MetaVill

Smart Contract Security Audit

Prepared by ShellBoxes

August 5th, 2022 - September 23rd, 2022

Shellboxes.com

contact@shellboxes.com

Document Properties

Client	Metavill
Version	1.0
Classification	Public

Scope

The MetaVill Contract in the MetaVill Repository

Repo	Commit Hash
https://gitlab.com/m6931/blockchain/audit/	7046e7febb97321dd66d0515442d75b02842600e

Files	MD5 Hash
Airdrop.sol	32ea71311e8af72c31fca7bdd7a2a117
Creative.sol	bf41d017dc5c81e30ec64c4f77cdf316
CreativeMarket.sol	eece1d0de84459f909ae8ebf3d957767
Donate.sol	a289c38cbdb778abe92e7d7a0c8f9e10
IDOPublic.sol	299c8e3a26711db4c2092a7a64b15696
IDOWhitelist.sol	4fb4dfdb42c533c5d92cb602e2f1b4a2
MVToken.sol	39237766fd1c9cb2577a1ccb7911342b
Private.sol	5d60ac63f1b2a4b1e9675506b413cf01
Seed.sol	b27ed5e0c50a222de9d17ce43737c84d
VestingSchedule.sol	eb1fcd940bc6e976d3ab65cf4dd798d5

Re-Audit Scope

Repo	Commit Hash
https://gitlab.com/m6931/blockchain/audit/	96765afed7a206751973a61487e5a6bb0dea4b05

Files	MD5 Hash
Airdrop.sol	ac2fcb0848976013032f12d514ebe567
Creative.sol	72ae12f2abf4b97717cb04e2dedadddb
CreativeMarket.sol	880deb4c2160ef8ea7d87d759d58e13d
Donate.sol	bb7a68aa9c95f353ffa1c50f8cc27667
IDOPublic.sol	4044bd34e62c3fa120b6f065e7213e59
IDOWhitelist.sol	41b14bf746f19dfcafab9499b7f75e53
MVToken.sol	39237766fd1c9cb2577a1ccb7911342b
Private.sol	7f1c3aaaace3dd285efaf51c45339661
Seed.sol	a35475d8d248350751229c3665c3218d
VestingSchedule.sol	2a9ddbdc8178ff326b6e9ab0864591e9
Whitelist.sol	b3e399a90b0c956c843732f1fadec6e0

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

1 Introduction

Metavill engaged ShellBoxes to conduct a security assessment on the MetaVill beginning on August 5th, 2022 and ending September 23rd, 2022. In this report, we detail our methodical approach to evaluate potential security issues associated with the implementation of smart contracts, by exposing possible semantic discrepancies between the smart contract code and design document, and by recommending additional ideas to optimize the existing code. Our findings indicate that the current version of smart contracts can still be enhanced further due to the presence of many security and performance concerns.

This document summarizes the findings of our audit.

1.1 About Metavill

METAVILL is a Social Entertainment Defi Platform, which allows users to connect with each other through broadcasting, creative NFTs, and earn from many activities: Livestream to earn, Watch to earn, engage to earn... and free to play.

Issuer	Metavill
Website	`https://metavill.io`
Type	Solidity Smart Contract
Audit Method	Whitebox

1.2 Approach & Methodology

ShellBoxes used a combination of manual and automated security testing to achieve a balance between efficiency, timeliness, practicability, and correctness within the audit's scope. While manual testing is advised for identifying problems in logic, procedure, and implementation, automated testing techniques help to expand the coverage of smart contracts and can quickly detect code that does not comply with security best practices.

1.2.1 Risk Methodology

Vulnerabilities or bugs identified by ShellBoxes are ranked using a risk assessment technique that considers both the LIKELIHOOD and IMPACT of a security incident. This framework is effective at conveying the features and consequences of technological vulnerabilities.

Its quantitative paradigm enables repeatable and precise measurement, while also revealing the underlying susceptibility characteristics that were used to calculate the Risk scores. A risk level will be assigned to each vulnerability on a scale of 5 to 1, with 5 indicating the greatest possibility or impact.

- Likelihood quantifies the probability of a certain vulnerability being discovered and exploited in the untamed.
- Impact quantifies the technical and economic costs of a successful attack.
- Severity indicates the risk's overall criticality.

Probability and impact are classified into three categories: H, M, and L, which correspond to high, medium, and low, respectively. Severity is determined by probability and impact and is categorized into four levels, namely Critical, High, Medium, and Low.

Impact				
	High	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Low
		High	Medium	Low

Likelihood

2 Findings Overview

2.1 Summary

The following is a synopsis of our conclusions from our analysis of the MetaVill implementation. During the first part of our audit, we examine the smart contract source code and run the codebase via a static code analyzer. The objective here is to find known coding problems statically and then manually check (reject or confirm) issues highlighted by the tool. Additionally, we check business logics, system processes, and DeFi-related components manually to identify potential hazards and/or defects.

