Smart Contract Audit Report for Bolide

Testers

1. Or Duan
2. Ido Shdaimah
3. Omri Shdaimah

Table of Contents

Table of Contents **2**
Management Summary **3**
Risk Methodology **4**
Vulnerabilities by Risk **5**
Approach **6**
Introduction 6
Scope Overview 6
Scope Validation 7
Threat Model 7
Security Evaluation **8**
Audit Findings **15**
Max Token Allowance 15
Insufficient of Input Validation 16
Inefficient Array Length Access 17
Unnecessary Initialization 18
Lack of Event Emission in a Critical Function 19

Management Summary

Bolide contacted Sayfer to perform a security audit on their smart contracts.

This report documents the research carried out by Sayfer targeting the selected resources defined under the research scope. Particularly, this report displays the security posture review for Bolide's smart contracts.

Over the research period of 40 research hours, we discovered 5 vulnerabilities in the contracts. All of them have been addressed by the Bolide team.

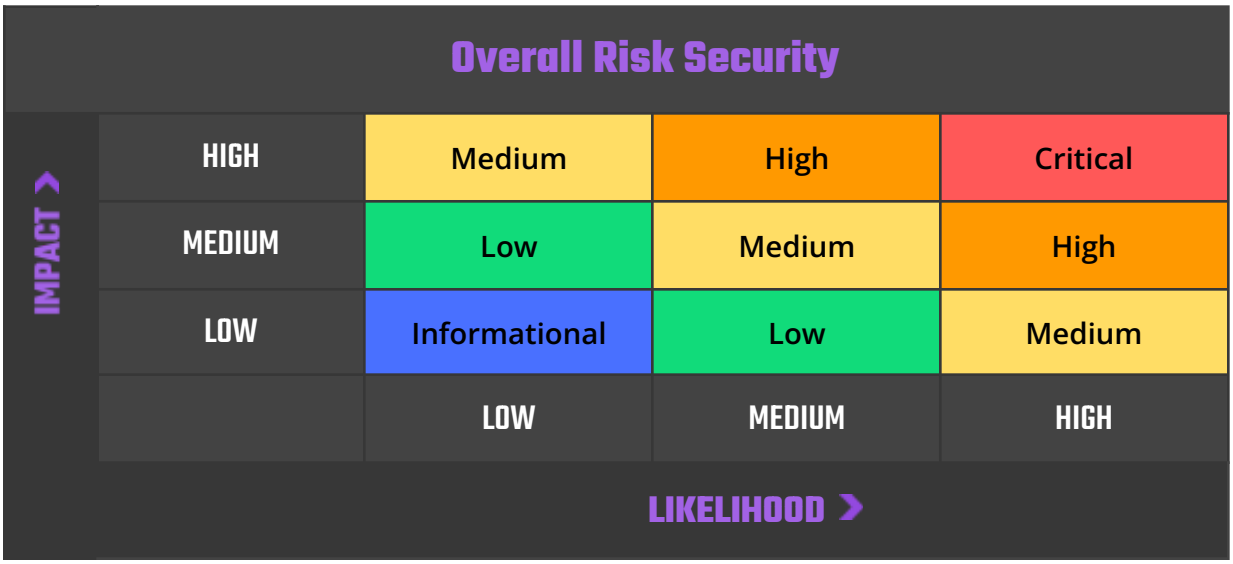
Risk Methodology

At Sayfer, we are committed to delivering the highest quality smart contract audits to our clients. That's why we have implemented a comprehensive risk assessment model to evaluate the severity of our findings and provide our clients with the best possible recommendations for mitigation.

