



Swell Network

Vault

SMART CONTRACT AUDIT

08.05.2022

Made in Germany by Chainsulting.de



Table of contents

1. Disclaimer.....	4
2. About the Project and Company	5
2.1 Project Overview.....	6
3. Vulnerability & Risk Level	7
4. Auditing Strategy and Techniques Applied.....	8
4.1 Methodology	8
5. Metrics	9
5.1 Tested Contract Files	9
5.2 Used Code from other Frameworks/Smart Contracts	10
5.3 CallGrap	11
5.4 Inheritance Graph	12
5.5 Source Lines & Risk.....	13
5.6 Capabilities	14
5.7 Source Unites in Scope	15
6. Scope of Work.....	17
6.1 Findings Overview	18
6.2 Manual and Automated Vulnerability Test.....	19
6.2.1 Rounding Errors Can Lead To Ignore Slippage	19
6.2.2 Racing Condition in Approve Function	20
6.2.3 Missing Zero Address Checks	21
6.2.4 For Loop Over Dynamic Array	22
6.2.5 Missing Value Verification.....	23

6.2.6 Floating And Multiple Pragma Version Identified	25
6.2.7 Unnecessary Call To The Balancer Vault For The Pool Address	25
6.2.8 Missing Natspec Documentation	26
6.3 SWC Attacks	27
6.4. Verify Claims	31
7. Executive Summary.....	32
8. Deployed Smart Contract	32
9. About the Auditor	33

1. Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of DL Labs Pte. Ltd.. If you are not the intended receptor of this document, remember that any disclosure, copying or dissemination of it is forbidden.

Major Versions / Date	Description
0.1 (01.05.2022)	Layout
0.4 (03.05.2022)	Automated Security Testing Manual Security Testing
0.5 (05.05.2022)	Verify Claims and Test Deployment
0.6 (06.05.2022)	Testing SWC Checks
0.9 (07.05.2022)	Summary and Recommendation
1.0 (07.05.2022)	Final document
1.1 (08.06.2022)	Re-check
1.2 (TBA)	Added deployed contract

2. About the Project and Company

Company address:

DL Labs Pte. Ltd.
Reg.: 202204142H
20 Tanjong Pagar Road
Singapore 088443



Website: <https://www.swellnetwork.io>

Twitter: <https://twitter.com/swellnetworkio>

Discord: <https://discord.gg/SeMQbGbeqC>

Medium: <https://medium.com/swell-network>

Forum: <https://forum.swellnetwork.io>

2.1 Project Overview

Swell delivers fast, simple and liquid staking. Swell Network is a decentralized, open, liquid, non-custodial, Ethereum staking DeFi protocol. Swell Network is organised as a Decentralised Autonomous Organisation (DAO). In return for staking ether, you receive a liquid derivative token (swETH which is pegged 1:1 to ether.) that can be used across DeFi to compound yield. Swell eliminates the complexity of setting up a validator and managing your own infrastructure or needing to have 32 ETH requirements.

Swell network supports 3 key pillars

- (a) Liquid Staking
- (b) DPools (decentralised mini pools)
- (c) Decentralised marketplace.

The connectivity between swETH and the staked ether is maintained by the sWETH protocol which factors in the total amount of staked ether, level of staking rewards, and any adjustments including any slashing penalties.

3. Vulnerability & Risk Level

Risk represents the probability that a certain source-threat will exploit vulnerability, and the impact of that event on the organization or system. Risk Level is computed based on CVSS version 3.0.

Level	Value	Vulnerability	Risk (Required Action)
Critical	9 – 10	A vulnerability that can disrupt the contract functioning in a number of scenarios, or creates a risk that the contract may be broken.	Immediate action to reduce risk level.
High	7 – 8.9	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.	Implementation of corrective actions as soon as possible.
Medium	4 – 6.9	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.	Implementation of corrective actions in a certain period.
Low	2 – 3.9	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.	Implementation of certain corrective actions or accepting the risk.
Informational	0 – 1.9	A vulnerability that have informational character but is not effecting any of the code.	An observation that does not determine a level of risk

4. Auditing Strategy and Techniques Applied

Throughout the review process, care was taken to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. To do so, reviewed line-by-line by our team of expert pentesters and smart contract developers, documenting any issues as there were discovered.