2.2 Key Findings

In general, these smart contracts are well-designed and constructed, but their implementation might be improved by addressing the discovered flaws, which include 3 critical-severity, 7 high-severity, 11 medium-severity, 25 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
D.1. The Owner Can Control All Transfer Parameters	CRITICAL	Fixed
E.1. The Contract Is Not Verified To Have mv Tokens	CRITICAL	Acknowledged
F.1. The Contract Is Not Verified To Have mv Tokens	CRITICAL	Acknowledged
A.1. The creative Contract Interface Cannot Be Set	HIGH	Fixed
A.2. The Operator Is The Center Of Each Buy	HIGH	Fixed
B.1. Buyer's Funds Can Be Lost	HIGH	Fixed
C.1. Buyer's Funds Can Be Lost	HIGH	Fixed
E.2. The Owner Can Whitelist Any Amount To Any User	HIGH	Fixed
F.2. The Owner Can Whitelist Any Amount To Any User	HIGH	Fixed
G.1. The Owner Can Airdrop Any Amount To Any User	HIGH	Acknowledged
A.3. Missing Transfer Verification	MEDIUM	Fixed
B.2. The end Variable Is Never Used	MEDIUM	Fixed
B.3. Missing Transfer Verification	MEDIUM	Fixed
C.2. The end Variable Is Never Used	MEDIUM	Fixed

C.3. Missing Transfer Verification	MEDIUM	Fixed
D.2. Missing Transfer Verification	MEDIUM	Fixed
E.3. vestingPeriods Elements permil Should Sum To 1000	MEDIUM	Fixed
E.4. Missing Transfer Verification	MEDIUM	Fixed
F.3. vestingPeriods Elements permil Should Sum To 1000	MEDIUM	Fixed
G.2. Missing Transfer Verification	MEDIUM	Fixed
I.1. vestingPeriods Elements permil Should Sum To 1000	MEDIUM	Fixed
A.4. Missing Address Verification	LOW	Fixed
A.5. Owner Can Renounce Ownership	LOW	Fixed
A.6. The Contract Can End Up Without Operators	LOW	Fixed
B.4. Missing Value Verification	LOW	Fixed
B.5. Missing Address Verification	LOW	Fixed
B.6. The userCount Variable Does Not Represent The Number Of The Users	LOW	Fixed
C.4. Missing Value Verification	LOW	Fixed
C.5. Missing Address Verification	LOW	Fixed
D.3. Missing Address Verification	LOW	Fixed
D.4. Owner Can Renounce Ownership	LOW	Fixed
E.5. Missing Address Verification	LOW	Fixed
E.6. Owner Can Renounce Ownership	LOW	Fixed
E.7. Usage of block.timestamp	LOW	Acknowledged
E.8. For Loop Over Dynamic Array	LOW	Fixed
F.4. Missing Address Verification	LOW	Fixed
F.5. Owner Can Renounce Ownership	LOW	Fixed
F.6. Usage of block.timestamp	LOW	Acknowledged
F.7. For Loop Over Dynamic Array	LOW	Fixed
G.3. Missing Address Verification	LOW	Fixed
G.4. Owner Can Renounce Ownership	LOW	Fixed

H.1. Owner Can Renounce Ownership	LOW	Fixed
H.2. The Contact Can End Up Without Operators	LOW	Not Fixed
I.2. Owner Can Renounce Ownership	LOW	Fixed
I.3. Usage of block.timestamp	LOW	Acknowledged
I.4. For Loop Over Dynamic Array	LOW	Fixed

3 Finding Details

A CreativeMarket.sol

A.1 The creative Contract Interface Cannot Be Set [HIGH]

Description:

The creative is a variable that contains a contract interface, which is used to execute transfer operations in the buy function. This variable is not initialized in the constructor, and it does not have a setter. Therefore, it will never have a value different from the address(0), this results in a denial of service in the buy function.

Code:

Listing 1: CreativeMarket.sol

```
9  ERC721 private creative;
```

Listing 2: CreativeMarket.sol

```
33 function buy(uint256 creativeId, address buyerAddress, address
       ↪ ownerAddress, uint price, uint fee) public onlyOperator {
34     mv.transferFrom(buyerAddress, market, price + fee);
35     mv.transferFrom(market, ownerAddress, price);
36     creative.transferFrom(ownerAddress, buyerAddress, creativeId);

38     emit Bought(creativeId, ownerAddress, buyerAddress);
39 }
```

Risk Level:

Likelihood – 4
Impact - 5

Recommendation:

Consider initializing the creative variable in the constructor.

Status - Fixed

The Metavill team has fixed the issue by initializing the creative variable in the constructor.

A.2 The Operator Is The Center Of Each Buy [HIGH]

Description:

The operator is the one responsible for executing the buy function, the buyer and the owner of the NFT do not have any interaction with the CreativeMarket contract, the only action they perform is approving the required assets for the buy function to pass. This represents a significant centralization risk where the operator is the center of all the buy operations and is able to manipulate all the parameters.

