BAIL security

BAILSEC.IO

EMAIL : OFFICE@BAILSEC.IO

TWITTER : @BAILSECURITY

TELEGRAM : @HELLOATBAILSEC

FINAL REPORT:

Amphor

September 2023

Disclaimer:

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

The content of this assessment is not an investment. The information provided in this report is for general informational purposes only and is not intended as investment, legal, financial, regulatory, or tax advice. The report is based on a limited review of the materials and documentation provided at the time of the audit, and the audit results may not be complete or identify all possible vulnerabilities or issues. The audit is provided on an "as-is," "where-is," and "as-available" basis, and the use of blockchain technology is subject to unknown risks and flaws.

The audit does not constitute an endorsement of any particular project or team, and we make no warranties, expressed or implied, regarding the accuracy, reliability, completeness, or availability of the report, its content, or any associated services or products. We disclaim all warranties, including the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

We assume no responsibility for any product or service advertised or offered by a third party through the report, any open-source or third-party software, code, libraries, materials, or information linked to, called by, referenced by, or accessible through the report, its content, and the related services and products. We will not be liable for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract.

The contract owner is responsible for making their own decisions based on the audit report and should seek additional professional advice if needed. The audit firm or individual assumes no liability for any loss or damages incurred as a result of the use or reliance on the audit report or the smart contract. The contract owner agrees to indemnify and hold harmless the audit firm or individual from any and all claims, damages, expenses, or liabilities arising from the use or reliance on the audit report or the smart contract.

By engaging in a smart contract audit, the contract owner acknowledges and agrees to the terms of this disclaimer.

1. Project Details

Project	Amphor
Website	https://amphor.io
Type	Vault
Language	Solidity
Methods	Manual Analysis
Github repository	https://github.com/mev-capital/amphor-core/blob/7223e93bf19dd1c9b2212d84785234c8787f2a70/src/AmphorSyntheticVault.sol

2. Detections Overview

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	0			
Medium	0			
Low	0			
Informational	5	1		4
Governance	1			1
Quality assurance	1	1		
Total	7	2		5

2.1 Detections Definitions

Severity	Description
High	The problem poses a significant threat to the confidentiality of a considerable number of users' sensitive data. It also has the potential to cause severe damage to the client's reputation or result in substantial financial losses for both the client and the affected users.
Medium	While medium level vulnerabilities may not be easy to exploit, they can still have a major impact on the execution of a smart contract. For instance, they may allow public access to critical functions, which could lead to serious consequences.
Low	Poses a very low-level risk to the project or users. Nevertheless the issue should be fixed immediately
Informational	Effects are small and do not post an immediate danger to the project or users
Governance	Governance privileges which can directly result in a loss of funds or other potential undesired behavior
Quality assurance	Aggregated minor issues, ensuring a high quality codebase.

3. Detections

AmphorSyntheticVault.sol

This audit delves into a distinctive vault contract that seamlessly amalgamates OpenZeppelin's reputable ERC4626 library functionalities with innovative features tailored for specific operational requirements.

At its core, the contract facilitates the standard deposit and withdrawal of tokens by users. However, the unique twist lies in its custodial mechanism: the owner can, at opportune moments, transition the vault into a 'custody mode', thereby assuming control over the vault's assets.

During this phase, standard user operations like deposits and withdrawals are temporarily suspended, effectively freezing the vault's activities. The ownership's subsequent ending of the 'custody mode' comes with expectations of a repayment amount that ideally exceeds the original withdrawal, suggesting a gainful interim use of the funds. If this scenario materializes, the contract astutely applies a fee on the differential profit.

While this fee rate is mutable at the discretion of the owner, it's been capped at 30% to foster a sense of balance and deter potential profiteering. The vault's operational heartbeat is regulated by the vaultIsOpen state variable, toggling between two primary modes:

The 'Open Mode': A user-centric phase permitting the free flow of deposits and withdrawals, also serving as the default stance after the contract's deployment.

The 'Custody Mode': An owner-centric phase, activated when the owner appropriates the vault's assets, leading to a temporary cessation of standard user activities.

Supplementing these core functionalities, the vault is fortified with a pause mechanism, a prudent inclusion to counteract unforeseen vulnerabilities or external adversities, ensuring a sanctuary for the staked assets, which at this point, only limitates supplying the vault, not withdrawing.

