the protocol for a large group of users through no fault of their own

4 Vulnerability Report

In this section, we describe the vulnerabilities found during our audit. For each issue found, we log the type of the issue, its severity, location in the code base, and its current status (i.e., acknowledged, fixed, etc.). Table 4.1 summarizes the issues discovered:

Table 4.1: Summary of Discovered Vulnerabilities.

ID	Description	Severity	Status
V-ACC-VUL-001	Incorrect share amount calculation in the thres...	High	Fixed
V-ACC-VUL-002	Time-based token presale DoS by making max_pai	High	Fixed
V-ACC-VUL-003	Threshold-based token presale DoS due to unboun	High	Fixed
V-ACC-VUL-004	Snipe amount may be atomically arbitraged via f..	High	Fixed
V-ACC-VUL-005	tx.origin can lead to incorrectly legitimize to...	Low	Fixed
V-ACC-VUL-006	Variables' values sanitization best practices	Warning	Fixed
V-ACC-VUL-007	Arithmetic operations best practices	Warning	Fixed
V-ACC-VUL-008	Misleading and inconsistent error messages	Info	Fixed
V-ACC-VUL-009	Unused and redundant code	Info	Fixed
V-ACC-VUL-010	Tokens' contracts are not factories	Info	Fixed
V-ACC-VUL-011	Tokens' presale contracts lack events emissions	Info	Fixed
V-ACC-VUL-012	Better use a two step ownership transfer	Info	Fixed
V-ACC-VUL-013	Disable initializers in the constructor for upg...	Info	Fixed

4.1 Detailed Description of Issues

4.1.1 V-ACC-VUL-001: Incorrect share amount calculation in the threshold-based token

Severity	High	**Commit**	cf59b57
Type	Logic Error	**Status**	Fixed
File(s)	`ThresholdBasedERC20WithTaxFactory.sol`		
Location(s)	getUserShare		
Confirmed Fix At	f3a66a8		

In the `TimeBasedERC20PresaleWithTax` contract, user shares are calculated by considering the contract's balance minus `snipeAmount`. This approach ensures that the snipe amount does not influence the distribution of shares among depositors since `snipeAmount` is contributed by the owner of the presale.

```
function getUserShare(uint256 ethAmount) public view returns (uint256) {
    uint256 tokensToDistribute = (ethAmount * presaleShare) /
     (address(this).balance - snipeAmount);

  return tokensToDistribute;
}
```

Snippet 4.1: Snippet from `TimeBasedERC20PresaleWithTax`.

However, in the `ThresholdBasedERC20WithTaxFactory` contract, the user share calculation does not account for the snipe amount, leading to an incorrect share distribution:

```
function getUserShare(uint256 ethAmount) public view returns (uint256) {
    uint256 tokensToDistribute = (ethAmount * presaleShare) /
     address(this).balance;

  return tokensToDistribute;
}
```

Snippet 4.2: Snippet from `ThresholdBasedERC20WithTaxFactory`.

Impact The users' contributions to the threshold-based presale are calculated inaccurately. Specifically, these contributors may receive fewer shares than intended, as the total balance used in the calculation may include the `snipeAmount` sent by the owner.

Recommendation It is recommended to make the share amount calculation the same as in the`TimeBasedERC20PresaleWithTax` contract.

Developer Response Threshold pre-sale was removed.

4.1.2 V-ACC-VUL-002: Time-based token presale DoS by making max_participant donations

Severity	High	**Commit**	cf59b57
Type	Denial of Service	**Status**	Fixed
File(s)	`TimeBasedERC20WithTaxFactory.sol`		
Location(s)	receive()		
Confirmed Fix At	f3a66a8		

During the time-based token presale, users participate by sending ether to the contract, triggering the `receive` function. This function checks to ensure the number of unique participants does not surpass `maxParticipants` to avoid gas exhaustion during the `airdropAll` function, which iterates over all participants.

However, this setup inadvertently introduces a potential attack vector. An attacker could effectively shut down the presale by sending `1 wei` from `maxParticipants` different accounts. With `maxParticipants` set at 801, this allows for 800 different accounts. The cost for executing 800 transactions could be as low as $300, assuming the average transaction cost is $0.368, making this a feasible attack.

Impact This vulnerability could prevent genuine users from participating in the presale if an attacker exploits this weakness by saturating the list of participants with minimal contributions.

Recommendation To mitigate this risk, it is advisable to implement a minimum contribution threshold. Setting a reasonable minimum `msg.value` for deposits will deter attackers by making it economically unfeasible to fill up the participant slots with trivial amounts.

Developer Response Recommendation was implemented.