4.1 Methodology

The auditing process follows a routine series of steps:

1. Code review that includes the following:
 - i. Review of the specifications, sources, and instructions provided to Chainsulting to make sure we understand the size, scope, and functionality of the smart contract.
 - ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
 - iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Chainsulting describe.
2. Testing and automated analysis that includes the following:
 - i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
 - ii. Symbolic execution, which is analysing a program to determine what inputs causes each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

5. Metrics

The metrics section should give the reader an overview on the size, quality, flows and capabilities of the codebase, without the knowledge to understand the actual code.

5.1 Tested Contract Files

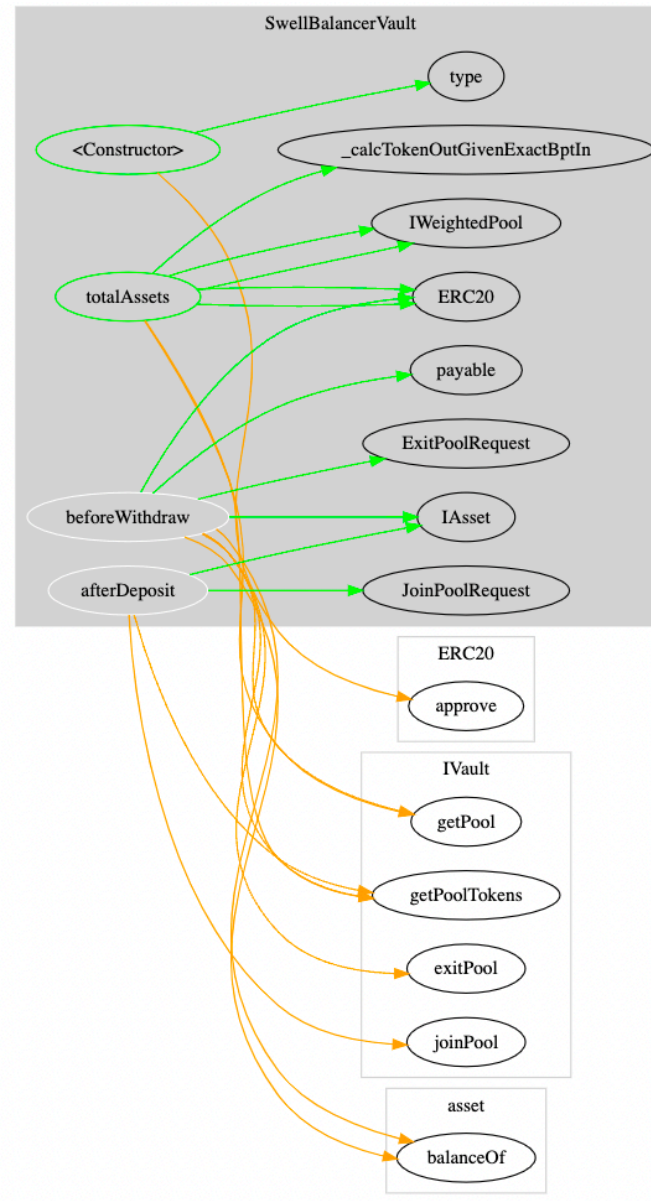
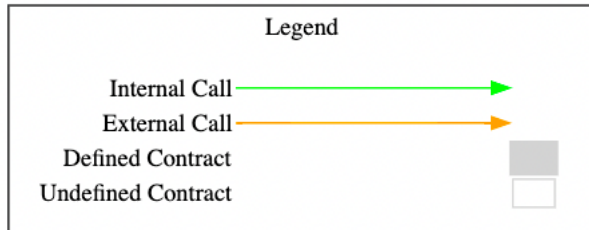
The following are the MD5 hashes of the reviewed files. A file with a different MD5 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different MD5 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review

