CERTIK

StakeWith.Us

Vault Refactor

Security Assessment

February 19th, 2021

By:

Alex Papageorgiou @ CertiK

alex.papageorgiou@certik.org

Camden Smallwood @ CertiK

camden.smallwood@certik.org

Disclaimer

CertiK reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security review.

CertiK Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

Project Summary

Project Name	StakeWith.Us Vault Refactor
Description	Round three audit of the StakeWith.Us vault implementation codebase, refactored to support ETH and a set of new strategies.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. a0092d5898ec9a2c55dd658f5f415ebeccf8b05d 2. bd3b3df8d8db58a136daab3dfdf8646f293b0a5f 3. 2e3f151a95600cc8a955bbb717a6cc0ef9e48777

Audit Summary

Delivery Date	February 19th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	2
Timeline	February 15th, 2021 - February 19th, 2021

Vulnerability Summary

Total Issues	13
● **Total Critical**	0
● **Total Major**	2
● **Total Medium**	0
● **Total Minor**	9
● **Total Informational**	3

Executive Summary

We were tasked with auditing the updated `uvault` codebase that contained revamped generalized vault and strategy mechanisms as well as introduced a set of new strategies that also handle accrued reward tokens from the strategy deposits.

Over the course of the audit, we validated that state transitions occur within reasonable bounds and that the strategies implemented under `contracts/strategies` are properly curated towards their designated pools i.e. strategies containing a Gauge V2 implementation prevent transfers of the Gauge token, reward tokens accrued are handled properly etc.

We were able to pinpoint 2 logical issues in the way funds are handled during partial withdrawals in both the ETH and ERC20 generalized strategy implementations that we urge the StakeWith.Us team to fix as soon as possible. Additionally, we were able to pinpoint certain inconsistencies in the way access control is managed across a set of functions in both certian strategy implementations as well as certain generalized functions that we believe should be remediated before a full launch of the new `uvault` implementation.

On the optimizational side, we pinpointed 2 ways that almost all strategies can be optimized via, greatly reducing the gas cost involved in unit conversions and detection of the most premium token.

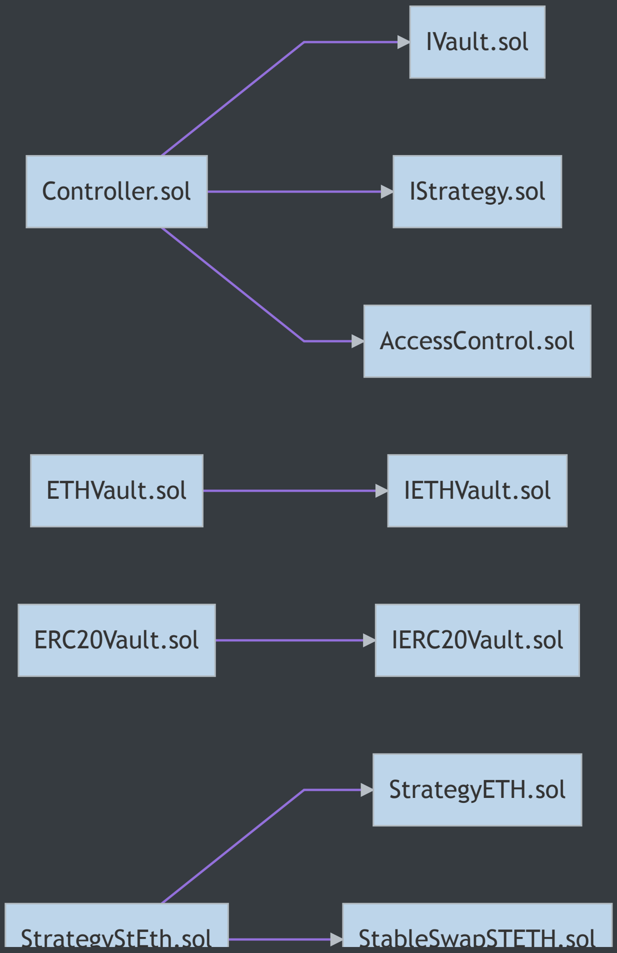
Ultimately, the codebase has no observable flaws in its design and has been developed conforming to the latest standards and security guidelines. While it should be noted that the Checks-Effects-Interactions pattern is not followed closely in certain segments, it is done so with relation to cross-contract interactions within the `uvault` system rather than with external parties and thus can be considered safe of any form of re-entrancies.

