SOLIDProof
Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC
Development | Marketing

MADE IN GERMANY

eZkalibur Tokens & Farming

Audit

Security Assessment
18. May, 2023

For

ezkalibur

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	15. May 2023 - 17. May 2023	• Layout project • Automated- /Manual-Security Testing • Summary

Note - This Audit report consists of a security analysis of the **ezKalibur** smart contracts. This analysis did not include functional testing (or unit testing) of the contract’s logic.

Network

zkSync

Website

https://ezkalibur.com

Telegram

https://t.me/ezkalibur

Twitter

https://twitter.com/eZKaliburDEX

Discord

https://discord.com/invite/ypqHnKE5KF

Description

ZKalibur is the first ecosystem-focused and community-driven DEX built on zkSync Era.
We have built a highly efficient and customizable protocol, allowing both builders and users to leverage our custom infrastructure for deep, sustainable, and adaptable liquidity.

Project Engagement

During the 30 of April 2023, **ezKalibu**r **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

Contract Links

v1.0

https://github.com/eZKalibur/contractsV2/tree/main/farming

https://github.com/eZKalibur/contractsV2/tree/main/launchpad

https://github.com/eZKalibur/contractsV2/tree/main/token

Commit: 4d48058

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.

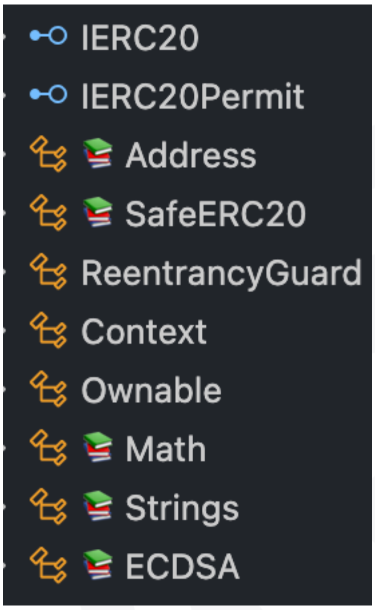
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.

4. Specific, itemized, actionable recommendations to help you take steps to secure your smart contracts.

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

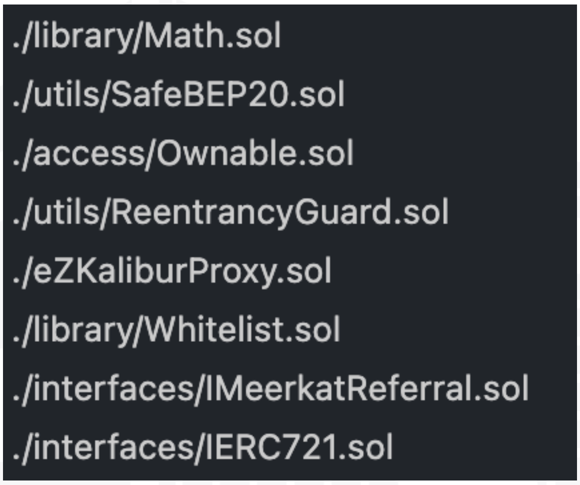
Imported packages:

Ico



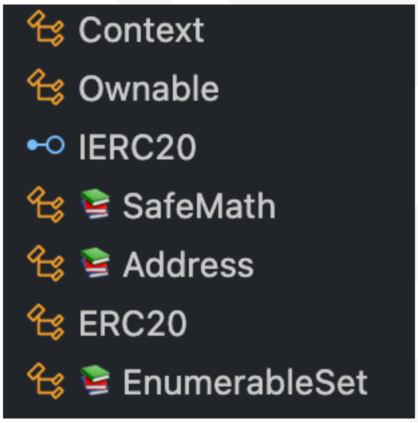
```
IERC20
IERC20Permit
Address
SafeERC20
ReentrancyGuard
Context
Ownable
Math
Strings
ECDSA
```

eZkaliburMaster



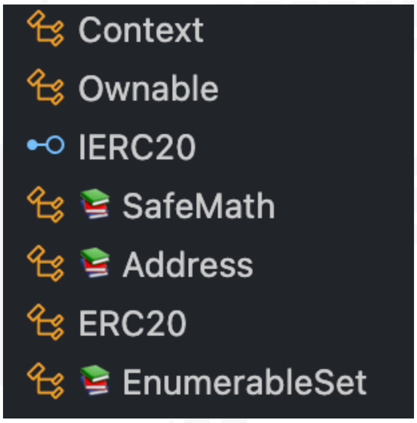
```
./library/Math.sol
./utils/SafeBEP20.sol
./access/Ownable.sol
./utils/ReentrancyGuard.sol
./eZKaliburProxy.sol
./library/Whitelist.sol
./interfaces/IMeerkatReferral.sol
./interfaces/IERC721.sol
```