Code:

Listing 3: CreativeMarket.sol

```
33 function buy(uint256 creativeId, address buyerAddress, address
       ↪ ownerAddress, uint price, uint fee) public onlyOperator {
34     mv.transferFrom(buyerAddress, market, price + fee);
35     mv.transferFrom(market, ownerAddress, price);
36     creative.transferFrom(ownerAddress, buyerAddress, creativeId);

38     emit Bought(creativeId, ownerAddress, buyerAddress);
39 }
```

Risk Level:

Likelihood – 4
Impact - 5

Recommendation:

Consider implementing a logic where the NFT holders would be able to create sell offers and the buyers would execute the buy function and fill the selected sell order.

Status - Fixed

The Metavill team has fixed the issue by only allowing the operator to execute the buy function using a signature provided by the buyerAddress.

A.3 Missing Transfer Verification [MEDIUM]

Description:

The ERC20 standard token implementation functions return the transaction status as a boolean. It is a good practice to check for the return status of the function call to ensure that the transaction was executed successfully. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails.

Code:

Listing 4: CreativeMarket.sol

```
33 function buy(uint256 creativeId, address buyerAddress, address
       ↪ ownerAddress, uint price, uint fee) public onlyOperator {
34     mv.transferFrom(buyerAddress, market, price + fee);
35     mv.transferFrom(market, ownerAddress, price);
36     creative.transferFrom(ownerAddress, buyerAddress, creativeId);

38     emit Bought(creativeId, ownerAddress, buyerAddress);
39 }
```

Risk Level:

Likelihood – 2
Impact - 4

Recommendation:

Use the safeTransfer function from the safeERC20 Implementation, or put the transfer call inside an assent or require verifying that it returned true.

Status - Fixed

The Metavill team has fixed the issue by using the safeTransferFrom function from the safeERC20 implementation.

A.4 Missing Address Verification [LOW]

Description:

Certain functions lack a safety check in the address, the address-type arguments should include a zero-address test, otherwise, the contract's functionality may become inaccessible. In the constructor, the contract must ensure that the mk and the mvContractAddress are different from address(0).

Code:

Listing 5: CreativeMarket.sol

```
43 constructor(address mk, address mvContractAddress) {
44     market = mk;
45     mv = ERC20(mvContractAddress);
46 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you make sure the addresses provided in the arguments are different from the address(0).

Status - Fixed

The Metavill team has fixed the issue by adding require statements to ensure the addresses provided in the arguments are different from the address(0).

A.5 Owner Can Renounce Ownership [LOW]

Description:

Typically, the account that deploys the contract is also its owner. Consequently, the owner is able to engage in certain privileged activities in his own name. In smart contracts, the renounceOwnership function is used to renounce ownership, which means that if the contract's ownership has never been transferred, it will never have an Owner, rendering some owner-exclusive functionality unavailable.

Code:

Listing 6: CreativeMarket.sol

```
7 contract CreativeMarket is Ownable {
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you prevent the owner from calling renounceOwnership without first transferring ownership to a different address.

Additionally, if you decide to use a multi-signature wallet, then the execution of the renounceOwnership will require for at least two or more users to be confirmed. Alternatively, you can disable Renounce Ownership functionality by overriding it.

Status - Fixed

The Metavill team has fixed the issue by overriding the renounce ownership functionality in order to disable it.

A.6 The Contract Can End Up Without Operators [LOW]

Description:

The contract has a role named operator, which is the group of addresses allowed to execute the buy function. The _operators mapping is used to store the operators addresses. The owner of the contract is able to call the setOperator and remove all the operators, which will result in a denial of service.

Code:

Listing 7: CreativeMarket.sol

```
42 function setOperator(address operatorAddress, bool value) public
       ↪ onlyOwner {
43     _operators[operatorAddress] = value;
44     emit OperatorSetted(operatorAddress, value);
45 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Consider adding a safety check in the setOperator function that will prevent the owner from removing all the operators from the contract.

Status - Fixed

The Metavill team has fixed the issue by preventing the owner from being removed from the operators.

B IDOPublic.sol

B.1 Buyer's Funds Can Be Lost [HIGH]

Description:

The buy function is used by the users to buy mv tokens using BUSD, the users can claim the bought amount over the course of vesting periods. If a user calls the buy with an amount that is lower than pricePerMv, the user will lose his funds without getting any mv tokens due to a type conversion error.

Code:

Listing 8: IDOPublic.sol

```
59 function buy(uint amount) external {
60     require(amount >= minAllocation, 'Min exceed');
61     require(amount + totals[msg.sender] <= maxAllocation, 'Max exceed');
62     require(block.timestamp >= start, 'Before IDO');

64     busd.transferFrom(msg.sender, busdWalletAddress, amount);
65     totals[msg.sender] += amount / pricePerMv * 10 ** 18;
66     userCount += 1;
67 }
```

Risk Level:

Likelihood – 4
Impact - 5

Recommendation:

Consider multiplying the amount by 10**18 before dividing it over the pricePerMv to avoid rounding errors.

Status - Fixed

The Metavill team has fixed the issue by performing the multiplication operation before the division.

B.2 The end Variable Is Never Used [MEDIUM]

Description:

The contract contains a variable called end, the buy function contains a check over the start variable, but it does not verify if the end has already passed.

Code:

Listing 9: IDOPublic.sol

```
16 uint private end;
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Consider implementing a check in the buy function that will make sure that block.timestamp is between start and end.

