Security Assessment

Blockzero Labs

July 14th, 2022

Secure3

secure3.io

Summary

BlockzeroLabs's Flashstake protocol is a novel financial infrastructure that allows users to receive yield on deposited assets and getting NFT when staking the assets. It also allows the creation of strategy and associate with a principal token.

This report has been prepared for BlockzeroLabs to identify issues and vulnerabilities in the smart contract source code of the BlockzeroLabs project. A comprehensive examination with Static Analysis and Manual Review techniques has been performed.

The examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static scanner to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Informational, Low, Medium, Critical. For each of the findings we have provided recommendation of a fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	BlockzeroLabs
Platform & Language	Ethereum, Solidity
Codebase	https://github.com/BlockzeroLabs/flashv3-contracts audit commit - 7b6e6b41ef496b516e9ce2ff23bea18b3db27af6 final commit - e0ea75a5457d8cd4b22151cbae051637ff8c8eba
Audit Methodology	• Business Logic Understanding and Review • Privileged Roles Review • Static Analysis • Code Review

Business Logic Review Summary

Total Number of Features	Caution	Information	Verified
9	0	1	8

Privileged Role Review Summary

Total Number of Privileged Roles	Caution	Information	Verified
9	0	0	9

Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowleged	Fixed	Mitigated
Critical	0	0	0	0	0
Medium	5	0	3	2	0
Low	6	0	3	2	1
Informational	6	0	3	3	0

Audit Scope

File	Commit Hash
contracts/FlashBack.sol	7b6e6b41ef496b516e9ce2ff23bea18b3db27af6
contracts/FlashFToken.sol	7b6e6b41ef496b516e9ce2ff23bea18b3db27af6
contracts/FlashFTokenFactory.sol	7b6e6b41ef496b516e9ce2ff23bea18b3db27af6
contracts/FlashNFT.sol	7b6e6b41ef496b516e9ce2ff23bea18b3db27af6
contracts/FlashToken.sol	7b6e6b41ef496b516e9ce2ff23bea18b3db27af6
contracts/interfaces/AAVE/DataTypes.sol	7b6e6b41ef496b516e9ce2ff23bea18b3db27af6
contracts/interfaces/AAVE/IAaveIncentivesController.sol	7b6e6b41ef496b516e9ce2ff23bea18b3db27af6
contracts/interfaces/AAVE/ILendingPool.sol	7b6e6b41ef496b516e9ce2ff23bea18b3db27af6
contracts/interfaces/AAVE/ILendingPoolAddressesProvider.sol	7b6e6b41ef496b516e9ce2ff23bea18b3db27af6
contracts/interfaces/IERC20C.sol	7b6e6b41ef496b516e9ce2ff23bea18b3db27af6
contracts/interfaces/IFlashFTokenFactory.sol	7b6e6b41ef496b516e9ce2ff23bea18b3db27af6
contracts/interfaces/IFlashNFT.sol	7b6e6b41ef496b516e9ce2ff23bea18b3db27af6
contracts/interfaces/IFlashStrategy.sol	7b6e6b41ef496b516e9ce2ff23bea18b3db27af6
contracts/interfaces/IFlashFToken.sol	7b6e6b41ef496b516e9ce2ff23bea18b3db27af6
contracts/interfaces/IUserIncentive.sol	7b6e6b41ef496b516e9ce2ff23bea18b3db27af6
contracts/FlashProtocol.sol	7b6e6b41ef496b516e9ce2ff23bea18b3db27af6
contracts/UserIncentive.sol	7b6e6b41ef496b516e9ce2ff23bea18b3db27af6
contracts/strategies/FlashStrategyAAVEv2.sol	7b6e6b41ef496b516e9ce2ff23bea18b3db27af6

Business Logic Review

In this section, we asked project team to provide a list of business features of their contracts, our team verified each feature one by one and provided the verification results below.