SWORD



```
Context
Ownable
IERC20
SafeMath
Address
ERC20
EnumerableSet
```

xSWORD



```
Context
Ownable
IERC20
SafeMath
Address
ERC20
EnumerableSet
```

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

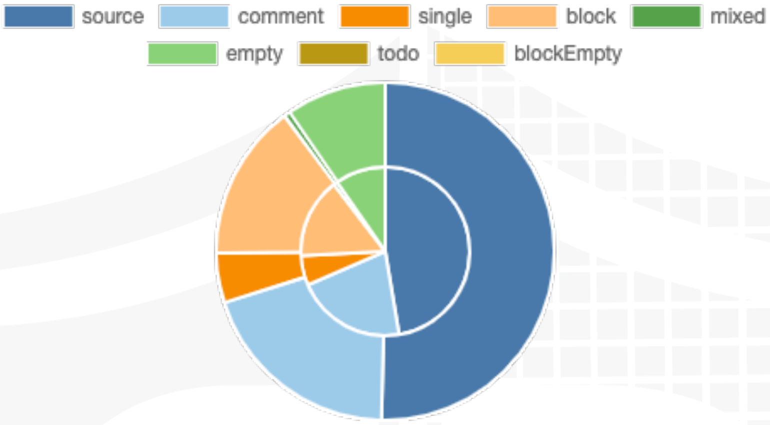
File Name	SHA-1 Hash
contracts/launchpad/icoPrivate.sol	e3698702a319479a8b310086f139db7cae53cfa8
contracts/launchpad/ico.sol	471e5b09fbedb9273355206b9f600f9fd8faa349
contracts/farming/eZKaliburProxy.sol	ef5df2f5a5dec60215a5df93a93bb44c5e742d41
contracts/farming/utils/AddressUpgradeable.sol	c8939d52cd5e15f93e1b14fb3bafcc727630a637
contracts/farming/utils/SafeBEP20.sol	c7feb7cd370ef8321a271d9c50827d474cf30b56
contracts/farming/utils/ContextUpgradeable.sol	0b9573383d939289977f200120392dacd629917f
contracts/farming/utils/Address.sol	58cc6e8fad92ee5b7cef524a1ef94677fe23eafe
contracts/farming/utils/ReentrancyGuard.sol	e87bbb6ad353ea74faace51953364cfd3edd0ede
contracts/farming/interfaces/IBEP20.sol	e4a8da7c50af79b6e5cff0c265c5b8e84a115388
contracts/farming/interfaces/IERC721.sol	8a508cebe3329e6bca41e6e44d312f7fb2279470
contracts/farming/interfaces/IMeerkatReferral.sol	1073946eecbf0f22b643340d0a590f55e7de4c91

contracts/farming/library/Math.sol	44f2973c0cb39f3a7e46582fea2b6c238ea796d3
contracts/farming/library/Whitelist.sol	54fd14495a2a50ab87777ff3719975fe149105e8
contracts/farming/library/SafeMath.sol	1a7688732b0260aacdd11cc2c2e4230118229e4e
contracts/farming/library/WhitelistUpgradeable.sol	c7f50d9a9dfd3bd6ee6b40a446aa1946e8891d08
contracts/farming/proxy/Initializable.sol	f499d19ef9ec2471f75802c169d5e5a861fb93a4
contracts/farming/access/Context.sol	6734a8153c1738efdb6f809c5c30687ad6d2af4c
contracts/farming/access/OwnableUpgradeable.sol	7e38d3d80fa042e58c3415b63973fff3c44fb821
contracts/farming/access/Ownable.sol	cb0a5ddfdf519ef6b42c6d495e8a649e3f20dd2e
contracts/farming/access/Ownable_flattened.sol	485c135c1cf6c9f193c1934f2d69ea1f57291704
contracts/farming/eZKaliburMaster.sol	fb90079cebaddd133e819c2f6879700bbce07c54
contracts/token/SWORD.sol	7a357816cc10941f307db86f70aa7b35a33004c8
contracts/token/xSWORD.sol	6166df966a9067dd8c8062322bd6f930af2b38ec

