SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY

Unicorn Hunter

Audit

Security Assessment
11. October, 2022

For

Unicorn
HUNTER

SolidProof_io @solidproof_io

Disclaimer

SolidProof.io 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. SolidProof.io do not cover testing or auditing the integration with external contract or services (such as Unicrypt, Uniswap, PancakeSwap etc'...)

SolidProof.io Audits 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 technology proprietors. SolidProof Audits 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.

SolidProof.io 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. SolidProof's position is that each company and individual are responsible for their own due diligence and continuous security. SolidProof in no way claims any guarantee of security or functionality of the technology we agree to analyze.

Version	Date	Description
1.0	11. October 2022	• Layout project • Automated- /Manual-Security Testing • Summary

Network
Ethereum

Website
https://unicornhunter.io/

Telegram
https://t.me/+wiAE-USt9BU0N2Jl

Twitter
https://twitter.com/Unicorn8668

Facebook
https://www.facebook.com/UnicornHunterCapital

Youtube
https://www.youtube.com/channel/UClGKpp_cnRqoSB2y0Lqov4w

Description
Unicorn Hunter is an Asia-based investment firm that was established by a group of experts who have experiences in cryptocurrency & blockchain industry since 2014 and high-return investments since 2006.

Project Engagement
During the 23rd of August 2022, **Unicorn Hunter Team** engaged Solidproof.io to audit smart contracts that they created. The engagement was technical in nature and focused on identifying security flaws in the design and implementation of the contracts. They provided Solidproof.io with access to their code repository and whitepaper.

Logo
Unicorn HUNTER

Contract Link
v1.0
- Github
    - https://github.com/Unicorn-Hunter-Venture-Capital/contract-eth
    - Commit: db266f0aebfc4f0f066bfdc5cb977d508bdbe6b2

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 aspossible.
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

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.

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 SolidProof 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 SolidProof describe.

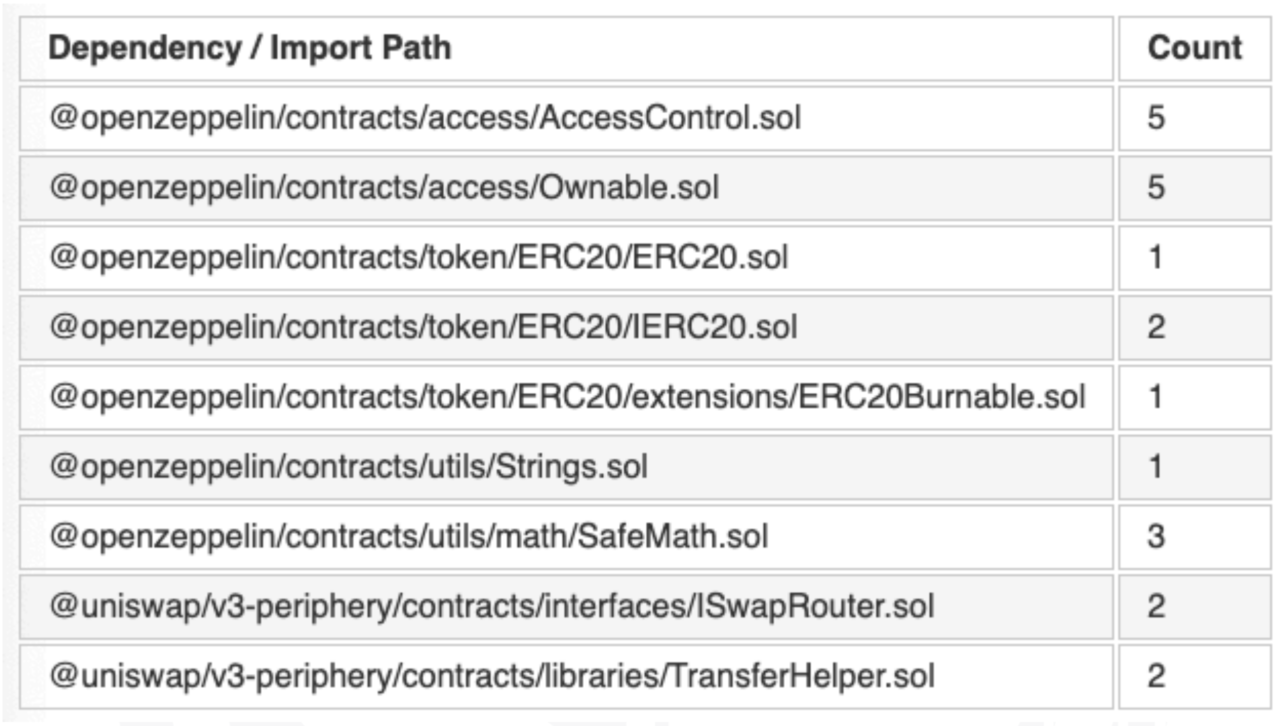
2. Testing and automated analysis that includes the following:
    - i) Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
    - ii) Symbolic execution, which is analysing a program 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.

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

Imported packages:



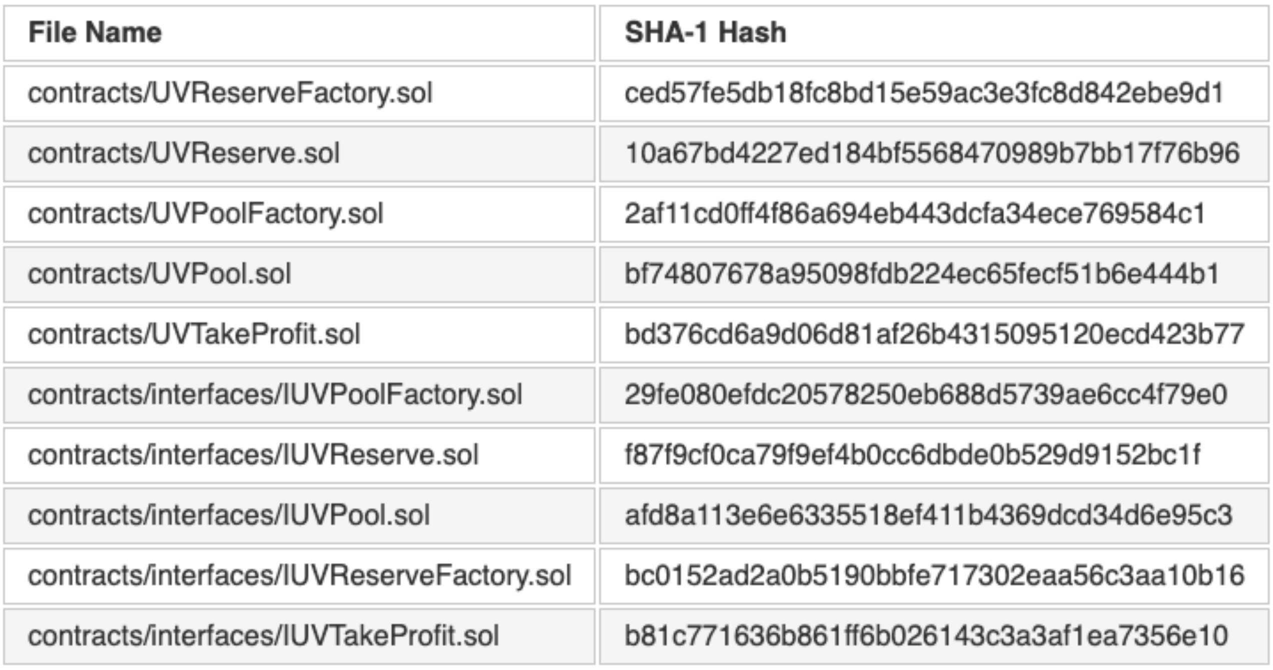
Dependency / Import Path	Count
@openzeppelin/contracts/access/AccessControl.sol	5
@openzeppelin/contracts/access/Ownable.sol	5
@openzeppelin/contracts/token/ERC20/ERC20.sol	1
@openzeppelin/contracts/token/ERC20/IERC20.sol	2
@openzeppelin/contracts/token/ERC20/extensions/ERC20Burnable.sol	1
@openzeppelin/contracts/utils/Strings.sol	1
@openzeppelin/contracts/utils/math/SafeMath.sol	3
@uniswap/v3-periphery/contracts/interfaces/ISwapRouter.sol	2
@uniswap/v3-periphery/contracts/libraries/TransferHelper.sol	2

Tested Contract Files

This audit covered the following files listed below with a SHA-1 Hash.

A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

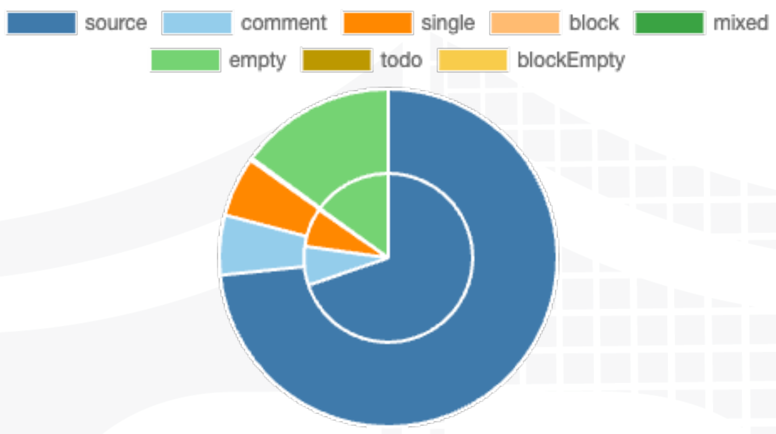


File Name	SHA-1 Hash
contracts/UVReserveFactory.sol	ced57fe5db18fc8bd15e59ac3e3fc8d842ebe9d1
contracts/UVReserve.sol	10a67bd4227ed184bf5568470989b7bb17f76b96
contracts/UVPoolFactory.sol	2af11cd0ff4f86a694eb443dcfa34ece769584c1
contracts/UVPool.sol	bf74807678a95098fdb224ec65fecf51b6e444b1
contracts/UVTakeProfit.sol	bd376cd6a9d06d81af26b4315095120ecd423b77
contracts/interfaces/IUVPoolFactory.sol	29fe080efdc20578250eb688d5739ae6cc4f79e0
contracts/interfaces/IUVReserve.sol	f87f9cf0ca79f9ef4b0cc6dbde0b529d9152bc1f
contracts/interfaces/IUVPool.sol	afd8a113e6e6335518ef411b4369dcd34d6e95c3
contracts/interfaces/IUVReserveFactory.sol	bc0152ad2a0b5190bbfe717302eaa56c3aa10b16
contracts/interfaces/IUVTakeProfit.sol	b81c771636b861ff6b026143c3a3af1ea7356e10

Metrics

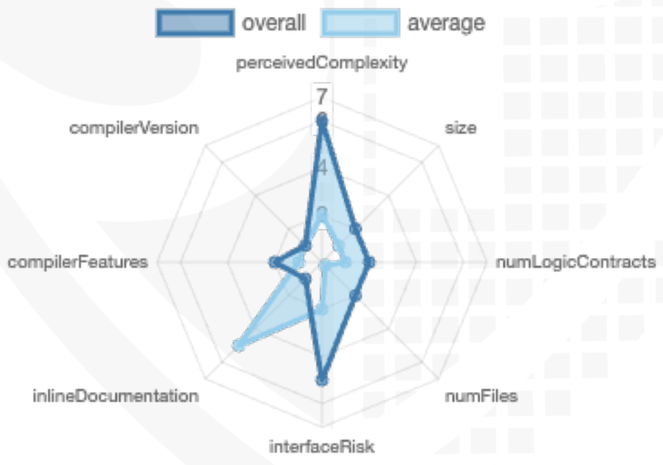
Source Lines

v1.0



Risk Level

v1.0



Capabilities

Components

Version	Contracts	Libraries	Interfaces	Abstract
1.0	5	0	5	0

Exposed Functions

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

Version	Public	Payable
1.0	114	8

Version	External	Internal	Private	Pure	View
1.0	61	98	0	0	15

State Variables

Version	Total	Public
1.0	55	51

Capabilities

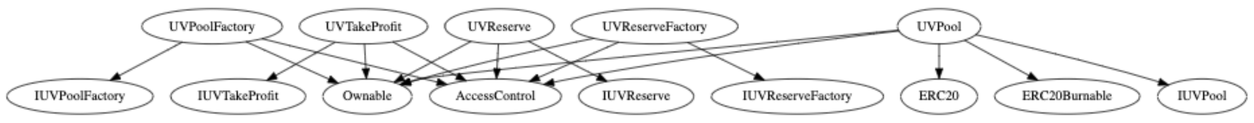
Version	Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
1.0	`^0.8.12`		`yes`		