File	Fingerprint (MD5)
./vendor/balancer/IWeightedPool.sol	9ae8c4bdc5028673b5c66789a39d0738
./vendor/balancer/IBalancerWeightedPoolFactory.sol	4f3e273b37f33052344a370021d6fbd9
./vendor/balancer/IVault.sol	742fc93526f9b61908834baff68fc85f
./vendor/balancer/WeightedMath.sol	5ad6523d74b6ff8583540401028e76bc
./vendor/IWETH.sol	2841be726fe26b7eb3ca9daf02bde3ba
./vendor/rari-capital/ERC4626.sol	fb3e772f6d1e578cf89bb915d6df89ad
./vendor/rari-capital/SafeTransferLib.sol	55308ff0738cc0ff1971818b394ed36f
./vendor/rari-capital/FixedPointMathLib.sol	61bb69231506158b2939554bc5bbeaa0
./vendor/rari-capital/ERC20.sol	e3569362f01c1388bee722f8eaa17337
./implementations/TestToken.sol	25c2a03faed51f11ccb1153a4cd81a14
./implementations/SwellBalancerVault.sol	0e764a507d7bff0dbe8ad4996e389b11

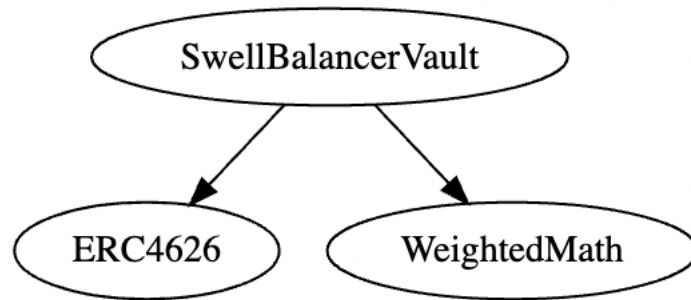
5.2 Used Code from other Frameworks/Smart Contracts (direct imports)

Dependency / Import Path	Source
@openzeppelin/contracts/token/ERC20/ERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.5.0/contracts/token/ERC20/ERC20.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/tree/v4.5.0/contracts/token/ERC20/IERC20.sol

5.3 CallGrap

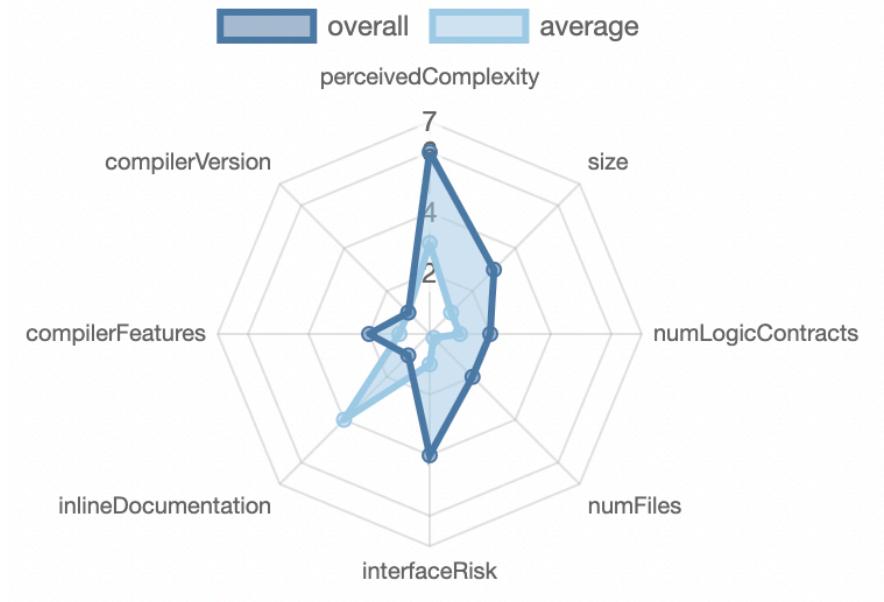


5.4 Inheritance Graph



5.5 Source Lines & Risk

source comment single block mixed
empty todo blockEmpty





5.6 Capabilities


Solidity Versions observed		 Experimental Features		 Can Receive Funds		 Uses Assembly		 Has Destroyable Contracts	
<div><div>^0.8.0</div><div>^0.8.13</div><div>>=0.8.0</div></div>				<div>yes</div>		<div>yes</div> <div>(10 asm blocks)</div>			
 Transfers ETH		 Low-Level Calls		 DelegateCall		 Uses Hash Functions		 ECRecover	
						<div>yes</div>		<div>yes</div>	
								 New/Create/Create2	