Metrics

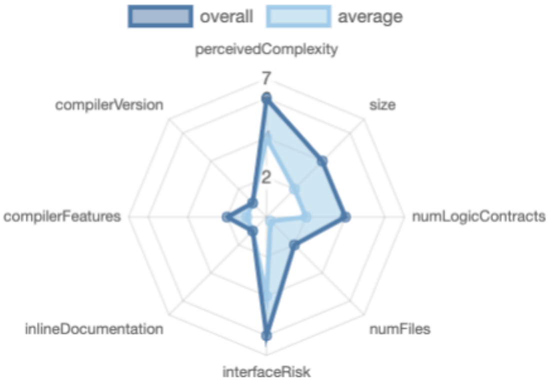
Source Lines

v1.0



Risk Level

v1.0



Capabilities

Components

📝Contracts	📚Libraries	🔍Interfaces	🎨Abstract
18	21	13	10

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
203	8

External	Internal	Private	Pure	View
109	445	26	95	143

StateVariables

Total	🌐Public
113	60

Capabilities

Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts

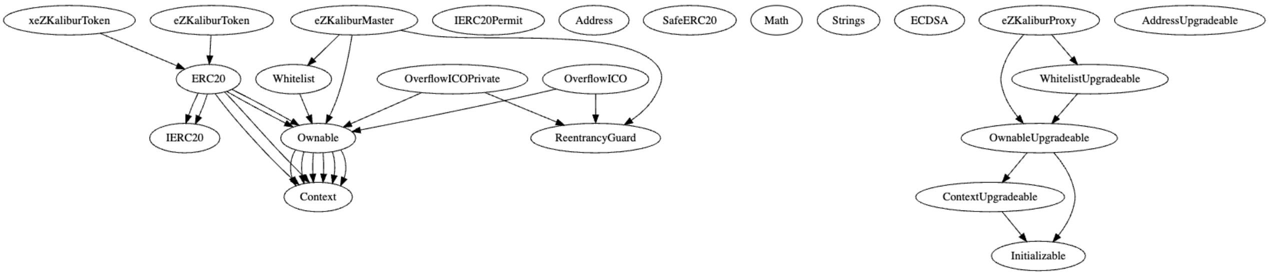
`^0.8.13`
`>=0.6.12`
`^0.6.0`
`>=0.6.0 <0.8.0`
`^0.6.2`
`>=0.4.0`
`^0.6.12`
`0.6.12`
`>=0.4.24 <0.8.0`
		`yes`	`yes`
			(22 asm blocks)
				—

📤 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧮 Uses Hash Functions	🔖 ECRecover	🌀 New/Create/Create2
`yes`	—	`yes`	`yes`	`yes`	—

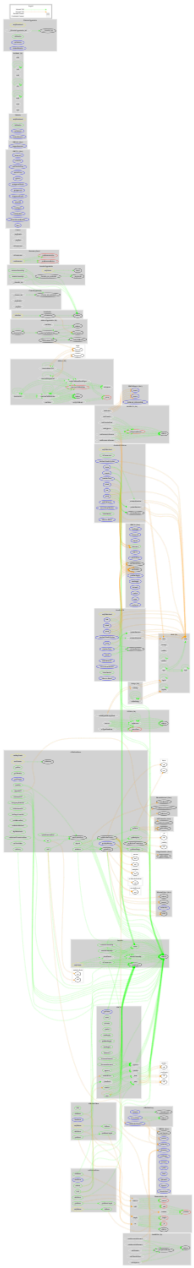
♻️ TryCatch	Σ Unchecked
—	`yes`

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. Overall checkup (Smart Contract Security)

Overall checkup (Smart Contract Security)

