SOLIDProof

Bring trust into your projects

Blockchain Security | Smart Contract Audits | KYC Development | Marketing

MADE IN GERMANY

ZkHero

Audit

Security Assessment

08. April, 2023

For

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	06. April 2023	• Layout project • Automated- /Manual-Security Testing • Summary

Network
ZkSync

Website
https://zkhero.app/

Telegram
https://t.me/ZksyncHero

Twitter
https://twitter.com/ZkSyncHero

Medium
https://medium.com/@zksynchero

Description

In the ZkHero game, there 4 kinds of monsters distinguished by the 4 level. The higher the level, the greater the reward and EXP. Although there is some weak and easy to fight monsters but be careful when facing them, there are not a joke and you will lose the fight,leading to many little gain in EXP and no gain in bounty reward.

Project Engagement

During the Date of 06 April 2023, **ZkHero 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 Link

v1.0

- https://explorer.zksync.io/address/0x4D0C07D4Ee72F3A5B49AFB1f7332C3CcC9C5d094#contract

Note for Investors: We only Audited one contract for **ZkHero Team**. However, If the project has other contracts (for example, a Presale contract etc), and they were not provided to us in the audit scope then we cannot comment on its security and we are not responsible for it in any way.

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:

```
./interfaces/IMuteSwitch.sol
./libs/ZkHeroERC20.sol
./libs/ReentrancyGuard.sol
```

Tested Contract Files

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

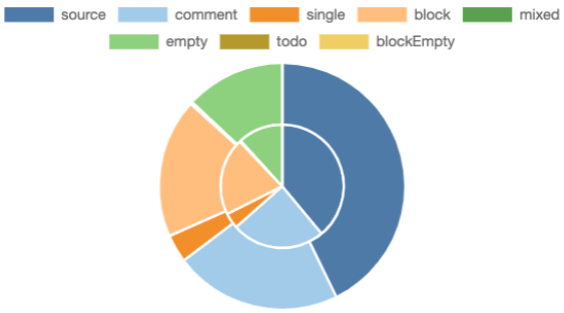
A file with a different Hash has been modified, intentionally or otherwise, after the security review. A different Hash could be (but not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of this review.

v1.0

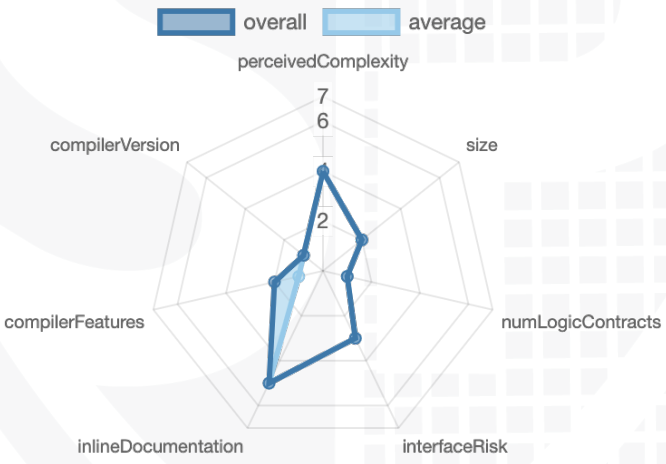
File Name	SHA-1 Hash
contracts/interfaces/IManager.sol	6e02c2083c24aac793a7fcdef1bae1a4290c7e73
contracts/interfaces/IMuteSwitch.sol	1edeb6c33f6ad6115da0f753d0f1adf0c61742cc
contracts/interfaces/IERC20.sol	ff75aac10b431f1097c0d6dfb1a16b0c75ecee4d
contracts/ZkHero.sol	5453f436bf389c3a7edf04366a8a30ec2a43f706
contracts/libs/Context.sol	707026989f1d8217aa9fed3d6a58f37b6a5f9c30
contracts/libs/ZkHeroERC20.sol	10adb96ad99f31854cfc8aed7120a64e995e4d73
contracts/libs/SafeMath.sol	3d176d7b223e34d655fd758154b0a3d6a28a4255
contracts/libs/Ownable.sol	23038c126ddf36ca3ebb14a57866d48ea560b36d
contracts/libs/ERC20.sol	2fd78d2cca1221141c2a17357a279875627e8480
contracts/libs/ReentrancyGuard.sol	887dde34900a5317c650af508f883d7a4cac1d91

