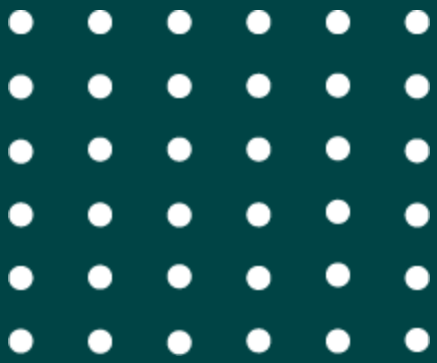
2heavens

Decentralized Crypto Wealth Wallet



IMMUNEBYTES

14 February 2023

Introduction

About 2heavens

2Heavens aims to provide a centralized view of crypto assets and financial education for families. You can carry out decisions about your crypto assets through our functionality TTOD (Token transfer on death), you can create joint accounts to educate children and companies can create smart contracts to bonus employees or even a vesting contract, and also offer a private pension plan.

With a lot of additional features:

- TTOD (Token Transfer on death)
- JOINT WALLET with children to educate them on investments
- Crypto Risk Prediction
- Asset Evolution Dashboard
- Life Insurance
- Private pension plan

Visit https://2heavens.com/ to learn more about the services.

About ImmuneBytes

ImmuneBytes is a security start-up that provides professional services in the blockchain space. The team has hands-on experience conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and understand DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, and dydx.

The team has secured 205+ blockchain projects by providing security services on different frameworks. The ImmuneBytes team helps start-ups with detailed system analysis, ensuring security and managing the overall project.

Visit http://immunebytes.com/ to learn more about the services.

Documentation Details

The 2heavens team has provided the following doc for audit:

1. https://github.com/guizostudios/2heavens#readme
2. https://immunebytes.notion.site/Standard-Contract-2heavens

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include

    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

Project Name	2heavens
Contract Name	HeirAccount.sol, Joint.sol, JointAccount.sol, TTOD.sol
Platform	CELO
Languages	Solidity
GitHub Link	https://github.com/guizostudios/2heavens
Commit - Initial Audit	548faed421a3b72a943c5b2e65362f1202932142
Commit - Final Audit	662de5fc8abfced42c77f4b134b347684a87a75b
Platforms & Tools	Remix IDE, Truffle, VScode, Contract Library, Slither, SmartCheck, Fuzz

Security Level References

Every issue in this report was assigned a severity level from the following:

CRITICAL

Issues may result in fund leakage or incorrect fund allocation.

HIGH

Issues affecting the business logic, performance, and functionality.

MEDIUM

Issues could lead to data loss or other manipulations.

LOW

Issues around smart contract code upgradability, libraries, and others.

INFORMATIONAL

Issues which can further improve the code on gas optimizations and reusability.

Issues	Critical	High	Medium	Low	Informational
Open	-	-	-	1	-
Closed	2	1	4	8	-
Acknowledged	-	-	-	1	-

Finding

#	Findings	Risk	Status
1.	Rugpull indicator	Critical	Fixed
2.	Unrestricted Heir Claiming	Critical	Fixed
3.	Insecure Contract Configuration	High	Fixed
4.	Create modifier to ensure proper access control	Medium	Fixed
5.	No event emission on critical functions	Medium	Fixed
6.	Missing data validation	Medium	Fixed
7.	Missing functionality	Medium	Fixed
8.	Use Non-Reentrant library	Low	Fixed
9.	Use pause/unpause openzeppelin library	Low	Fixed
10.	Merge withdraw and transferFunds function	Low	Fixed
11.	Code practice/ Gas Optimisation	Low	Fixed
12.	Don't make withdrawal functions payable.	Low	Fixed
13.	Relevant checks should be placed first.	Low	Fixed
14.	Naming convention	Low	Pending
15.	Improper Contract functions configuration	Low	Fixed
16.	Contract Order of layout	Low	Fixed
17.	Use Call instead of Transfer to transfer ETH	Low	Acknowledged

Critical Severity Issues

1. **Rugpull indicator:**

The code in question has a vulnerability known as a rug pull. A rugpull is a type of malicious attack that occurs in decentralized finance (DeFi) applications. In this type of attack, an adversary sets themselves as the beneficiary of a contract, and then calls the "withdraw" function with their own recipient address. This allows the attacker to drain all of the funds from the contract to their own address, effectively "pulling the rug" out from under the unsuspecting users of the DeFi application.