Tested	Verified
✓	✓

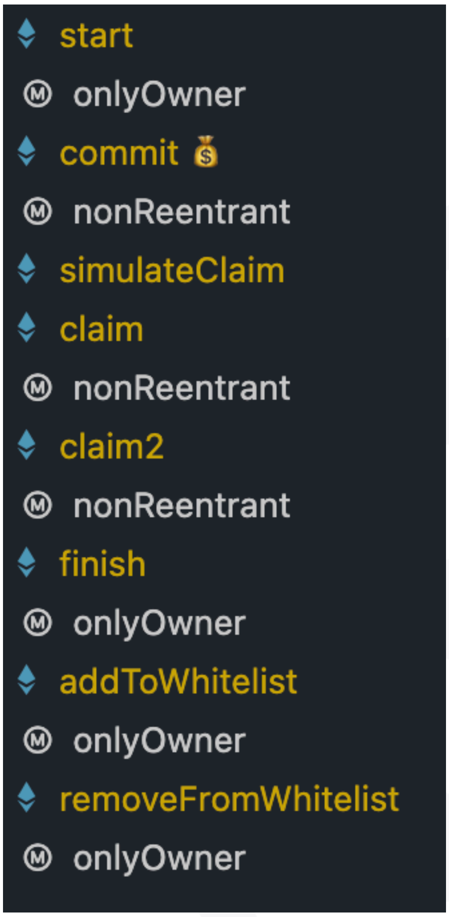
Legend

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

Modifiers and public functions

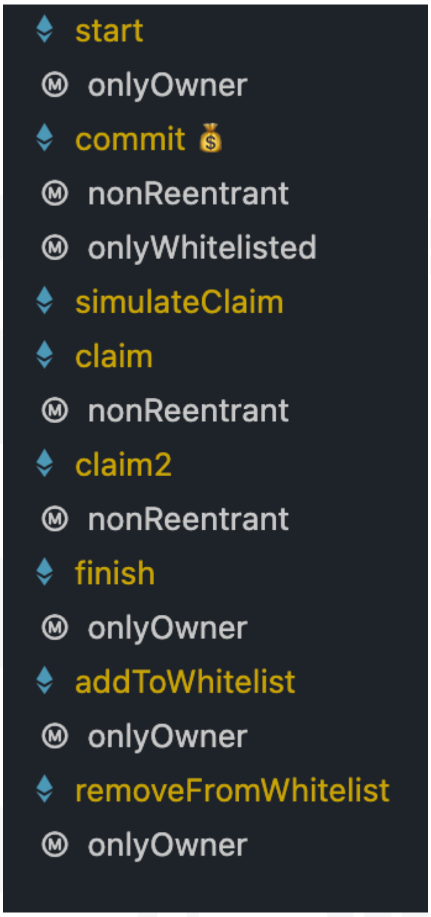
v1.0

ico.sol



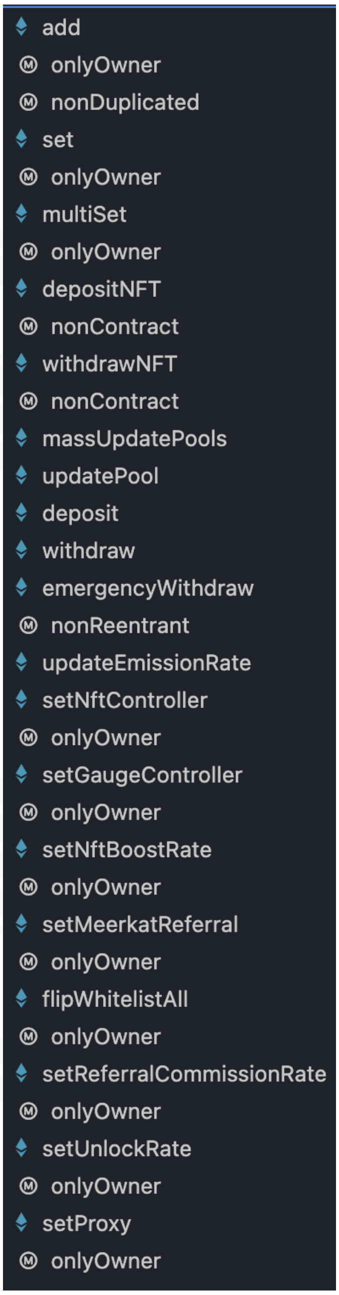
♦ start
Ⓜ onlyOwner
♦ commit 💰
Ⓜ nonReentrant
♦ simulateClaim
♦ claim
Ⓜ nonReentrant
♦ claim2
Ⓜ nonReentrant
♦ finish
Ⓜ onlyOwner
♦ addToWhitelist
Ⓜ onlyOwner
♦ removeFromWhitelist
Ⓜ onlyOwner

