Swell Network

Staking Protocol

SMART CONTRACT AUDIT

11.03.2022

Made in Germany by Chainsulting.de

Table of contents

1. Disclaimer

The audit makes no statements or warrantees 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 (21.02.2022)	Layout
0.4 (27.02.2022)	Automated Security Testing Manual Security Testing
0.5 (28.02.2022)	Verify Claims and Test Deployment
0.6 (01.03.2022)	Testing SWC Checks
0.9 (05.03.2022)	Summary and Recommendation
1.0 (06.03.2022)	Final document
1.1 (11.03.2022)	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

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.

4.2 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)
contracts/helpers.sol	b82aedcd8bfd0c0d0deaaca14a20e664
contracts/swETH.sol	ededbfc83b23486523eb0959094ce99f
contracts/Strategy.sol	301f678d258c1352bb27ddcebf20448d
contracts/swNFTUpgrade.sol	9a7b53996949d4bc8278e162c033b782
contracts/swDAO.sol	1d24d714c52606d8b71a93b0420864e8
contracts/interfaces/ISWETH.sol	e988bef4d2e7e83cc3855c522b22f25c
contracts/interfaces/ISWNFT.sol	92b9180b44d0502a5d144157b5d096ac
contracts/interfaces/IStrategy.sol	502b412c7d660290d89e43f2eac83c62
contracts/libraries/NFTDescriptor.sol	6cbbb31de61b77d19b9b8f3a0065cf7d
contracts/libraries/NFTSVG.sol	65c2615b2f3d0adf7de37e9a252bcb28
contracts/libraries/HexStrings.sol	48f28bd45d3293d67a56d28b34797f26

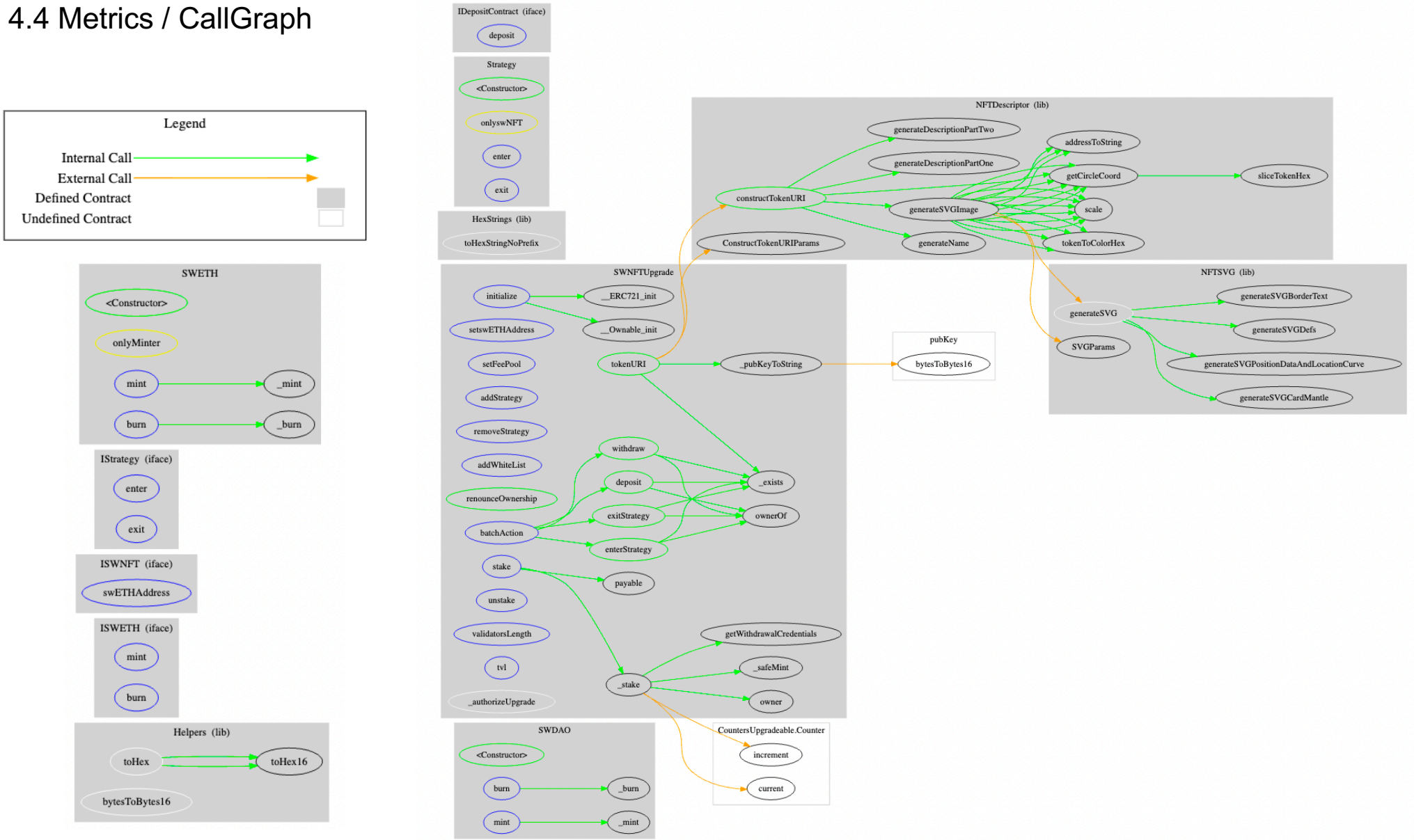
Language	Solidity
Token Standards	ERC20 / ERC721
Most Used Framework	OpenZeppelin
Compiler Version	0.8.9
Burn Function	Yes
Mint	Yes
Lock Mechanism	No
Vesting Function	No