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

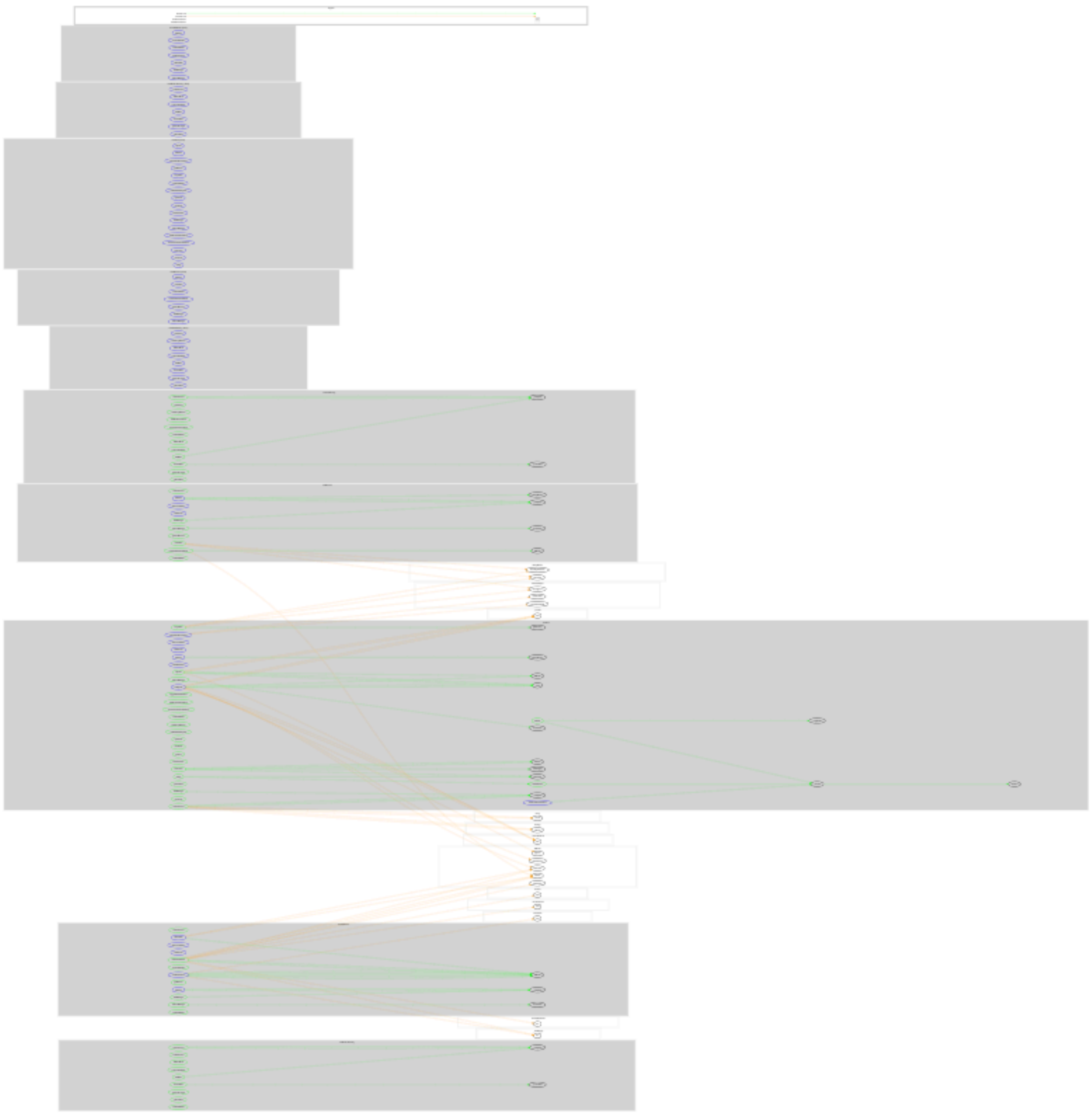
1.0	`yes`			`yes`		`yes`
						`→ NewContract:UVTakeProfit`
						`→ NewContract:UVReserve`

Inheritance Graph

v1.0



CallGraph
v1.0



Scope of Work/Verify Claims

The above token Team provided us with the files that needs to be tested (Github, Bscscan, Etherscan, files, etc.). The scope of the audit is the main contract (usual the same name as team appended with .sol).

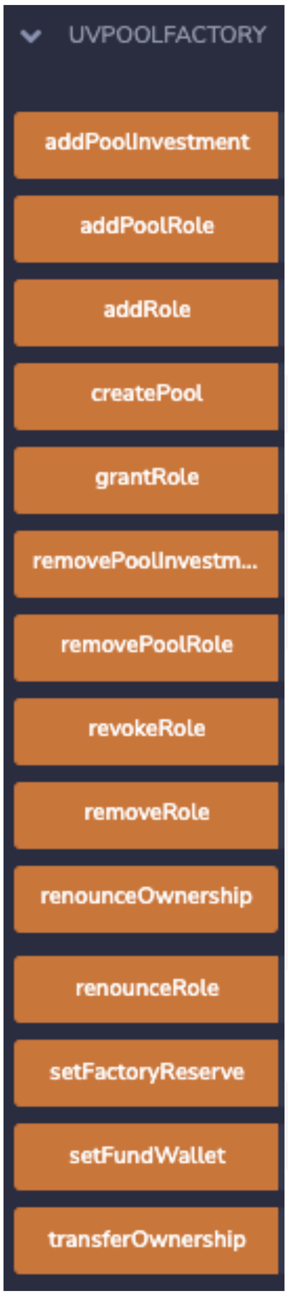
We will verify the following claims:

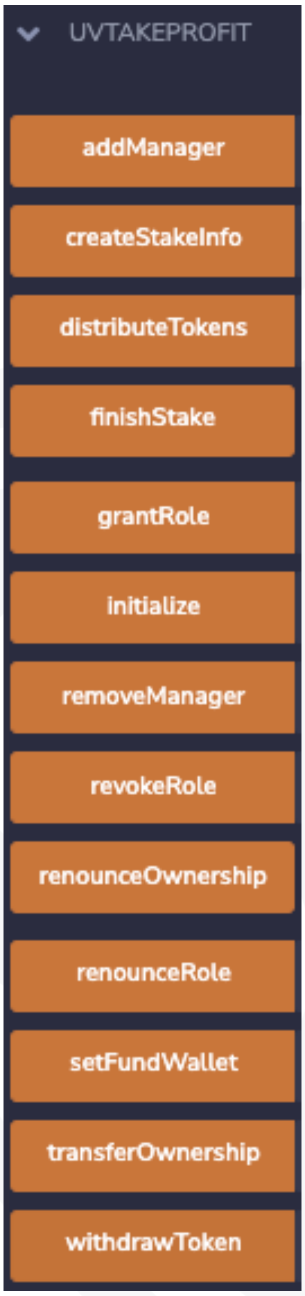
1. Is contract an upgradeable
2. Deployer cannot mint any new tokens
3. Deployer cannot burn or lock user funds
4. Deployer cannot pause the contract
5. Deployer cannot set fees
6. Deployer cannot blacklist/antisnipe addresses
7. Overall checkup (Smart Contract Security)

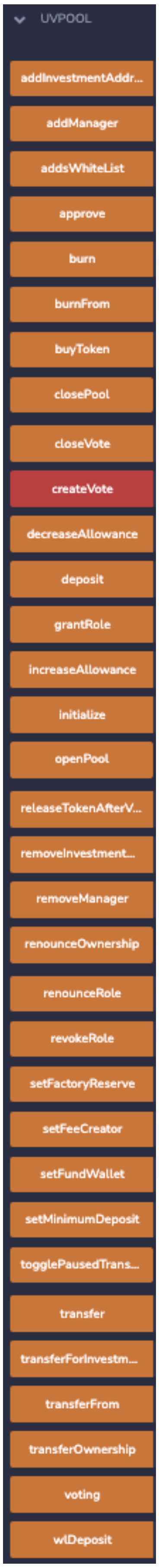
Is contract an upgradeable

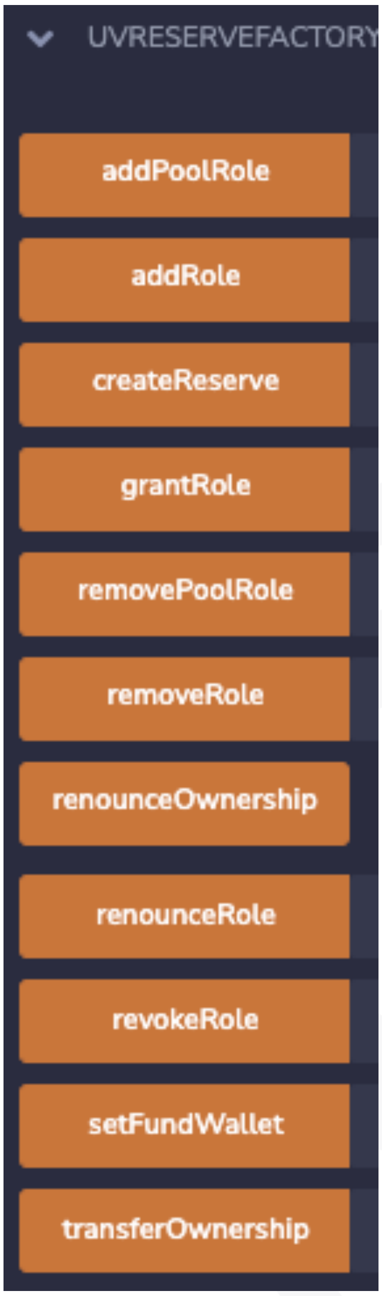
Name	
Is contract an upgradeable?	No

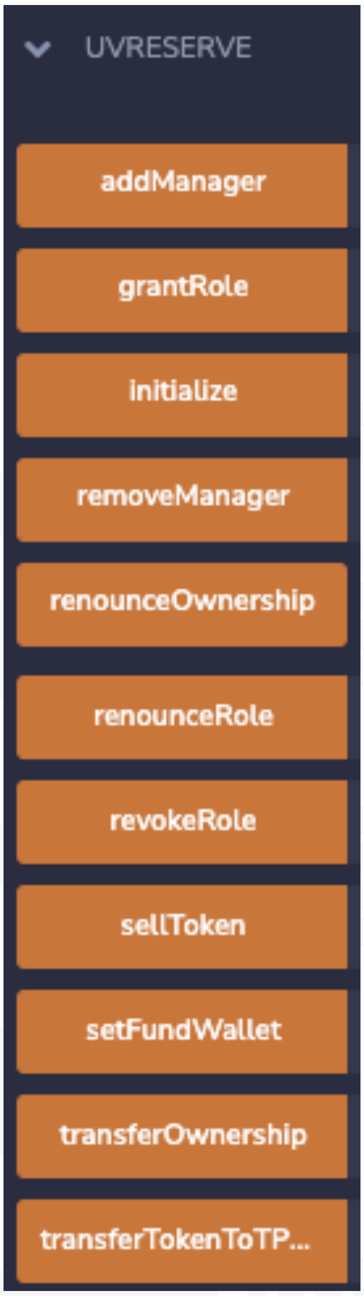
Write functions of contract v1.0








UVRESERVEFACTORY
addPoolRole
addRole
createReserve
grantRole
removePoolRole
removeRole
renounceOwnership
renounceRole
revokeRole
setFundWallet
transferOwnership


UVRESERVE
addManager
grantRole
initialize
removeManager
renounceOwnership
renounceRole
revokeRole
sellToken
setFundWallet
transferOwnership
transferTokenToTP...

Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint	✓	✓	✓

Comments:

v1.0

- Tokens will be minted while
  - depositing in UVPool
  - wIdeposit in UVPool

Deployer cannot burn or lock user funds

Name	Exist	Tested	Status
Deployer cannot lock	✓	✓	✗
Deployer cannot burn	✓	✓	✓

Comments:

v1.0

- Owner can lock user funds by
  - Pausing in UVPool

- Tokens
  - can be burned by msg.sender in UVPool
  - will be burned by calling “finishStake” function in UVTakeProfit