Status - Fixed

The Metavill team has fixed the issue by implementing the use of the end variable and verifying it when calling the buy function.

B.3 Missing Transfer Verification [MEDIUM]

Description:

The ERC20 standard token implementation functions return the transaction status as a boolean. It is a good practice to check for the return status of the function call to ensure that the transaction was executed successfully. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails.

Code:

Listing 10: IDOPublic.sol

```
59 function buy(uint amount) external {
60     require(amount >= minAllocation, 'Min exceed');
61     require(amount + totals[msg.sender] <= maxAllocation, 'Max exceed');
62     require(block.timestamp >= start, 'Before IDO');

64     busd.transferFrom(msg.sender, busdWalletAddress, amount);
65     totals[msg.sender] += amount / pricePerMv * 10 ** 18;
66     userCount += 1;
67 }
```

Listing 11: IDOPublic.sol

```
71 function claim() external {
72     require(totals[msg.sender] > 0, 'Not available');
73     uint amount = sumPermil() * totals[msg.sender] / 1000 - claimed[msg.
           ↪ sender];
74     require(amount > 0, 'Not available to claimed');
75     claimed[msg.sender] += amount;
```

```
76      mv.transferFrom(mvWalletAddress, msg.sender, amount);
77  }
```

Risk Level:

Likelihood – 2
Impact - 4

Recommendation:

Use the safeTransfer function from the safeERC20 Implementation, or put the transfer call inside an assert or require verifying that it returned true.

Status - Fixed

The Metavill team has fixed the issue by using the safeTransferFrom function from the safeERC20 implementation.

B.4 Missing Value Verification [LOW]

Description:

Certain functions lack a value safety check, the values of the arguments should be verified to allow only the ones that comply with the contract's logic. In the constructor function, the contract must ensure that pricePerMv_ is different from 0, and the start_ variable is higher than now and lower than end_, in addition to that, the minAllocation_ should be verified to be higher than maxAllocation_.

Code:

Listing 12: IDOPublic.sol

```
23  constructor(
24      address busdAddress_,
25      address mvAddress_,
26      uint minAllocation_,
```

```
27      uint maxAllocation_,
28      uint start_,
29      uint end_,
30      uint pricePerMv_
31  ) {
32      busd = IERC20(busdAddress_);
33      mv = IERC20(mvAddress_);
34      busdWalletAddress = msg.sender;
35      mvWalletAddress = msg.sender;
36      minAllocation = minAllocation_;
37      maxAllocation = maxAllocation_;
38      start = start_;
39      end = end_;
40      pricePerMv = pricePerMv_;
41      userCount = 0;
42  }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you verify the values provided in the arguments. The issue can be addressed by utilizing a require statement.

Status - Fixed

The Metavill team has fixed the issue by verifying the values provided from the arguments.

B.5 Missing Address Verification [LOW]

Description:

Certain functions lack a safety check in the address, the address-type arguments should include a zero-address test, otherwise, the contract's functionality may become inaccessible. In the constructor, the contract must ensure that the busdAddress_ and the mvAddress_ are different from address(0).

Code:

Listing 13: IDOPublic.sol

```
23 constructor(
24     address busdAddress_,
25     address mvAddress_,
26     uint minAllocation_,
27     uint maxAllocation_,
28     uint start_,
29     uint end_,
30     uint pricePerMv_
31 ) {
32     busd = IERC20(busdAddress_);
33     mv = IERC20(mvAddress_);
34     busdWalletAddress = msg.sender;
35     mvWalletAddress = msg.sender;
36     minAllocation = minAllocation_;
37     maxAllocation = maxAllocation_;
38     start = start_;
39     end = end_;
40     pricePerMv = pricePerMv_;
41     userCount = 0;
42 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you make sure the addresses provided in the arguments are different from the address(0).

Status - Fixed

The Metavill team has fixed the issue by adding require statements to ensure the addresses provided in the arguments are different from the address(0).

B.6 The userCount Variable Does Not Represent The Number Of The Users [LOW]

Description:

The userCount variable is initialized to zero, and it is incremented every time a user buys an mv package. However, this variable does not represent its name as the buy function can be called 2 times from the same user, therefore its value will not represent the number of users.

