SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: WhiteBIT
Date: 8 September, 2023

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for WhiteBIT
Approved By	Paul Fomichov | Lead Solidity SC Auditor at Hacken OU
Platform	EVM
Language	Solidity
Methodology	Link
Website	https://whitebit.com
Changelog	04.07.2023 - Initial Review 01.08.2023 - Second Review 08.09.2023 - Third Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by WhiteBIT (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

System Overview

WB Soul Ecosystem is a WB Network blockchain - based ecosystem designed to bring a comprehensive decentralized identity and attributes management system. Soulbound enables users to create unique identifiers called Souls, which are associated with their wallets, by supplying relevant information to the network. The network, in turn, associates Souls with two types of features - dynamic and immutable, through a system of smart contracts. Soulbound provides users with a decentralized platform for creating and managing digital identities with associated dynamic and permanent features on the blockchain.

The files in the scope:

- **EnumerableSet.sol** - OpenZeppelin library for managing abstract data type of primitive types.
- **SoulRegistry.sol** - Is a contract that enables the registration of Souls and the management of its addresses (associating/dissociating secondary addresses, changing the primary address). This contract is controlled by the owner (WhiteBIT), with the possibility of granting primary Soul addresses the ability to manage their list of secondary addresses.
- **SoulAttributeRegistry.sol** - Is a contract that allows registering Soul Attributes and binding specific attributes to specific Souls. This contract facilitates the registration of Attribute and provides functionality to bind specific Attributes to specific Souls.
- **SoulBoundTokenRegistry.sol** - The SoulBoundTokenRegistry contract is responsible for managing the binding of SoulBound tokens to Souls. The contract facilitates the association of a token from a specified collection to a particular Soul.
- **Ownable.sol** - contract module from OpenZeppelin, which provides a basic access control mechanism, where there is an account (an owner) that can be granted exclusive access to specific functions.
- **SoulRegistryConfig.sol** - Simple registry configuration contract that provides addresses assignment rules.
- **Deployer.sol** - Basic deployer contract for deploying all registries in a single place
- **SoulLevel.sol** - This contract implements the ISoulAttribute interface and represents the current Hold level of a user on WhiteBIT

- **IsVerified.sol** - This contract implements the ISoulAttribute interface and represents the current KYC verification status of a user on WhiteBIT
- **ISoulAttributeRegistry.sol** - The Interface of the SoulAttributeRegistry.sol
- **ISoulBoundTokenRegistry.sol** - The Interface of the SoulBoundTokenRegistry.sol
- **Context.sol** - Classic Context contract from OpenZeppelin.
- **SoulAttribute.sol** - Contract with a predefined IERC165 methods.
- ISoulBoundTokenCollection.sol - Interface of the SoulBoundTokenRegistry.sol
- **ISoulRegistry.sol** - The Interface of the SoulRegistry.sol
- **ISoulFeature.sol** - ISoulFeature is an interface for defining specific soul features.
- **ISoulFeatureRegistry.sol** - Interface for
- **IERC165.sol** - The Interface of the ERC165 standard.

Privileged roles

- Owner privilege roles for Ownable.sol:
  - Transfers ownership of the contract to a new account.
  - Renounce ownership of the contract.

- Owner privilege roles SoulFeatureRegistry.sol:
  - Ability to register new features.
  - Ability to pause registered features.
  - Ability to unpause paused features.

- Owner privilege roles SoulRegistry.sol:
  - Register new soul using specified address as a primary address.
  - Change registered soul's primary address.
  - Assign new address to existing soul.

- Owner privilege roles SoulRegistryConfig.sol:
  - Allow souls to manage addresses list.
  - Disallow souls to manage addresses list.
  - Update addresses per soul limit.

Executive Summary

The score measurement details can be found in the corresponding section of the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are provided.
- Technical description is provided.
- NatSpecs are very good.

Code quality

The total Code Quality score is **10** out of **10**.

- The development environment is configured.

Test coverage

Code coverage of the project is **100%** (branch coverage).

- Deployment and user interactions are covered with tests.

Security score

As a result of the audit, the code does not contain issues. The security score is **10** out of **10**.

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **10**. The system users should acknowledge all the risks summed up in the risks section of the report.