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

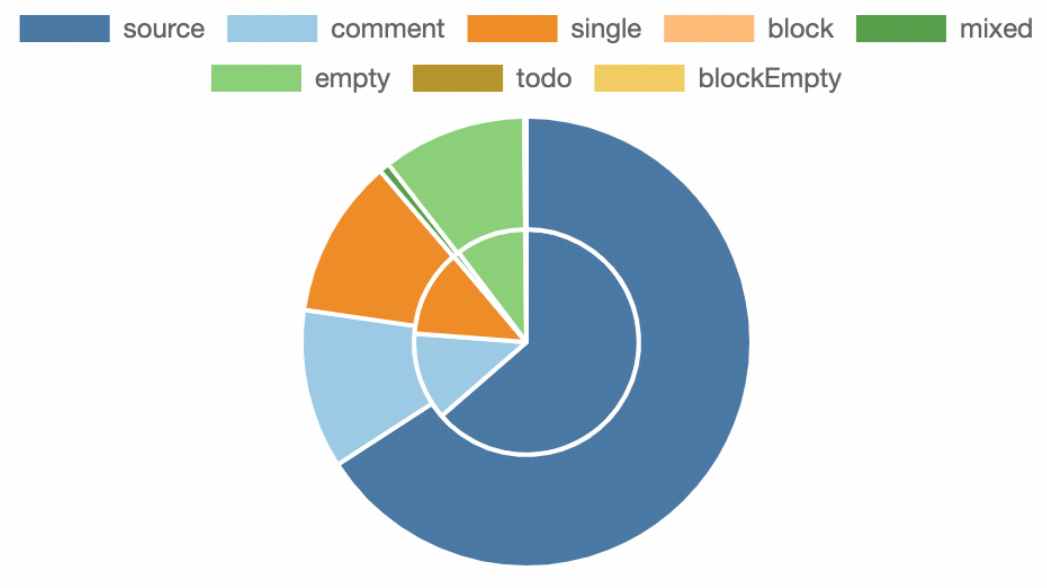
Dependency / Import Path	Source
@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/release-v4.5/contracts/access/OwnableUpgradeable.sol
@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/release-v4.5/contracts/proxy/utils/UUPSUpgradeable.sol
@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/release-v4.5/contracts/token/ERC721/ERC721Upgradeable.sol
@openzeppelin/contracts-upgradeable/token/ERC721/extensions/ERC721EnumerableUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/release-v4.5/contracts/token/ERC721/extensions/ERC721EnumerableUpgradeable.sol
@openzeppelin/contracts-upgradeable/utils/CountersUpgradeable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/release-v4.5/contracts/utils/CountersUpgradeable.sol
@openzeppelin/contracts/access/Ownable.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/access/Ownable.sol
@openzeppelin/contracts/token/ERC20/ERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/token/ERC20/ERC20.sol
@openzeppelin/contracts/token/ERC20/IERC20.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/token/ERC20/IERC20.sol