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer cannot pause	✓	✓	✗

Comments:

v1.0

- Manager can pause contract in UVPool

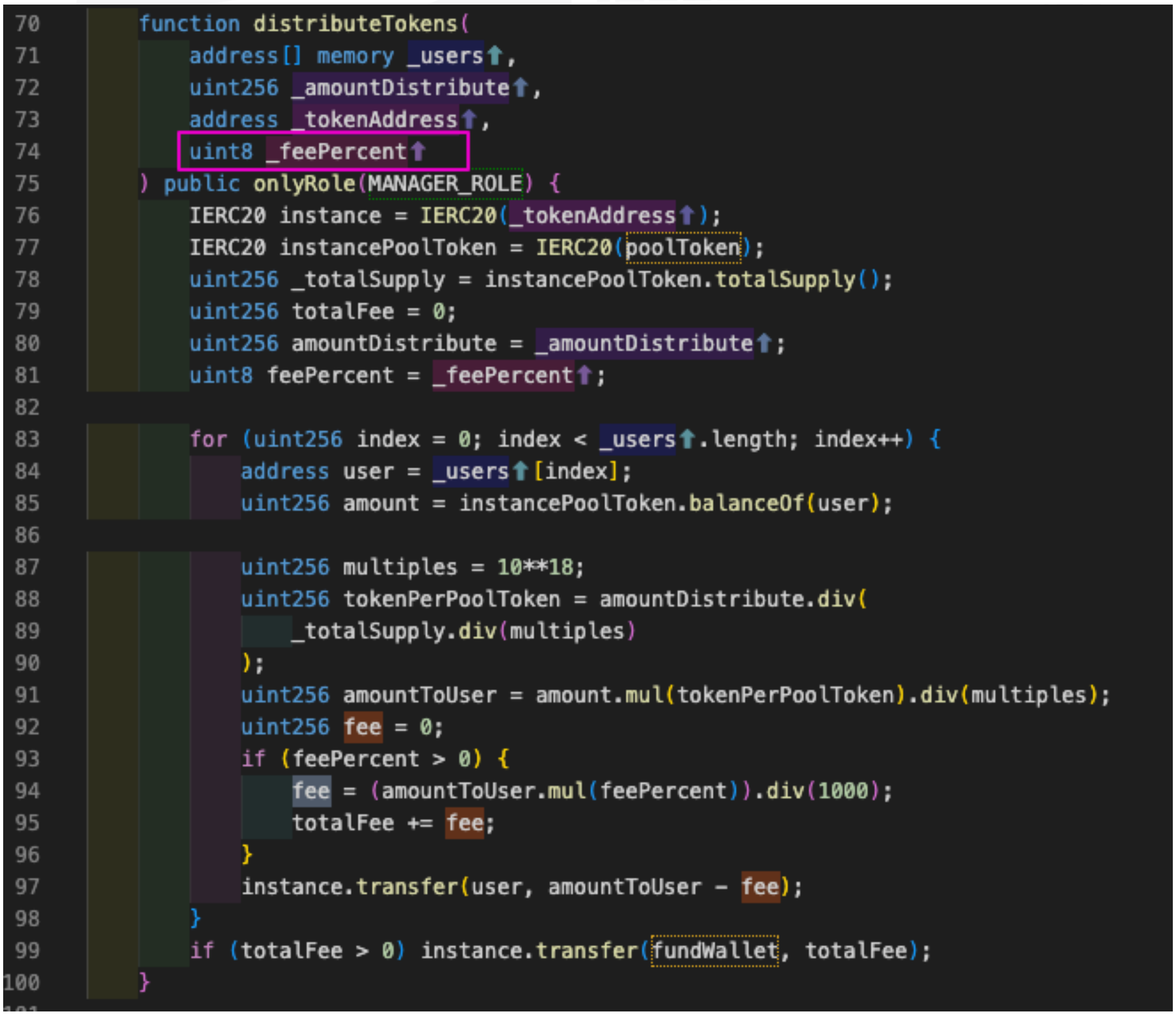
Deployer cannot set fees

Name	Exist	Tested	Status
Deployer cannot set fees over 25%	✓	✓	✗
Deployer cannot set fees to nearly 100% or to 100%	✓	✓	✗

Comments:

v1.0

- Fees can be set up to 100% in UVTakeProfit (function: distributeToken)



```
function distributeTokens(
    address[] memory _users,
    uint256 _amountDistribute,
    address _tokenAddress,
    uint8 _feePercent
) public onlyRole(MANAGER_ROLE) {
    IERC20 instance = IERC20(_tokenAddress);
    IERC20 instancePoolToken = IERC20(poolToken);
    uint256 _totalSupply = instancePoolToken.totalSupply();
    uint256 totalFee = 0;
    uint256 amountDistribute = _amountDistribute;
    uint8 feePercent = _feePercent;

    for (uint256 index = 0; index < _users.length; index++) {
        address user = _users[index];
        uint256 amount = instancePoolToken.balanceOf(user);

        uint256 multiples = 10**18;
        uint256 tokenPerPoolToken = amountDistribute.div(
            _totalSupply.div(multiples)
        );
        uint256 amountToUser = amount.mul(tokenPerPoolToken).div(multiples);
        uint256 fee = 0;
        if (feePercent > 0) {
            fee = (amountToUser.mul(feePercent)).div(1000);
            totalFee += fee;
        }
        instance.transfer(user, amountToUser - fee);
    }
    if (totalFee > 0) instance.transfer(fundWallet, totalFee);
}
```

Deployer can blacklist/antisnipe addresses

Name	Exist	Tested	Status
Deployer cannot blacklist/antisnipe addresses	–	–	–

Overall checkup (Smart Contract Security)

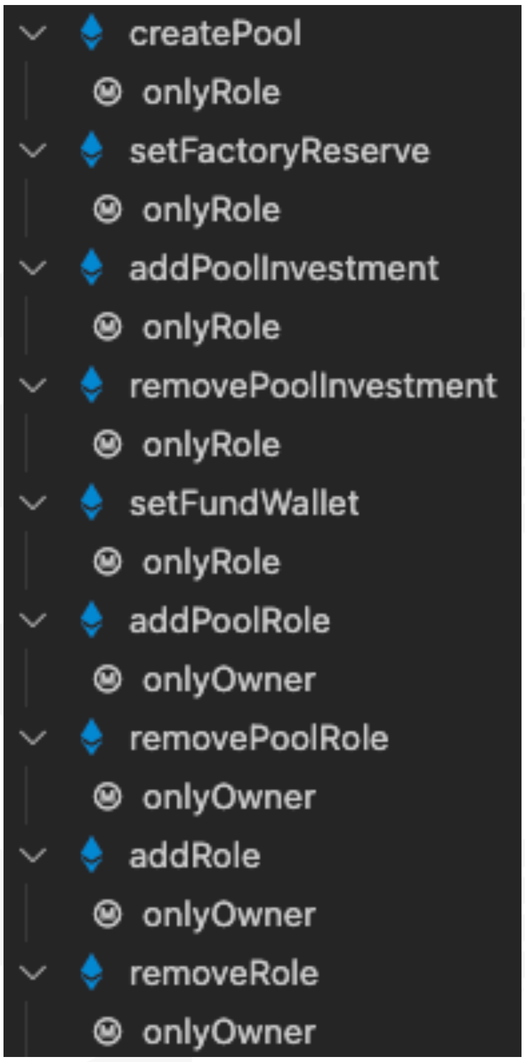
Tested	Verified
✓	✓

Legend

Attribute	Symbol
Verified / Checked	✓
Partly Verified	⚑
Unverified / Not checked	✗
Not available	–