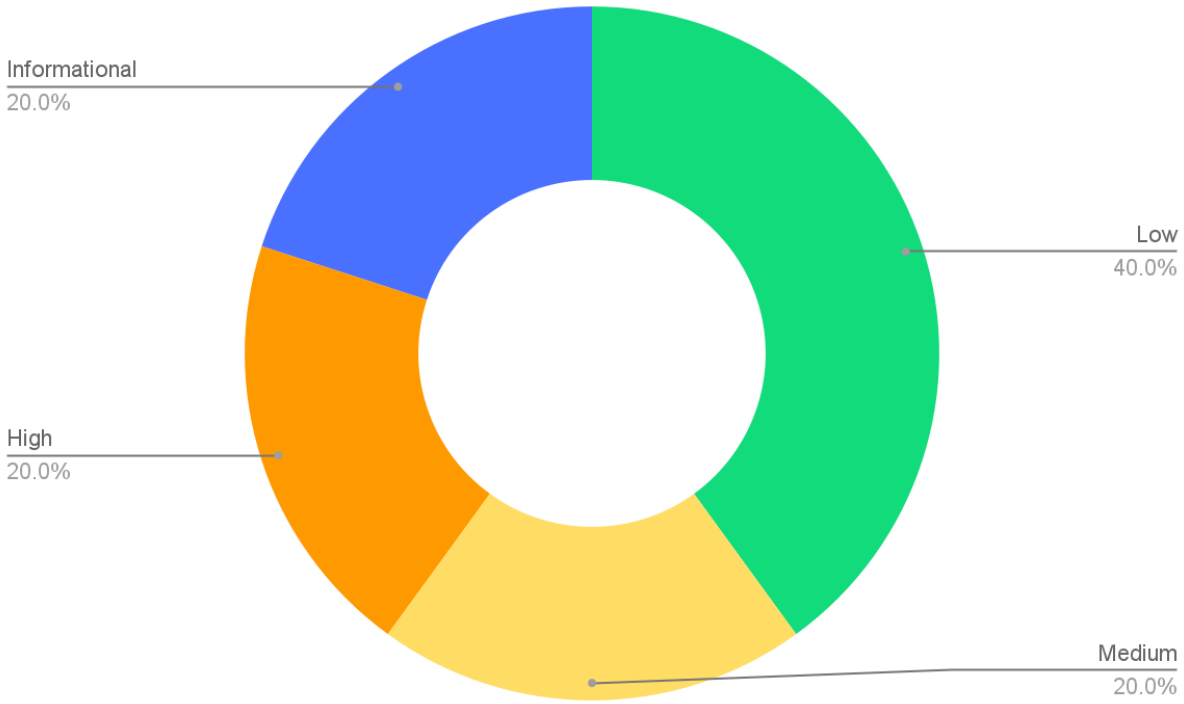
Our risk assessment model is based on two key factors: **IMPACT** and **LIKELIHOOD**. Impact refers to the potential harm that could result from an issue, such as financial loss, reputational damage, or a non-operational system. Likelihood refers to the probability that an issue will occur, taking into account factors such as the complexity of the contract and the number of potential attackers.

By combining these two factors, we can create a comprehensive understanding of the risk posed by a particular issue and provide our clients with a clear and actionable assessment of the severity of the issue. This approach allows us to prioritize our recommendations and ensure that our clients receive the best possible advice on how to protect their smart contracts.

Risk is defined as follows:



Vulnerabilities by Risk



Risk	Low	Medium	High	Critical	Informational
# of issues	2	1	1	0	1

Approach

Introduction

Bolide contacted Sayfer to perform a security audit on their smart contracts.

This report documents the research carried out by Sayfer targeting the selected resources defined under the research scope. Particularly, this report displays the security posture review for the aforementioned contracts.

Scope Overview

Together with the client team we defined the following contract as the scope of the project.
Audit commit: [1531de8aac65aaffebf3a85f5b9e81314b480bcd]
Fixes commit: [226308db99bb6078473e5a53643fa3ae7b70a3d4]

Contract	Sha-256
contracts/Automation.sol	26e82ac5e70c9e22adf862a0fbbcbe7894505b0a106bc1cb1bf47471739a1024
contracts/BaseLogic.sol	7f060b120469279bd85befc8ac3180c39016a7376b514cbfa9a884a64d13fcbf
contracts/Bolide.sol	d5ee8ee29e39ad67c93250e97a0073cfa06a24774e51f4ab480348f5f5b56940
contracts/Booster.sol	7c6278bf0088fa267ac4405f566a5c0e6f9496da18dd89669962992f4ffe658e
contracts/LendingLogic.sol	986bee0d7d7cb7b979265e61972c4e6792cca69accdaf172375cff4d751daeb7
contracts/Migrations.sol	e01768271524e4ab9fea2138b301b41af2a8f9116fb325da57117e53bdce200c
contracts/MultiLogic.sol	8a0d3f8f9894ba6c66d325900dab1262eb290e3ccac7a840a4e466a7fa41d2ab
contracts/StatisticsBase.sol	82dc64d23088728bf6a203e58b692b6fdd9e3231303797849c8f85ec41489783
contracts/StorageV3.sol	5eadbf485e3ad69e3c2615eb25e5909722668bfa14fd12554a5818fe33dba4e3
contracts/SwapGateway.sol	68464763a0eb38e35f243fba30653a7f3f7287414c26a68f21b4c4dc6eff58f7
contracts/strategies/lbl/dforce/DForceLogic.sol	b6c23c3bb31ad813b8d97c19f1ddf181bdcdf54e81f5b97d0d767cae916225ea
contracts/strategies/lbl/dforce/DForceStatistics.sol	b30749f931d9b1bfd77c659cae5e2bdc7310782f075e8e3397a1a884fd69f916
contracts/strategies/lbl/	8600386c807cd78e98379bfef08c24373b4126de98fe6c7a1d5f6c4ee9c78220