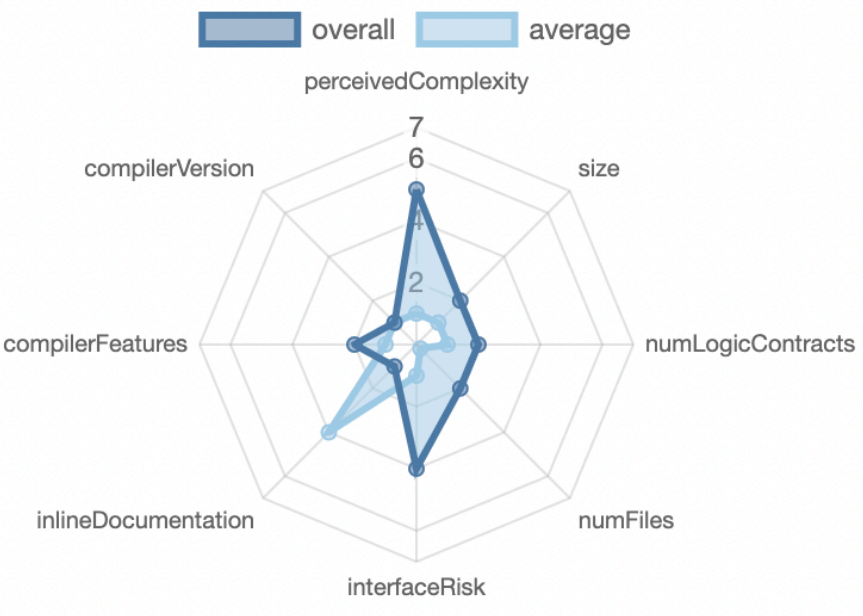
Dependency / Import Path	Source
@openzeppelin/contracts/utils/Strings.sol	https://github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.5/contracts/utils/Strings.sol

4.4 Metrics / CallGraph



4.5 Metrics / Source Lines & Risk





4.6 Metrics / Capabilities

Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
`0.8.9` `=0.8.9`		`yes`		

Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2
`yes`					

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
31	2

External	Internal	Private	Pure	View
23	36	10	21	7

StateVariables

Total	Public
19	15

4.7 Metrics / Source Unites in Scope

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📚	contracts/helpers.sol	1		44	44	27	14	18	
🔍	contracts/interfaces/ISWETH.sol		1	13	10	5	2	7	
🔍	contracts/interfaces/ISWNFT.sol		1	92	31	23	3	3	
🔍	contracts/interfaces/IStrategy.sol		1	21	7	3	3	5	
📝	contracts/swETH.sol	1		32	32	20	4	18	
📝	contracts/swDAO.sol	1		22	22	13	3	16	
📚	contracts/libraries/NFTDescriptor.sol	1		162	143	126	2	72	
📚	contracts/libraries/NFTSVG.sol	1		220	209	196	10	37	
📚	contracts/libraries/HexStrings.sol	1		29	29	12	14	19	
📝	contracts/Strategy.sol	1		43	43	32	3	28	🧢

Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝🔍	contracts/swNFTUpgrade.sol	1	1	363	340	220	79	218	💰📩
📝📚🔍	**Totals**	**8**	**4**	**1041**	**910**	**677**	**137**	**441**	💰📩

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, ...)

5. Scope of Work

The Swell Network Team provided us with the files that needs to be tested. The scope of the audit are the staking protocol contracts.

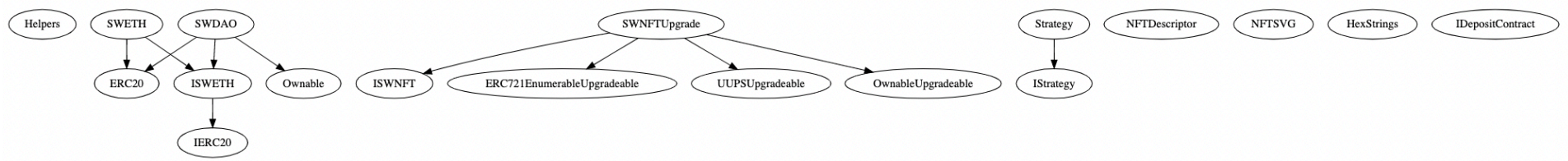
Following contracts with the direct imports has been tested:

- Strategy.sol
- swNFTUpgrade.sol
- swETH.sol
- swDAO.sol

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

- Staked ETH can't be withdrawn by deployer/contract owner
- The Deployer/Owner cannot burn, lock user funds (ETH)
- The Deployer/Owner cannot pause the contract
- The swNFTs are compatible with the ERC-721 standard
- The owner of this NFT can modify or redeem position
- NFT owner cannot withdraw more than the position value
- 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.