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

 Public	 Payable				
38	2				
External	Internal	Private	Pure	View	
17	84	2	49	23	

















StateVariables

Total	 Public
179	10

5.7 Source Unites in Scope

Source: <https://github.com/SwellNetwork/vault>

Last commit: 566e0c8edbb86e20d592e448000ff4d5b38300de

Type	File	Logic Contract s	Interface s	Lin es	nLin es	nSL OC	Com ment Lines	Comp lex. Score	Capabili ties
	contracts/vendor/balancer/IWeightedPool.sol	_____	1	10	5	3	1	7	_____
	contracts/vendor/balancer/IBalancerWeightedPoolFactory.sol	_____	1	17	9	5	1	3	_____
	contracts/vendor/balancer/IVault.sol	_____	2	205	63	35	122	17	
 	contracts/vendor/balancer/WeightedMath.sol	6	1	1540	1419	760	502	515	
	contracts/vendor/IWETH.sol	_____	1	8	7	4	1	8	
	contracts/vendor/rari-capital/ERC4626.sol	1	_____	227	180	103	33	83	_____
	contracts/vendor/rari-capital/SafeTransferLib.sol	1	_____	144	131	73	43	173	
	contracts/vendor/rari-capital/FixedPointMathLib.sol	1	_____	232	220	140	65	456	
	contracts/vendor/rari-capital/ERC20.sol	1	_____	227	207	121	40	50	

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
	contracts/implementations/TestToken.sol	1	_____	10	10	7	2	5	_____
	contracts/implementations/SwellBalanceVault.sol	1	_____	144	135	105	14	120	_____
	Totals	12	6	2764	2386	1356	824	1437	

Legend: []

- **Lines:** total lines of the source unit
- **nLines:** normalized lines of the source unit (e.g. normalizes functions spanning multiple lines)
- **nSLOC:** normalized source lines of code (only source-code lines; no comments, no blank lines)
- **Comment Lines:** lines containing single or block comments
- **Complexity Score:** a custom complexity score derived from code statements that are known to introduce code complexity (branches, loops, calls, external interfaces, ...)

6. Scope of Work

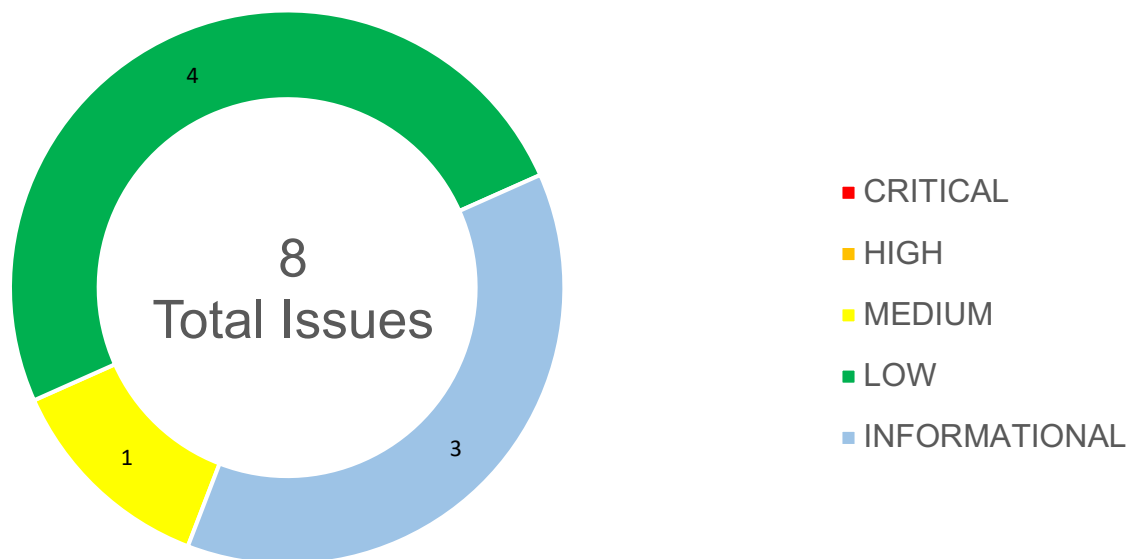
The Swell Network Team provided us with the files that needs to be tested. The scope of the audit is the vault contract.