LendBorrowLendStrategy.sol	
contracts/strategies/lbl/ LendBorrowLendStrategyHelper.sol	c1ccf3bb2c30caa25c4c11a6fb3d31b46d9421b68f14baf8778f48889cfc4df3
contracts/utils/UpgradeableBase.sol	35930ca7565a4d832927163d451770c7f4546b9da089648ff0f7844ce39b7759
contracts/utils/ UpgradeableMultiAdminableBase.sol	2dbb7041c8e969f1ee9a752e67b5dbe17d60c1d1c278fc244e7eecfa97eb33a8
contracts/utils/ OwnableUpgradeableVersionable.sol	24ab9ad3300f451269265a86ab941742f64ae32bd075ce944ace0f9f8586b5be
contracts/utils/ OwnableUpgradeableAdminable.sol	10587ee0da02bf4088696078482cde74873e92050775251b1060a68d1643d968
contracts/utils/ OwnableUpgradeableMultiAdminable.sol	75b3792017eddb597e6919f8c62758c04bd6b161131c1986db8932c7311d6c7a

Our tests were performed from the 20th of July to the 2nd of August 2023

Scope Validation

We began by ensuring that the scope defined to us by the client was technically logical. Deciding what scope is right for a given system is part of the initial discussion.

Threat Model

We defined that the largest current threat to the system is the ability of malicious users to steal funds from the contract.

Security Evaluation

The following test cases were the guideline while auditing the system. This checklist is a modified version of the SCSVS v1.2, with improved grammar, clarity, conciseness, and additional criteria. Where there is a gap in the numbering, an original criterion was removed. Criteria that are marked with an asterisk were added by us.

Architecture, Design and Threat Modeling	Test Name
G1.2	Every introduced design change is preceded by threat modeling.
G1.3	The documentation clearly and precisely defines all trust boundaries in the contract (trusted relations with other contracts and significant data flows).
G1.4	The SCSVS, security requirements or policy is available to all developers and testers.
G1.5	The events for the (state changing/crucial for business) operations are defined.
G1.6	The project includes a mechanism that can temporarily stop sensitive functionalities in case of an attack. This mechanism should not block users' access to their assets (e.g. tokens).
G1.7	The amount of unused cryptocurrencies kept on the contract is controlled and at the minimum acceptable level so as not to become a potential target of an attack.
G1.8	If the fallback function can be called by anyone, it is included in the threat model.
G1.9	Business logic is consistent. Important changes in the logic should be applied in all contracts.
G1.10	Automatic code analysis tools are employed to detect vulnerabilities.
G1.11	The latest major release of Solidity is used.
G1.12	When using an external implementation of a contract, the most recent version is used.
G1.13	When functions are overridden to extend functionality, the super keyword is used to maintain previous functionality.
G1.14	The order of inheritance is carefully specified.
G1.15	There is a component that monitors contract activity using events.
G1.16	The threat model includes whale transactions.
G1.17	The leakage of one private key does not compromise the security of the entire project.

Policies and Procedures	Test Name

G2.2	The system's security is under constant monitoring (e.g. the expected level of funds).
G2.3	There is a policy to track new security vulnerabilities and to update libraries to the latest secure version.
G2.4	The security department can be publicly contacted and that the procedure for handling reported bugs (e.g., thorough bug bounty) is well-defined.
G2.5	The process of adding new components to the system is well defined.
G2.6	The process of major system changes involves threat modeling by an external company.
G2.7	The process of adding and updating components to the system includes a security audit by an external company.
G2.8	In the event of a hack, there's a clear and well known mitigation procedure in place.
G2.9	The procedure in the event of a hack clearly defines which persons are to execute the required actions.
G2.10	The procedure includes alarming other projects about the hack through trusted channels.
G2.11	A private key leak mitigation procedure is defined.