5.1 Manual and Automated Vulnerability Test

CRITICAL ISSUES

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

5.1.1 Initialize not protected

Severity: CRITICAL
Status: ACKNOWLEDGED
Code: NA
File(s) affected: swNFTUpgrade.sol
Update: On the proxy level it’s guarded by the initializer modifier. On implementation level that doesn’t matter as storage is not being read from there.

Attack / Description	Code Snippet	Result/Recommendation
SWNFTUpgrade is an upgradeable contract that uses an initializer. The initialize function is an unprotected external function. Anyone can call it before the owner or caller with right intentions; and pass in address _eth1WithdrawalAddress	swNFTUpgrade.sol (line 63) function initialize(address _eth1WithdrawalAddress) external initializer {	It is recommended to protect the function using access control e.g use onlyOwner modifier.

HIGH ISSUES

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

5.1.2 Possible reentrancy attack vector

Severity: HIGH
Status: FIXED
Code: NA
File(s) affected: swNFTUpgrade.sol
Commit: 041aaa40fbf1d141ada341695974e88ed4825e8a

Attack / Description	Code Snippet	Result/Recommendation

Reentrancy due to _safeMint() usage in staking function. If receiver is a contract which needs to implement onERC721Received can call back into the staking function. The position of earlier item Ids may not be captured. Note that in reentering the contract still needs to stake minimum 1ETH it's the changes to state that are affected instead.	swNFTUpgrade.sol (line 137)
	
	`_safeMint(msg.sender, newItemId);`
	
	`ISWETH(baseTokenAddress).mint(msg.value);`
	
	`positions[newItemId] = Position(`
	`pubKey,`
	`msg.value,`
	`msg.value`
	`);`
		It is recommended to use the Check Effects Interactions pattern by moving position updates to the top of _safeMint. Reentrancy guard may be useful but note it increases cost of function.

MEDIUM ISSUES

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

5.1.3 Missing require check

Severity: MEDIUM
Status: FIXED
Code: NA
File(s) affected: swNFTUpgrade.sol
Commit: 51dea8f9b05439b588bc54cd357f972b367dccdf

Attack / Description	Code Snippet	Result/Recommendation

Missing require statement can decrease the quote of failures.	swNFTUpgrade.sol (line 95-102)
	`function removeStrategy(uint strategy) onlyOwner external{`
	`require(strategies[strategy] != address(0), "strategy does not exist");`
	`uint length = strategies.length;`
	`address last = strategies[length-1];`
	`emit LogRemoveStrategy(strategy, strategies[strategy]);`
	`strategies[strategy] = last;`
	`strategies.pop();`
	`}`
		It is recommended to add a require to check the balance before removing.
		
		uint length = strategies.length;
		require(length >= 1, "nothing to remove");
		address last = strategies[length-1];

LOW ISSUES

During the audit, Chainsulting's experts found **6 Low issues** in the code of the smart contract

5.1.4 Error strings in require

Severity: LOW
Status: ACKNOWLEDGED
Code: NA
File(s) affected: Strategy.sol

Attack / Description	Code Snippet	Result/Recommendation

Require statements without error strings	Strategy.sol (line 20)
	`modifier onlyswNFT {`
	`require(msg.sender == swNFT);`
	`_;`
	`}`
		It is recommended that all require statements have an error message. This allows for off chain monitoring to notify on failing conditions. Lack of error messages impacts user experience, thus lowering the system's quality.

5.1.5 Variables that can be made constant or immutable

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: swETH.sol, Strategy.sol
Commit: 95b31eea33ef584a029f50508cea0509e58a7ae1

Attack / Description	Code Snippet	Result/Recommendation

State variables that do not change during life of contract or are set at construction and never change can be made constant or immutable	swETH.sol(line 11) address public minter;
	Strategy.sol (line 14) address public swNFT;
	swNFTUpgrade.sol (line 50) uint256 public ETHER = 1e18;
		It is recommended to make the minter and swNFT immutable variables and ETHER a constant variable. Variables with immutable keyword are read cheaper than state variables as they behave like constant variables, their values are directly inserted into runtime code.

