SIVIRA

SMART CONTRACT AUDIT

zokyo

September 26th, 2022 | v. 1.0

Security Audit Score

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

TECHNICAL SUMMARY

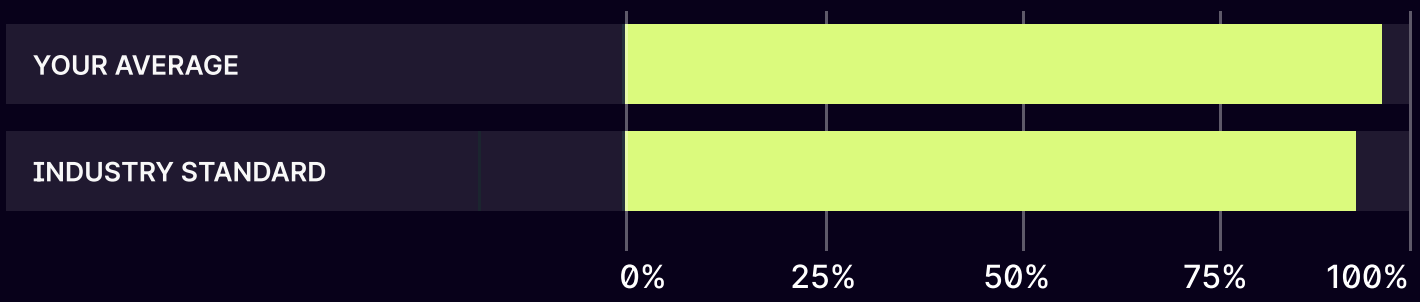
This document outlines the overall security of the SIVIRA smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the SIVIRA smart contract codebase for quality, security, and correctness.

Contract Status

There were 0 critical issue found during the audit. (See Complete Analysis)

Testable Code



The testable code is 100%, which is above the industry standard of 95%. (See Complete Analysis)

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the SIVIRA team put in place a bug bounty program to encourage further and active analysis of the smart contract.

Table of Contents

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the SIVIRA repository.

Repository: https://github.com/SIVIRA/unwallet-contracts

Last commit: b4b686e2a5ea5aa21895a0e33106fcfaaaa9aae1

Contracts under the scope:

- Identity.sol
- Proxy.sol
- Factory.sol
- IdentityProxyFactory.sol
- LockManager.sol
- ModuleManager.sol
- ModuleRegistry.sol
- Ownable.soll
- CoreBaseModule.sol
- CoreModuleAggregate.sol
- CoreRelayerModule.sol
- DelegateModule.sol
- Address.sol
- ECDSA.sol
- Math.sol
- SafeCast.sol

Throughout the review process, Zokyo Security ensures that the contract:

- Implements and adheres to existing Token standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices inefficient use of gas, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of SIVIRA smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Tru e testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

01	Due diligence in assessing the overall code quality of the codebase.
02	Cross-comparison with other, similar smart contracts by industry leaders.
03	Testing contract logic against common and uncommon attack vectors.
04	Thorough manual review of the codebase, line by line.

Executive Summary

Contracts are well written and structured. There was no critical issue found during the audit . All the mentioned findings may have an effect only in case of specific conditions performed by the contract owner and the investors interacting with it. They are described in detail in the “Complete Analysis” section. All of them were resolved by the SIVIRA team.

STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Acknowledged means that the issue was sent to the client team and the client team are aware of it, but they have chosen to not solved it. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

SYSTEM OVERVIEW

The main use case of the contracts audited by the Zokyo team is to serve as components for the wallet the SIVIRA team is building. During the manual and testing stages of the contracts audit, multiple security issues were found. All those can be found in the following sections. Beside these findings, there are also remarks that have to be made about the overall security of the contracts submitted for audit.

During the initial assessment of the protocol, it has been discovered that the module manager component can perform a lot of owner related actions. These include sending funds from the Identity contract to arbitrary addresses. After further inspection, the Zokyo team established that these actions constitute a centralization risk for the security of the project's components. It's important to note that while some of these do not have a direct security impact on the contracts and project as a whole, there could be a scenario where this design can cause issues; such as complete loss of funds, undefined behavior due to either misconfiguration or malicious intent. Zokyo advises the team to acknowledge these design decisions and take extra care while operating the contracts in their current design.