1
2
3
4
5
6
7
8
9
10

The final score

Table. The distribution of issues during the audit

Review date	Low	Medium	High	**Critical**
04 July 2023	2	0	0	0
01 August 2023	0	0	0	0
08 September 2023	0	0	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and specific vulnerabilities. Here are some items considered:

Item	Description	Status	Related Issues
Default Visibility	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed	
Integer Overflow and Underflow	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed	
Outdated Compiler Version	It is recommended to use a recent version of the Solidity compiler.	Passed	
Floating Pragma	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed	
Unchecked Call Return Value	The return value of a message call should be checked.	Passed	
Access Control & Authorization	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed	
SELFDESTRUCT Instruction	The contract should not be self-destructible while it has funds belonging to users.	Passed	
Check-Effect-Interaction	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed	
Assert Violation	Properly functioning code should never reach a failing assert statement.	Passed	
Deprecated Solidity Functions	Deprecated built-in functions should never be used.	Passed	
Delegatecall to Untrusted Callee	Delegatecalls should only be allowed to trusted addresses.	Not Relevant	
DoS (Denial of Service)	Execution of the code should never be blocked by a specific contract state unless required.	Passed	
Race Conditions	Race Conditions and Transactions Order Dependency should not be possible.	Passed	

Authorization through tx.origin	tx.origin should not be used for authorization.	Passed	
Block values as a proxy for time	Block numbers should not be used for time calculations.	Passed	
Signature Unique Id	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery. EIP-712 should be followed during a signer verification.	Passed	
Shadowing State Variable	State variables should not be shadowed.	Passed	
Weak Sources of Randomness	Random values should never be generated from Chain Attributes or be predictable.	Not Relevant	
Incorrect Inheritance Order	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed	
Calls Only to Trusted Addresses	All external calls should be performed only to trusted addresses.	Passed	
Presence of Unused Variables	The code should not contain unused variables if this is not justified by design.	Passed	
EIP Standards Violation	EIP standards should not be violated.	Passed	
Assets Integrity	Funds are protected and cannot be withdrawn without proper permissions or be locked on the contract.	Passed	
User Balances Manipulation	Contract owners or any other third party should not be able to access funds belonging to users.	Passed	
Data Consistency	Smart contract data should be consistent all over the data flow.	Passed	
Flashloan Attack	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used. Contracts shouldn’t rely on values that can be changed in the same transaction.	Not Relevant	
Token Supply Manipulation	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the Customer.	Passed	

Gas Limit and Loops	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	Passed	
Style Guide Violation	Style guides and best practices should be followed.	Passed	
Requirements Compliance	The code should be compliant with the requirements provided by the Customer.	Passed	
Environment Consistency	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed	
Secure Oracles Usage	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant	
Tests Coverage	The code should be covered with unit tests. Test coverage should be sufficient, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed	
Stable Imports	The code should not reference draft contracts, which may be changed in the future.	Passed	

Findings

🟥🟥🟥🟥 Critical

No critical severity issues were found.

🟧🟧🟧 High

No high severity issues were found.

🟨🟨 Medium

No medium severity issues were found.

🟦 Low

L01. Missing Event Emitting

Impact	Low
Likelihood	Low

Critical state changes should emit events for tracking things off-chain. The functions do not emit events on change of important values.

Path: ./contracts/SoulRegistryConfig.sol : allowPublicModification(), disallowPublicModification(), updateMaxAddressesPerSoul()

Recommendation: Emit events on critical state changes.

Found in: 3e2f867

Status: Fixed (Revised commit: 102c891)

L02. Copy Of Well Known Contract

Impact	Low
Likelihood	Low

Well-known contracts from projects like OpenZeppelin should be imported directly from source as the projects are in development and may update the contracts in future. The system uses a copy of OpenZeppelin's EnumerableSet, Context and Ownable.

Paths: ./libraries/EnumerableSet.sol,

./contracts/Context.sol,

./contracts/Ownable.sol,

Recommendation: Import contracts directly from OpenZeppelin's package.

Found in: 3e2f867

Status: Fixed (Revised commit: 102c891)

Informational

I01. Floating Pragma

The project uses floating pragmas ^0.8.19.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively

Paths: ./contracts/Context.sol :

./contracts/Ownable.sol :

./libraries/EnumerableSet.sol :

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Found in: 3e2f867

Status: Fixed (Revised commit: 102c891)

I02. Style Guide Violation

Contract readability and code quality are influenced significantly by adherence to established style guidelines. In Solidity programming, there exist certain norms for code arrangement and ordering. These guidelines help to maintain a consistent structure across different contracts, libraries, or interfaces, making it easier for developers and auditors to understand and interact with the code.