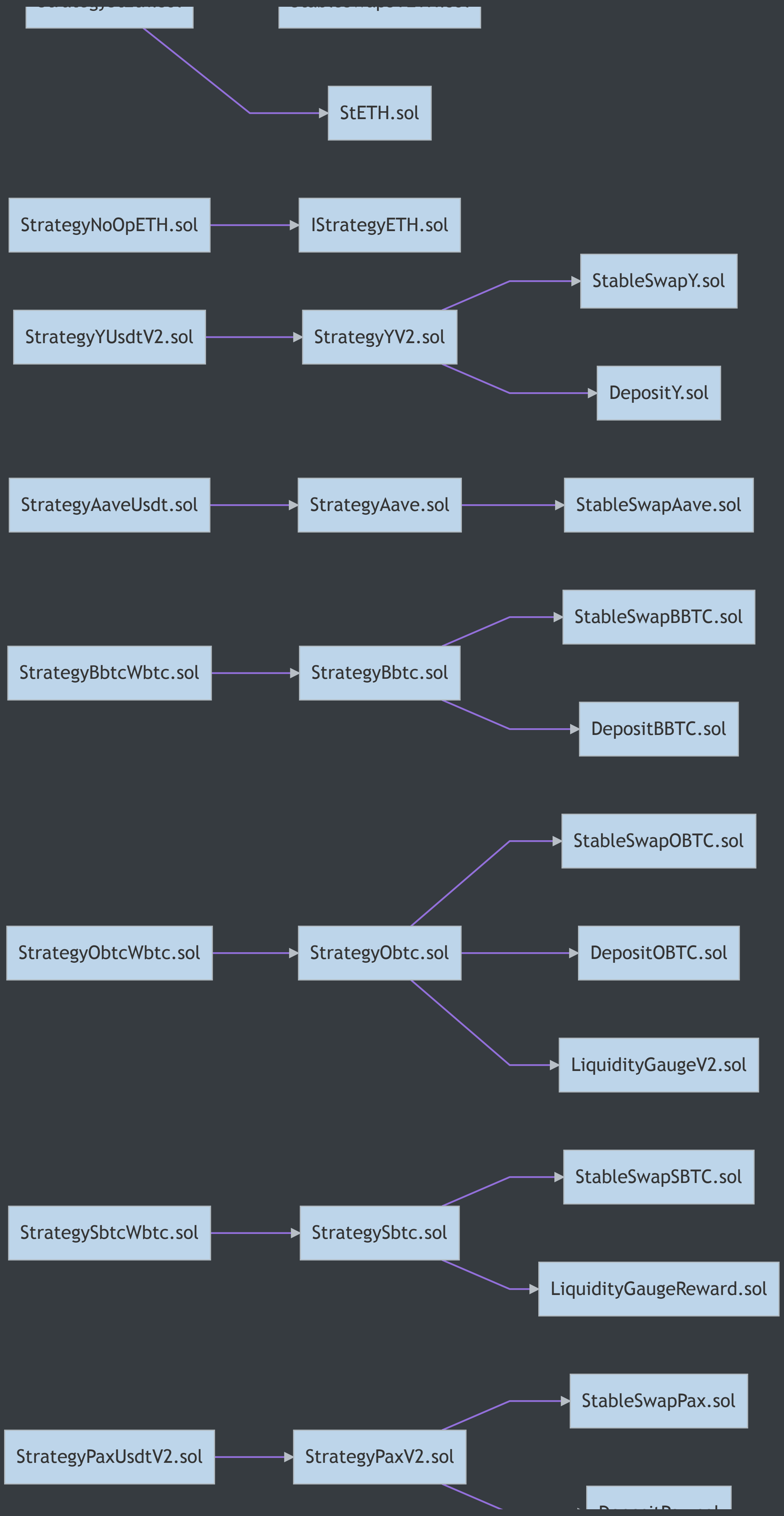
Files In Scope

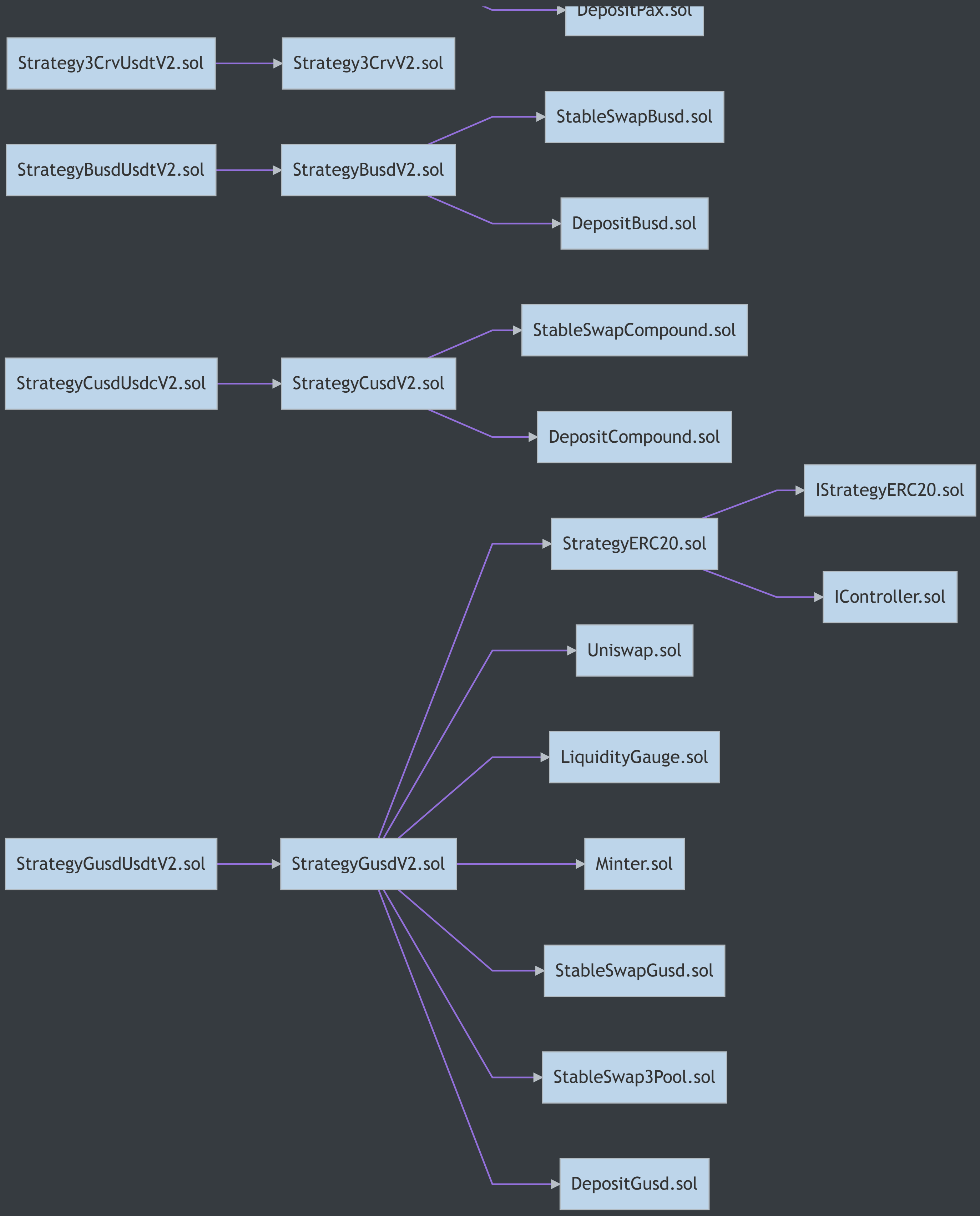
ID	Contract	Location
CON	Controller.sol	contracts/Controller.sol
ETH	ETHVault.sol	contracts/ETHVault.sol
ERC	ERC20Vault.sol	contracts/ERC20Vault.sol
SET	StrategyETH.sol	contracts/StrategyETH.sol
SYV	StrategyYV2.sol	contracts/strategies/StrategyYV2.sol
SAE	StrategyAave.sol	contracts/strategies/StrategyAave.sol
SBC	StrategyBbtc.sol	contracts/strategies/StrategyBbtc.sol
SOC	StrategyObtc.sol	contracts/strategies/StrategyObtc.sol
SSC	StrategySbtc.sol	contracts/strategies/StrategySbtc.sol
SER	StrategyERC20.sol	contracts/StrategyERC20.sol
SPV	StrategyPaxV2.sol	contracts/strategies/StrategyPaxV2.sol
SSE	StrategyStEth.sol	contracts/strategies/StrategyStEth.sol
SCV	Strategy3CrvV2.sol	contracts/strategies/Strategy3CrvV2.sol
SBV	StrategyBusdV2.sol	contracts/strategies/StrategyBusdV2.sol
CON	StrategyCusdV2.sol	contracts/strategies/StrategyCusdV2.sol
SGV	StrategyGusdV2.sol	contracts/strategies/StrategyGusdV2.sol
SYD	StrategyYDaiV2.sol	contracts/strategies/StrategyYDaiV2.sol
SAD	StrategyAaveDai.sol	contracts/strategies/StrategyAaveDai.sol
SNO	StrategyNoOpETH.sol	contracts/strategies/StrategyNoOpETH.sol
SYU	StrategyYUsdcV2.sol	contracts/strategies/StrategyYUsdcV2.sol
SUV	StrategyYUsdtV2.sol	contracts/strategies/StrategyYUsdtV2.sol
SAU	StrategyAaveUsdc.sol	contracts/strategies/StrategyAaveUsdc.sol
CON	StrategyAaveUsdt.sol	contracts/strategies/StrategyAaveUsdt.sol