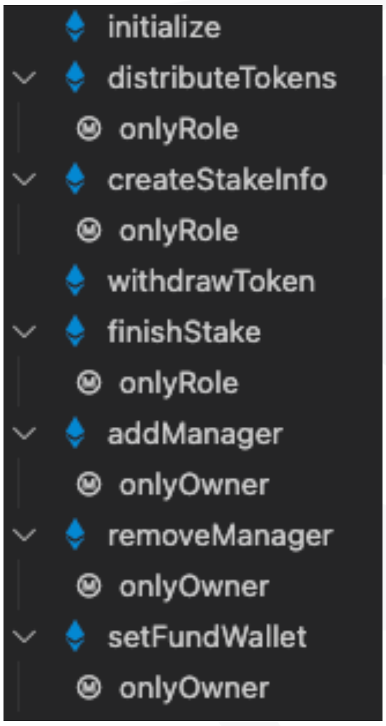
Modifiers and public functions
v1.0

UVPoolFactory



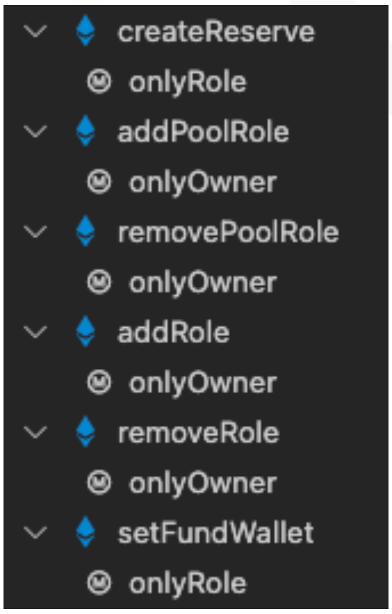
- createPool
  - onlyRole
- setFactoryReserve
  - onlyRole
- addPoolInvestment
  - onlyRole
- removePoolInvestment
  - onlyRole
- setFundWallet
  - onlyRole
- addPoolRole
  - onlyOwner
- removePoolRole
  - onlyOwner
- addRole
  - onlyOwner
- removeRole
  - onlyOwner

UVTakeProfit



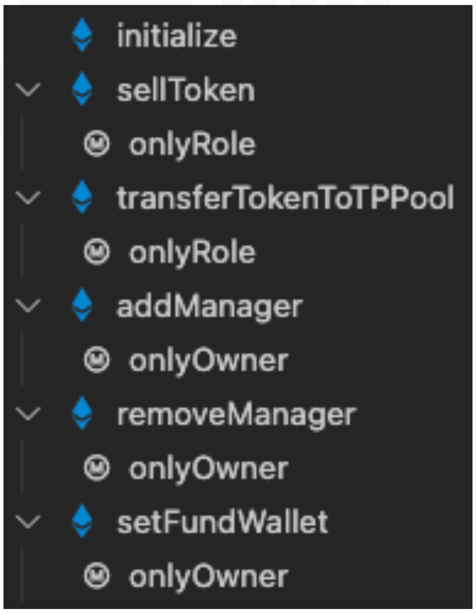
- initialize
- distributeTokens
  - onlyRole
- createStakeInfo
  - onlyRole
- withdrawToken
- finishStake
  - onlyRole
- addManager
  - onlyOwner
- removeManager
  - onlyOwner
- setFundWallet
  - onlyOwner

UVReserveFactory



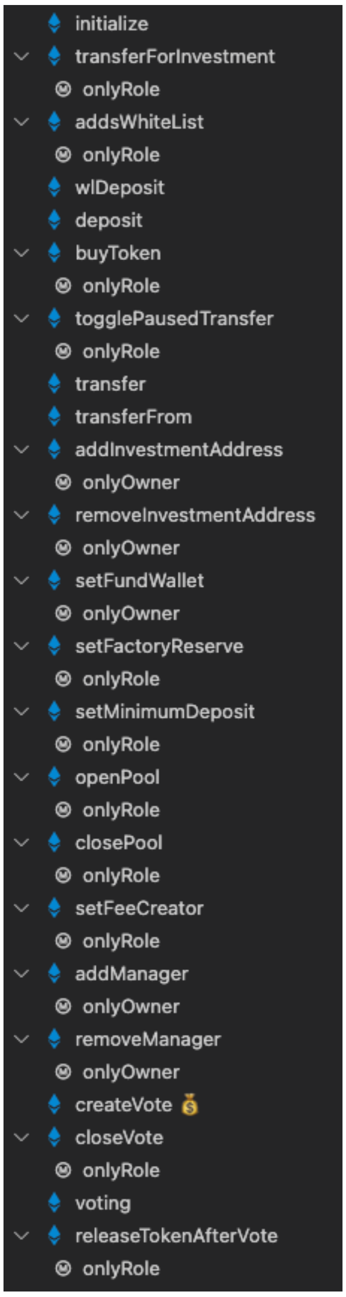
- createReserve
  - onlyRole
- addPoolRole
  - onlyOwner
- removePoolRole
  - onlyOwner
- addRole
  - onlyOwner
- removeRole
  - onlyOwner
- setFundWallet
  - onlyRole

UVReserve



- initialize
- sellToken
  - onlyRole
- transferTokenToTPPool
  - onlyRole
- addManager
  - onlyOwner
- removeManager
  - onlyOwner
- setFundWallet
  - onlyOwner