5.1.6 Missing zero-address checks

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: Strategy.sol, swETH.sol, swNFTUpgarde.sol
Commit: 95b31eea33ef584a029f50508cea0509e58a7ae1

Attack / Description	Code Snippet	Result/Recommendation

In the current implementation, there are several addresses set without checking for the zero address. This can lead to unintended behaviour.	Strategy.sol (line 17)
	constructor(address _swNFT) {
	swNFT = _swNFT;
	}
	
	swETH.sol (line 16)
	constructor(address _minter) ERC20("Swell Ether", "swETH") {
	minter = _minter;
	}
	
	swNFTUpgrade.sol (line 63)
		It is highly recommended to check addresses for zero address _swNFT, _minter,_eth1WithdrawalAddress by adding require statement
		
		require(_address != address(0))

	function initialize(address _eth1WithdrawalAddress)	

5.1.7 Hardcoded address

Severity: LOW
Status: ACKNOWLEDGED
Code: NA
File(s) affected: swNFTUpgrade.sol
Update: With an upgradable contract we can't have const public state variables. But the ETH2 deposit contract has a fixed code and address. So as long as the test and audit makes no mistake on the address, code and address will never change.

Attack / Description	Code Snippet	Result/Recommendation
Hardcoded values like addresses can impact the life time of the contract, as they may change in the future.	swNFTUpgrade.sol(line 72) depositContract = IDepositContract(0x00000000219ab540356cBB839Cbe05303d7705Fa);	It is recommended to save the DepositContract address as a const public state variable. This gives transparency into the address which can be called by a public getter. Additionally it reduces chances of making errors with addresses, consider tests that check if the value of this address is the correct value. Another option is it may also be passed in the constructor. address constant public depositAddress = 0x00000000219ab540356cBB839Cbe05303d7705Fa; depositContract = IDepositContract(

		depositAddress);

5.1.8 Events without indexed parameters

Severity: LOW
Status: FIXED
Code: NA
File(s) affected: ISWNFT.sol, IStrategy.sol
Commit: 95b31eea33ef584a029f50508cea0509e58a7ae1

Attack / Description	Code Snippet	Result/Recommendation
State variables that do not change during life of contract or are set at construction and never change can be made constant or immutable	All events	It is recommended to index parameters, especially those that will be searched. Events without indexed parameters may lead to challenges for off-chain tooling that are expecting indexed events. Indexed parameters allow web3 applications to filter events by those parameters

5.1.9 Ownership control

Severity: LOW
Status: FIXED
Code: CWE-282
File(s) affected: swNFTUpgrade.sol
Commit: 95b31eea33ef584a029f50508cea0509e58a7ae1
Update: Applied and we will be using Protocol DAO Gnosis multisig for the deployment

Attack / Description	Code Snippet	Result/Recommendation
SWNFTUpgrade contract is Ownable. While it allows setting of restricted aspects and only owner functions like upgrade. It centralizes power in one address. Owner can call renounceOwnership() leaving the address with no owner so the contract can't add strategies or upgrade contracts. If the owner is malicious or control owner functions can't be called or can be called in a malicious manner.	Inherits OwnableUpgradeable to have an owner	It is recommended to use Multisig for ownership of address or other more decentralized control of the address. (Gnosis Safe) It is recommended to prevent renounceOwnership() from being called. It is recommended to use a two-step process when transferring ownership, to ensure the new owner can confirm has access and control to the new Owner address. That avoids loss of ownership over the contract.

INFORMATIONAL ISSUES

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

5.1.10 Uncomplete repository clean-up

Severity: INFORMATIONAL
Status: FIXED
Code: CWE-459
File(s) affected: swDAO.sol
Update: The swDAO.sol will be deployed and address being saved on swNFT. And once there's LP and price we could add the

require check on stake function.

Attack / Description	Code Snippet	Result/Recommendation
File swDAO.sol appears not to be used in the current implementation. Additionally it creates SWETH contract similar to SWETH contract in swETH.sol file	`swDAO.sol`	It is recommended to remove this file as it may lead to confusion in testing, auditing and code maintainability.

5.2. 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	✅

ID	Title	Relationships	Test Result
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	✅
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	✅

ID	Title	Relationships	Test Result
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')	✅
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	✅

ID	Title	Relationships	Test Result
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	✅
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	✅