By setting themselves as the beneficiary of the JointAccount contract and then calling the "withdraw" function, an attacker is able to transfer all of the funds from the contract to their own address.

Code:
The vulnerability arises on line 60 (require statement for owner) in the file JointAccounts.sol:

```
// Function to set the beneficiary and the time to withdraw all the money
function setBeneficiary(
    address payable _beneficiary,
    uint256 _delay,
    address _sender
) public {
    require(owner == _sender, "Only the owner can set the beneficiary.");
    require(!mutex, "The function is already in execution.");
    mutex = true;
    beneficiary = _beneficiary;
    delay = _delay;
    mutex = false;
    InitiatedAt = block.timestamp;
}
```

Here, the function can be bypassed by any user on the blockchain if they fetch the address of the owner from the blockchain then provide the same address as _sender in the above function to set the _beneficiary as the attacker's recipient address to receive the funds from the owner of the JointAccount

Recommendation:
The recommendation is to change the require statement from:
require(owner == _sender, "Only the owner can set the beneficiary.");
To
require(owner == msg.sender, "Only the owner can set the beneficiary.");

Amended (February 14th, 2023): The team has fixed the issue, and it is no longer present in commit 662de5fc8abfced42c77f4b134b347684a87a75b.

2. **Unrestricted Heir Claiming**

Description: The code allows any user designated as the "heir" to initiate the claim process, and then to claim the ownership of an account without restrictions, even if the account is frozen. This means that a malicious heir can potentially gain unauthorized access to an account's funds, even if the original owner has taken steps to freeze the account to prevent such unauthorized access.

Steps to Reproduce:
- A user creates a TTOD account and designates another user as their "heir".
- The original owner decides to freeze the account for security reasons.
- The designated heir initiates the claim process and then claims the ownership of the account.
- The newly-claimed owner can unfreeze the account and withdraw the funds.

Recommendation: To prevent this vulnerability, the claim function should have an additional check to ensure that the account is not frozen. This will ensure that the funds can only be accessed by the rightful owner of the account, even if the heir designation has changed.

Amended (February 14th, 2023): The team has fixed the issue, and it is no longer present in commit 662de5fc8abfced42c77f4b134b347684a87a75b.

High Severity Issues

1. **Insecure Contract Configuration**

Description: The contract function createTTOD sets values for the Accounts contract's _feeAddress and _transferFee variables through user input, without any restrictions on the values that can be provided. This opens the contract up to the possibility of malicious actors setting _feeAddress to their own address, and setting _transferFee to an excessively high value in order to capture a large portion of any transferred funds.

Recommendation: To mitigate this vulnerability, the contract should enforce an upper and lower cap on the value of _transferFee, and hardcode the value of _feeAddress rather than allowing it to be

set through user input. This will ensure that the fee address is known and controlled by the contract owner, and that the transfer fee is set to a reasonable and secure value.The function should implement a function to change the _transferFee and _feeAddress values.

Amended (February 14th, 2023): The team has fixed the issue, and it is no longer present in commit 662de5fc8abfced42c77f4b134b347684a87a75b.

Medium severity issues

1. **Create modifier to ensure proper access control**

   Description: It is recommended to create a modifier to encapsulate the check for the contract owner if the same check is repeated in multiple functions. Modifiers allow you to encapsulate frequently used checks and apply them to multiple functions, making the code more readable, maintainable, and less prone to errors.

   Eg.
   require(owner == _sender, "Only the owner can set the heir.");
   require(heir == _sender, "Only the heir can claim the account.");

   Recommendation: Create a Modifier to check owner/heir and place them into relevant functions.
   Amended (February 14th, 2023): The team has fixed the issue, and it is no longer present in commit 662de5fc8abfced42c77f4b134b347684a87a75b.

2. **No event emission on critical functions**

   Description: Contract does not emit respective events over crucial functionality eg. createJointWallet function. Emitting events can make it easier to monitor the state and activity of a contract, and can also be useful for debugging. While the critical functions of a contract do not emit events, this will make it more difficult to understand what is happening in the contract and to track important changes to its state. Although events can be use to in such ways:

   - Logging: Events are used to log important events or state changes in the contract, which can later be used for auditing and debugging.
   - Notifications: Events are used to send notifications to clients or other contracts about certain events that have occurred in the contract.
   - Data indexing: Events are used to index data on the blockchain, making it possible to search and filter events based on specific criteria.
   - Front-end integration: Events are used to communicate with the front-end of decentralized applications, providing a way to update the UI in real-time.