Upgradability	**Test Name**
G2.2	Before upgrading, an emulation is made in a fork of the main network and everything works as expected on the local copy.
G2.3	The upgrade process is executed by a multisig contract where more than one person must approve the operation.
G2.4	Timelocks are used for important operations so that the users have time to observe upcoming changes (please note that removing potential vulnerabilities in this case may be more difficult).
G2.5	*initialize()* can only be called once.
G2.6	*initialize()* can only be called by an authorized role through appropriate modifiers (e.g. *initializer*, *onlyOwner*).
G2.7	The update process is done in a single transaction so that no one can front-run it.
G2.8	Upgradeable contracts have reserved gap on slots to prevent overwriting.
G2.9	The number of reserved (as a gap) slots has been reduced appropriately if new variables have been added.
G2.10	There are no changes in the order in which the contract state variables are declared, nor their types.
G2.11	New values returned by the functions are the same as in previous versions of the contract (e.g. *owner()*, *balanceOf(address)*).
G2.12	The implementation is initialized.
G2.13	The implementation can't be destroyed.

Business Logic	Test Name
G4.2	The contract logic and protocol parameters implementation corresponds to the documentation.
G4.3	The business logic proceeds in a sequential step order and it is not possible to skip steps or to do it in a different order than designed.
G4.4	The contract has correctly enforced business limits.
G4.5	The business logic does not rely on the values retrieved from untrusted contracts (especially when there are multiple calls to the same contract in a single flow).
G4.6	The business logic does not rely on the contract's balance (e.g., *balance == 0*).
G4.7	Sensitive operations do not depend on block data (e.g., *block hash*, *timestamp*).
G4.8	The contract uses mechanisms that mitigate transaction-ordering (front-running) attacks (e.g. pre-commit schemes).
G4.9	The contract does not send funds automatically, but lets users withdraw funds in separate transactions instead.

Access Control	Test Name
G5.2	The principle of the least privilege is upheld. Other contracts should only be able to access functions and data for which they possess specific authorization.
G5.3	New contracts with access to the audited contract adhere to the principle of minimum rights by default. Contracts should have a minimal or no permissions until access to the new features is explicitly granted.
G5.4	The creator of the contract complies with the principle of the least privilege and their rights strictly follow those outlined in the documentation.
G5.5	The contract enforces the access control rules specified in a trusted contract, especially if the dApp client-side access control is present and could be bypassed.
G5.6	Calls to external contracts are only allowed if necessary.
G5.7	Modifier code is clear and simple. The logic should not contain external calls to untrusted contracts.
G5.8	All user and data attributes used by access controls are kept in trusted contracts and cannot be manipulated by other contracts unless specifically authorized.
G5.9	the access controls fail securely, including when a revert occurs.
G5.10	If the input (function parameters) is validated, the positive validation approach (whitelisting) is used where possible.

Communication	Test Name
G6.2	Libraries that are not part of the application (but the smart contract relies on to operate) are identified.

G6.3	Delegate call is not used with untrusted contracts.
G6.4	Third party contracts do not shadow special functions (e.g. revert).
G6.5	The contract does not check whether the address is a contract using *extcodesize* opcode.
G6.6	Re-entrancy attacks are mitigated by blocking recursive calls from other contracts and following the Check-Effects-Interactions pattern. Do not use the *send* function unless it is a must.
G6.7	The result of low-level function calls (e.g. *send*, *delegatecall*, *call*) from other contracts is checked.
G6.8	Contract relies on the data provided by the right sender and does not rely on tx.origin value.

Arithmetic	**Test Name**
G7.2	The values and math operations are resistant to integer overflows. Use SafeMath library for arithmetic operations before solidity 0.8.*.
G7.3	the unchecked code snippets from Solidity ≥ 0.8.* do not introduce integer under/overflows.
G7.4	Extreme values (e.g. maximum and minimum values of the variable type) are considered and do not change the logic flow of the contract.
G7.5	Non-strict inequality is used for balance equality.
G7.6	Correct orders of magnitude are used in the calculations.
G7.7	In calculations, multiplication is performed before division for accuracy.
G7.8	The contract does not assume fixed-point precision and uses a multiplier or store both the numerator and denominator.