How to read the table

1. **Left column is from project team**, describing their business intent
2. **Right column is from auditing team**, verifying if the code implementation meets the claimed business intent

Business Feature Claimed	Business Feature Audit Result
Token ERC20 - FlashToken is a ERC20 token	◉ **Auditor Evaluation:** **Verified,** ◉ **Code Reference:** contracts/FlashToken.sol:10 ◉ **Detail:** The `Flashstake/FLASH` token is `ERC20` token with `150,000,000` total supply.
Token ERC20 - fToken yield-bearing token is a ERC20 token that can only be created by owner	◉ **Auditor Evaluation:** **Verified** ◉ **Code Reference:** contracts/FlashFToken.sol:15 ◉ **Detail:** The `FlashFToken` token is `ERC20` token which can only be minted by the contract owner.
FlashStrategyAAVEv2 - can be used for any underlying principal token	◉ **Auditor Evaluation:** **Verified,** ◉ **Code Reference:** contracts/strategies/FlashStrategyAAVEv2.sol:31 ◉ **Detail:** The `_principalTokenAddress` is passed in the constructor for the chosen token.
FlashStrategyAAVEv2 - the principle token is deposited into the lending protocol such as AAVE	◉ **Auditor Evaluation:** **Informational,** ◉ **Code Reference:** contracts/strategies/FlashStrategyAAVEv2.sol:55 ◉ **Detail:** The `depositPrincipal()` internally calls `ILendingPool::deposit()` to deposit the amount. However the address of `lendingPoolAddress` is determined during deployment, reader should verify the final actual address is the AAVE address on Mainnet

Business Feature Claimed	Business Feature Audit Result
FlashStrategyAAVEv2 - user can withdraw principle token	◉ **Auditor Evaluation: Verified,** ◉ **Code Reference:** contracts/strategies/FlashStrategyAAVEv2.sol:72 ◉ **Detail:** The `withdrawPrincipal()` internally calls `ILendingPool::withdraw()` and `IERC20::safeTransfer()` to withdraw the principal token from LendingPool and transfer the amount back to `msg.sender`.
FlashProtocol - when stake the principal token, fToken is minted	◉ **Auditor Evaluation: Verified,** ◉ **Code Reference:** contracts/FlashProtocol.sol:116,120 ◉ **Detail:** The `stake()` internally calls `IFlashFToken::mint()` to mint the fToken to `_fTokensTo` address for the lending pool yield entitlement.
FlashProtocol - staking, minting fTokens and burning all operations are in one transaction	◉ **Auditor Evaluation: Verified,** ◉ **Code Reference:** contracts/FlashProtocol.sol:272 ◉ **Detail:** The `flashStake()` has `stake()`, `mint()` and `burnFToken()` in the call stack, which guarantees the operations are in one transaction.
FlashProtocol - user can unstake the principal toke	◉ **Auditor Evaluation: Verified,** ◉ **Code Reference:** contracts/FlashProtocol.sol:148 ◉ **Detail:** The `unstake()` function burn the yield bearing fToken and transfer the principle tokens from strategy to the user.
FlashProtocol - user can build and register a new strategies into the FlashProtocol	◉ **Auditor Evaluation: Verified,** ◉ **Code Reference:** contracts/FlashProtocol.sol:68 ◉ **Detail:** The `registerStrategy()` function can register the strategy address with principal token address. It also creates the fToken internally with the `IFlashFTokenFactory`.

Privilege Role Review

In this section, we reviewed all the privileged roles in the contracts. We listed all the findings in the following table.