Recommendation: It is recommended to emit respective events while events play a crucial role in making contracts more transparent and accessible to users, and they help to increase the overall functionality and flexibility of decentralized applications.

Amended (February 14th, 2023): The team has fixed the issue, and it is no longer present in commit 662de5fc8abfced42c77f4b134b347684a87a75b.

3. **Missing data validation**

Description: Contract does not check for input data passed into function parameter eg. createJointWallet, jointSetBeneficiary and setWithdrawLimit, these functions do not check for zero address and zero amount which can be easily set by malicious users.

Recommendation: It is recommended to place relevant checks to ensure invalid data should not be acceptable in the smart contract.

Amended (February 14th, 2023): The team has fixed the issue, and it is no longer present in commit 662de5fc8abfced42c77f4b134b347684a87a75b.

4. **Missing functionality**

Description: During the deployment of the contract, _transferFee set to something which is not specified anywhere in the contract to we are assuming the value could be anything as there is no check even in constructor to verify the amount reserved for protocol fee, further there is no functionality in contract to change the fee parameter in future if required.

Code

```
constructor(
    address payable _owner,
    address payable _feeAddress,
    uint256 _transferFee //@audit assuming fee can not be change after deployment
) {
    owner = _owner;
    transferFee = _transferFee;
    feeAddress = _feeAddress;
}
```

Recommendation: It is recommended to fix the fee value in the contract by placing a state variable and use that variable in constructor but to set arbitrary fee value while deploying each contract, also

place a function to change fee if in case need to change this parameter in future and place a lower and upper bound respective.

Amended (February 14th, 2023): The team has fixed the issue, and it is no longer present in commit 662de5fc8abfced42c77f4b134b347684a87a75b.

Low severity issues

1. **Use Non-Reentrant library**

   Description: The mutex variable is used to prevent reentrancy by manually setting it to true before executing critical sections of code and setting it back to false after the critical section is finished. This approach can be effective, but it is error-prone and not as secure as using a non-reentrant library.

   Recommendation: A better approach would be to use a non-reentrant library such as OpenZeppelin's SafeERC20 or ReentrancyGuard, which provides a secure and easy-to-use implementation of reentrancy protection. These libraries use the onlyPayable modifier, which prevents external contracts from calling your contract again before the first call has finished.

   Amended (February 14th, 2023): The team has fixed the issue, and it is no longer present in commit 662de5fc8abfced42c77f4b134b347684a87a75b.

2. **Use pause/unpause openzeppelin library**

   Description: As a general rule, using well-vetted and battle-tested libraries such as OpenZeppelin is considered a best practice in smart contract development, as it can greatly reduce the risk of security vulnerabilities. OpenZeppelin has been thoroughly tested and audited by a large community of developers and security experts, which makes it more secure and reliable compared to manually implementing similar functionality. You can make mistakes while implementing the freeze/unfreeze functions manually, there is a risk that the contract could be exploited, leading to the loss of funds or other unintended consequences. This is especially true if the freeze/unfreeze functions are a critical part of the contract's security.

   Recommendation: It is highly recommended to use a well-established library such as OpenZeppelin to implement the freeze/unfreeze functionality, rather than manually implementing it yourself.

   Amended (February 14th, 2023): The team has fixed the issue, and it is no longer present in commit 662de5fc8abfced42c77f4b134b347684a87a75b.