icoPrivate



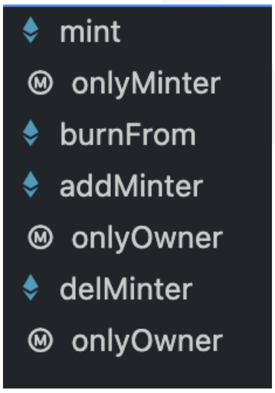
♦ start
Ⓜ onlyOwner
♦ commit 💰
Ⓜ nonReentrant
Ⓜ onlyWhitelisted
♦ simulateClaim
♦ claim
Ⓜ nonReentrant
♦ claim2
Ⓜ nonReentrant
♦ finish
Ⓜ onlyOwner
♦ addToWhitelist
Ⓜ onlyOwner
♦ removeFromWhitelist
Ⓜ onlyOwner

eZkaliburMaster



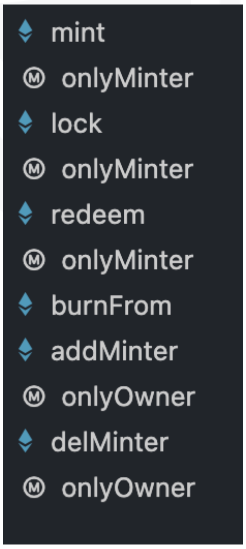
♦ add
Ⓜ onlyOwner
Ⓜ nonDuplicated
♦ set
Ⓜ onlyOwner
♦ multiSet
Ⓜ onlyOwner
♦ depositNFT
Ⓜ nonContract
♦ withdrawNFT
Ⓜ nonContract
♦ massUpdatePools
♦ updatePool
♦ deposit
♦ withdraw
♦ emergencyWithdraw
Ⓜ nonReentrant
♦ updateEmissionRate
♦ setNftController
Ⓜ onlyOwner
♦ setGaugeController
Ⓜ onlyOwner
♦ setNftBoostRate
Ⓜ onlyOwner
♦ setMeerkatReferral
Ⓜ onlyOwner
♦ flipWhitelistAll
Ⓜ onlyOwner
♦ setReferralCommissionRate
Ⓜ onlyOwner
♦ setUnlockRate
Ⓜ onlyOwner
♦ setProxy
Ⓜ onlyOwner

SWORD



♦ mint
Ⓜ onlyMinter
♦ burnFrom
♦ addMinter
Ⓜ onlyOwner
♦ delMinter
Ⓜ onlyOwner

xSWORD



♦ mint
Ⓜ onlyMinter
♦ lock
Ⓜ onlyMinter
♦ redeem
Ⓜ onlyMinter
♦ burnFrom
♦ addMinter
Ⓜ onlyOwner
♦ delMinter
Ⓜ onlyOwner

Note:

- ❖ General fork from Mad MeerKat Finance and OverFlow ICO
  - Contracts inside are the same as the Links Mention Below with very minor changes
    - Token: https://arbiscan.io/token/0x56b251d4b493ee3956E3f899D36b7290902D2326
    - xToken: https://arbiscan.io/token/0xB8635f1644422e7EbcA07C06b839075A74f57dBB
    - Master Farming: https://arbiscan.io/address/0xa73ae666ceb460d5e884a20fb30de2909604557a#code
    - Differences between ezKalibur, Mad MeerKat and OverFlow ICO contract is:
      - The Farm and token contracts only have name changes and the logic of all contracts remain unchanged
      - The Private ICO contract has an added functionality of Whitelisting. The normal one does too, but the functionality has not been implemented in the 'public ico' contract.

Ownership Privileges

- ❖ *ico.sol* -
  - Start and Finish ICO. Moreover, it can be ended only after the end time has passed

- ❖ *icoPrivate.sol* -
  - Start and Finish ICO. Moreover, it can be ended only after the end time has passed
  - Add/Remove addresses in the whitelist, and only the whitelisted address can deposit tokens.