SBW	StrategyBbtcWbtc.sol	contracts/strategies/StrategyBbtcWbtc.sol
SOW	StrategyObtcWbtc.sol	contracts/strategies/StrategyObtcWbtc.sol
SPD	StrategyPaxDaiV2.sol	contracts/strategies/StrategyPaxDaiV2.sol
SSW	StrategySbtcWbtc.sol	contracts/strategies/StrategySbtcWbtc.sol
SCD	Strategy3CrvDaiV2.sol	contracts/strategies/Strategy3CrvDaiV2.sol
SBD	StrategyBusdDaiV2.sol	contracts/strategies/StrategyBusdDaiV2.sol
SDV	StrategyCusdDaiV2.sol	contracts/strategies/StrategyCusdDaiV2.sol
SGD	StrategyGusdDaiV2.sol	contracts/strategies/StrategyGusdDaiV2.sol
SNE	StrategyNoOpERC20.sol	contracts/strategies/StrategyNoOpERC20.sol
SPU	StrategyPaxUsdcV2.sol	contracts/strategies/StrategyPaxUsdcV2.sol
PUV	StrategyPaxUsdtV2.sol	contracts/strategies/StrategyPaxUsdtV2.sol
SCU	Strategy3CrvUsdcV2.sol	contracts/strategies/Strategy3CrvUsdcV2.sol
CUV	Strategy3CrvUsdtV2.sol	contracts/strategies/Strategy3CrvUsdtV2.sol
SBB	StrategyBusdBusdV2.sol	contracts/strategies/StrategyBusdBusdV2.sol
SBU	StrategyBusdUsdcV2.sol	contracts/strategies/StrategyBusdUsdcV2.sol
BUV	StrategyBusdUsdtV2.sol	contracts/strategies/StrategyBusdUsdtV2.sol
CON	StrategyCusdUsdcV2.sol	contracts/strategies/StrategyCusdUsdcV2.sol
SGG	StrategyGusdGusdV2.sol	contracts/strategies/StrategyGusdGusdV2.sol
SGU	StrategyGusdUsdcV2.sol	contracts/strategies/StrategyGusdUsdcV2.sol
GUV	StrategyGusdUsdtV2.sol	contracts/strategies/StrategyGusdUsdtV2.sol
ZSE	ZapStEth.vy	repo/ZapStEth.vy