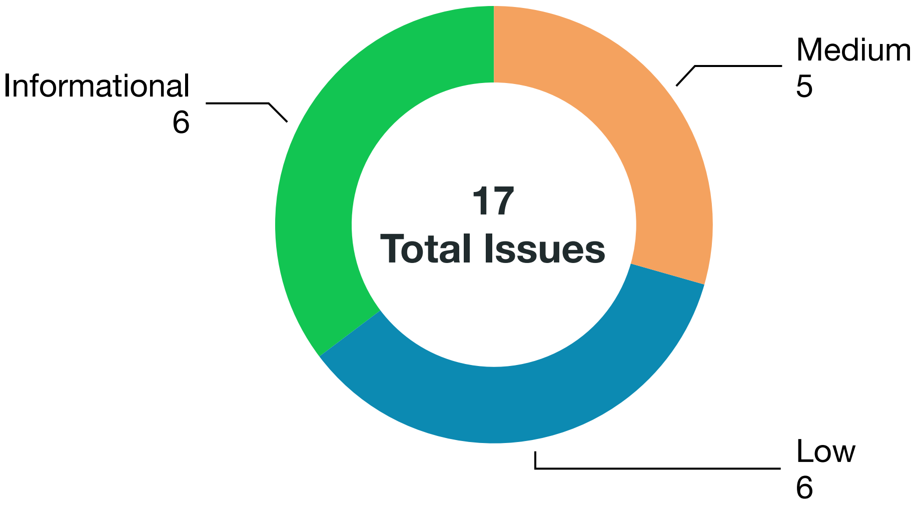
How to read the table

1. **Left column:** privileged role name
2. **Middle column:** privileged permission of the role
3. **Right column:** verified code implementation and roles permission by auditing team

Contract Role	Privileged Functionalities	Audit Review
FlashBack Owner Address	◉ setForfeitRewardAddress ◉ setRewardRate	◉ **Auditor Evaluation:** **Verified**, ◉ **Code Reference:** contracts/**FlashBack.sol** ◉ **Detail:** critical functionalities can only be called by contract owner
FlashFToken Owner Address	◉ mint	◉ **Auditor Evaluation:** **Verified** ◉ **Code Reference:** contracts/ **FlashFToken.sol** ◉ **Detail:** critical functionalities can only be called by contract owner
FlashFTokenFactory Owner Address	◉ createFToken	◉ **Auditor Evaluation:** **Verified**, ◉ **Code Reference:** contracts/ **FlashFTokenFactory.sol** ◉ **Detail:** critical functionalities can only be called by contract owner
FlashNFT Owner Address	◉ burn ◉ mint	◉ **Auditor Evaluation:** **Verified**, ◉ **Code Reference:** contracts/**FlashNFT.sol** ◉ **Detail:** critical functionalities can only be called by contract owner

Contract Role	Privileged Functionalities	Audit Review
FlashProtocol Owner Address	◉ setMintFeeInfo	◉ **Auditor Evaluation: Verified,** ◉ **Code Reference:** contracts/ **FlashProtocol.sol** ◉ **Detail:** critical functionalities can only be called by contract owner
UserIncentive Owner Address	◉ depositReward ◉ addRewardTokens ◉ setRewardRatio	◉ **Auditor Evaluation: Verified,** ◉ **Code Reference:** contracts/ **UserIncentive.sol** ◉ **Detail:** critical admin functions can only be called by contract owner
UserIncentive Strategy Owner Address	◉ claimReward	◉ **Auditor Evaluation: Verified,** ◉ **Code Reference:** contracts/ **UserIncentive.sol** ◉ **Detail:** critical admin functions can only be called by strategy contract owner
FlashStrategyAAVEv2 Owner Address	◉ withdrawERC20 ◉ claimAAVEv2Rewards ◉ setUserIncentiveAddress	◉ **Auditor Evaluation: Verified,** ◉ **Code Reference:** contracts/strategies/ **FlashStrategyAAVEv2.sol** ◉ **Detail:** critical admin functions can only be called by contract owner
FlashStrategyAAVEv2 Contract Address or **FlashProtocol Contract** Address	◉ depositPrincipal ◉ withdrawPrincipal ◉ setFTokenAddress	◉ **Auditor Evaluation: Verified,** ◉ **Code Reference:** contracts/strategies/ **FlashStrategyAAVEv2.sol** ◉ **Detail:** critical admin functions can only be called by contract itself or FlashProtocol