- ❖ *eZKalibur.sol* -
  - Add new LP to the pool
  - Set a pool's allocation point to any arbitrary value
  - Set NFT controller, proxy, and Gauge Controller address
  - Toggle All Whitelist
  - Set unlock rate basis points but it must be less than or equal to 10000

- ❖ *SWORD.sol* -

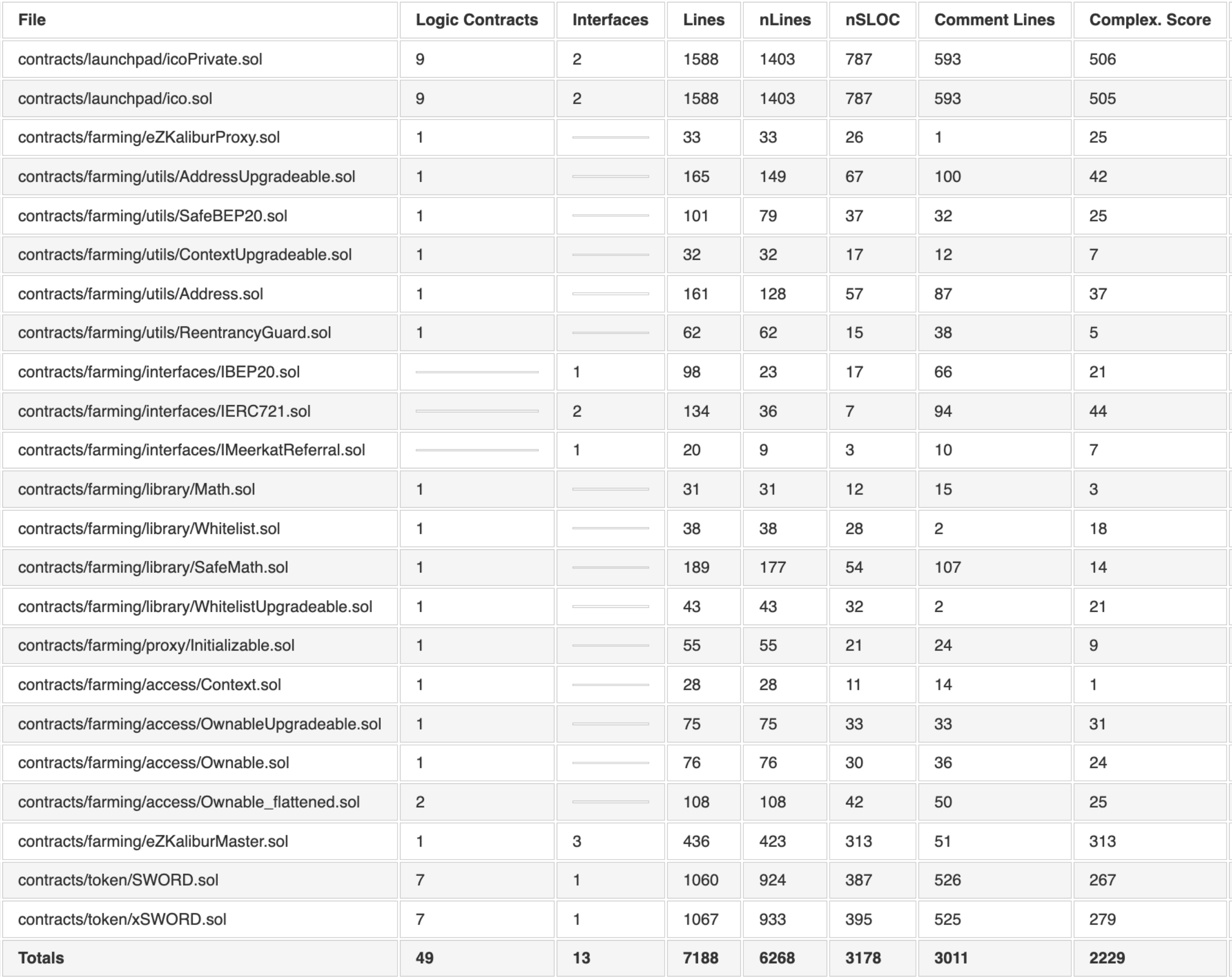
    - The owner can add/remove minter addresses, and these addresses with he minting permission can mint unlimited tokens