#	Title	Risk
1	Pragma version lock	Low
2	Unchecked possible zero address	Medium
3	Unchecked possible zero address	Medium
4	Unchecked possible zero address	High
5	Unchecked possible zero address	Medium
6	Use of assembly for contract creation	Informational
7	Unprotected initialize function	Low
8	Improper transfer of ownership	Medium
9	Unchecked address before assignment	Informational
10	Unchecked address before assignment	Informational
11	Signature length verification	Low
12	Signature length verification	Low
13	Signature index verification	Low
14	Centralization risk	High

LOW | RESOLVED

Pragma version lock
It's recommended to have the same compiler version that the contracts were tested with the most. This way it reduces the risk of introducing unknown bugs.

Recommendation:
Lock pragma versions.

MEDIUM | RESOLVED

Unchecked possible zero address
In contract Identity.sol, at line 40 in function setOwner the newOwner function parameter can be zero and is not checked. The following function calls will not revert in case of a zero newOwner address, leading to undefined behavior, such as setting the owner to zero address.

Recommendation:
Add a sanity check for the newOwner address to not be zero and revert otherwise.

MEDIUM | RESOLVED

Unchecked possible zero address
In contract Identity.sol, at line 55 in function setModuleManager the newModuleManager function parameter can be zero and is not checked. In case the address is zero, the _moduleManager variable would be an interface initialized at a zero address. This would lead the call in the modifier onlyModule to fail, blocking functions, such as setOwner or execute, which use the modifier.

Recommendation:
Add a sanity check for the newModuleManager address to not be zero and revert otherwise.

HIGH | RESOLVED

Unchecked possible zero address
In contract Identity.sol, at line 90 in function execute the to function parameter can be zero and is not checked. In case the to address is zero, the following to.call{value: value}(data) would return true and send the value to the zero address.

Recommendation:
Add a sanity check for the to address variable to not be zero and revert otherwise.

MEDIUM | RESOLVED

Unchecked possible zero address
In contract Proxy.sol, at line 7 inside the constructor the impl parameter can be zero and is not checked. In case the address is zero, the proxy would not function properly and there is no fallback method to set the correct implementation after deployment.

Recommendation:
Add a sanity check for the to address variable to not be zero and revert otherwise.

INFORMATIONAL | ACKNOWLEDGE

Use of assembly for contract creation
In contract Factory.sol, the method create that deploys a new contract uses the opcode create2 in order to create a new contract from the bytecode. In newer Solidity versions, >=8, there is an alternative to this by using the new keyword to deploy a contract. This approach would eliminate the need for inline assembly.

Recommendation:
If the intent is to deploy a single type of contract, of which you have the source code, refactor and use the new keyword.

Comment from client:
we don't fix this, because Factory is intended to deploy arbitrary contracts.

LOW | ACKNOWLEDGE

Unprotected initialize function
In contract Identity.sol, at line 31, the initialize function is not protected and therefore anybody can call and initialize the contract with an arbitrary initialOwner address. As the function is blocked after the first call, due to the _isInitialized flag, this essentially locks the contract owner.

Recommendation:
Add a modifier or check to protect the initialize function.

Comment from client:
We don't fix this, because Proxy of Identity needs to be atomically deployed and initialized by an external contract account like IdentityProxyFactory.

MEDIUM | RESOLVED

Improper transfer of ownership
In contract Identity.sol, an arbitrary module can change the owner of the contract. This action should be an only owner action.

Recommendation:
Add a sanity check in function _setOwner to ensure that only the owner of the contract can modify the ownership.

INFORMATIONAL | RESOLVED

Unchecked address before assignment
In contract Identity, in function _setModuleManager, the address of variable _moduleManager is set. This address must be a contract address that implements the IModuleManager interface, but no check is made on it.

Recommendation:
Add a sanity check in function _setModuleManager to verify that newModuleManager is a contract address, to prevent un undefined behavior

INFORMATIONAL | RESOLVED

Unchecked address before assignment
In contract ModuleRegistry, in function registerModule the address of a module is stored. This address must be a contract address.

Recommendation:
Add a sanity check in function “registerModule” to verify that module is a contract address, to prevent un undefined behavior

LOW | RESOLVED

Signature length verification
In contract ECDSA, in function recover, from line 26, the length of sig is not validated. A signature should be 65 bytes and any mismatch can result in unclear errors due to use of assembly.