Code:

Listing 14: IDOPublic.sol

```
59 function buy(uint amount) external {
60     require(amount >= minAllocation, 'Min exceed');
61     require(amount + totals[msg.sender] <= maxAllocation, 'Max exceed');
62     require(block.timestamp >= start, 'Before IDO');

64     busd.transferFrom(msg.sender, busdWalletAddress, amount);
65     totals[msg.sender] += amount / pricePerMv * 10 ** 18;
```

```
66      userCount += 1;
67  }
```

Risk Level:

Likelihood – 1
Impact - 2

Recommendation:

Consider adding a mapping that will allow the contract to identify the users that have already called the buy function, then only increment the userCount variable when the msg.sender is calling the buy function for the first time.

Status - Fixed

The Metavill team has fixed the issue by only incrementing userCount variable when the msg.sender is calling the buy function for the first time.

C IDOWhitelist.sol

C.1 Buyer's Funds Can Be Lost [HIGH]

Description:

The buy function is used by the users to buy mv tokens using BUSD, the users can claim the bought amount over the course of vesting periods. If a user calls the buy with an amount that is lower than pricePerMv, the user will lose his funds without getting any mv tokens due to a type conversion error.

Code:

Listing 15: IDOWhitelist.sol

```
62  function buy(uint amount) external {
```

```
63      require(amount >= minAllocation, 'Min exceed');
64      require(amount + totals[msg.sender] <= maxAllows[msg.sender], 'Max
            ↪ exceed');
65      require(block.timestamp >= start, 'Before IDO');

67      busd.transferFrom(msg.sender, busdWalletAddress, amount);
68      totals[msg.sender] += amount / pricePerMv * 10 ** 18;
69  }
```

Risk Level:

Likelihood – 4
Impact - 5

Recommendation:

Consider multiplying the amount by 10**18 before dividing it over the pricePerMv to avoid rounding errors.

Status - Fixed

The Metavill team has fixed the issue by performing the multiplication operation before the division.

C.2 The end Variable Is Never Used [MEDIUM]

Description:

The contract contains a variable called end, the buy function contains a check over the start variable, but it does not verify if the end has already passed.

Code:

Listing 16: IDOWhitelist.sol

```
16  uint private end;
```

Risk Level:

Likelihood – 3
Impact - 3

Recommendation:

Consider implementing a check in the buy function that will make sure that block.timestamp is between start and end.

Status - Fixed

The Metavill team has fixed the issue by implementing the use of the end variable and verifying it when calling the buy function.

C.3 Missing Transfer Verification [MEDIUM]

Description:

The ERC20 standard token implementation functions return the transaction status as a boolean. It is a good practice to check for the return status of the function call to ensure that the transaction was executed successfully. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails.

Code:

Listing 17: IDOWhitelist.sol

```
62 function buy(uint amount) external {
63     require(amount >= minAllocation, 'Min exceed');
64     require(amount + totals[msg.sender] <= maxAllows[msg.sender], 'Max
       ↪ exceed');
65     require(block.timestamp >= start, 'Before IDO');

67     busd.transferFrom(msg.sender, busdWalletAddress, amount);
```

```
68      totals[msg.sender] += amount / pricePerMv * 10 ** 18;
69  }
```

Listing 18: IDOWhitelist.sol

```
71  function claim() external {
72      require(totals[msg.sender] > 0, 'Not available');
73      uint amount = sumPermil() * totals[msg.sender] / 1000 - claimed[msg.
            ↪ sender];
74      require(amount > 0, 'Not available to claimed');
75      claimed[msg.sender] += amount;
76      mv.transferFrom(mvWalletAddress, msg.sender, amount);
77  }
```

Risk Level:

Likelihood – 2
Impact - 4

Recommendation:

Use the safeTransfer function from the safeERC20 Implementation, or put the transfer call inside an assert or require verifying that it returned true.

Status – Fixed

The Metavill team has fixed the issue by using the safeTransferFrom function from the safeERC20 implementation.

C.4 Missing Value Verification [LOW]

Description:

Certain functions lack a value safety check, the values of the arguments should be verified to allow only the ones that comply with the contract's logic. In the constructor function, the contract must ensure that pricePerMv_ is different from 0, and the start_ variable is higher than now and lower than end_, in addition to that, the minAllocation_ should be verified to be higher than maxAllocation_.