Code Assessment Findings



ID	Name	Category	Severity	Status
BZL-1	**Solidity compiler version is not fixed and consistent across the project**	Language Specific	Low	Mitigated
BZL-2	`FlashBack.maximumStakeDuration` **does not count for leap year**	Logical	Informational	Acknowledged
BZL-3	`FlashBack::constructor()` **does not validate** `_stakingTokenAddress`	Logical	Low	Acknowledged
BZL-4	`FlashBack::stake()` **should use the defined state variable instead of magic value literal address**	Logical	Medium	Acknowledged
BZL-5	`FlashBack::unstake()` **does not check** `transfer()` **return value**	Logical	Medium	Fixed
BZL-6	`FlashBack::setForfeitRewardAddress()` **missing event**	Code Style	Informational	Fixed

ID	Name	Category	Severity	Status
BZL-7	`FlashBack::setRewardRate()` missing event	Code Style	Informational	Fixed
BZL-8	`FlashProtocol::constructor()` does not validate input parameter address	Logical	Low	Acknowledged
BZL-9	`FlashProtocol::registerStrategy()` does not validate input parameter address	Logical	Informational	Fixed
BZL-10	`FlashToken::decimals()` not set explicitly	Logical	Informational	Acknowledged
BZL-11	`UserIncentive::depositReward()` does not check `transfer()` return value	Logical	Medium	Acknowledged
BZL-12	`UserIncentive::claimReward()` does not check `transfer()` return value	Logical	Medium	Acknowledged
BZL-13	`FlashStrategyAAVEv2::constructor()` does not validate input parameter address	Logical	Low	Acknowledged
BZL-14	`FlashStrategyAAVEv2` is unable to stop or decrease AVVE lending pool approved amount	Logical	Informational	Acknowledged
BZL-15	`FlashStrategyAAVEv2::withdrawYield()` ignoring AVVE lending pool withdraw returned value	Logical	Informational	Fixed
BZL-16	`FlashStrategyAAVEv2::withdrawPrincipal()` ignoring AVVE lending pool withdraw returned value	Logical	Medium	Fixed
BZL-17	`FlashStrategyAAVEv2::getMaxStakeDuration()` comment typo	Code Style	Informational	Fixed

BZL-1: Solidity compiler version is not fixed and consistent across the project

Category	Severity	Code Reference	Status
Language Specific	Low	All contracts	Mitigated

Code

```
2: pragma solidity ^0.8.4;
2: pragma solidity >=0.8.4;
```

Description

There are `^0.8.4` and `>=0.8.4` solidity versions used in the contracts and the compiler version is floating. Having non fixed compiler version is not the best practice.

Recommendation

Fix the compiler version to `0.8.4` or a preferred version.

Client Response

Client changed all the version pragma to `^0.8.4`, meaning all the patch versions of `0.8.4` and higher versions in the `0.8.x` branch. Since the caret range is for non-breaking changes, this is better than `>=0.8.4` which can include breaking changes in the future.

BZL-2: `FlashBack.maximumStakeDuration` does not count for leap year

Category	Severity	Code Reference	Status
Logical	Informational	contracts/FlashBack.sol:13	Acknowledged

Code

```
13:     uint256 constant maximumStakeDuration = 31536000; // 365 days in seconds
```

Description

With leap year considered and averaged out in four years, `maximumStakeDuration` should be `365.25 x 24 x 60 x 60 = 31557600` seconds.

Recommendation

Consider if the leap year case is needed and modify the value accordingly.

Client Response

No change required.

BZL-3: `FlashBack::constructor()` does not validate `_stakingTokenAddress`

Category	Severity	Code Reference	Status
Logical	Low	contracts/FlashBack.sol:36	Acknowledged

Code

```
35:     constructor(address _stakingTokenAddress) public {
36:         stakingTokenAddress = _stakingTokenAddress;
37:     }
```

Description

The input parameter `_stakingTokenAddress` can be zero address.

Recommendation

Add a require statement to validate `_stakingTokenAddress != address(0)`.

Client Response

No change required - this is part of due diligence around deployment.