3. Merge withdraw and transferFunds function

Description: While Withdraw and TransferFunds functions doing the same job, it is recommended to merge the two functions into one function with an additional parameter to determine if the transfer is a withdraw or a transfer to a recipient. The code could look something like this:

Code Example:

```
function transferOrWithdraw(
    bool _isWithdraw,
    bool _isCelo,
    address payable _recipient,
    uint256 _amount,
    address _ERC20Address,
    address _sender
) public payable whenNotPaused nonReentrant onlyOwner{
    address payable recipient = _isWithdraw ? owner : _recipient;
    if (_isCelo) {
        require(_amount <= address(this).balance, "Insufficient funds.");
        fee = _amount.mul(transferFee).div(100);
        recipient.transfer(_amount.sub(fee));
        feeAddress.transfer(fee);
    } else {
        ERC20 = IERC20(_ERC20Address);
        require(
            _amount <= ERC20.balanceOf(address(this)),
            "Insufficient funds."
        );
        fee = _amount.mul(transferFee).div(100);
        ERC20.safeTransfer(recipient, _amount.sub(fee));
        ERC20.safeTransfer(feeAddress, fee);
    }
}
```

Recommendation: It is recommended to merge withdraw and transferFunds functions in HeirAccount and JoitAccount contracts.

Amended (February 14th, 2023): The team has fixed the issue, and it is no longer present in commit 662de5fc8abfced42c77f4b134b347684a87a75b.

4. **Code practice/ Gas Optimisation**

   Description: Using "!= 0" is a valid way to check if a uint value is non-zero nevertheless it has the same efficiency as using "> 0". In solidity specifically while using uint, it might make the code more readable, as it explicitly checks for non-zero values.

   Code Example:

   require(address(this).balance > 0, "There are no funds to withdraw.");

   This code example is just one case however the practice has been used all over the contract.

   Recommendation: It is recommended to use " != 0" instead of "> 0".

   Amended (February 14th, 2023): The team has fixed the issue, and it is no longer present in commit 662de5fc8abfced42c77f4b134b347684a87a75b.

5. **Don't make withdrawal functions payable.**

   Description: While the withdraw function does not imply a functionality to receive eth and the function is not meant to handle any Ether transfers, it is not necessary to make it payable.

   Recommendation: Remove the payable keyword as if the withdraw function is not meant to receive Ether, then marking it as payable is unnecessary and could potentially lead to confusion. If the withdraw function does not need to receive Ether, it should not be marked as payable.

   Amended (February 14th, 2023): The team has fixed the issue, and it is no longer present in commit 662de5fc8abfced42c77f4b134b347684a87a75b.

6. **Relevant checks should be placed first.**

   Description: In jointAccount and heirAccount contract there is a function name Withdraw which checks for :

   require(!frozen, "Account is frozen, cannot withdraw funds.");

   This check should be placed first in the function to make sure other functionality will be processed after that, there is no point to move forward before checking for that.

Recommendation: Move this check first while withdrawing funds from the contract, moving towards other checks first will cost more gas.

Amended (February 14th, 2023): The team has fixed the issue, and it is no longer present in commit 662de5fc8abfced42c77f4b134b347684a87a75b.

7. **Naming convention**

Description: Naming conventions are important in smart contract development because they help make the code more readable. This can help prevent mistakes and misunderstandings. Additionally, well-defined naming conventions can make the code easier to maintain and test, and can help ensure that the code meets best practices and standards for smart contract development. By providing clear, concise, and consistent names for functions, variables, and other elements of the contract, naming conventions can make it easier to identify the purpose and functionality of each component, improving the overall quality and maintainability of the code.

Few suggestions:

ttodReturnOwner => getTtoOwner

ttodReturnHeir => getTtoHeir

ttodReturnDeployer => getTtoDeployer

ttodReturnTransferFee => getTtoTransferFee

findDeployer => deployers

findOwner => owners

findHeir => heirs

returnDelay => getDelay

returnInitiateAt => getInitiateAt

8. Improper Contract functions configuration

In the protocol, the current coding practice is costly and inefficient intruding redundancy in the code base. By following these steps, the functionality of the factory contracts in the factory design pattern will be improved, resulting in a more efficient and user-friendly solution.

a. Removing Wrapper Functions: All wrapper functions from both factory contracts will be removed and only the createInstance functions will be used.
b. Renaming Function: The createTTOD function will be renamed to createTTODWallet for better clarity.
c. Using createJointWallet Only: The createJointWallet function will be used for creating joint wallets.
d. Adding New Functions: New functions will be added to the factory accounts to increase their usability. These functions will include:
    i. getJointWalletById: To retrieve the joint wallet by its ID
    ii. getAllJointWalletsByFeeAddress: To retrieve all joint wallets associated with a specific fee address.
    iii. etc: Any other necessary functions that improve the overall functionality of the factory design pattern.