Following the Solidity naming convention is strongly recommended.

Paths: ./contracts/attributes/SoulLevel.sol : description,

./contracts/attributes/IsVerified.sol : description

Recommendation: Consistent adherence to the official Solidity style guide is recommended. This enhances readability and maintainability of the code, facilitating seamless interaction with the contracts. Following Solidity's naming conventions further enriches the quality of the code.

Found in: 3e2f867

Status: Mitigated (The Customer stated constants are part of the *ISoulFeature.sol* interface.)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts Hacken is using a risk-based approach that considers the potential impact of any vulnerabilities and the likelihood of them being exploited. The matrix of impact and likelihood is a commonly used tool in risk management to help assess and prioritize risks.

The impact of a vulnerability refers to the potential harm that could result if it were to be exploited. For smart contracts, this could include the loss of funds or assets, unauthorized access or control, or reputational damage.

The likelihood of a vulnerability being exploited is determined by considering the likelihood of an attack occurring, the level of skill or resources required to exploit the vulnerability, and the presence of any mitigating controls that could reduce the likelihood of exploitation.

Risk Level	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

Risk Levels

Critical: Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.

High: High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.

Medium: Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.

Low: Major deviations from best practices or major Gas inefficiency. These issues won't have a significant impact on code execution, don’t affect security score but can affect code quality score.

Impact Levels

High Impact: Risks that have a high impact are associated with financial losses, reputational damage, or major alterations to contract state. High impact issues typically involve invalid calculations, denial of service, token supply manipulation, and data consistency, but are not limited to those categories.

Medium Impact: Risks that have a medium impact could result in financial losses, reputational damage, or minor contract state manipulation. These risks can also be associated with undocumented behavior or violations of requirements.

Low Impact: Risks that have a low impact cannot lead to financial losses or state manipulation. These risks are typically related to unscalable functionality, contradictions, inconsistent data, or major violations of best practices.

Likelihood Levels

High Likelihood: Risks that have a high likelihood are those that are expected to occur frequently or are very likely to occur. These risks could be the result of known vulnerabilities or weaknesses in the contract, or could be the result of external factors such as attacks or exploits targeting similar contracts.

Medium Likelihood: Risks that have a medium likelihood are those that are possible but not as likely to occur as those in the high likelihood category. These risks could be the result of less severe vulnerabilities or weaknesses in the contract, or could be the result of less targeted attacks or exploits.

Low Likelihood: Risks that have a low likelihood are those that are unlikely to occur, but still possible. These risks could be the result of very specific or complex vulnerabilities or weaknesses in the contract, or could be the result of highly targeted attacks or exploits.

Informational

Informational issues are mostly connected to violations of best practices, typos in code, violations of code style, and dead or redundant code.

Informational issues are not affecting the score, but addressing them will be beneficial for the project.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Initial review scope

Repository	https://github.com/whitebit-exchange/souls-ecosystem-contracts
Commit	3e2f8675abc1cca244d23285cb3cd25e88aa3a1e
Whitepaper	-
Requirements	Confidential
Technical Requirements	Confidential
Contracts	File: contracts/Context.sol SHA3: fbed13e8608f0be145eb6a1ed6660d99ea7538519c8f5dec6513ec62893b6918 File: contracts/Deployer.sol SHA3: 154e8ba18920380313876471868c022703f7251106594dc9fa34881b5dec63dd File: contracts/Ownable.sol SHA3: d2ad9b0c946b5c87e480d4caab8ac9f4adf4180a953ce9b7c4de1426ed68986e File: contracts/SoulAttribute.sol SHA3: cfabab2a56fc27050c4bb9bd53f3b58ec82f0f1ec975bb97a5a537945715790b File: contracts/SoulAttributeRegistry.sol SHA3: 7d428958ed3f27dd9fce73a355f9e8f696a8ec9899d7ca022e6be4c2ba49f848 File: contracts/SoulBoundTokenRegistry.sol SHA3: 4e016dee5b92f7e30924171dc371d806e0ebfe0c616737e683cc05ddeb5b4b45 File: contracts/SoulFeatureRegistry.sol SHA3: 7eabf67dca015a10eee777d6ec15168d118a2d209e30d3ce774dde301a5e6b1f File: contracts/SoulRegistry.sol SHA3: 8ff2634845af3491696e4e8b3f63b9bab22b294e4f8b2af4c20716a6a7cb21b5 File: contracts/SoulRegistryConfig.sol SHA3: 28928288828bc54daa530c310bd7ab6ec0fb74a2bbaa703eff95648305a9f0ba File: contracts/attributes/IsVerified.sol SHA3: 77b5264737ac229308d356e432b5f869990621351729c1e551c6b407065a12e5 File: contracts/attributes/SoulLevel.sol SHA3: 426d13926882b22530aa9a59c2ec16367c527034b82083e08b3af13406b1b72e File: interfaces/IERC165.sol SHA3: 0b6324b1ecdab0c61e4fb3f6f991000b1ae36ee947b8ea0100f55224a0b92f85 File: interfaces/ISoulAttribute.sol SHA3: 1c2576f719eb0b9ce8fbff0b5ff467c8b7098449a523935da68831e094213d68 File: interfaces/ISoulAttributeRegistry.sol SHA3: 8911a5e60b5ab157043e907de7c5e5d534269d965eab9b29538b7fe6ac1db057