The team put forward the following assumptions regarding the security, usage of the contracts:

- Implementations of the vendors Balancer and Rari-Capital are safe to use
- Deposits are working as expected
- Withdrawals are working as expected
- The smart contract is coded according to the newest standards and in a secure way.

The main goal of this audit was to verify these claims. The auditors can provide additional feedback on the code upon the client's request.

6.1 Findings Overview



No	Title	Severity	Status
6.2.1	Rounding Errors Can Lead To Ignore Slippage	MEDIUM	ACKNOWLEDGED
6.2.2	Racing Condition in Approve Function	LOW	CLOSED
6.2.3	Missing Zero Address Checks	LOW	CLOSED
6.2.4	For Loop Over Dynamic Array	LOW	ACKNOWLEDGED
6.2.5	Missing Value Verification	LOW	CLOSED
6.2.6	Floating And Multiple Pragma Version Identified	INFORMATIONAL	CLOSED
6.2.7	Unnecessary Call To The Balancer Vault For The Pool Address	INFORMATIONAL	CLOSED
6.2.8	Missing Natspec Documentation	INFORMATIONAL	ACKNOWLEDGED

6.2 Manual and Automated Vulnerability Test

CRITICAL ISSUES

During the audit, Chainsulting's experts found **0 Critical issues** in the code of the smart contract.

HIGH ISSUES

During the audit, Chainsulting's experts found **0 High issues** in the code of the smart contract.

MEDIUM ISSUES

During the audit, Chainsulting's experts found **1 Medium issue** in the code of the smart contract.

6.2.1 Rounding Errors Can Lead To Ignore Slippage

Severity: MEDIUM

Status: ACKNOWLEDGED

Code: NA

File(s) affected: SwellBalancerVault.sol

Update: Assets amount to withdraw cannot be less than 100 due to the pool balancer revert condition (EXIT_BELOW_MIN) is triggered. A unit test was added to testify this.

Attack / Description	In the beforeWithdraw hook, the amountsOut[i] is calculated by removing a 1% of slippage. If assets are less than 100 the results of amountsOut[i] will be equal to, assets thus ignoring the slippage.
Code	Line 87 - 96 (SwellBalancerVault.sol) <pre>IAsset[] memory assetList = new IAsset[](tokens.length); uint256[] memory amountsOut = new uint256[](tokens.length); uint256 exitTokenIndex; for (uint256 i; i < tokens.length; i++) {</pre>

	<pre> assetList[i] = IAsset(address(tokens[i])); if (address(tokens[i]) == address(asset)) { amountsOut[i] = assets - (assets / 100); // 1% slippage exitTokenIndex = i; } } </pre>
Result/Recommendation	Consider verifying that assets exceed 100.

LOW ISSUES

During the audit, Chainsulting's experts found **4 Low issues** in the code of the smart contract.

6.2.2 Racing Condition in Approve Function

Severity: LOW

Status: CLOSED

Code: CWE-362

File(s) affected: ERC4626.sol, SwellBalancerVault.sol

Update: Implemented OpenZeppelin's version of increaseAllowance and decreaseAllowance functions in the ERC20.

Attack / Description	<p>The ERC4626 contract implements the standard ERC20, which contains a widely known racing condition in the approve function. A known race condition exists within the present implementation of the ERC20 standard.</p> <p>The scenario for exploitation is as follows:</p> <ol style="list-style-type: none"> 1. Alice calls approve(Bob, 1000), allocating 1000 tokens for Bob to spend 2. Alice opts to change the amount approved for Bob to spend to a lesser amount via approve(Bob, 500). Once mined, this decreases the number of tokens that Bob can spend to 500.
-----------------------------	---

	<ol style="list-style-type: none"> Bob sees the transaction and calls <code>transferFrom(Alice, X, 1000)</code> before <code>approve(Bob, 500)</code> has been mined. If Bob's transaction is mined prior to Alice's, 1000 tokens will be transferred by Bob. However, once Alice's transaction is mined, Bob can call <code>transferFrom(Alice, X, 500)</code>, transferring a total of 1500 tokens despite Alice attempting to limit the total token allowance to 500. <p>The particular exploit requires the usage of both the <code>transferFrom</code> and <code>approve</code> functions. As demonstrated above, the race condition occurs when one calls <code>approve</code> a second time on a spender that has already been allowed. If the spender sees the transaction containing the call before it has been mined, then the spender can call <code>transferFrom</code> to transfer the previous value and still receive the authorization to transfer the new value. While a multitude of approaches do exist to prevent this particular behavior from being exploited (one example of such is included below), they unfortunately may cause downstream functionality issues with those who implement such changes.</p>
Code	Line 10 (SwellBalancerVault.sol) <code>contract SwellBalancerVault is ERC4626, WeightedMath</code>
Result/Recommendation	Use <code>increaseAllowance</code> and <code>decreaseAllowance</code> functions, to modify the approval amount instead of using the <code>approve</code> function to modify it.