5.3. Verify Claims

5.3.1 Staked ETH can't be withdrawn by deployer/contract owner
Status: tested and verified ✅

5.3.2 The Deployer/Owner cannot burn, lock user funds (ETH)
Status: tested and verified ✅

5.3.3 The Deployer/Owner cannot pause the contract
Status: tested and verified X

5.3.4 The swNFTs are compatible with the ERC-721 standard
Status: tested and verified ✅

5.3.5 The owner of this NFT can modify or redeem position
Status: tested and verified ✅

5.3.6 NFT owner cannot withdraw more than the position value
Status: tested and verified ✅

5.3.7 The smart contract is coded according to the newest standards and in a secure way.
Status: tested and verified ✅

5.4. Unit Tests

```
  NFTDescriptor
    #addressToString
      ✓ returns the correct string for a given address (101ms)
    #tokenToColorHex
      ✓ returns the correct hash for the first 3 bytes of the toke
n address (52ms)
      ✓ returns the correct hash for the last 3 bytes of the addre
ss (91ms)
```

```
  SWNFTUpgrade
Warning: Potentially unsafe deployment of TestswNFTUpgrade

    You are using the `unsafeAllow.external-library-linking` flag
to include external libraries.
    Make sure you have manually checked that the linked libraries
are upgrade safe.

    ✓ cannot stake less than 1 Ether (1ms)
    ✓ can stake 1 Ether (5170ms)
    ✓ cannot stake 1 Ether again (58ms)
    ✓ can add validator into whiteList (78ms)
    ✓ can stake 1 Ether again (1285ms)
    ✓ cannot stake more than 32 Ether (1ms)
    ✓ cannot withdraw 2 swETH (2ms)
    ✓ can withdraw 1 swETH (29ms)
    ✓ cannot deposit 2 swETH (27ms)
    ✓ can deposit 1 swETH (41ms)
    ✓ can add strategy (46ms)
    ✓ can enter strategy (1ms)
    ✓ can exit strategy (1ms)
    ✓ can batch actions (2ms)
    ✓ can remove strategy (1ms)

  NFTDescriptor
    #addressToString
      ✓ returns the correct string for a given address (27ms)
    #tokenToColorHex
      ✓ returns the correct hash for the first 3 bytes of the toke
```

```
·------------|----------------------------·
|        Solc version: 0.8.9         ·  Optimizer enabled: true  ·
 Runs: 200  ·  Block limit: 30000000 gas  |
·····························|·····|·····|·····|·····|·····|·····|
·····|·····|·····|·····|·····|·····|·····|
| Methods                                                        |
·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|
·····|·····|·····|·····|·····|·····|·····|
| Contract      · Method          · Min        · Max        ·
 Avg        · # calls      · usd (avg)  |
·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|
·····|·····|·····|·····|·····|·····|·····|
| SWETH         · approve         ·      29582 ·      46682 ·
    38132  ·           2  ·          –  |
·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|
·····|·····|·····|·····|·····|·····|·····|
| SWNFTUpgrade  · addStrategy     ·      57373 ·      74437 ·
    65905  ·           4  ·          –  |
·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|
·····|·····|·····|·····|·····|·····|·····|
| SWNFTUpgrade  · addWhiteList    ·          – ·          – ·
    54689  ·           2  ·          –  |
·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|
·····|·····|·····|·····|·····|·····|·····|
| SWNFTUpgrade  · deposit         ·          – ·          – ·
    72934  ·           2  ·          –  |
·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|
·····|·····|·····|·····|·····|·····|·····|
```

```
    ✓ returns a valid SVG (210ms)
```

```
·-----------------------|----------------------------|-------------
|----------------------------·
|  Solc version: 0.8.9  ·  Optimizer enabled: true  ·  Runs: 200
·  Block limit: 30000000 gas  |
·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|·····
|·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|·····
|  Methods                                                        |
·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|·····
|·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|·····
|  Contract  ·  Method  ·  Min        ·  Max        ·  Avg
·  # calls      ·  usd (avg)  |
·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|·····
|·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|·····
|  Deployments          ·                                         |
·  % of limit   ·                                                |
·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|·····
|·····|·····|·····|·····|·····|·····|·····|·····|·····|·····|·····
|  NFTDescriptorTest    ·          –  ·          –  ·    2409624
·         8 %  ·           –  |
·-----------------------|-------------|-------------|-------------
|---------------|-------------·
```

```
  4 passing (9s)
```

6. 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 of the smart contract and the functions. During the audit the following issues have been found: 1 critical, 1 high, 1 medium, 6 low and 1 informational. Please address the issues with your development team and get back to your auditor for re-check.

Update (11.03.2022): All issues have been addressed and codebase got re-checked.

7. Deployed Smart Contract

PENDING

8. About the Auditor

Chainsulting is a professional software development firm based in Germany that provides comprehensive distributed ledger technology (DLT) 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 instil trust and transparency into the vibrant crypto community. They have also reviewed and secure 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.

The blockchain security provider brings the highest security standards to crypto and blockchain platforms, helping to foster growth and transparency within the fast-growing ecosystem.

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