Denial of Service	**Test Name**
G8.2	The contract does not iterate over unbound loops.
G8.3	Self-destruct functionality is used only if necessary. If it is included in the contract, it should be clearly described in the documentation.
G8.4	The business logic isn't blocked if an actor (e.g. contract, account, oracle) is absent.
G8.5	The business logic does not disincentivize users to use contracts (e.g. the cost of transaction is higher than the profit).
G8.6	Expressions of functions assert or require have a passing variant.
G8.7	If the fallback function is not callable by anyone, it is not blocking contract functionalities.
G8.8	There are no costly operations in a loop.
G8.9	There are no calls to untrusted contracts in a loop.
G8.10	If there is a possibility of suspending the operation of the contract, it is also

	possible to resume it.
G8.11	If whitelists and blacklists are used, they do not interfere with normal operation of the system.
G8.12	There is no DoS caused by overflows and underflows.

Blockchain Data	Test Name
G9.2	Any saved data in contracts is not considered secure or private (even private variables).
G9.3	No confidential data is stored in the blockchain (passwords, personal data, token etc.).
G9.4	Contracts do not use string literals as keys for mappings. Global constants are used instead to prevent Homoglyph attack.
G9.5	Contract does not trivially generate pseudorandom numbers based on the information from blockchain (e.g. seeding with the block number).

Gas Usage and Limitations	Test Name
G10.2	Gas usage is anticipated, defined and has clear limitations that cannot be exceeded. Both code structure and malicious input should not cause gas exhaustion.
G10.3	Function execution and functionality does not depend on hard-coded gas fees (they are bound to vary).

Clarity and Readability	Test Name
G11.2	The logic is clear and modularized in multiple simple contracts and functions.
G11.3	Each contract has a short 1-2 sentence comment that explains its purpose and functionality.
G11.4	Off-the-shelf implementations are used, this is made clear in comment. If these implementations have been modified, the modifications are noted throughout the contract.
G11.5	The inheritance order is taken into account in contracts that use multiple inheritance and shadow functions.
G11.6	Where possible, contracts use existing tested code (e.g. token contracts or mechanisms like *ownable*) instead of implementing their own.
G11.7	Consistent naming patterns are followed throughout the project.
G11.8	Variables have distinctive names.
G11.9	All storage variables are initialized.
G11.10	Functions with specified return type return a value of that type.

G11.11	All functions and variables are used.
G11.12	*require* is used instead of *revert* in *if* statements.
G11.13	The *assert* function is used to test for internal errors and the *require* function is used to ensure a valid condition in input from users and external contracts.
G11.14	Assembly code is only used if necessary.

Test Coverage	Test Name
G12.2	Abuse narratives detailed in the threat model are covered by unit tests.
G12.3	Sensitive functions in verified contracts are covered with tests in the development phase.
G12.4	Implementation of verified contracts has been checked for security vulnerabilities using both static and dynamic analysis.
G12.5	Contract specification has been formally verified.
G12.6	The specification and results of the formal verification is included in the documentation.

Decentralized Finance	Test Name
G14.1	The lender's contract does not assume its balance (used to confirm loan repayment) to be changed only with its own functions.
G14.2	Functions that change lenders' balance and/or lend cryptocurrency are non-re-entrant if the smart contract allows borrowing the main platform's cryptocurrency (e.g. Ethereum). It blocks the attacks that update the borrower's balance during the flash loan execution.
G14.3	Flash loan functions can only call predefined functions on the receiving contract. If it is possible, define a trusted subset of contracts to be called. Usually, the sending (borrowing) contract is the one to be called back.
G14.4	If it includes potentially dangerous operations (e.g. sending back more ETH/tokens than borrowed), the receiver's function that handles borrowed ETH or tokens can be called only by the pool and within a process initiated by the receiving contract's owner or another trusted source (e.g. multisig).
G14.5	Calculations of liquidity pool share are performed with the highest possible precision (e.g. if the contribution is calculated for ETH it should be done with 18 digit precision - for Wei, not Ether). The dividend must be multiplied by the 10 to the power of the number of decimal digits (e.g. dividend * 10^18 / divisor).
G14.6	Rewards cannot be calculated and distributed within the same function call that deposits tokens (it should also be defined as non-re-entrant). This protects from momentary fluctuations in shares.
G14.7	Governance contracts are protected from flash loan attacks. One possible

	mitigation technique is to require the process of depositing governance tokens and proposing a change to be executed in different transactions included in different blocks.
G14.8	When using on-chain oracles, contracts are able to pause operations based on the oracles' result (in case of a compromised oracle).
G14.9	External contracts (even trusted ones) that are allowed to change the attributes of a project contract (e.g. token price) have the following limitations implemented: thresholds for the change (e.g. no more/less than 5%) and a limit of updates (e.g. one update per day).
G14.10	Contract attributes that can be updated by the external contracts (even trusted ones) are monitored (e.g. using events) and an incident response procedure is implemented (e.g. during an ongoing attack).
G14.11	Complex math operations that consist of both multiplication and division operations first perform multiplications and then division.
G14.12	When calculating exchange prices (e.g. ETH to token or vice versa), the numerator and denominator are multiplied by the reserves (see the *getInputPrice* function in the *UniswapExchange* contract).