Metrics

Source Lines
v1.0



Risk Level
v1.0



Capabilities

Components

📝Contracts	📚Libraries	🔍Interfaces	💊Abstract
3	1	5	3

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
61	3

External	Internal	Private	Pure	View
42	61	3	8	31

StateVariables

Total	🌐Public
30	16

Capabilities

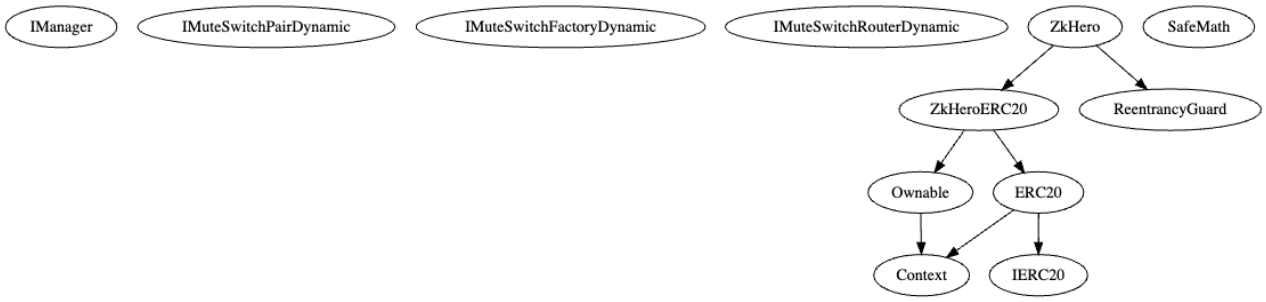
Solidity Versions observed	🧪 Experimental Features	💰 Can Receive Funds	🖥 Uses Assembly	💣 Has Destroyable Contracts
`^0.8.0`		`yes`		

📩 Transfers ETH	⚡ Low-Level Calls	👥 DelegateCall	🧱 Uses Hash Functions	🦘 ECRecover	🌀 New/Create/Create2

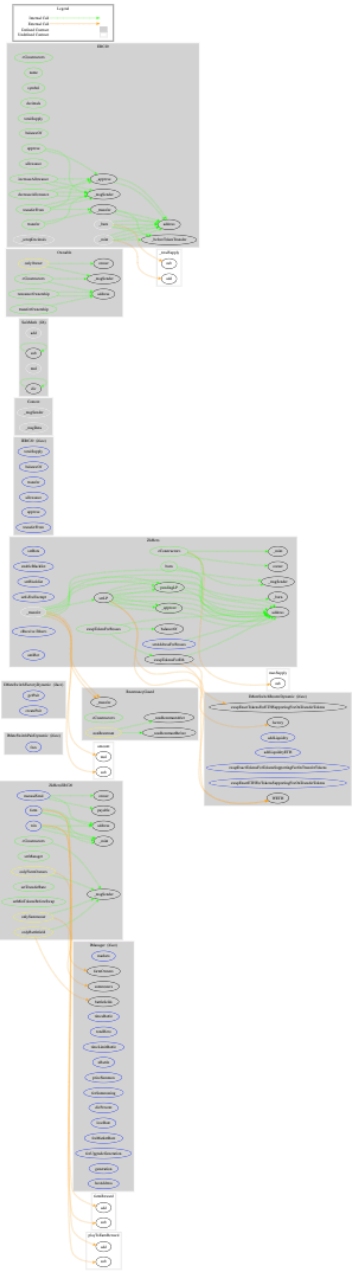
♻️ TryCatch	Σ Unchecked

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. Correct implementation of Token standard
3. Deployer cannot mint any new tokens
4. Deployer cannot burn or lock user funds
5. Deployer cannot pause the contract
6. Deployer cannot set fees
7. Deployer cannot blacklist/antisnipe addresses
8. Overall checkup (Smart Contract Security)

Is contract an upgradeable

Name	
Is contract an upgradeable?	No

Correct implementation of Token standard

ERC20				
Function	**Description**	**Exist**	**Tested**	**Verified**
TotalSupply	Provides information about the total token supply	✓	✓	✓
BalanceOf	Provides account balance of the owner's account	✓	✓	✓
Transfer	Executes transfers of a specified number of tokens to a specified address	✓	✓	✓
TransferFrom	Executes transfers of a specified number of tokens from a specified address	✓	✓	✓
Approve	Allow a spender to withdraw a set number of tokens from a specified account	✓	✓	✓
Allowance	Returns a set number of tokens from a spender to the owner	✓	✓	✓

Write functions of contract v1.0

1. antiBot
2. approve
3. burn
4. decreaseAllowance
5. enableBlacklist
6. farm
7. increaseAllowance
8. renounceOwnership
9. setBlacklist
10. setBots
11. setIsFeeExempt
12. setManager
13. setTransferRate
14. transfer
15. transferFrom
16. transferOwnership
17. win

Deployer cannot mint any new tokens

Name	Exist	Tested	Status
Deployer cannot mint	✓	✓	✓
Max / Total Supply	300.000.000		

Comments:

v1.0

- Owner cannot mint new tokens

Deployer cannot burn or lock user funds

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

Comments:

v1.0

- Tokens can be burned by msg.sender, and the owner cannot burn tokens from other accounts.