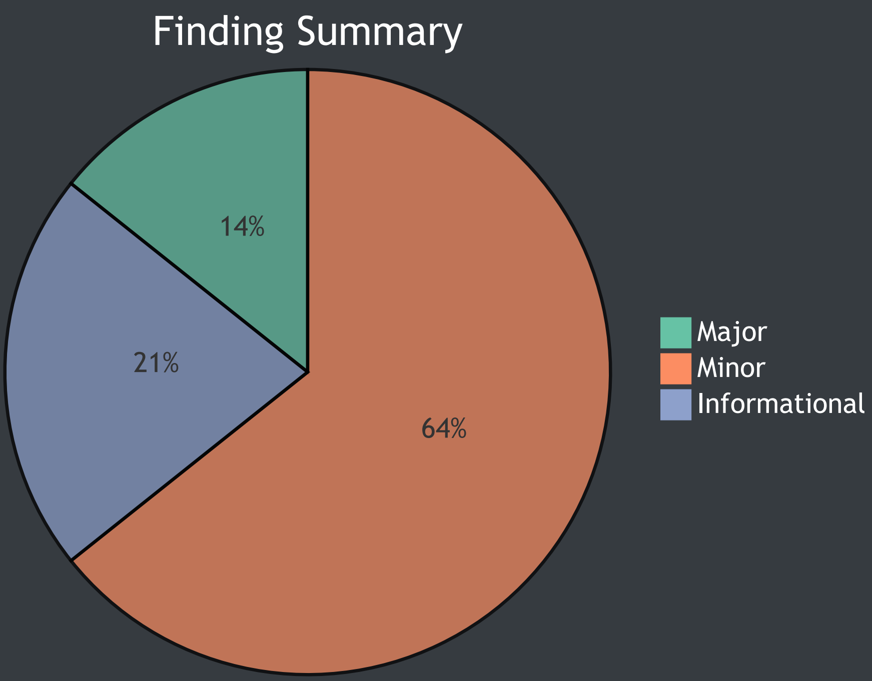
File Dependency Graph




StETH.sol
StrategyNoOpETH.sol
IStrategyETH.sol
StableSwapY.sol
StrategyYUsdtV2.sol
StrategyYV2.sol
DepositY.sol
StrategyAaveUsdt.sol
StrategyAave.sol
StableSwapAave.sol
StableSwapBBTC.sol
StrategyBbtcWbtc.sol
StrategyBbtc.sol
DepositBBTC.sol
StableSwapOBTC.sol
StrategyObtcWbtc.sol
StrategyObtc.sol
DepositOBTC.sol
LiquidityGaugeV2.sol
StableSwapSBTC.sol
StrategySbtcWbtc.sol
StrategySbtc.sol
LiquidityGaugeReward.sol
StableSwapPax.sol
StrategyPaxUsdtV2.sol
StrategyPaxV2.sol


DepositPax.sol
Strategy3CrvUsdtV2.sol
Strategy3CrvV2.sol
StableSwapBusd.sol
StrategyBusdUsdtV2.sol
StrategyBusdV2.sol
DepositBusd.sol
StableSwapCompound.sol
StrategyCusdUsdcV2.sol
StrategyCusdV2.sol
DepositCompound.sol
IStrategyERC20.sol
StrategyERC20.sol
IController.sol
Uniswap.sol
LiquidityGauge.sol
StrategyGusdUsdtV2.sol
StrategyGusdV2.sol
Minter.sol
StableSwapGusd.sol
StableSwap3Pool.sol
DepositGusd.sol

Findings



ID	Title	Type	Severity	Resolved
SER-01	Incorrect Underlying Withdrawal	Logical Issue	● Major	✓
SER-02	Redundant Statement	Logical Issue	● Minor	✓
SET-01	Incorrect Underlying Withdrawal	Logical Issue	● Major	✓
SET-02	Redundant Statement	Logical Issue	● Minor	✓
ETH-01	Potential Devaluation of Deposits	Language Specific	● Minor	✓