In summation, the vault contract is a harmonious blend of traditional ERC20 functions and avant-garde custodial and profit-sharing protocols. While its architectural aspirations are aimed at augmenting profitability and ensuring asset security, it mandates an exhaustive audit to guarantee its reliability in the decentralized finance landscape.

The Amphor development team ensured that the start and end operations will be done using a multisig contract, which provides additional safety mechanisms in the scenario of a private key leakage.

Issue	Governance privileges
Severity	Governance
Description	The vault grants the owner extensive centralized control over its operations, including asset transfers and opening/closing mechanisms. If the owner's private keys were compromised, an attacker could potentially misuse these privileges, leading to loss of funds, inaccurate asset accounting, and disruption of the vault's functions.
Recommendations	The owner should be strictly kept under a multisig with trusted participants.
Comments / Resolution	Acknowledged.

Issue	Virtual shares concept is not a guarantee for preventing inflation attacks
Severity	Informational
Description	The virtual shares concept was implemented by OpenZeppelin to prevent the inflation attack. This happens by increasing the totalSupply artificially, with the following desired effect: a) Preventing the following calculation to round down to zero:

assets * totalSupply / totalAssets

b) Decreasing the value which is received during a withdrawal

We will illustrate this logic with a simple PoC, using OpenZeppelin's standard configuration offset = 0:

1) Alice deposits 1 WEI

totalAssets = 1
totalSupply = 1
shares[Alice] = 1

2) Alice sends 100 000e18 tokens directly to the vault

totalAssets = 100 000e18 + 1 wei (to simplify this PoC, we will ignore this 1 wei)
totalSupply = 1
shares[Alice] = 1

3) Bob deposits 10 000e18 tokens

assets * totalSupply +1 / totalAssets +1

10 000e18 * 2 / 100000e18 = 0
totalAssets = 110 000e18
totalSupply = 1
shares[Alice] = 1
shares[Bob] = 0

Bob therefore did not receive any shares.

4) Alice attempts to redeem her 1 share

1 * 110 000e18 / 2 = 55 000e18

As one can see, this attack was not profitable for Alice due to the virtual shares concept. However, within the normal business logic, this will not be the case, if Bob would have deposited multiple times or other deposits would have followed, Alice's attack would then be profitable.

This issue can be fixed with increasing the offset to a larger value, however, it must be noted that the larger the offset, the larger potential losses for users during the withdrawal, since the virtual shares will gather a part of the deposits, OpenZeppelin commented on this issue as follows:

The drawback of this approach is that the virtual shares do capture (a very small) part of the value being accrued
to the vault. Also, if the vault experiences losses, the users try to exit the vault, the virtual shares and assets
will cause the first user to exit to experience reduced losses in detriment to the last users that will experience
bigger losses. Developers willing to revert back to the pre-v4.9 behavior just need to override the
`_convertToShares` and `_convertToAssets` functions.

While the offset is increased, this will make it almost impossible for an attacker to become profitable, however, it can still result in a loss of user funds, which is illustrated in the two following PoC's:

For both PoC's, an offset of 4 is selected, which is derived from Uniswap V2's logic of burning an initial amount:

https://github.com/Uniswap/v2-core/blob/master/contracts/UniswapV2Pair.sol#L121

1) Alice deposits 1 WEI and sends 10_000_000e18 tokens directly to the contract

assets * (totalSupply + 1e4) / (totalAssets + 1)

1 * 1e4 / 1 = 1e4

-> totalSupply = 1e4
-> totalAssets = 100_000e18 + 1 wei (we will ignore the 1 wei to simplify this PoC)

2) Bob deposits 100e18 tokens, now we will use the virtual share solution using a valid amount of 1e4 as offset:

assets * (totalSupply + 1e4) / (totalAssets + 1)

100e18 * (1e4+1e4) / 10_000_000e18 = 0.2 -> 0 since solidity rounds down

As one can see, Bob's value still rounds down to zero, which is essentially determined by the offset <-> totalAssets ratio, the higher the offset, the larger the totalAssets, to execute this attack.

Of course, this attack will not be profitable for Alice, but funds can still be lost for Bob.

The strategy to prevent this issue is simply by reverting if the shares are zero.