Amended (February 14th, 2023): The team has fixed the issue, and it is no longer present in commit 662de5fc8abfced42c77f4b134b347684a87a75b.

9. Contract order of layout

Description: The layout of a Solidity smart contract should be according to recommended practices for organizing and structuring code that can make it easier to read, maintain, and debug. These are some best practices that can help ensure a clean and well-structured smart contract:

- Variable Declarations: Variables should be declared at the top of the contract and grouped by data type and visibility.
- Function Declarations: Functions should be declared in the order they are used, and grouped by related functionality.
- Modifiers: Modifiers should be declared at the top of the contract, before any functions, and grouped by related functionality.
- Constructor: The constructor function should be placed immediately after the variable and modifier declarations.

- Event Declarations: Event declarations should be placed after the constructor and grouped by related functionality.
- External Functions: External functions should be declared after internal functions and grouped by related functionality.
- Commenting: Comments should be used to explain the purpose of the code and to provide context where necessary.
- Naming Conventions: Consistent and descriptive naming conventions should be used for variables, functions, and events.
- These recommendations can help ensure a clean and well-structured smart contract, making it easier to read, maintain, and debug.
- Receive/Fallback Function: These functions should be declared right after the constructor.

Amended (February 14th, 2023): The team has fixed the issue, and it is no longer present in commit 662de5fc8abfced42c77f4b134b347684a87a75b.

10. Use Call instead of Transfer to transfer ETH

Description: Since transfer() has typically been recommended by the security community because it helps guard against reentrancy attacks. This guidance made sense under the assumption that gas costs wouldn't change, but that assumption turned out to be incorrect. EIP 1884 just headed our way in the Istanbul hard fork. This change increases the gas cost of the SLOAD operation and therefore breaks some existing smart contracts. Contracts will break because their fallback functions used to consume less than 2300 gas, and they'll now consume more. Which makes 2300 gas not significant while It's the amount of gas a contract's fallback function receives if it's called via Solidity's transfer() or send() methods.

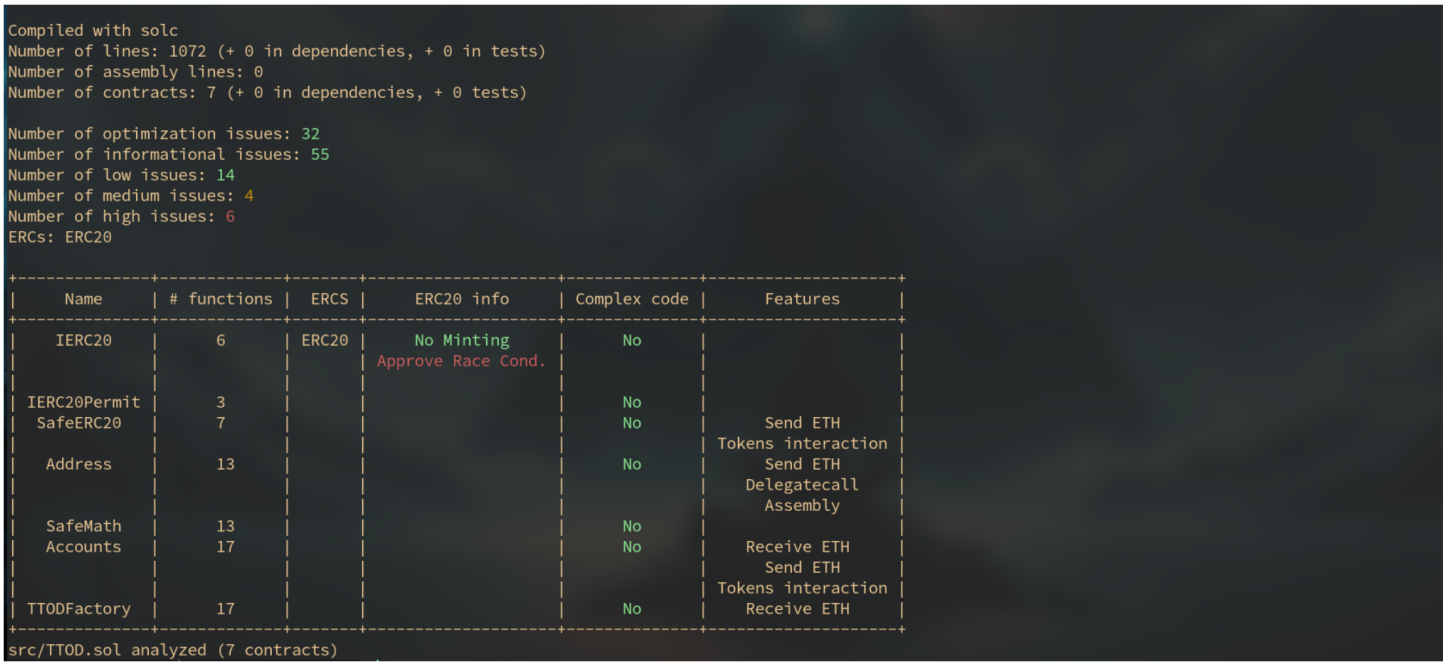
Recommendation: Using transfer() made sense under the assumption that gas costs are constant. Gas costs are not constant while ethereum is changing gas costs in past EIPs. Smart contracts should be robust to this fact. Solidity's transfer() and send() use a hardcoded gas amount. These methods should be avoided. Use .call.value(...)("") instead. This carries a risk regarding reentrancy. Be sure to use one of the robust methods available for preventing reentrancy vulnerabilities.