CON-01	Inconsistent Access Control	Control Flow	● Minor	✓
SCV-01	Mutability Optimization	Gas Optimization	● Informational	✓
SCV-02	Inefficient Identification of Most Premium Token	Gas Optimization	● Informational	✓
SOC-01	Insufficient Token Protection	Logical Issue	● Minor	✓
SNO-01	Redundant `require` Check	Gas Optimization	● Informational	✓
SSE-01	Inefficient Token Protection	Logical Issue	● Minor	✓
SSC-01	Unutilized Reward Token	Logical Issue	● Minor	✓
SSC-02	Inefficient Token Protection	Logical Issue	● Minor	✓
ZSE-01	Unchecked Function Return	Logical Issue	● Minor	✓

SER-01: Incorrect Underlying Withdrawal

Type	Severity	Location
Logical Issue	● Major	StrategyERC20.sol L202-L206

Description:

The linked code segment is located within the `withdraw` function which is meant to withdraw the input `_underlyingAmount` from the system by calculating the `shares` associated with it, withdrawing them and consequently decreasing the underlying debt. The current implementation, however, ignores the `_underlyingAmount` value passed in and withdraws the full balance regardless of the amount of underlying set to be withdrawn, causing the system to come to a state of desync.

Recommendation:

We advise that the `_underlyingAmount` variable is properly passed in to the `_decreaseDebt` function and that the preceding `balanceOf` getter and surrounding `if` clause are omitted from the segment as `_underlyingAmount` is guaranteed to be above `0` at L193.

Alleviation:

Refer to SET-01.

SER-02: Redundant Statement

Type	Severity	Location
Logical Issue	● Minor	StrategyERC20.sol L218

Description:

The linked assignment zeroes out the `totalDebt` variable right after a `_decreaseDebt` invocation with the full `balanceOf` of the contract.

Recommendation:

As the `totalDebt` represents the balance that has been deposited to the strategies, the invocation of `_decreaseDebt` with the full balance held by the contract after burning the shares should reduce `totalDebt` to `0` unless the withdrawn funds from the strategy are actually less than the debt. In the latter case, the debt should properly be reflected on the contract even after complete withdrawal of shares, so the linked statement should be omitted from the codebase.

Alleviation:

Refer to SET-02.

SET-01: Incorrect Underlying Withdrawal

Type	Severity	Location
Logical Issue	● Major	StrategyETH.sol L189-L193

Description:

The linked code segment is located within the `withdraw` function which is meant to withdraw the input `_ethAmount` from the system by calculating the `shares` associated with it, withdrawing them and consequently decreasing the underlying debt. The current implementation, however, ignores the `_ethAmount` value passed in and withdraws the full balance regardless of the amount of underlying set to be withdrawn, causing the system to come to a state of desync.

Recommendation:

We advise that the `_ethAmount` variable is properly passed in to the `_decreaseDebt` function and that the preceding `address(this).balance` getter and surrounding `if` clause are omitted from the segment as `_ethAmount` is guaranteed to be above `0` at L180.

Alleviation:

The StakeWith.Us team has stated that they expect the possibility of `_ethAmount > ethBal` to be achievable as an edge case and thus the statements within the code block are meant to guarantee that only the actual balance is being decreased as debt. It is assumed the `vault`, `controller` or `admin` that will invoke the function will do so taking care the security consideration laid out in this exhibit's description. To this end, comments were added accompanying the function and explicitly stating the dangers we outlined.

SET-02: Redundant Statement

Type	Severity	Location
Logical Issue	● Minor	StrategyETH.sol L206

Description:

The linked assignment zeroes out the `totalDebt` variable right after a `_decreaseDebt` invocation with the full `address(this).balance` of the contract.

Recommendation:

As the `totalDebt` represents the balance that has been deposited to the strategies, the invocation of `_decreaseDebt` with the full balance held by the contract after burning the shares should reduce `totalDebt` to `0` unless the withdrawn funds from the strategy are actually less than the debt. In the latter case, the debt should properly be reflected on the contract even after complete withdrawal of shares, so the linked statement should be omitted from the codebase.