Audit Findings

Max Token Allowance

ID	SAY-01
Status	Acknowledged
Risk	High
Business Impact	While this decision could perhaps be justified, it could be a bit dangerous. It could allow attackers to perform gigantic transactions, potentially draining the contract of funds.
Location	LendingLoglic.sol; lines 57-68; addXTokens(address, address)

Description	The function addXTokens explicitly sets the allowance of the token being added to max.
	
	• addXTokens(address, address):
	`if ((token) != ZERO_ADDRESS) {`
	`  IERC20Upgradeable(token).approve(xToken, type(uint256).max);`
	`  IERC20Upgradeable(token).approve(`
	`  multiLogicProxy,`
	`  type(uint256).max`
	`  );`
	`  approveTokenForSwap(swapGateway, token);`
	
	`  XTokens[token] = xToken;`
	`} else {`
	`  xETH = xToken;`
	`}`

Mitigation	Ideally, the allowance should be precisely set, depending on the size of transactions required.

Insufficient of Input Validation

ID	SAY-02
Status	Fixed
Risk	Medium
Business Impact	Lack of input validation can lead to unexpected behavior. For instance, the *setPercentages* function could validate that the *_token* address is not null before proceeding. Similarly, the *initStrategies* function could validate that the lengths of *_strategyName* and *_multiStrategy* are equal and are not zero.
Location	The entire codebase.
Description	Several functions such as *setPercentages*, *initStrategies*, *addStrategy*, *deactivateStrategy*, and *setLogicTokenAvailable* do not place validation for the incoming inputs. Similarly, in several functions in the Automation contract like *setKeeper*, *gelatoCheckerUseToken*, and *gelatoCheckerRebalance*, there is no check to ensure that the input address is not a zero address.
Mitigation	Implement input validation for all functions. This will ensure that the functions behave as expected and reduce the risk of vulnerabilities.

Inefficient Array Length Access

ID	SAY-03
Status	Invalidated
Risk	Low
Business Impact	In Solidity, accessing the length property of an array is a constant-time operation and does not cost additional gas each time it is accessed. Storing it in a local variable is unnecessary and leads to extra storage operations.
Location	LendBorrowFarmingPair.sol; line 194; findPath(uint256, address)

Description	The *findPath* function currently stores the length of the *reserve.path* array in a local variable before the loop.
	
	• findPath(uint256, address):
	`uint256 length = reserve.path.length;`
	
	`for (uint256 i = 0; i < length; ) {`
	`  if (reserve.path[i][reserve.path[i].length - 1] == token) {`
	`  return reserve.path[i];`
	`  }`
	`  unchecked {`
	`  ++i;`
	`  }`

Mitigation	The loop should be written like this, directly accessing the *length* property.
	`for (uint256 i = 0; i < reserve.path.length; ) {`
	`  ...`
	`}`

Unnecessary Initialization

ID	SAY-04
Status	Fixed
Risk	Low
Business Impact	This costs gas for each initialization and is not necessary since variables in Solidity are automatically initialized to their default value, which is zero for numeric types.
Location	StatisticsBase.sol; lines 46-49; getStorageAmount(address, address[])

Description	This contract unnecessarily initializes several variables to zero.
	
	• getStorageAmount(address, address[]):
	`strategyAmountUSD = 0;`
	`takenAmountUSD = 0;`
	`balanceUSD = 0;`
	`availableAmountUSD = 0;`

Mitigation	Just ditch the initialization and simply use your variables.

Lack of Event Emission in a Critical Function

ID	SAY-05
Status	Fixed
Risk	Informational
Business Impact	Event emissions can serve as an important way to track changes and actions in the contract. Events provide a way to trigger functionality and transfer information from the contract to security tools, dApps or analytics tools, enabling users or developers to react to specific contract changes.
Location	LendingLogic.sol; addXTokens(address, address)
Description	The specified function, although it appears to fill a very important role (namely, adding new tokens to be deposited), emits no event.
Mitigation	Simply add an event for this function.