UVPool


```
initialize
transferForInvestment
    onlyRole
addsWhiteList
    onlyRole
wlDeposit
deposit
buyToken
    onlyRole
togglePausedTransfer
    onlyRole
transfer
transferFrom
addInvestmentAddress
    onlyOwner
removeInvestmentAddress
    onlyOwner
setFundWallet
    onlyOwner
setFactoryReserve
    onlyRole
setMinimumDeposit
    onlyRole
openPool
    onlyRole
closePool
    onlyRole
setFeeCreator
    onlyRole
addManager
    onlyOwner
removeManager
    onlyOwner
createVote 💰
closeVote
    onlyRole
voting
releaseTokenAfterVote
    onlyRole
```

Note: Not listed functions/modifiers was implemented from libraries

Comments

- *Deployer can set following state variables without any limitations*
    - UVPool
        - feeCreator
        - minimumDeposit
    - UVTakeProfit
        -
        -

- *Deployer can enable/disable following state variables*
  - UVPool
    - whiteList
    - pausedTransfer
    - investmentAddreses
    - isClose
  - UVPoolFactory
    - managers
    -
- *Deployer can set following addresses*
  - UVPool
    - feeCreator
    - factoryReserve
    - fundWallet
  - UVPoolFactory
    - factoryReserve
  - UVReserve
    - fundWallet
  - UVTakeProfit

fundWallet

-

Comments

- UVPool
  - Initialize function can be called more than once
  - Owner can revoke/add manager
  - Only manager can call all functions except
    - addInvestmentAddress
    - removeInvestmentAddress
    - setFundWallet
    - addManager
    - removeManager
  - Manager can close vote anytime
    -
- UVPoolFactory
  - Admin can
    - create pools
    - Add/remove investment
    - Set fund wallet
    - Add admin role to address
    - Add new manager in the UVPool
- UVReserve

        - Manager can
            - Sell tokens
            - Sell native tokens
            - Transfer tokens to staking pool
        - Admin can
            - Grant manager role
        - Initialize function can be called more than once
- UVReserveFactory
    - Admin can
        - Create new reserve
    - Owner can
        - Add/remove role
- UVTakeProfit
    - Initialize function can be called more than once
    - Manager can
        - Distribute tokens
        - Create stake info
        - Finish pool with “finishStake” function. This will send every poolToken balance to burn address
    - Owner can
        - Add/remove manager role
    - Unnecessary “require” statement in “withdrawToken” L135. If the “_amountStake” is above 0 the function will be continued. “_amountStake” will be set then to “users[_stakeId][msg.sender].amount” which is also not 0.

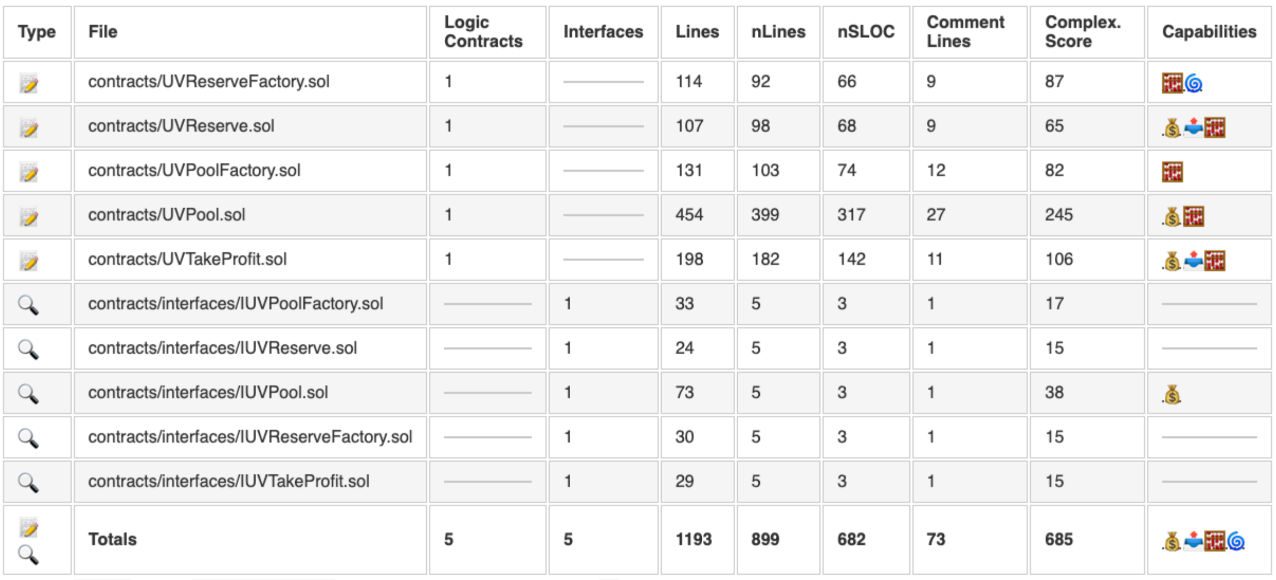
Please check every type of uint max type and the variable whether it is possible to set to it or not:

- uint8 => max: 2^8-1 = 255
- Uint16 => max: 2^16-1 = 65.535
- Uint64 => max: 2^64-1 = 1,84467441e19
- And so on

Please check if an OnlyOwner or similar restrictive modifier has been forgotten.

Source Units in Scope

v1.0



Type	File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
📝	contracts/UVReserveFactory.sol	1	—	114	92	66	9	87	🧮🌀
📝	contracts/UVReserve.sol	1	—	107	98	68	9	65	💰📥🧮
📝	contracts/UVPoolFactory.sol	1	—	131	103	74	12	82	🧮
📝	contracts/UVPool.sol	1	—	454	399	317	27	245	💰🧮
📝	contracts/UVTakeProfit.sol	1	—	198	182	142	11	106	💰📥🧮
🔍	contracts/interfaces/IUVPoolFactory.sol	—	1	33	5	3	1	17	—
🔍	contracts/interfaces/IUVReserve.sol	—	1	24	5	3	1	15	—
🔍	contracts/interfaces/IUVPool.sol	—	1	73	5	3	1	38	💰
🔍	contracts/interfaces/IUVReserveFactory.sol	—	1	30	5	3	1	15	—
🔍	contracts/interfaces/IUVTakeProfit.sol	—	1	29	5	3	1	15	—
📝🔍	**Totals**	**5**	**5**	**1193**	**899**	**682**	**73**	**685**	💰📥🧮🌀

Legend

Attribute	Description
Lines	total lines of the source unit
nLines	normalised lines of the source unit (e.g. normalises functions spanning multiple lines)
nSLOC	normalised 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, ...)