	File: interfaces/ISoulBoundTokenCollection.sol SHA3: e5f107d13aa7368f38defa524f77588045ea9fcc6cade4611a0cd044e106a12b File: interfaces/ISoulBoundTokenRegistry.sol SHA3: aec4cf893067b55e36e266f2039a79e253e0ed2e7850640683 2718ec467b4d7b File: interfaces/ISoulFeature.sol SHA3: f72ddea8371da5fdeae172c53117de5256c86d9e36f8e5a7fb35850f75065e68 File: interfaces/ISoulFeatureRegistry.sol SHA3: 9d7fc2ad838d6688e230a25366c180073ccbd5b6b7b0f6ff3534f00596a3d0a2 File: interfaces/ISoulRegistry.sol SHA3: abc57709e7bf67923d53a3e2da2d617bb8b725289910de822746131196808bce File: libraries/EnumerableSet.sol SHA3: 7cdb59b33bab09cb82e8b087e8a2207cae99b79dce95e14ccf7054705424e319

Second review scope

Repository	https://github.com/whitebit-exchange/souls-ecosystem-contracts
Commit	102c8911a74eb294516206a0bd43f5857fabc00f
Whitepaper	-
Requirements	Confidential
Technical Requirements	Confidential
Contracts	File: contracts/SoulAttribute.sol SHA3: 7a4c3f281d36e3591b69dd3d5ec5bdcf7249d7557dd69d9d5ba0b30cc4a32f61 File: contracts/SoulAttributeRegistry.sol SHA3: 6e0d6dd51c9d0efa8730761f715af25b61f8deb247c58d92eee16285cc7f22d1 File: contracts/SoulBoundTokenRegistry.sol SHA3: 73a9fbf71af368097e8fed8946eef8b9a840158e65af3194a5c6b1054eac6178 File: contracts/SoulFeatureRegistry.sol SHA3: f760c25dda7ca55390b67bb39f4e53df88da78f03efb851ae568ed11e458a076 File: contracts/SoulRegistry.sol SHA3: d8beb69c132ae0ea005813a3ea3ed0d6abb04e19be83113cd3bdb69ff007ffe4 File: contracts/SoulRegistryConfig.sol SHA3: e3cae7016592cc1f23f3d238642baace566e41ba62bfaf6fec31e11c6cdc717c File: contracts/attributes/HoldLevel.sol SHA3: 2a6b9209e029df6a6a82eb4f543d8d864db2e7164296a1f841268ceff3223a44 File: contracts/attributes/IsVerified.sol SHA3: f761a24a3554dfcab547b9a526250fc2619440b5c5c3f1cc8e1d39cf67e3c285 File: contracts/tokens/EarlyBird.sol SHA3: 5daa5399c0dda11bfe8048a2a92ece70714f16cbd7c32e2fc923f96b402e9a21 File: interfaces/IERC165.sol SHA3: 0b6324b1ecdab0c61e4fb3f6f991000b1ae36ee947b8ea0100f55224a0b92f85