Acknowledged (February 14th, 2023): The team has acknowledged the issue.

Informational

No issues were found.

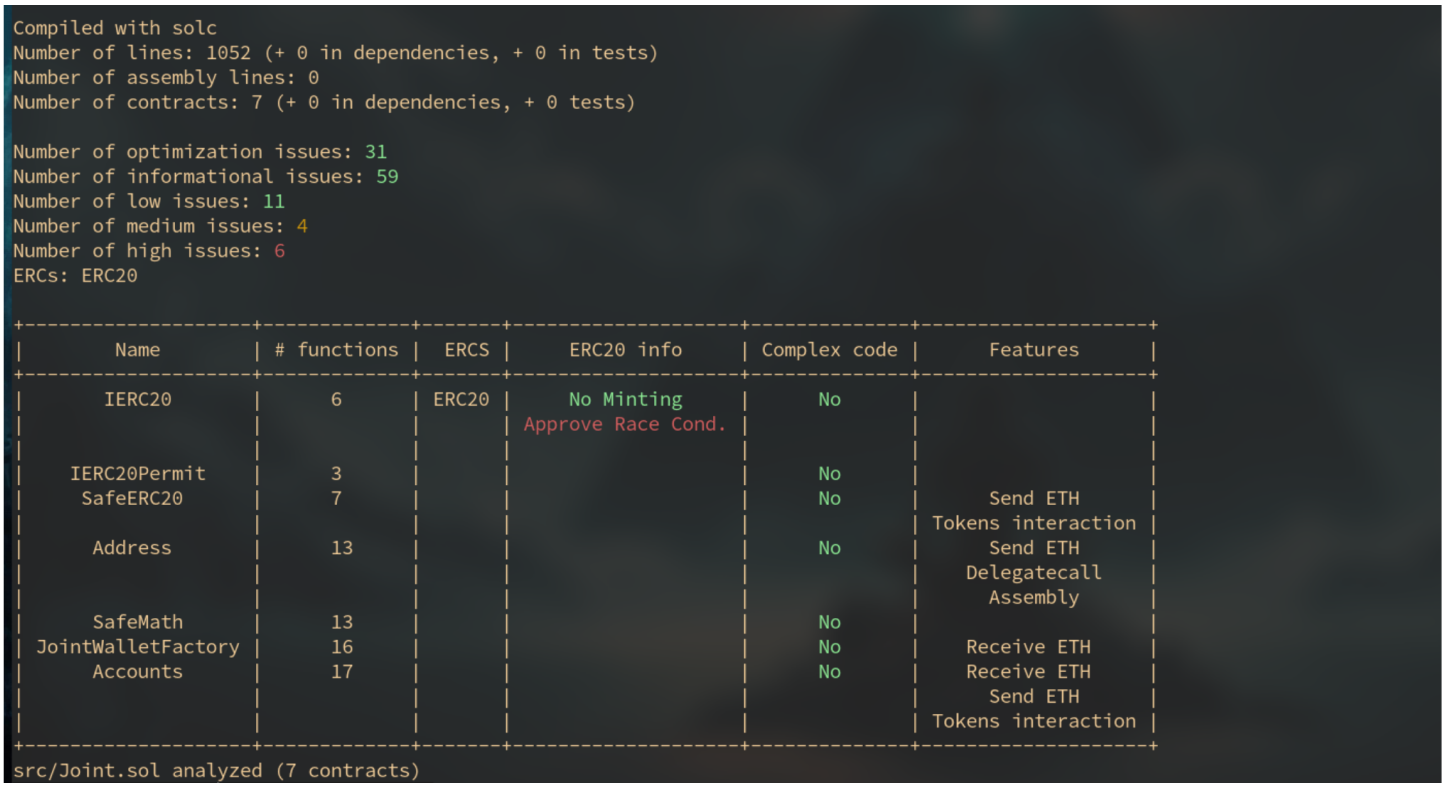
Automated Audit Result



```
Compiled with solc
Number of lines: 1072 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 7 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 32
Number of informational issues: 55
Number of low issues: 14
Number of medium issues: 4
Number of high issues: 6
ERCs: ERC20

+--------------+------------+-------+--------------------+--------------+--------------------+
|     Name     | # functions | ERCS |     ERC20 info     | Complex code |      Features      |
+--------------+------------+-------+--------------------+--------------+--------------------+
|    IERC20    |      6     | ERC20 |     No Minting     |      No      |                    |
|              |            |       | Approve Race Cond. |              |                    |
|              |            |       |                    |              |                    |
| IERC20Permit |      3     |       |                    |      No      |                    |
|  SafeERC20   |      7     |       |                    |      No      |      Send ETH      |
|              |            |       |                    |              | Tokens interaction |
|   Address    |      13    |       |                    |      No      |      Send ETH      |
|              |            |       |                    |              |    Delegatecall    |
|              |            |       |                    |              |      Assembly      |
|   SafeMath   |      13    |       |                    |      No      |                    |
|   Accounts   |      17    |       |                    |      No      |    Receive ETH     |
|              |            |       |                    |              |      Send ETH      |
|              |            |       |                    |              | Tokens interaction |
| TTODFactory  |      17    |       |                    |      No      |    Receive ETH     |
+--------------+------------+-------+--------------------+--------------+--------------------+
src/TTOD.sol analyzed (7 contracts)
```



```
Compiled with solc
Number of lines: 1052 (+ 0 in dependencies, + 0 in tests)
Number of assembly lines: 0
Number of contracts: 7 (+ 0 in dependencies, + 0 tests)

Number of optimization issues: 31
Number of informational issues: 59
Number of low issues: 11