6.2.3 Missing Zero Address Checks

Severity: LOW

Status: CLOSED

Code: NA

File(s) affected: SwellBalancerVault.sol

Update: Added 0 address check for `_vault` address

Attack / Description	In the current implementation, there is an address set without checking for the zero address. This can lead to unintended behaviour.
Code	Line 29 (SwellBalancerVault.sol) <pre>// approve the balancer token vault contract in the asset token contract for this vault _asset.approve(address(_vault), type(uint256).max); }</pre>
Result/Recommendation	It is recommended to make sure the _vault address provided in the arguments is different from the address(0).

6.2.4 For Loop Over Dynamic Array

Severity: LOW

Status: ACKNOWLEDGED

Code: NA

File(s) affected: SwellBalancerVault.sol

Update: In this instance, the max loop is 50 iterations, with production likely to only run through 2 iterations (weighted two token pool)

Attack / Description	When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows over time can result in a Denial-of-Service. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.
Code	Line 54 - 59 (SwellBalancerVault.sol) <pre>for (uint256 i; i < tokens.length; ++i) { if (address(tokens[i]) == address(asset)) {</pre>

	<pre> assetIndex = i; break; } } Line 90 - 96 (SwellBalancerVault.sol) for (uint256 i; i < tokens.length; i++) { assetList[i] = IAsset(address(tokens[i])); if (address(tokens[i]) == address(asset)) { amountsOut[i] = assets - (assets / 100); // 1% slippage exitTokenIndex = i; } } Line 125 - 130 (SwellBalancerVault.sol) for (uint256 i = 0; i < tokensFromPool.length; i++) { tokens[i] = IAsset(address(tokensFromPool[i])); if (address(asset) == address(tokensFromPool[i])) { tokenAmounts[i] = assets; } } </pre>
Result/Recommendation	<p>Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocks and thus multiple transactions.</p>

6.2.5 Missing Value Verification

Severity: LOW

Status: CLOSED

Code: NA

File(s) affected: SwellBalancerVault.sol

Update: Now verified _poolID using the getPool function. Expect revert if the poolID provided is not registered.

Attack / Description	Certain functions lack a safety check in the values, the values of the arguments should be verified to allow only the ones that go with the contract's logic. In the constructor of the SwellBalancerVault, the _poolID is not verified, thus inserting a non-existing pool which can cause all the transactions to revert.
Code	<p>Line 18 - 30 (SwellBalancerVault.sol)</p> <pre>constructor(ERC20 _asset, string memory _name, string memory _symbol, IVault _vault, bytes32 _poolId) ERC4626(_asset, _name, _symbol) { balancerVault = _vault; poolId = _poolId; // approve the balancer token vault contract in the asset token contract for this vault _asset.approve(address(_vault), type(uint256).max); }</pre>
Result/Recommendation	Consider verifying _poolID using the getPool function which verify the existence of the pool using the modifier withRegisteredPool.

INFORMATIONAL ISSUES

During the audit, Chainsulting's experts found **3 Informational issues** in the code of the smart contract.