Deployer cannot pause the contract

Name	Exist	Tested	Status
Deployer cannot pause	–	–	–

Deployer cannot set fees

Name	Exist	Tested	Status
Deployer cannot set fees over 10%	✓	✓	✓
Deployer cannot set fees to nearly 100% or to 100%	✓	✓	✓

Deployer can blacklist/antisnipe addresses

Name	Exist	Tested	Status
Deployer cannot blacklist/antisnipe addresses	✓	✓	✗

Comments:

v1.0

- Owner is able to blacklist addresses

Overall checkup (Smart Contract Security)

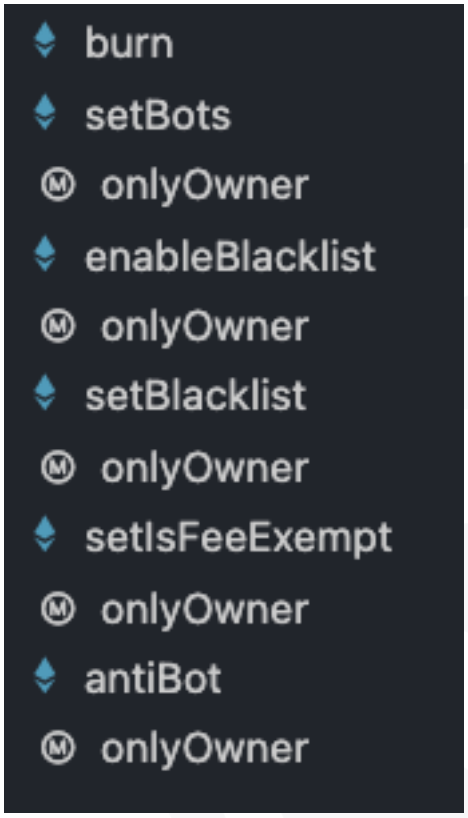
Tested	Verified
✓	✓

Legend

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

Modifiers and public functions
v1.0

ZkHero



ZkHeroERC20

Ownership Privileges

- *ZkHero.sol*
  - Set bot address, and the amount for the anti-bot
  - Enable/Disable blacklist
  - Include/Exclude addresses from fees

- *ZkHeroER20.sol*
  - Set manager address, and this contract will verify whether an address is a farmOwner, Summoner, or BattleField

- *Existing Modifiers*
  - onlyOwner
  - onlyFarmOwners
  - onyBattlefield

- There are several authorities which are authorized to call some functions, that means, if the owner is renounced, another address is still authorized to call functions

- Be aware of this

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/interfaces/IManager.sol		1	37	6	3	1	33
contracts/interfaces/IMuteSwitch.sol		3	65	7	5	1	29
contracts/interfaces/IERC20.sol		1	27	5	3	1	13
contracts/ZkHero.sol	1		181	177	124	16	135
contracts/libs/Context.sol	1		12	12	9	1	1
contracts/libs/ZkHeroERC20.sol	1		95	95	75	1	70
contracts/libs/SafeMath.sol	1		41	41	32	3	8
contracts/libs/Ownable.sol	1		56	56	27	21	24
contracts/libs/ERC20.sol	1		280	280	89	161	79
contracts/libs/ReentrancyGuard.sol	1		67	67	21	38	7
Totals	**7**	**5**	**861**	**746**	**388**	**244**	**399**

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	All	Contract doesn't import npm packages from source (like OpenZeppelin etc.)	—	We recommend to import all packages from npm directly without flatten the contract. Functions could be modified or can be susceptible to vulnerabilities
#2	All	A floating pragma is set	—	The current pragma Solidity directive is „“^0.8.0”.
#3	ZkHero.sol	Missing Zero Address Validation (missing-zero-check)	49, 59, 63	Check that the address is not zero
#4	ZkHerERC20.sol	Missing Events Arithmetic	All	Emit an event for critical parameter changes

Informational issues

Issue	File	Type	Line	Description
#1	ZkHero.sol	State variables that could be declared constant (constable-states)	22	Add the `constant` attributes to state variables that never change
#2	ZkHero.sol	Error message is missing	50, 116, 139	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.

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.

08. April 2023:

- There is still an owner (Owner still has not renounced ownership)
- Owner can deploy a new version of the contract which can change any limit and give owner new privileges
- The Manager contract was only provided as an interface to us in the audit scope, so we cannot comment on the security of the contract because we were not provided with the complete code
- 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	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	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