BZL-4: `FlashBack::stake()` should use the defined state variable instead of magic value literal address

Category	Severity	Code Reference	Status
Logical	Medium	contracts/FlashBack.sol:44,45	Acknowledged

Code

```
44:         require(msg.sender != 0x5089722613C2cCEe071C39C59e9889641f435F15, "BLACKLISTED
ADDRESS");
45:         require(msg.sender != 0x8603FfE7B00CCd759f28aBfE448454A24cFba581, "BLACKLISTED
ADDRESS");
```

Description

The two blacklisted addresses are already defined as state variables in the contract `FlashBack.forfeitRewardAddress` and `FlashProtocol.globalMintFeeRecipient`. The logic should reference them instead of using hardcode magic address. Besides good code style, when `forfeitRewardAddress` and `globalMintFeeRecipient` get updated by the setter functions, the checks will fail to pick up the new values. Also, the two error messages should be distinct to differentiate each failure cause.

Recommendation

Reference the `forfeitRewardAddress` in the require. For `globalMintFeeRecipient`, either add a setter in `FlashBack` to update the `globalMintFeeRecipient` in the contract or to get the updated value from `FlashProtocol` every call with a higher gas cost. Update the revert error messages.

Client Response

No change required, we have intentionally put these addresses there so it is clear to those reading that these two addresses are blacklisted from participating in FlashBacks. This is really just for optics.

BZL-5: `FlashBack::unstake()` does not check `transfer()` return value

Category	Severity	Code Reference	Status
Logical	Medium	contracts/FlashBack.sol:82,83,87	Fixed

Code

```
81:         if (unstakedEarly) {
82:             IERC20C(stakingTokenAddress).transfer(msg.sender, p.stakedAmount);
83:             IERC20C(stakingTokenAddress).transfer(forfeitRewardAddress,
p.reservedReward);
84:
85:             emit Unstaked(_stakeId, 0, p.reservedReward);
86:         } else {
87:             IERC20C(stakingTokenAddress).transfer(msg.sender, p.stakedAmount +
p.reservedReward);
88:
89:             emit Unstaked(_stakeId, p.reservedReward, 0);
90:         }
```

Description

The ERC20 `transfer()` function has a return value, and in case of failure it returns false. The best practice is to check the return value of the `transfer()` function and revert in case of failure.

Recommendation

Use `SafeERC20` from OpenZeppelin by `using SafeERC20 for IERC20C` in the contract and `IERC20C(stakingTokenAddress).safeTransfer(address, amount)` to use it.

Client Response

Fixed by using `safeTransfer`.

BZL-6: `FlashBack::setForfeitRewardAddress()` missing event

Category	Severity	Code Reference	Status
Code Style	**Informational**	contracts/FlashBack.sol:113	Fixed

Code

```
112:     function setForfeitRewardAddress(address _forfeitRewardAddress) external onlyOwner {
113:         forfeitRewardAddress = _forfeitRewardAddress;
114:     }
```

Description

The `forfeitRewardAddress` state is changed but there is no event emitted.

Recommendation

Emit an event

Client Response

Event added.

BZL-7: `FlashBack::setRewardRate()` missing event

Category	Severity	Code Reference	Status
Code Style	Low	contracts/FlashBack.sol:118	Fixed

Code

```
116:     function setRewardRate(uint256 _rewardRate) external onlyOwner {
117:         require(_rewardRate <= 63419583968, "INVALID REWARD RATE");
118:         rewardRate = _rewardRate;
119:     }
```

Description

The `rewardRate` state is changed but there is no event emitted.

Recommendation

Emit an event

Client Response

Event added.

BZL-8: `FlashProtocol::constructor()` does not validate input parameter address

Category	Severity	Code Reference	Status
Logical	Low	contracts/FlashProtocol.sol:63	Acknowledged

Code

```
63:     constructor(address _flashNFTAddress, address _flashFTokenFactoryAddress) public {
64:         flashNFTAddress = _flashNFTAddress;
65:         flashFTokenFactoryAddress = _flashFTokenFactoryAddress;
66:     }
```

Description

The input parameter `_flashNFTAddress` and `_flashFTokenFactoryAddress` can be zero address.

