Audit Report

Cypher Autoload Simple

v1.0

August 8, 2024

Table of Contents

License

Disclaimer

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED “AS IS”, WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND.

THE AUTHOR AND HIS EMPLOYER DISCLAIM ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT.

THIS AUDIT REPORT WAS PREPARED EXCLUSIVELY FOR AND IN THE INTEREST OF THE CLIENT AND SHALL NOT CONSTRUE ANY LEGAL RELATIONSHIP TOWARDS THIRD PARTIES. IN PARTICULAR, THE AUTHOR AND HIS EMPLOYER UNDERTAKE NO LIABILITY OR RESPONSIBILITY TOWARDS THIRD PARTIES AND PROVIDE NO WARRANTIES REGARDING THE FACTUAL ACCURACY OR COMPLETENESS OF THE AUDIT REPORT.

FOR THE AVOIDANCE OF DOUBT, NOTHING CONTAINED IN THIS AUDIT REPORT SHALL BE CONSTRUED TO IMPOSE ADDITIONAL OBLIGATIONS ON COMPANY, INCLUDING WITHOUT LIMITATION WARRANTIES OR LIABILITIES.

This audit has been performed by

Oak Security GmbH

https://oaksecurity.io/
info@oaksecurity.io

Introduction

Purpose of This Report

Oak Security GmbH has been engaged by CypherD Wallet Inc to perform a security audit of Cypher Autoload Simple.

The objectives of the audit are as follows:

1. Determine the correct functioning of the protocol, in accordance with the project specification.
2. Determine possible vulnerabilities, which could be exploited by an attacker.
3. Determine smart contract bugs, which might lead to unexpected behavior.
4. Analyze whether best practices have been applied during development.
5. Make recommendations to improve code safety and readability.

This report represents a summary of the findings.

As with any code audit, there is a limit to which vulnerabilities can be found, and unexpected execution paths may still be possible. The author of this report does not guarantee complete coverage (see disclaimer).

Codebase Submitted for the Audit

The audit has been performed on the following target:

Repository	https://github.com/CypherD-IO/Contract-Solid-Cyd/tree/main
Commit	`51d18c7928f44757571c35e541748889e6ab7613`
Scope	The scope of this audit was restricted to the contract in `contracts/ERC20_AUTOLOAD_SIMPLE.sol`
Fixes verified at commit	`d6bae11153bab8c66d3f480a9220f828eba1e12b` Note that only fixes to the issues described in this report have been reviewed at this commit. Any further changes such as additional features have not been reviewed.

Methodology

The audit has been performed in the following steps:

1. Gaining an understanding of the code base's intended purpose by reading the available documentation.
2. Automated source code and dependency analysis.
3. Manual line-by-line analysis of the source code for security vulnerabilities and use of best practice guidelines, including but not limited to:
    a. Race condition analysis
    b. Under-/overflow issues
    c. Key management vulnerabilities
4. Report preparation

Functionality Overview

Cypher Autoload Simple is a helper contract for automatically topping up users' credit cards from an approved list of tokens. To do so, users approve the contract and an off-chain executioner (assumed to be trusted) triggers the funding.

How to Read This Report

This report classifies the issues found into the following severity categories:

Severity	Description
Critical	A serious and exploitable vulnerability that can lead to loss of funds, unrecoverable locked funds, or catastrophic denial of service.
Major	A vulnerability or bug that can affect the correct functioning of the system, lead to incorrect states or denial of service.
Minor	A violation of common best practices or incorrect usage of primitives, which may not currently have a major impact on security, but may do so in the future or introduce inefficiencies.
Informational	Comments and recommendations of design decisions or potential optimizations, that are not relevant to security. Their application may improve aspects, such as user experience or readability, but is not strictly necessary. This category may also include opinionated recommendations that the project team might not share.

The status of an issue can be one of the following: **Pending,** **Acknowledged**, or **Resolved**.

Note that audits are an important step to improving the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of the system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**. We include a table with these criteria below.

Note that high complexity or low test coverage does not necessarily equate to a higher risk, although certain bugs are more easily detected in unit testing than in a security audit and vice versa.

Code Quality Criteria

The auditor team assesses the codebase's code quality criteria as follows:

Criteria	Status	Comment
Code complexity	Low	-
Code readability and clarity	High	-
Level of documentation	Medium	No external documentation was available, but the client provided an internal description.
Test coverage	Low	No tests were provided.