Alleviation:

The code block within the `if` clause was adjusted to replicate an invocation of `_decreaseDebt` with a full debt removal, in doing so setting the `totalDebt` to zero as the ability to be able to fully reset debt was desired by the StakeWith.Us team.

ETH-01: Potential Devaluation of Deposits

Type	Severity	Location
Language Specific	● Minor	ETHVault.sol L146-L148

Description:

The linked statement is utilized by other sub-functions such as `_totalAssets`, utilizing it as a divisor for example in share minting. The issue with the variable is that it is assumed the `require` check imposed on the `receive` function is sufficient to prevent deposits of ether by external sources, which is not the case and can lead to permanent devaluation of all deposits if ether is forcibly deposited to the contract.

Recommendation:

We advise that a non-dynamic variable, such as a state variable incremented during `receive`, is utilized instead of the `address(this).balance` which can deviate from the actual values deposited to the contracts i.e. due to forcibly deposited ether due to a `selfdestruct` instruction.

Alleviation:

After discussing the issue at hand with the StakeWith.Us team, they concluded that the changes necessary to alleviate this issue are significant and the game theory behind this exploitation vector, which is also prevalent in the ERC20 equivalent contract, does not incentivize it to be exploited as an attacker would be able to acquire more profit by depositing correctly rather than forcing ETH / ERC20s to the contract and expect consequent shares to arrive and overvalue their deposit. As a result, we concluded that the issue at hand does not need to be remediated for the security of the project.

CON-01: Inconsistent Access Control

Type	Severity	Location
Control Flow	● Minor	Controller.sol L133-L139, L157-L164

Description:

The linked functions interact with a `_vault` without validating it is an approved vault via `onlyApprovedVault` as `invest` does.

Recommendation:

We advise that the same modifier is also applied to the linked functions to ensure consistent access control is imposed on the codebase as even though the roles evaluated by the `onlyAuthorized` modifier differ, the `ADMIN_ROLE` does not necessarily mean that it can authorize new vaults since the `approveVault` function utilizes the `onlyAdmin` modifier which in turns evaluates the `admin` of the contract rather than a role which can be arbitrarily granted via `grantRole` .

Alleviation:

Proper `modifier` invocations were introduced in the function declarations to ensure that this issue has been alleviated.

SCV-01: Mutability Optimization

Type	Severity	Location
Gas Optimization	● Informational	Strategy3CrvV2.sol L22

Description:

The linked declaration `PRECISION_DIV` is meant to provide the precisions for each respective token based on the `underlyingIndex` , however, it is not `constant` and utilizes redundant gas.

Recommendation:

We advise that the `underlyingIndex` is passed to the `constructor` of the linked contract, the array of L22 is instead declared in-memory on the `constructor` of the contract and a new `immutable` variable is introduced that holds the precision divisor and is initialized during the `constructor` . This will significantly reduce the gas cost involved with utilizing the variable. This optimization can be replicated across multiple strategies but will only be mentioned here for brevity.

Alleviation:

The `UNDERLYING_INDEX` variable was properly set to `immutable` benefitting greatly in terms of gas optimization.

SCV-02: Inefficient Identification of Most Premium Token

Type	Severity	Location
Gas Optimization	● Informational	Strategy3CrvV2.sol L130-L143

Description:

As the comparison is only done between 3 tokens, it is possible to optimize the linked segment by structuring valid `if` statements that capture all cases and are more optimal.

Recommendation:

We advise that the recommendation laid out in the description is followed to optimize the gas cost of the function. This optimization can be replicated across multiple strategies but will only be mentioned here for brevity.

Alleviation:

The identification approach was altered to not use a `for` loop and instead use `if` clauses directly optimizing the gas cost involved in executing it.

SOC-01: Insufficient Token Protection

Type	Severity	Location
Logical Issue	● Minor	StrategyObtc.sol L329-L333

Description:

The linked `sweep` mechanism is meant to protect withdrawal of tokens utilized by the strategy, however, it fails to guard against withdrawals of `BOR` tokens that were left in the contract either due to the `shouldSellBor` flag being `false` or due to them being indirectly accrued as rewards from Gauge `deposit` or `withdraw` invocations.