Code:

Listing 19: IDOWhitelist.sol

```
24 constructor(
25     address busdAddress_,
26     address mvAddress_,
27     uint minAllocation_,
28     uint maxAllocation_,
29     uint start_,
30     uint end_,
31     uint pricePerMv_
32 ) {
33     busd = IERC20(busdAddress_);
34     mv = IERC20(mvAddress_);
35     busdWalletAddress = msg.sender;
36     mvWalletAddress = msg.sender;
37     minAllocation = minAllocation_;
38     maxAllocation = maxAllocation_;
39     start = start_;
40     end = end_;
41     pricePerMv = pricePerMv_;
42     whitelistCount = 0;
43 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you verify the values provided in the arguments. The issue can be addressed by utilizing a require statement.

Status - Fixed

The Metavill team has fixed the issue by verifying the values provided from the arguments.

C.5 Missing Address Verification [LOW]

Description:

Certain functions lack a safety check in the address, the address-type arguments should include a zero-address test, otherwise, the contract's functionality may become inaccessible. In the constructor, the contract must ensure that the busdAddress_ and the mvAddress_ are different from address(0).

Code:

Listing 20: IDOWhitelist.sol

```
24 constructor(
25     address busdAddress_,
26     address mvAddress_,
27     uint minAllocation_,
28     uint maxAllocation_,
29     uint start_,
30     uint end_,
31     uint pricePerMv_
32 ) {
```

```
33      busd = IERC20(busdAddress_);
34      mv = IERC20(mvAddress_);
35      busdWalletAddress = msg.sender;
36      mvWalletAddress = msg.sender;
37      minAllocation = minAllocation_;
38      maxAllocation = maxAllocation_;
39      start = start_;
40      end = end_;
41      pricePerMv = pricePerMv_;
42      whitelistCount = 0;
43  }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you make sure the addresses provided in the arguments are different from the address(0).

Status - Fixed

The Metavill team has fixed the issue by adding require statements to ensure the addresses provided in the arguments are different from the address(0).

D Donate.sol

D.1 The Owner Can Control All Transfer Parameters [CRITICAL]

Description:

The owner is the one responsible for executing the donate function, and he can also manipulate all the transfers' parameters with no restrictions. For instance, he can simply execute the donate function with zero as the receiveAmount, causing the receiverAddress to receive the entire sendAmount. This represents a significant centralization risk where the owner controls all aspects of the contract.

Code:

Listing 21: Donate.sol

```
20 function donate(address from, address to, uint sendAmount, uint
       ↪ receiveAmount) external onlyOwner {
21     require(sendAmount > receiveAmount);
22     token.transferFrom(from, receiverAddress, sendAmount);
23     token.transferFrom(receiverAddress, to, receiveAmount);
24 }
```

Risk Level:

Likelihood – 5
Impact - 5

Recommendation:

Consider changing the logic of the contract to be more interactive with the users to avoid centralization risks.

Status - Fixed

The Metavill team has fixed the issue by only allowing the owner to execute the donate function using a signature provided by the from and that verifies the transfer parameters.

D.2 Missing Transfer Verification [MEDIUM]

Description:

The ERC20 standard token implementation functions return the transaction status as a boolean. It is a good practice to check for the return status of the function call to ensure that the transaction was executed successfully. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails.

Code:

Listing 22: Donate.sol

```
20 function donate(address from, address to, uint sendAmount, uint
       ↪ receiveAmount) external onlyOwner {
21     require(sendAmount > receiveAmount);
22     token.transferFrom(from, receiverAddress, sendAmount);
23     token.transferFrom(receiverAddress, to, receiveAmount);
24 }
```

Risk Level:

Likelihood – 2
Impact - 4

Recommendation:

Use the safeTransfer function from the safeERC20 Implementation, or put the transfer call inside an assert or require verifying that it returned true.

Status - Fixed

The Metavill team has fixed the issue by using the safeTransferFrom function from the safeERC20 implementation.

D.3 Missing Address Verification [LOW]

Description:

Certain functions lack a safety check in the address, the address-type arguments should include a zero-address test, otherwise, the contract's functionality may become inaccessible. In the constructor, the contract must ensure that the mvTokenAddress_ is different from address(0).

Code:

Listing 23: Donate.sol

```
11 constructor(address mvTokenAddress_) {
12     receiverAddress = msg.sender;
13     token = IERC20(mvTokenAddress_);
14 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you make sure the addresses provided in the arguments are different from the address(0).

Status - Fixed

The Metavill team has fixed the issue by adding require statements to ensure the addresses provided in the arguments are different from the address(0).

D.4 Owner Can Renounce Ownership [LOW]

Description:

Typically, the account that deploys the contract is also its owner. Consequently, the owner is able to engage in certain privileged activities in his own name. In smart contracts, the renounceOwnership function is used to renounce ownership, which means that if the contract's ownership has never been transferred, it will never have an Owner, rendering some owner-exclusive functionality unavailable.

Code:

Listing 24: Donate.sol

```
6 contract Donate is Ownable {
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you prevent the owner from calling renounceOwnership without first transferring ownership to a different address. Additionally, if you decide to use a multi-signature wallet, then the execution of the renounceOwnership will require for at least two or more users to be confirmed. Alternatively, you can disable Renounce Ownership functionality by overriding it.

Status - Fixed

The Metavill team has fixed the issue by overriding the renounce ownership functionality in order to disable it.