Recommendation

Add a require statement to validate `_flashNFTAddress` and `_flashFTokenFactoryAddress` is not `address(0)`.

Client Response

No change required - this is part of due diligence around deployment.

BZL-9: `FlashProtocol::registerStrategy()` does not validate input parameter address

Category	Severity	Code Reference	Status
Logical	Low	contracts/FlashProtocol.sol:69,70	Fixed

Code

```
68:     function registerStrategy(
69:         address _strategyAddress,
70:         address _principalTokenAddress,
71:         string calldata _fTokenName,
72:         string calldata _fTokenSymbol
73:     ) external {
```

Description

The input parameter `_strategyAddress` and `_principalTokenAddress` can be zero address. And when `_strategyAddress` is zero, the subsequent require would default to zero and bypass the check.

Recommendation

Add a require statement to validate `_strategyAddress` and `_principalTokenAddress` is not `address(0)`.

Client Response

Fixed by adding validation.

BZL-10: `FlashToken::decimals()` not set explicitly

Category	Severity	Code Reference	Status
Logical	Informational	contracts/FlashToken.sol:10	Acknowledged

Code

```
09:     constructor() ERC20("Flashstake", "FLASH") ERC20Permit("Flashstake") {
10:         _mint(msg.sender, 150000000 * 10**decimals());
11:     }
```

Description

Token's `decimals` is not set explicitly. The default value of decimals is 18. To select a different value for decimals you should overload it.

Recommendation

Please confirm if 18 is the desired `decimals` value.

Client Response

No change required, 18 decimals is expected.

BZL-11: `UserIncentive::depositReward()` does not check `transfer()` return value

Category	Severity	Code Reference	Status
Logical	Medium	contracts/UserIncentive.sol:36	Acknowledged

Code

```
35:         require(block.timestamp > rewardLockoutTs, "LOCKOUT IN FORCE");
36:         IERC20C(rewardTokenAddress).transfer(msg.sender, rewardTokenBalance);
```

Description

The ERC20 `transfer()` function has a return value, and in case of failure it returns false. The best practice is to check the return value of the `transfer()` function and revert in case of failure.

Recommendation

Use `SafeERC20` from OpenZeppelin by `using SafeERC20 for IERC20C` in the contract and `IERC20C(rewardTokenAddress).safeTransfer(address, amount)` to use it.

Client Response

No change required, `UserIncentive` contract will only be used with ERC-20 compliant tokens (specifically Flash token).

BZL-12: `UserIncentive::claimReward()` does not check `transfer()` return value

Category	Severity	Code Reference	Status
Logical	Medium	contracts/UserIncentive.sol:86	Acknowledged

Code

```
85:         // Transfer and update balance locally
86:         IERC20C(rewardTokenAddress).transfer(_yieldTo, rewardAmount);
87:         rewardTokenBalance = rewardTokenBalance - rewardAmount;
```

Description

The ERC20 `transfer()` function has a return value, and in case of failure it returns false. The best practice is to check the return value of the `transfer()` function and revert in case of failure.

Recommendation

Use `SafeERC20` from OpenZeppelin by `using SafeERC20 for IERC20C` in the contract and `IERC20C(rewardTokenAddress).safeTransfer(address, amount)` to use it.

Client Response

No change required, UserIncentive contract will only be used with ERC-20 compliant tokens (specifically Flash token).

BZL-13: `FlashStrategyAAVEv2::constructor()` does not validate input parameter address

Category	Severity	Code Reference	Status
Logical	Low	contracts/strategies/ FlashStrategyAAVEv2.sol:37-40	Acknowledged

Code

```
31:     constructor(
32:         address _lendingPoolAddress,
33:         address _principalTokenAddress,
34:         address _interestBearingTokenAddress,
35:         address _flashProtocolAddress
36:     ) public {
37:         lendingPoolAddress = _lendingPoolAddress;
38:         principalTokenAddress = _principalTokenAddress;
39:         interestBearingTokenAddress = _interestBearingTokenAddress;
40:         flashProtocolAddress = _flashProtocolAddress;
41:
42:         increaseAllowance();
43:     }
```

Description

The input parameter address can be zero address.