6.2.6 Floating And Multiple Pragma Version Identified

Severity: INFORMATIONAL

Status: CLOSED

Code: SWC-103

File(s) affected: ALL

Update: Fixed all solidity versions to 0.8.13 for all contract implementations

Attack / Description	It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.
Code	Line 1 <code>^0.8.0</code> <code>^0.8.13</code> <code>>=0.8.0</code>
Result/Recommendation	It is recommended to follow the latter example, as future compiler versions may handle certain language constructions in a way the developer did not foresee. It is advised that floating pragma should not be used in production. Both truffle-config.js and hardhat.config.js support locking the pragma version. i.e. Pragma solidity 0.8.0

6.2.7 Unnecessary Call To The Balancer Vault For The Pool Address

Severity: INFORMATIONAL

Status: CLOSED

Code: NA

File(s) affected: SwellBalancerVault.sol

Update: Added `_getPoolAddress` function which costs less gas to retrieve pool address

Attack / Description	For getting the pool address, it will less costly in term of gas if it's calculated automatically instead of performing a call to the Balancer Vault. The address is calculated using the following code
Code	NA
Result/Recommendation	<pre>function _getPoolAddress(bytes32 poolId) internal pure returns (address) { // 12 byte logical shift left to remove the nonce and specialization // setting. We do not need to mask, // since the logical shift already sets the upper bits to zero. return address(uint256(poolId) >> (12 * 8)); }</pre>

6.2.8 Missing Natspec Documentation

Severity: INFORMATIONAL

Status: ACKNOWLEDGED

Code: NA

File(s) affected: SwellBalancerVault.sol

Attack / Description	Solidity contracts can use a special form of comments to provide rich documentation for function, return variables, and more. This special form is named Ethereum Natural Language Specification Format(NatSpec).
Code	//
Result/Recommendation	It is recommended to include natspec documentation and follow the doxygen style including <code>@author</code> , <code>@title</code> , <code>@notice</code> , <code>@dev</code> , <code>@param</code> , <code>@return</code> and make it easier to review and understand your smart contract.

6.3 SWC Attacks

ID	Title	Relationships	Test Result
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	✓
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	✓
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	✓
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	✓
SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	✓
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	✓
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	✓
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	✓


ID	Title	Relationships	Test Result
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	✓
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	✓
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	✓
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	✓
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	✓
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	✓
SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	✓
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	✓

ID	Title	Relationships	Test Result
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	✓
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	✓
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	✓
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	✓
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	✓
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	✓
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	✓
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	✓
SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	✓
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	✓


ID	Title	Relationships	Test Result
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	✓
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	✓
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	✓
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	✓

6.4. Verify Claims


6.4.1 Implementations of the vendors Balancer and Rari-Capital are safe to use

Status: tested and verified 


6.4.2 Deposits are working as expected

Status: tested and verified 

6.4.3 Withdrawals are working as expected

Status: tested and verified 

6.4.4 The smart contract is coded according to the newest standards and in a secure way.

Status: tested and verified 

7. Executive Summary

Two (2) independent Chainsulting experts performed an unbiased and isolated audit of the smart contract codebase.

The main goal of the audit was to verify the claims regarding the security and functions of the smart contract. During the audit, no critical, 1 medium, 4 low and 3 informational issues have been found, after the manual and automated security testing. We advise the Swell Network team to implement the recommendations to further enhance the code's security and readability.

Update (08.06.2022) : All issues have been addressed and re-check done.

8. Deployed Smart Contract

PENDING

9. About the Auditor

Chainsulting is a professional software development firm, founded in 2021 and based in Germany. They show ways, opportunities, risks and offer comprehensive blockchain solutions. Some of their services include blockchain development, smart contract audits and consulting.

Chainsulting conducts code audits on market-leading blockchains such as Hyperledger, Tezos, Ethereum, Binance Smart Chain, and Solana to mitigate risk and instill trust and transparency into the vibrant crypto community. They have also reviewed and secured the smart contracts of 1Inch, POA Network, Unicrypt, Amun, Furucombo among numerous other top DeFi projects.

Chainsulting currently secures [\\$100 billion](#) in user funds locked in multiple DeFi protocols. The team behind the leading audit firm relies on their robust technical know-how in the blockchain sector to deliver top-notch smart contract audit solutions, tailored to the clients' evolving business needs.

Check our website for further information: <https://chainsulting.de>

How We Work



1 -----

PREPARATION

Supply our team with audit ready code and additional materials



2 -----

COMMUNICATION

We setup a real-time communication tool of your choice or communicate via e-mails.



3 -----

AUDIT

We conduct the audit, suggesting fixes to all vulnerabilities and help you to improve.



4 -----

FIXES

Your development team applies fixes while consulting with our auditors on their safety.



5 -----

REPORT

We check the applied fixes and deliver a full report on all steps done.