However, there is yet another PoC that even circumvents the zero check:

1) Alice deposits 1 WEI and sends 10_000_000e18 tokens directly to the contract

assets * (totalSupply + 1e4) / (totalAssets + 1)

	-> totalSupply = 1e4 -> totalAssets = 10_00_000e18 + 1 wei (we will ignore the 1 wei to simplify this PoC) 2) Bob deposits 9900e18 tokens, now we will use the virtual share solution using a valid amount of 4 as offset: assets * (totalSupply + 1e4) / (totalAssets + 1) 9900e18 * (1e4+1e4) / 10_000_000e18 = 19.8 -> 19 since solidity rounds down. As one can see, it is not only possible to manipulate the vault to return zero shares, but also possible to create such an environment to round shares down so that a user receives less than he should and bypass a potential non-zero check.
Recommendations	We understand that multiple PoC's might be confusing, therefore we will keep the fix recommendation as simple as possible: a) Reverting if the share calculation returns zero b) Allowing the user to input a minShares/AssetsOut parameter (this should be applied to deposit/withdraw/redeem/mint, with the corresponding parameter. This will then prevent if accidentally a user receives less shares than expected. While resolution b) is quite trivial, it is unfortunately not common knowledge yet. BailSec attempts to improve the web3 security space by recommending best-practices which we think should be implemented by all protocols.
Comments / Resolution	Resolved, the Amphor team implemented a depositMinShares and mintMaxAssets function.

Issue	*claimToken can be bypassed for tokens with multiple entry points*
Severity	Informational
Description	Specific tokens, such as TrueUSD, might have different entry points, which can be abused to bypass address checks, such as in the claimToken function. More information on this topic can be found here: https://medium.com/chainsecurity/trueusd-compound-vulnerability-bc5b696d29e2
Recommendations	While we do not see any bad impact at this point, we still are of the opinion that it is necessary to raise awareness for this potential issue. No action is needed here besides of double checking that no such tokens are added.
Comments / Resolution	Acknowledged

Issue	Vault does not work with transfer-tax tokens
Severity	Informational
Description	Within the vault + custody architecture are multiple spots which do not align with transfer-tax tokens. As an example, within the start and end functions, tokens will be lost during the transfer, which will leave users with a loss. Moreover, users would receive shares based on the amount pre-tax, when depositing.
Recommendations	Consider simply not using such tokens or ensure that they are

	whitelisted. Making vaults transfer-tax compatible will come with potential reentrancy risk which must be carefully inspected, since the Amphor team does not plan to use such tokens, we do not see a fix necessity here.
Comments / Resolution	Acknowledged, such tokens will not be used.

Issue	Checks-effects-interactions pattern is violated
Severity	**Informational**
Description	The checks-effects-interactions pattern is a fundamental best practice in smart contract design to prevent reentrancy attacks. In the end function, the `vaultIsOpen` status is set to `true` after funds have already been transferred out. This sequence deviates from the recommended pattern, wherein state changes (like setting `vaultIsOpen`) should be executed before any external calls or transfers. Generally speaking, such deviations can expose the contract to vulnerabilities, potentially leading to unauthorized access or fund theft. Adhering to the checks-effects-interactions pattern is essential to ensure the contract's robustness and security. More information about this topic can be found here: https://docs.soliditylang.org/en/v0.6.11/security-considerations.html
Recommendations	The state variable change should be executed before the asset transfer.

Comments / Resolution	Acknowledged Amphor Team comment: *"We've decided not to apply this recommendation because setting the vaultIsOpen variable to false locks out withdrawals. It therefore seems more logical to us to perform this lock before calling an external contract."*

Issue	Codebase is vulnerable for frontend phishing
Severity	Informational
Description	Multiple functions such as the deposit or mint function allow for a receiver parameter, this will be the address receiving the shares. In the scenario of a compromised frontend, this can lead to a direct loss of user funds.
Recommendations	Maintaining a strong emphasis on the security of the web2 component is crucial, especially given the increasing prevalence of frontend attacks in today's landscape.
Comments / Resolution	Acknowledged

Issue	Quality assurance
Severity	**Quality assurance**
Description	All typographical and minor issues have been aggregated in the Quality Assurance section for clarity and comprehensive review. L 63:

	address public immutable asset; This variable can be directly casted to type IERC20, which saves operational overhead during the business logic. L 455: An inconsistency between withdraw/redeem and deposit/mint is present, regarding the visibility, to comply with best practices, all functions should have the same visibility. Moreover, the virtual keyword can be removed as these functions are not overridden. L 614: For consistency reasons, the lastSavedBalance variable should be reset during the end function.
Recommendations	All aforementioned issues should be fixed.
Comments / Resolution	Resolved