	File: interfaces/ISoulAttribute.sol SHA3: a552db916f592c104bdeaf544b0234387033f84895ceb22401dc3a67526bc8e7 File: interfaces/ISoulAttributeRegistry.sol SHA3: 3f0404462b872b35fa870a5dbdb6688bbc03b7ad6dc7cd5f610dbd5c6f1a2b4b File: interfaces/ISoulBoundTokenCollection.sol SHA3: 277546552957e27b990b76dca631e2b5905bcfe0160c6be1f7d538a116415ec3 File: interfaces/ISoulBoundTokenRegistry.sol SHA3: aec4cf893067b55e36e266f2039a79e253e0ed2e78506406832718ec467b4d7b File: interfaces/ISoulFeature.sol SHA3: d74f54b1c264301b20cc758d49fcd3391b400522fcc0ba8f493c34c24fd268c6 File: interfaces/ISoulFeatureRegistry.sol SHA3: 9d7fc2ad838d6688e230a25366c180073ccbd5b6b7b0f6ff3534f00596a3d0a2 File: interfaces/ISoulRegistry.sol SHA3: b1bdac5922e985c1d4990b8858b7fd9ef5ec7cb296c21f65b81d59d3da0dec66

Third review scope

Repository	https://github.com/whitebit-exchange/souls-ecosystem-contracts
Commit	f332570abecf5897e4ae9719577d78ab9f8ef0ab
Whitepaper	-
Requirements	Confidential
Technical Requirements	Confidential
Contracts	File: contracts/SoulAttribute.sol SHA3: b5399cb2a3662d401f232d46503cdd3845cb9de33fe1d8e36adf9a67083754c3 File: contracts/SoulAttributeRegistry.sol SHA3: 2345907184706d3316d88ddde527ed531bab843ba60b580108b5e2db07c9062a File: contracts/SoulBoundTokenRegistry.sol SHA3: 73a9fbf71af368097e8fed8946eef8b9a840158e65af3194a5c6b1054eac6178 File: contracts/SoulFeatureRegistry.sol SHA3: f760c25dda7ca55390b67bb39f4e53df88da78f03efb851ae568ed11e458a076 File: contracts/SoulRegistry.sol SHA3: d8beb69c132ae0ea005813a3ea3ed0d6abb04e19be83113cd3bdb69ff007ffe4 File: contracts/SoulRegistryConfig.sol SHA3: e3cae7016592cc1f23f3d238642baace566e41ba62bfaf6fec31e11c6cdc717c File: contracts/attributes/HoldAmount.sol SHA3: 53bce3448acd2613294276b0780615bb570100e5a52239ff41cb7e89406559e9 File: contracts/attributes/HoldLevel.sol SHA3: f750ab70f24f2f9d3f1757b4f88907020803f2bc0be9516e313442ef3cff48d2 File: contracts/attributes/IsVerified.sol

	SHA3: 609f12c9241b7278f68751bb777f9f3f311ea9c644ef56717e301b6860287d5f File: contracts/tokens/EarlyBird.sol SHA3: 5daa5399c0dda11bfe8048a2a92ece70714f16cbd7c32e2fc923f96b402e9a21 File: interfaces/IERC165.sol SHA3: 0b6324b1ecdab0c61e4fb3f6f991000b1ae36ee947b8ea0100f55224a0b92f85 File: interfaces/ISoulAttribute.sol SHA3: 45549c3b055bc220bfdce7bd7a4360415321725032a099613149c2bc31695ec8 File: interfaces/ISoulAttributeRegistry.sol SHA3: 44714958075d82a744818f7bba5c347a29a5bc55cc08e23e00b6b2c870912421 File: interfaces/ISoulBoundTokenCollection.sol SHA3: 277546552957e27b990b76dca631e2b5905bcfe0160c6be1f7d538a116415ec3 File: interfaces/ISoulBoundTokenRegistry.sol SHA3: aec4cf893067b55e36e266f2039a79e253e0ed2e78506406832718ec467b4d7b File: interfaces/ISoulFeature.sol SHA3: d74f54b1c264301b20cc758d49fcd3391b400522fcc0ba8f493c34c24fd268c6 File: interfaces/ISoulFeatureRegistry.sol SHA3: 9d7fc2ad838d6688e230a25366c180073ccbd5b6b7b0f6ff3534f00596a3d0a2 File: interfaces/ISoulRegistry.sol SHA3: b1bdac5922e985c1d4990b8858b7fd9ef5ec7cb296c21f65b81d59d3da0dec66