4.1.3 V-ACC-VUL-003: Threshold-based token presale DoS due to unbounded amount of participants

Severity	High	**Commit**	cf59b57
Type	Denial of Service	**Status**	Fixed
File(s)	`ThresholdBasedERC20WithTaxFactory.sol`		
Location(s)	receive()		
Confirmed Fix At	f3a66a8		

In the threshold-based token system, there is currently no `max_participants` check within the `receive()` function. This omission leaves the system vulnerable to an attack where a malicious party could make numerous minimal deposits, for example 1 wei, from a large number of different accounts. Such a strategy could exhaust gas limits when executing the `_airdropAll()` function, which is designed to distribute tokens among all participants at the end of the pre-sale.

Impact If exploited, this vulnerability could prevent the successful completion of the token pre-sale phase. As the `_airdropAll()` function fails due to gas exhaustion, participants would not receive their tokens as expected. While users could still withdraw their funds, they would incur a 1% fee, leading to financial loss and potential damage to the credibility of the project.

Recommendation To mitigate this risk, it is advisable to implement a maximum number of participants to prevent gas exhaustion issues on the selected deployment chain. Additionally, implementing a minimum contribution threshold is important to avoid the issues outlined in V-ACC-VUL-002.

Developer Response Threshold pre-sale was removed.

4.1.4 V-ACC-VUL-004: Snipe amount may be atomically arbitraged via flashloan

Severity	High	**Commit**	cf59b57
Type	Flashloan	**Status**	Fixed
File(s)	`ThresholdBasedERC20WithTaxFactory.sol`, `TimeBasedERC20WithTaxFactory.sol`		
Location(s)			
Confirmed Fix At	f3a66a8		

After the pre-sale event concludes, the pre-sale contract executes the following actions:

1. Distributes 49% of the initial supply among investors based on their eth contribution.
2. Deploys the trading pair WETH-TOKEN on the Camelot DEX with the following initial state:
   - It sends all the investor WETH accumulated by the pre-sale contract.
   - It allocates 49% of the initial supply for the TOKEN amount.
3. Purchases TOKEN on behalf of the token owner by exchanging the `snipeAmount` locked within the pre-sale contract.

Here, it is important to consider that in step 3, the owner is essentially exchanging WETH for tokens, which drives up the price of tokens relative to WETH. Since the new price will be greater than the initial one, the investors can now sell an amount of tokens at a profit until the price is driven down to the original price. This profit is at expense of the first buyer which is the owner of the presale.

Due to the nature of a Constant Product AMM such as Camelot, the profit leaked by the owner (which is also a loss for the owner) during the first purchase is proportional to the amount of 'snipeAmount' relative to the initial WETH reserves in the pool. This is because trades are not executed at the theoretical price of the pool but at the execution price. See how they are computed in the context of the current contracts:

- Theoretical price, the price computed given the current reserves, is computed as:
  - `wethReserves / tokensReserve`
- Execution price, the real price at which an order is executed, is computed as:
  - `(wethReserves + snipeAmount) / tokenReserves`

When `snipeAmount` is small relative to `wethReserves`, the execution price starts becoming close to the theoretical price, but as `snipeAmount` grows the execution price starts becoming bigger that the theoretical price.

The issue is that the current logic does not limit the value of the `snipeAmount` relative to the expected initial reserves of WETH leaving the possibility for unaware presale creators to shoot themselves in the foot.

Impact How much profit is leaked to the investors at the expense of the owner depends on the ratio of `snipeAmount` and the initial WETH reserves. We provide a specific scenario where the last investor nets a 35% profit risk free at the expense of the owner of the presale. The parametrized PoC is available here.

It is also important to consider, that in the case of the Threshold pre-sale, the last investor can perform the trade atomically via a flashloan netting a risk-free profit.

Recommendation To reduce the value leaked at the expense of the owner, it is important to enforce a limit on how much the `snipeAmount` can grow relative to the expected initial reserves. This can be computed using the values of the theoretical price and execution price and enforcing a minimum ratio between them.

Developer Response The flash-loan attack vector was mitigated by avoiding to enter and end the pre-sale in the same block. The risk of having a big `snipeAmount` relative to the reserves is accepted.

4.1.5 V-ACC-VUL-005: tx.origin can lead to incorrectly legitimize token presales

Severity	Low	**Commit**	cf59b57
Type	Usability Issue	**Status**	Fixed
File(s)	`TokenLauncher.sol`		
Location(s)	launchTokenPresale		
Confirmed Fix At	f3a66a8		

Using `tx.origin` in the `launchTokenPresale` function exposes the contract to phishing attacks. These attacks could occur when a well-known or reputable account is tricked into interacting with a malicious contract that calls `launchTokenPresale` deploying a new token presale.