Recommendation:

We advise that the function properly guards against the withdrawal of `BOR` by the administrator.

Alleviation:

A corresponding `require` check was introduced that properly prevents the `BOR` token from being withdrawn from the contract.

SNO-01: Redundant `require` Check

Type	Severity	Location
Gas Optimization	● Informational	StrategyNoOpETH.sol L118

Description:

The linked `require` check ensures the `_token` to be withdrawn is not equal to the `underlying` token, however, the `underlying` token in this `NoOp` is a placeholder as ETH cannot be withdrawn via `transfer` invocations. As such, the `require` check can be safely omitted.

Recommendation:

We advise that the redundant check is removed from the codebase.

Alleviation:

The `require` check was indeed removed from the `sweep` function as outlined in the finding's recommendation.

SSE-01: Inefficient Token Protection

Type	Severity	Location
Logical Issue	● Minor	StrategyStEth.sol L242-L245

Description:

The linked `sweep` mechanism is meant to protect withdrawal of tokens utilized by the strategy, however, it fails to guard against withdrawals of `LDO` tokens that were left in the contract due to them being indirectly accrued as rewards from Gauge `deposit` or `withdraw` invocations.

Recommendation:

We advise that the function properly guards against the withdrawal of `LDO` by the administrator.

Alleviation:

A corresponding `require` check was added to guard against withdrawals of the `LDO` token in a similar fashion to other such strategies.

SSC-01: Unutilized Reward Token

Type	Severity	Location
Logical Issue	● Minor	StrategySbtc.sol L179-L190

Description:

The `Sbtc` gauge rewards `BPT` (Balancer Pool Tokens) to the callers of `claim_rewards` as well as indirectly via deposits and withdrawals. This token remains unutilized by the contract and potentially forever locked.

Recommendation:

We advise a handling mechanism similar to how `LDO` is handled on the `Steth` vault is introduced to ensure no funds are wasted by the strategy.

Alleviation:

The StakeWith.Us team stated that it is indeed the intention for the administrator to be able to sweep the `BPT` token as their rewards have expired and the liquidity is very low on decentralized exchanges. To this end, this exhibit can be considered null.

SSC-02: Inefficient Token Protection

Type	Severity	Location
Logical Issue	● Minor	StrategySbtc.sol L229-L232

Description:

The linked `sweep` mechanism is meant to protect withdrawal of tokens utilized by the strategy, however, it fails to guard against withdrawals of `BPT` tokens that were left in the contract either due to them not being properly handled or due to them being indirectly accrued as rewards from Gauge `deposit` or `withdraw` invocations.

Recommendation:

We advise that the function properly guards against the withdrawal of `BPT` by the administrator.

Alleviation:

Refer to SSC-01.

ZSE-01: Unchecked Return Value

Type	Severity	Location
Logical Issue	● Minor	ZapStEth.vy L42-L85

Description:

The linked code block contains unchecked `transfer` and `transferFrom` invocations that should be asserted to be `true`.

Recommendation:

We advise that the results are appropriately validated via an `assert` instruction.

Alleviation:

All `transfer` and `transferFrom` invocations were adjusted to be properly validated in the linked code block. We should note that the `StEth` token as well as the vault itself are fully compatible with the ERC20 standard and thus do return a `bool` so there is no need to use a wrapper library in this case.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings refer to exhibits that do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation exhibits entail findings that relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings are exhibits that detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

Control Flow

Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Data Flow

Data Flow findings describe faults in the way data is handled at rest and in memory, such as the result of a `struct` assignment operation affecting an in-memory `struct` rather than an in-storage one.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.

Magic Numbers

Magic Number findings refer to numeric literals that are expressed in the codebase in their raw format and should otherwise be specified as `constant` contract variables aiding in their legibility and maintainability.

Compiler Error

Compiler Error findings refer to an error in the structure of the code that renders it impossible to compile using the specified version of the project.

Dead Code

Code that otherwise does not affect the functionality of the codebase and can be safely omitted.