Recommendation:
Add a sanity check for sig, to have exactly 65 bytes and revert otherwise.

LOW | RESOLVED

Signature length verification
In contract ECDSA, in function recover, from line 44, the length of sig is not validated. A signature should be 65 bytes and any mismatch can result in unclear errors due to use of assembly. Taking into account the fact that the sig parameter contains several signatures, its length should be a multiple of 65.

Recommendation:
Add a sanity check for length of sig to be a multiple of 65 bytes and revert otherwise.

LOW | RESOLVED

Signature index verification
The contract ECDSA, in function recover, from line 44, the index of signature is not validated. For a given index, there must be a signature in the sigs variable. In other words, the value of the index cannot be greater than the length of a signature (65) multiplied by the number of signatures.

Recommendation:
Add a sanity check to ensure that exists a signature corresponding to the index and revert otherwise.

HIGH | RESOLVED

Centralization risk
In contract Identity, in function execute, an arbitrary module can send the funds from Identity contract to any other address. This behavior creates a major risk of centralization.

Recommendation:
Modify the behavior of the contract, so that only the owner of the funds can have access to them.

	Identity.sol	Proxy.sol	
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions/Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	IdentityProxyFactory.sol	LockManager.sol	
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions/Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	ModuleRegistry.sol	Ownable.sol	
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions/Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	CoreModuleAggregate.sol	CoreRelayerModule.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions/Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

	Address.sol	ECDSA.sol	
Re-entrancy	Pass	Pass	Pass
Access Management Hierarchy	Pass	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass	Pass
Unexpected Ether	Pass	Pass	Pass
Delegatecall	Pass	Pass	Pass
Default Public Visibility	Pass	Pass	Pass
Hidden Malicious Code	Pass	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass	Pass
External Contract Referencing	Pass	Pass	Pass
Short Address/Parameter Attack	Pass	Pass	Pass
Unchecked CALL Return Values	Pass	Pass	Pass
Race Conditions/Front Running	Pass	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass	Pass
Floating Points and Precision	Pass	Pass	Pass
Tx.Origin Authentication	Pass	Pass	Pass
Signatures Replay	Pass	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass	Pass

	DelegateModule.sol	SafeCast.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions/Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Security team

As part of our work assisting SIVIRA in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Hardhat testing framework.

Tests were based on the functionality of the code, as well as a review of the SIVIRA contract requirements for details about issuance amounts and how the system handles these.

```
DelegateModule
  supportsInterface
    ✓ supportsInterface (243ms)
  onERC721Received
    ✓ onERC721Received (64ms)
  onERC1155Received
    ✓ supportsInterface (58ms)
  onERC1155BatchReceived
    ✓ supportsInterface (59ms)
  isValidSignature
    ✓ reverts with invalid signature length (88ms)
    ✓ reverts with invalid signer (55ms)
    ✓ Returns true for valid signer (43ms)
ECDSA
  Operation without exceptions:
    ✓ #recover - recover the address given sig
    ✓ #recover - recover the correct index of signature
    ✓ #recover - recover address given r,s,v
    ✓ #toEthSignedMessageHash -
  Exceptions
    ✓ #recover - invalid s
    ✓ #recover - invalid v
    ✓ #recover - invalid signature; signer = address(0)
ECDSA
  Math Operation:
    ✓ #min -
    ✓ #min - MAXUINT and zero
    ✓ #max -
    ✓ #max - MAXUINT and zero
    ✓ #ceilDiv- 0/1
```

- ✓ #ceilDiv- 1/2 = 1, underflow ceil → 1
- ✓ #ceilDiv - revert div by 0
- ✓ #ceilDiv- 10/5 = 2
- ✓ #ceilDiv- 10/6 = 2 (1.66)

SafeCast Operations:

- ✓ #toUint128 - toUint128(0)
- ✓ #toUint128 - toUint128(1)
- ✓ #toUint128 - toUint128(2**128-1)
- ✓ #toUint128 - toUint128(2**128); should revert
- ✓ #toUint64 - toUint64(1)
- ✓ #toUint64 - toUint64(2**64-1)
- ✓ #toUint64 - toUint128(2**64); should revert