Audit Results

AUDIT PASSED

Critical issues

No critical issues

High issues

No high issues

Medium issues

No medium issues

Low issues

Issue	File	Type	Line	Description
#1	All	A floating pragma is set	See description	See pragma versions, especially the one that started with “^” sign
#2	UVPool	Missing Zero Address Validation (missing-zero-check)	85-87, 308, 303,	Check that the address is not zero
#3	UVPool Factory	Missing Zero Address Validation (missing-zero-check)	55	Check that the address is not zero
#4	UVTake Profit	Missing Zero Address Validation (missing-zero-check)	54, 55, 195	Check that the address is not zero
#5	UVRese rve	Missing Zero Address Validation (missing-zero-check)	36, 104	Check that the address is not zero
#6	UVPool Factory	State variable visibility is not set	15	It is best practice to set the visibility of state variables explicitly

#7	UVReserveFactory	State variable visibility is not set	20	It is best practice to set the visibility of state variables explicitly
#8	UVTakeProfit	Missing Events Arithmetic	58	Emit an event for critical parameter changes
#9	UVPool	Missing Events Arithmetic	92	Emit an event for critical parameter changes
#10	UVPool	State variable shadowing	55	Remove the state variable shadowing because it is declared already in the ERC20 contract of OZ
#11	UVPool	Local State variable shadowing	285	Rename the local variables that shadow another component
#12	UVPool	Wrong error message or statement	400	Check the isClosed require statement in L400. Change error message or statement

Informational issues

Issue	File	Type	Line	Description
#1	UVReserve	Error message is missing	82, 83	Provide an error message for require statement
#2	UVTakeProfit	Error message is missing	111, 116	Provide an error message for require statement
#3	All	NatSpec documentation missing	-	If you started to comment your code, also comment all other functions, variables etc.
#4	UVPool	Unnecessary check	See description	The Manager_Role was checked in the “if condition” in L370 but you are checking the role again in the “else” condition in L378. If the “else” should be called, the msg.sender doesn’t have the role

Audit Comments

We recommend you to use the special form of comments (NatSpec Format, Follow link for more information https://docs.soliditylang.org/en/v0.5.10/natspec-format.html) for your contracts to provide rich documentation for functions, return variables and more. This helps investors to make clear what that variables, functions etc. do.

11. October 2022:

- Please check max type of uint's
- Whitepaper was not provided from customer
- Read whole report and modifiers section for more information

SWC Attacks

ID	Title	Relationships	Status
SWC-136	Unencrypted Private Data On-Chain	CWE-767: Access to Critical Private Variable via Public Method	PASSED
SWC-135	Code With No Effects	CWE-1164: Irrelevant Code	PASSED
SWC-134	Message call with hardcoded gas amount	CWE-655: Improper Initialization	PASSED
SWC-133	Hash Collisions With Multiple Variable Length Arguments	CWE-294: Authentication Bypass by Capture-replay	PASSED
SWC-132	Unexpected Ether balance	CWE-667: Improper Locking	PASSED
SWC-131	Presence of unused variables	CWE-1164: Irrelevant Code	PASSED
SWC-130	Right-To-Left-Override control character (U+202E)	CWE-451: User Interface (UI) Misrepresentation of Critical Information	PASSED
SWC-129	Typographical Error	CWE-480: Use of Incorrect Operator	PASSED
SWC-128	DoS With Block Gas Limit	CWE-400: Uncontrolled Resource Consumption	PASSED

SWC-127	Arbitrary Jump with Function Type Variable	CWE-695: Use of Low-Level Functionality	**PASSED**
SWC-125	Incorrect Inheritance Order	CWE-696: Incorrect Behavior Order	**PASSED**
SWC-124	Write to Arbitrary Storage Location	CWE-123: Write-what-where Condition	**PASSED**
SWC-123	Requirement Violation	CWE-573: Improper Following of Specification by Caller	**PASSED**
SWC-122	Lack of Proper Signature Verification	CWE-345: Insufficient Verification of Data Authenticity	**PASSED**
SWC-121	Missing Protection against Signature Replay Attacks	CWE-347: Improper Verification of Cryptographic Signature	**PASSED**
SWC-120	Weak Sources of Randomness from Chain Attributes	CWE-330: Use of Insufficiently Random Values	**PASSED**
SWC-119	Shadowing State Variables	CWE-710: Improper Adherence to Coding Standards	**NOT PASSED**
SWC-118	Incorrect Constructor Name	CWE-665: Improper Initialization	**PASSED**
SWC-117	Signature Malleability	CWE-347: Improper Verification of Cryptographic Signature	**PASSED**

SWC-116	Timestamp Dependence	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SWC-115	Authorization through tx.origin	CWE-477: Use of Obsolete Function	PASSED
SWC-114	Transaction Order Dependence	CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition')	PASSED
SWC-113	DoS with Failed Call	CWE-703: Improper Check or Handling of Exceptional Conditions	PASSED
SWC-112	Delegatecall to Untrusted Callee	CWE-829: Inclusion of Functionality from Untrusted Control Sphere	PASSED
SWC-111	Use of Deprecated Solidity Functions	CWE-477: Use of Obsolete Function	PASSED
SWC-110	Assert Violation	CWE-670: Always-Incorrect Control Flow Implementation	PASSED
SWC-109	Uninitialized Storage Pointer	CWE-824: Access of Uninitialized Pointer	PASSED
SWC-108	State Variable Default Visibility	CWE-710: Improper Adherence to Coding Standards	NOT PASSED
SWC-107	Reentrancy	CWE-841: Improper Enforcement of Behavioral Workflow	PASSED
SWC-106	Unprotected SELFDESTRUCT Instruction	CWE-284: Improper Access Control	PASSED

SWC-105	Unprotected Ether Withdrawal	CWE-284: Improper Access Control	**PASSED**
SWC-104	Unchecked Call Return Value	CWE-252: Unchecked Return Value	**PASSED**
SWC-103	Floating Pragma	CWE-664: Improper Control of a Resource Through its Lifetime	**NOT PASSED**
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	**PASSED**
SWC-101	Integer Overflow and Underflow	CWE-682: Incorrect Calculation	**PASSED**
SWC-100	Function Default Visibility	CWE-710: Improper Adherence to Coding Standards	**PASSED**

SolidProof_io

@solidproof_io

Solid Proofed

Blockchain Security | Smart Contract Audits | KYC

MADE IN GERMANY