- *xSWORD.sol* -
    - The owner can add/remove minter addresses, and these addresses with he minting permission can mint unlimited tokens
    - The minter addresses can also withdraw tokens from the contract balance

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

Source Units in Scope

v1.0



File	Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score
contracts/launchpad/icoPrivate.sol	9	2	1588	1403	787	593	506
contracts/launchpad/ico.sol	9	2	1588	1403	787	593	505
contracts/farming/eZKaliburProxy.sol	1		33	33	26	1	25
contracts/farming/utils/AddressUpgradeable.sol	1		165	149	67	100	42
contracts/farming/utils/SafeBEP20.sol	1		101	79	37	32	25
contracts/farming/utils/ContextUpgradeable.sol	1		32	32	17	12	7
contracts/farming/utils/Address.sol	1		161	128	57	87	37
contracts/farming/utils/ReentrancyGuard.sol	1		62	62	15	38	5
contracts/farming/interfaces/IBEP20.sol		1	98	23	17	66	21
contracts/farming/interfaces/IERC721.sol		2	134	36	7	94	44
contracts/farming/interfaces/IMeerkatReferral.sol		1	20	9	3	10	7
contracts/farming/library/Math.sol	1		31	31	12	15	3
contracts/farming/library/Whitelist.sol	1		38	38	28	2	18
contracts/farming/library/SafeMath.sol	1		189	177	54	107	14
contracts/farming/library/WhitelistUpgradeable.sol	1		43	43	32	2	21
contracts/farming/proxy/Initializable.sol	1		55	55	21	24	9
contracts/farming/access/Context.sol	1		28	28	11	14	1
contracts/farming/access/OwnableUpgradeable.sol	1		75	75	33	33	31
contracts/farming/access/Ownable.sol	1		76	76	30	36	24
contracts/farming/access/Ownable_flattened.sol	2		108	108	42	50	25
contracts/farming/eZKaliburMaster.sol	1	3	436	423	313	51	313
contracts/token/SWORD.sol	7	1	1060	924	387	526	267
contracts/token/xSWORD.sol	7	1	1067	933	395	525	279
Totals	**49**	**13**	**7188**	**6268**	**3178**	**3011**	**2229**

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

Critical issues

No critical issues

High issues

No high issues

Medium issues

No medium issues

Low issues

Issue	File	Type	Line	Description
#1	SWORD .sol	Missing Events Arithmetic	All	Emit an event for critical parameter changes
#2	xSWOR D.sol	Missing Events Arithmetic	All	Emit an event for critical parameter changes
#3	All	Old Compiler Version	—	The contracts use a very old compiler version which is not recommended for deployment as it is susceptible to known vulnerabilities
#4	eZkalib ur.sol	Missing Events Arithmetic	188-214, 402, 406, , 411, 416	Emit an event for critical parameter changes

Informational issues

Issue	File	Type	Line	Description
#1	All	Contract doesn't import npm packages from source (like OpenZeppelin etc.)	—	We recommend importing all packages from npm directly without flattening the contract. Functions could be modified or can be susceptible to vulnerabilities

Audit Comments

We recommend you to use the special form of comments (NatSpec Format, Follow link for more information https://docs.soliditylang.org/en/latest/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.

18. May 2023:

- This project consists of the following forks
  - Mad Meer Kat Finance
  - OverflowICO
- Unit tests with 100% code coverage was not provided to SolidProof so we cannot ensure complete functional correctness of the code's logic.
- We recommend ezKalibur team to conduct unit and fuzz tests thoroughly to rule out possibilities of an unwanted logical and calculation errors.
- Read whole report and modifiers section for more information
- The low issues that remain unfixed in the Mad MeerKat codebase still exist in the forked code.
- We recommend using a multisig wallet for the owner address to prevent any risk of the loss of private key
- Do your own research here

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	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	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	PASSED
SWC-102	Outdated Compiler Version	CWE-937: Using Components with Known Vulnerabilities	NOT 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 Development | Marketing

MADE IN GERMANY