Impact This vulnerability could mislead potential investors by falsely associating the presale with a trusted party, thereby enhancing the credibility of a potentially fraudulent offering. Even though the malicious party will not own the owner token share, legitimate users might still purchase the token via the chosen Automated Market Maker (AMM).

Recommendation Replace the use of `tx.origin` with `msg.sender` within the `launchTokenPresale` function to ensure that the direct caller of the function is recorded as the creator and the owner of the presale. Additionally, modify the token contract's constructor to include an explicit `creator` parameter to be saved into the `initialCreator` variable.

Developer Response Recommendation was implemented.

4.1.6 V-ACC-VUL-006: Variables' values sanitization best practices

Severity	Warning	**Commit**	cf59b57
Type	Maintainability	**Status**	Fixed
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	f3a66a8		

Sanitizing variables' values is considered good practice to prevent users from using meaningless values that could lead to issues ranging from simple re-execution to more severe consequences due to overlooked edge cases. Several variables have been identified as lacking necessary validations:

- In the token launcher:
  - `LaunchParams params.name` and `params.symbol` should be validated to ensure they are not empty strings.
  - `LaunchParams params.totalSupply` should be validated to fall within specific lower and upper bounds.
  - The variables `presaleSecondDuration` and `thresholdGoal` have upper bounds checks but lack lower bound checks.
- In the tokens' presale contracts:
  - Validate in the constructor that the sum of `LP_SHARE`, `PRESALE_SHARE` and `OWNER_SHARE` is equal to `100`.

Impact To the extent of the knowledge of Veridise auditors, the lack of these validations does not pose a direct security impact in the current codebase.

Recommendation It is recommended to implement meaningful lower and upper bound validation checks (aligned with the business logic) for the aforementioned inputs.

Developer Response Recommendations were implemented.

4.1.7 V-ACC-VUL-007: Arithmetic operations best practices

Severity	Warning	**Commit**	cf59b57
Type	Logic Error	**Status**	Fixed
File(s)	`TimeBasedERC20WithTaxFactory.sol`, `ThresholdBasedERC20WithTaxFactory.sol`		
Location(s)	constructor, airdropAll, _transfer		
Confirmed Fix At	f3a66a8		

Throughout the code, there are several places where division operation occurs before multiplication.

```
1 uint256 tax = (amount / 1000) * 3;
```

Snippet 4.3: Snippet from `_transfer()`

Impact This order of operation introduce rounding errors. For example, in case the `amount` equals 900, the `tax` would be `0` , since the division operation rounds down in Solidity. If we change the formula to `uint256 tax = (amount * 3) / 1000` , the result would be `2`.

Recommendation While this does not seem to be a significant issue in the current codebase, as there are some rounding errors that occur due to this order, they appear to be negligible. However, it would be advisable to change the order so that multiplication comes before division. This would make rounding errors less likely in case the codebase continues to evolve.

Developer Response Recommendations were implemented.

4.1.8 V-ACC-VUL-008: Misleading and inconsistent error messages

Severity	Info	**Commit**	cf59b57
Type	Usability Issue	**Status**	Fixed
File(s)	`TimeBasedERC20WithTaxFactory.sol` , `ThresholdBasedERC20WithTaxFactory.sol`		
Location(s)	See description		
Confirmed Fix At	f3a66a8		

There are several places in the code with inaccurate error messages put into `require` checks.

- In `TimeBasedERC20PresaleWithTax::isAirdropTs` modifier, the require error message says "TimeBasedERC20PresaleWithTax: presale !started", but should be "TimeBasedERC20PresaleWithTax: presale has not ended"
- In `ThresholdBasedERC20WithTaxFactory::withdrawNative` function, the require error message says "ThresholdBasedERC20WithTaxFactory: amount < 0" , but should be "ThresholdBasedERC20WithTaxFactory: amount == 0"
- The error message for the`isPresaleActive` modifier is different among tokens.

Impact Users and developers may be misled by the inaccurate error messages. Usability of contracts may suffer.

Recommendation It is recommended that messages be made clear and consistent across both tokens.

Developer Response Recommendation was implemented.

4.1.9 V-ACC-VUL-009: Unused and redundant code

Severity	Info	**Commit**	cf59b57
Type	Maintainability	**Status**	Fixed
File(s)	See description		
Location(s)	See description		
Confirmed Fix At	f3a66a8		

Throughout the code, some unused code is present. It is recommended to remove this code from the code-base as it can reduce the cognitive load on the developer.