E Private.sol

E.1 The Contract Is Not Verified To Have mv Tokens [CRITICAL]

Description:

The owner is able to allow any amount to the users, this amount can be claimed by the user using the claim function. However, when allowing an amount to a user, the owner does not fund the contract with the allowed amount. As a result, the users may find themselves in a situation where they are unable to collect their money from the contract due to a lack of balance.

Code:

Listing 25: Private.sol

```
23 function addWhitelist(address user, uint amount) external onlyOwner {
24     totals[user] += amount;
25 }
```

Listing 26: Private.sol

```
39 function claim() external {
40     require(totals[msg.sender] > 0, 'Not in whitelist');
41     uint amount = claimable(msg.sender);
42     require(amount > 0, 'Amount is zero');
43     claimed[msg.sender] += amount;
44     mv.transfer(msg.sender, amount);
45 }
```

Risk Level:

Likelihood – 5

Impact - 5

Recommendation:

Consider making it mandatory for the owner to fund the contract with the amount of mv tokens permitted by the addWhitelist function.

Status - Acknowledged

The Metavill team has acknowledged the risk, stating the owner will fund the contract with the sufficient funds.

E.2 The Owner Can Whitelist Any Amount To Any User [HIGH]

Description:

The addWhitelist function allows the owner to whitelist a user allowing him any amount of mv tokens, this implementation cannot assure a good distribution of tokens over the users as the owner can just whitelist one user with all the available amount. This represents a significant centralization risk.

Code:

Listing 27: Private.sol

```
23 function addWhitelist(address user, uint amount) external onlyOwner {
24     totals[user] += amount;
25 }
```

Risk Level:

Likelihood – 4
Impact - 5

Recommendation:

Consider constructing a Merkle tree that contains all the whitelisted users, and storing the Merkle root in the contract as a constant, then verify that the caller is whitelisted before

allowing him any amount. In addition to that, the users should be getting the same amount to assure a fair distribution of tokens.

Status - Fixed

The Metavill team has fixed the issue by removing the addWhitelist function.

E.3 vestingPeriods Elements permil Should Sum To 1000 [MEDIUM]

Description:

The vestingPeriods contains the vesting periods that will decide the amount that the user will be able to claim in each period. However, the sum of the pemil attribute of all elements should be equal to 1000 to assure that the user will be able to get all of his funds by the end of the vesting period.

Code:

Listing 28: Private.sol

```
13 Period[] private vestingPeriods;
```

Risk Level:

Likelihood – 3
Impact - 5

Recommendation:

Consider requiring the sum of the pemil attribute of all elements to be equal to 1000.

Status - Fixed

The Metavill team has fixed the issue by requiring the sum of pemil to be equal to 1000.

E.4 Missing Transfer Verification [MEDIUM]

Description:

The ERC20 standard token implementation functions return the transaction status as a boolean. It is a good practice to check for the return status of the function call to ensure that the transaction was executed successfully. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails.

Code:

Listing 29: Private.sol

```
39 function claim() external {
40     require(totals[msg.sender] > 0, 'Not in whitelist');
41     uint amount = claimable(msg.sender);
42     require(amount > 0, 'Amount is zero');
43     claimed[msg.sender] += amount;
44     mv.transfer(msg.sender, amount);
45 }
```

Risk Level:

Likelihood – 2
Impact - 4

Recommendation:

Use the safeTransfer function from the safeERC20 Implementation, or put the transfer call inside an assert or require verifying that it returned true.

Status - Fixed

The Metavill team has fixed the issue by using the safeTransferFrom function from the safeERC20 implementation.

E.5 Missing Address Verification [LOW]

Description:

Certain functions lack a safety check in the address, the address-type arguments should include a zero-address test, otherwise, the contract's functionality may become inaccessible. In the constructor, the contract must ensure that the tokenAddress_ is different from address(0).

Code:

Listing 30: Private.sol

```
19 constructor(address tokenAddress_) {
20     mv = IERC20(tokenAddress_);
21 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you make sure the addresses provided in the arguments are different from the address(0).

Status - Fixed

The Metavill team has fixed the issue by adding require statements to ensure the addresses provided in the arguments are different from the address(0).

E.6 Owner Can Renounce Ownership [LOW]

Description:

Typically, the account that deploys the contract is also its owner. Consequently, the owner is able to engage in certain privileged activities in his own name. In smart contracts, the renounceOwnership function is used to renounce ownership, which means that if the contract's ownership has never been transferred, it will never have an Owner, rendering some owner-exclusive functionality unavailable.

Code:

Listing 31: Private.sol

```
6 contract Private is Ownable {
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you prevent the owner from calling renounceOwnership without first transferring ownership to a different address. Additionally, if you decide to use a multi-signature wallet, then the execution of the renounceOwnership will require for at least two or more users to be confirmed. Alternatively, you can disable Renounce Ownership functionality by overriding it.