Address

- ✓ #isContract - return true; checking a contract
- ✓ #isContract - return false; checking an EOA
- ✓ #isContract - PoC; taking advantage of vulnerability (110ms)

CoreBaseModule

- ✓ Should be able to deploy correctly (88ms)
- ✓ Should be able to verify ping (77ms)
- ✓ Should be able to verify itself (57ms)
- ✓ Should be able to check if identity is locked (54ms)
- ✓ Should be able to ping (69ms)

CoreModuleAggregate

- ✓ Should be able to deploy correctly
- ✓ Should be able to execute through Identity (125ms)
- ✓ Should be able to execute (244ms)
- ✓ Should be able to get nonce (113ms)
- ✓ Should be able to issue refunds (75ms)

Factory

- ✓ Should be able to create (114ms)

IdentityProxyFactory

- ✓ Should be able to deploy correctly (108ms)
- ✓ Should be able to create proxy (119ms)
- ✓ Should be able to get proxy address (135ms)

LockManager

- ✓ Should be able to deploy correctly
- ✓ Should be able to lock identity (169ms)
- ✓ Should be able to unlock identity (151ms)
- ✓ Should be able to check if identity is locked (101ms)
- ✓ Should be able to get when identity lock will expire (56ms)

ModuleManager
- ✓ Should be able to deploy correctly (172ms)
- ✓ Should be able to initialize (126ms)
- ✓ Should be able to enable module (192ms)
- ✓ Should be able to disable module (280ms)
- ✓ Should be able to verify if module is enabled (201ms)
- ✓ Should be able to fix module (217ms)
- ✓ Should be able to check fix status of module (189ms)
- ✓ Should be able to enable delegation (229ms)
- ✓ Should be able to disable delegation (231ms)
- ✓ Should be able to get delegate (199ms)

ModuleRegistry
- ✓ Should be able to register module (55ms)
- ✓ Should be able to deregister module (50ms)
- ✓ Should be able to check registration status of modules (57ms)

Ownable
- ✓ Should be able to deploy correctly
- ✓ Should be able to to transfer ownership
- ✓ Should be able to to renounce ownership

Identity
- ✓ should create a proxy (116ms)
- ✓ should create a identity contract with initial modules (57ms)
- ✓ failure → should revert
- ✓ failure → should not allow zero address manager
- ✓ failure → should not allow double initialization (58ms)
- ✓ should change owner (139ms)
- ✓ should change moduleManager address (137ms)
- ✓ should fail to call a module (92ms)
- ✓ should call a module successfully (120ms)
- ✓ success → should call execute (192ms)
- ✓ failure → should not complete execute call (157ms)

Proxy
- ✓ success → Delegate call from proxy to identity implementation (276ms)
- ✓ success → Call recieve in proxy contract
- ✓ failure → implementation should be a contract (44ms)

82 passing (27s)

CODE COVERAGE

The resulting code coverage (i.e., the ratio of tests-to-code) is as follows:

FILE	% STMTS	% BRANCH	% FUNCS	% LINES	% UNCOVERED LINES
contracts/					
Identity.sol	100	100	100	100	
Proxy.sol	100	100	100	100	
contracts/infrastructure	100	97.62	100	100	
Factory.sol	100	75	100	100	
IdentityProxyFactory.sol	100	100	100	100	
LockManager.sol	100	100	100	100	
ModuleManager.sol	100	100	100	100	
ModuleRegistry.sol	100	100	100	100	
contracts/infrastruc ture/base/	100	100	100	100	
Ownable.sol	100	100	100	100	
contracts/module/core	100	100	100	100	
CoreBaseModule.sol	100	100	100	100	
CoreModuleAggregate.sol	100	100	100	100	
CoreRelayerModule.sol	100	100	100	100	
contracts/module/non core/					
DelegateModule.sol	100	100	100	100	
contracts/utils/	100	81.25	100	100	
Address.sol	100	100	100	100	
ECDSA.sol	100	75	100	100	
Math.sol	100	100	100	100	
SafeCast.sol	100	100	100	100	
All files	**100**	**95.83**	**100**	**100**	

We are grateful to have been given the opportunity to work with the SIVIRA team.

The statements made in this document should not be interpreted as an investment or legal advice, nor should its authors be held accountable for the decisions made based on them.

Zokyo's Security Team recommends that the SIVIRA team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

zokyo