Recommendation

Add a require statement to validate input parameters are not `address(0)`.

Client Response

No change required - this is part of due diligence around deployment.

BZL-14: `FlashStrategyAAVEv2` is unable to stop or decrease AVVE lending pool approved amount

Category	Severity	Code Reference	Status
Logical	Informational	contracts/strategies/ FlashStrategyAAVEv2.sol:47	Acknowledged

Code

```
46:     function increaseAllowance() public {
47:         IERC20(principalTokenAddress).safeApprove(lendingPoolAddress, type(uint256).max);
48:     }
49:
```

Description

While it is unlikely AAVE is compromised, it is crucial that the contract owner can decrease the approved amount from an external contract allowance and have full control on the allowance.

Recommendation

Consider add a new function to decrease or stop the allowance with `onlyOwner` modifier.

Client Response

No change required - the risk is known but we favour decentralisation. Users will be made aware of the inherit risk surrounding the protocol and its dependencies.

BZL-15: `FlashStrategyAAVEv2::withdrawYield()` ignoring AVVE lending pool withdraw returned value

Category	Severity	Code Reference	Status
Logical	Informational	contracts/strategies/FlashStrategyAAVEv2.sol: 62	Fixed

Code

```
60:     function withdrawYield(uint256 _tokenAmount) private {
61:         // Withdraw from AAVE
62:         ILendingPool(lendingPoolAddress).withdraw(principalTokenAddress, _tokenAmount,
address(this));
63:
64:         uint256 aTokenBalance =
IERC20(interestBearingTokenAddress).balanceOf(address(this));
65:         require(aTokenBalance >= getPrincipalBalance(), "PRINCIPAL BALANCE INVALID");
66:     }
```

Description

`ILendingPool::withdraw()` returns the final amount withdrawn, and this could be different that the input `_tokenAmount`.

Recommendation

Confirm in the case that the final yield `withdrawn` amount is different than requested, do you want to revert the transaction.

Client Response

Addressed, added in a check.

BZL-16: `FlashStrategyAAVEv2::withdrawPrincipal()` ignoring AVVE lending pool withdraw returned value

Category	Severity	Code Reference	Status
Logical	Medium	contracts/strategies/FlashStrategyAAVEv2.sol: 70	Fixed

Code

```
68:     function withdrawPrincipal(uint256 _tokenAmount) external override onlyAuthorised {
69:         // Withdraw from AAVE
70:         ILendingPool(lendingPoolAddress).withdraw(principalTokenAddress, _tokenAmount,
address(this));
71:
72:         IERC20(principalTokenAddress).safeTransfer(msg.sender, _tokenAmount);
73:
74:         principalBalance = principalBalance - _tokenAmount;
75:     }
```

Description

`ILendingPool::withdraw()` returns the final amount withdrawn, and this could be less than the input `_tokenAmount`. When that happens, in the line 72 the `msg.sender` would receive more than what is staked in AAVE.

However, we understand this function is guarded by `onlyAuthorised` modifier so the `msg.sender` can only be strategy contract itself or `flashProtocolAddress`.

Recommendation

Confirm in the case that the final yield `withdrawn` amount is different than requested, do you want to revert the transaction or use the actual `withdrawn` amount for the `safeTransfer()` call.

Client Response

Addressed, added in a check.

BZL-17: `FlashStrategyAAVEv2::getMaxStakeDuration()` comment typo

Category	Severity	Code Reference	Status
Code Style	Informational	Contracts/strategies/ FlashStrategyAAVEv2.sol:166	Fixed

Code

```
165:     function getMaxStakeDuration() public pure override returns (uint256) {
166:         return 63072000; // Static 720 days (2 years)
167:     }
```

Description

63072000 seconds is 730 days (2 years), the comment says 720 days a typo.

Recommendation

Correct the typo to be 730 days.

Client Response

Addressed, that was a typo.

Disclaimer

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 Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. 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 Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. 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.