Summary of Findings

No	Description	Severity	Status
1	Debit transfers always fail for tokens that return no value on success	Major	Resolved
2	Withdrawal limit can be easily circumvented	Minor	Acknowledged
3	The executioner can debit any approved tokens from the users	Minor	Acknowledged
4	Missing event for critical parameter change	Informational	Resolved
5	Redundant input validations	Informational	Resolved
6	Redundant checks during token transfer	Informational	Resolved
7	Global withdrawal limit does not allow granular control	Informational	Acknowledged
8	The withdrawal limit can be set to zero implicitly pausing the contract's functionality	Informational	Acknowledged
9	Miscellaneous	Informational	Resolved

Detailed Findings

1. Debit transfers always fail for tokens that return no value on success

Severity: Major

The `debit` function allows transferring tokens from a user's account to a beneficiary. The `transferFrom` function is being called inside the following require statement:

```
require(token.transferFrom(userAddress, beneficiaryAddress, amount), "Token transfer failed");
```

This would lead to failing transfers for tokens that return no value on success and revert or throw on failure, like USDT.

Recommendation

We recommend using the `safeTransferFrom` function from the `SafeERC20` library by OpenZeppelin.

Status: Resolved

2. Withdrawal limit can be easily circumvented

Severity: Minor

Whenever the function `debit` is called, the modifier `checkWithdrawalLimit` checks that the requested amount is below the global withdrawal limit. However, this check is only applied per call and can be easily circumvented by splitting larger withdrawals into multiple transactions where each one is below the withdrawal limit.

Recommendation

We recommend considering a different approach for rate limiting, for instance applying a time-based limit per token or address, depending on the exact goals of the limit.

Status: Acknowledged

3. The executioner can debit any approved tokens from the users

Severity: Minor

According to the client's comment, *the main motive of this contract is to auto-top-up users' cards from their approved list of tokens.*

However, the current implementation has no checks against any approved lists. This allows the executioner to debit any tokens from the users that have approved spending.

Recommendation

We recommend implementing an approved list functionality as intended.

Status: Acknowledged

4. Missing event for critical parameter change

Severity: Informational

The `setMaxWithdrawalLimit` function sets a new withdrawal limit, which is a critical change. However, it does not emit an event. Events help off-chain tools to track changes, and hence increase usability.

Recommendation

We recommend adding a `setMaxWithdrawalLimit` event and emitting it at the end of `setMaxWithdrawalLimit` function.

Status: Resolved

5. Redundant input validations

Severity: Informational

In a couple of instances, redundant validations are present:

- The `checkBeneficiaryRole` modifier checks whether the `beneficiaryAddress` is not `address(0)` which can only happen if a privileged admin has granted `BENEFICIARY_ROLE` to `address(0)`. Because this is highly unlikely and a mistake by the admin, the abovementioned check can be removed to save gas.
- The `checkWithdrawalLimit` modifier checks whether the `tokenAddress` argument is not `address(0)`. However, because any `ERC20` call to `address(0)` would revert anyway, this check can be removed to save gas.

Recommendation

We recommend removing unnecessary checks to save gas and improve readability.

Status: Resolved

6. Redundant checks during token transfer

Severity: Informational

The `debit` function allows transferring tokens from a user's account to a beneficiary. However, before the token transfer, the function performs allowance and balance checks which are then checked again by the tokens' contracts. This leads to unnecessary gas spending and reduces overall readability.

Recommendation

We recommend removing allowance and balance checks in the debit function.

Status: Resolved

7. Global withdrawal limit does not allow granular control

Severity: Informational

The contract has one global limit `_maxWithdrawalLimit` that can be updated by an admin. Within `checkWithdrawalLimit`, this value is scaled by the token's decimals and compared with the requested amount. This approach has multiple downsides:

- An admin can only set one value, which will have very different monetary values for different tokens. For instance, a value of 10,000 results in a limit of 10,000 USD for USDC, more than 3 million USD for ETH and more than 60 million USD for WBTC.
- Because the value is scaled by the decimals, the smallest possible value is 1 whole unit for every token. For some tokens like WBTC, 1 WBTC may already be too large. Allowing fractional values may be desirable.

Recommendation

We recommend adding a limit per token. We also recommend not to scale limits, but instead storing raw values.

Status: Acknowledged

8. The withdrawal limit can be used to implicitly pause the contract

Severity: Informational

The `setMaxWithdrawalLimit` function allows an admin to set the `_maxWithdrawalLimit` to any value including zero. However, due to the `checkWithdrawalLimit` modifier, setting the withdrawal limit to zero would not allow the users to debit tokens and would lead to the same outcome as pausing the contract via the `pause` function. This may mislead users.

Recommendation

We recommend adding a check for a minimum value of the `newLimit` argument.

Status: Acknowledged

9. Miscellaneous

Severity: Informational

- Use `!= 0` instead of `> 0` for unsigned integer comparison to save gas.
- The modifier name `anyPrivilagedUser` contains a typo, replace it with `anyPrivilegedUser`.

Recommendation

We recommend fixing aforementioned details.

Status: Resolved