Number of medium issues: 4
Number of high issues: 6
ERCs: ERC20

+--------------------+-------------+-------+--------------------+--------------+--------------------+
|        Name        | # functions | ERCS  |     ERC20 info     | Complex code |      Features      |
+--------------------+-------------+-------+--------------------+--------------+--------------------+
|       IERC20       |      6      | ERC20 |     No Minting     |      No      |                    |
|                    |             |       | Approve Race Cond. |              |                    |
|                    |             |       |                    |              |                    |
|    IERC20Permit    |      3      |       |                    |      No      |                    |
|     SafeERC20      |      7      |       |                    |      No      |      Send ETH      |
|                    |             |       |                    |              | Tokens interaction |
|      Address       |      13     |       |                    |      No      |      Send ETH      |
|                    |             |       |                    |              |    Delegatecall    |
|                    |             |       |                    |              |      Assembly      |
|      SafeMath      |      13     |       |                    |      No      |                    |
| JointWalletFactory |      16     |       |                    |      No      |    Receive ETH     |
|      Accounts      |      17     |       |                    |      No      |    Receive ETH     |
|                    |             |       |                    |              |      Send ETH      |
|                    |             |       |                    |              | Tokens interaction |
+--------------------+-------------+-------+--------------------+--------------+--------------------+
src/Joint.sol analyzed (7 contracts)
```

Concluding Remarks

While conducting the audits of the 2heavens smart contracts, it was observed that the contracts contain Critical, High, Medium, and Low severity issues along with a few recommendations.

Our auditors suggest that Critical, High, Medium and Low severity issues should be resolved by the developers. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process; therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the 2heavens platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

IMMUNEBYTES

AUDITS