In `ThresholdBasedERC20WithTaxFactory.sol`, the following code has been identified as unused:

- `import "hardhat/console.sol";` at line 12
- `console.log(lastParticipantId);` at line 152
- `LP_SHARE` constant at line 20

In `TimeBasedERC20WithTaxFactory.sol` , the following code has been identified as unused:

- `LP_SHARE` constant at line 18

In `TokenLauncher.sol`, the following code has been identified as unused or redundant:

- `import "./struct/Structs.sol";` at line 6. The struct defined in the file is not used.

Impact Unused or redundant code clutters the code base and increases a cognitive load on developers.

Recommendation It is advised to remove used code, and refactor the redundant code.

Developer Response Recommendation was implemented.

4.1.10 V-ACC-VUL-010: Tokens' contracts are not factories

Severity	Info	**Commit**	cf59b57
Type	Maintainability	**Status**	Fixed
File(s)	`TimeBasedERC20WithTaxFactory.sol` , `ThresholdBasedERC20WithTaxFactory.sol`		
Location(s)			
Confirmed Fix At	f3a66a8		

The project contain files `TimeBasedERC20WithTaxFactory.sol`,`ThresholdBasedERC20WithTaxFactory.sol` . Those file names suggest that they contain factories of contracts. However, in fact, both files contain parametrized ERC20-compatible token contracts, not factories. As for the factory, the `TokenLauncher.sol` is a typical example of a factory.

Impact Misleading file names may hinder the developers understanding of the code base and business intent.

Recommendation Rename files into something more relevant. For example, `TimeBasedERC20WithTaxToken.sol` and `ThresholdBasedERC20WithTaxToken.sol`

Developer Response Recommendation was implemented.

4.1.11 V-ACC-VUL-011: Tokens' presale contracts lack events emissions

Severity	Info	**Commit**	cf59b57
Type	Missing/Incorrect Eve	**Status**	Fixed
File(s)	`TimeBasedERC20WithTaxFactory.sol`, `ThresholdBasedERC20WithTaxFactory.sol`		
Location(s)	See description		
Confirmed Fix At	f3a66a8		

Upon a state update, it is strongly recommended that developers emit an event to indicate that a change has been made. This allows both external users and protocol administrators to monitor the protocol for various reasons, including potentially suspicious activity. Therefore, it is critical for significant changes to the protocol to be accompanied by events in order to enable this monitoring. However, neither Tokens nor the Token Launcher emit any events.

Impact Functions executed on tokens and the Token Launcher will stay unnoticed. External users and protocol administrators may miss a potentially suspicious activity.

Recommendation It is recommended to add corresponding events for contract methods.

Developer Response Recommendation was implemented.

4.1.12 V-ACC-VUL-012: Better use a two step ownership transfer

Severity	Info	**Commit**	cf59b57
Type	Maintainability	**Status**	Fixed
File(s)	`TimeBasedERC20WithTaxFactory.sol`, `ThresholdBasedERC20WithTaxFactory.sol`, `TokenLauncher.sol`		
Location(s)	See description		
Confirmed Fix At	f3a66a8		

The contract `TokenLauncherArbitrum` inherits the `OwnableUpgradeable` contract from OpenZeppelin library to implement the ownership transfer functionality. The 1-step ownership is considered unsafe, considering the significance of this step.

Impact Specifying a wrong address for the ownership transfer will make `rescueFunds` function unavailable.

Recommendation It is recommended to use an upgradable pattern with a two step ownership transfer, implemented with `Ownable2StepUpgradeable` contract from the OpenZeppelin library.

Developer Response Recommendation was implemented.

4.1.13 V-ACC-VUL-013: Disable initializers in the constructor for upgradeable contracts

Severity	Info	**Commit**	cf59b57
Type	Maintainability	**Status**	Fixed
File(s)	`TokenLauncher.sol`		
Location(s)	constructor()		
Confirmed Fix At	f3a66a8		

In upgradable smart contracts utilizing the proxy pattern one important aspect is to ensure that the implementation contract is properly initialized. To prevent the implementation contract from being initialized by an attacker, the `_disableInitializers` function should be invoked in the constructor to automatically lock it when it is deployed.

Impact If an implementation contract is left uninitialized, it could potentially be taken over by an attacker. This security vulnerability could then impact the proxy and, by extension, the entire system, as attackers could exploit the open initialization function.

To the extent of the knowledge of Veridise auditors, this issue does not pose a direct security impact in the current codebase. However, they recommend implementing the suggested fixes to prevent potential issues in future iterations of the code.

Recommendation Call `_disableInitializers` in the constructor:

```
constructor() {
    _disableInitializers();
}
```

Snippet 4.4: Example of calling `_disableInitializers` in the constructor.

Developer Response Recommendation was implemented.