Status - Fixed

The Metavill team has fixed the issue by overriding the renounce ownership functionality in order to disable it.

E.7 Usage of block.timestamp [LOW]

Description:

Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all what is guaranteed is that it is higher than the timestamp of the previous block.

Code:

Listing 32: Private.sol

```
47 function _precheckPeriod(uint permil) private view returns (bool) {
48         uint total = 0; //= init
49         for(uint i = 0; i < vestingPeriods.length; ++i) {
50             if (block.timestamp > vestingPeriods[i].timestamp) { //=
                ↪ block.timestamp //= For loop
51                 total += vestingPeriods[i].permil;
52             }
53         }
54         require(total + permil <= 1000, 'Above 1');
55         return true;
56     }
```

Listing 33: Private.sol

```
58     function claimable(address user) public view returns (uint) {
59         uint permil = 0; //= init
60         for(uint i = 0; i < vestingPeriods.length; ++i) {
61             if (block.timestamp > vestingPeriods[i].timestamp) { //=
                ↪ block.timestamp //= For loop
62                 permil += vestingPeriods[i].permil;
63             }
64         }
65         require(permil <= 1000, 'Above 1');
```

```
66          return totals[user] * permil / 1000 - claimed[user];
67      }
68  }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Verify that a delay of 900 seconds will not harm the logic of the contract.

Status - Acknowledged

The Metavill team has acknowledged the risk, stating that a 900 seconds delay will not harm the logic of the contract.

E.8 For Loop Over Dynamic Array [LOW]

Description:

When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial of Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Code:

Listing 34: Private.sol

```
47  function _precheckPeriod(uint permil) private view returns (bool) {
48          uint total = 0;
49          for(uint i = 0; i < vestingPeriods.length; ++i) {
```

```
            if (block.timestamp > vestingPeriods[i].timestamp) {
                total += vestingPeriods[i].permil;
            }
        }
        require(total + permil <= 1000, 'Above 1');
        return true;
    }
```

Listing 35: Private.sol

```
function claimable(address user) public view returns (uint) {
    uint permil = 0;
    for(uint i = 0; i < vestingPeriods.length; ++i) {
        if (block.timestamp > vestingPeriods[i].timestamp) {
            permil += vestingPeriods[i].permil;
        }
    }
    require(permil <= 1000, 'Above 1');
    return totals[user] * permil / 1000 - claimed[user];
}
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocs and thus multiple transactions.

Status - Fixed

The Metavill team has fixed the issue by requiring the length of the vestingPeriods array to be lower than 50.

F Seed.sol

F.1 The Contract Is Not Verified To Have mv Tokens [CRITICAL]

Description:

The owner is able to allow any amount to the users, this amount can be claimed by the user using the claim function. However, when allowing an amount to a user, the owner does not fund the contract with the allowed amount. As a result, the users may find themselves in a situation where they are unable to collect their money from the contract due to a lack of balance.

Code:

Listing 36: Seed.sol

```
35 function addWhitelist(address user, uint amount) external onlyOwner {
36     totals[user] += amount;
37 }
```

Listing 37: Seed.sol

```
61 function claim() external {
62     require(totals[msg.sender] > 0, 'Not in whitelist');
63     uint amount = claimable(msg.sender);
64     require(amount > 0, 'Amount is zero');
65     claimed[msg.sender] += amount;
66     mv.transfer(msg.sender, amount);
67 }
```

Risk Level:

Likelihood – 5

Impact - 5

Recommendation:

Consider making it mandatory for the owner to fund the contract with the amount of mv tokens permitted by the addWhitelist function.

Status - Acknowledged

The Metavill team has acknowledged the risk, stating the owner will fund the contract with the sufficient funds.

F.2 The Owner Can Whitelist Any Amount To Any User [HIGH]

Description:

The addWhitelist function allows the owner to whitelist a user allowing him any amount of mv tokens, this implementation cannot assure a good distribution of tokens over the users as the owner can just whitelist one user with all the available amount. This represents a significant centralization risk.

Code:

Listing 38: Seed.sol

```
35 function addWhitelist(address user, uint amount) external onlyOwner {
36     totals[user] += amount;
37 }
```

Risk Level:

Likelihood – 4
Impact - 5

Recommendation:

Consider constructing a Merkle tree that contains all the whitelisted users, and storing the Merkle root in the contract as a constant, then verify that the user is whitelisted before al-

lowing him any amount. In addition to that, the users should be getting the same amount to assure a fair distribution of tokens.

Status - Fixed

The Metavill team has fixed the issue by removing the addWhitelist function.

F.3 vestingPeriods Elements permil Should Sum To 1000 [MEDIUM]

Description:

The vestingPeriods contains the vesting periods that will decide the amount that the user will be able to claim in each period. However the sum of the pemil attribute of all elements should be equal to 1000 to assure that the user will be able to get all of his funds by the end of the vesting period.

Code:

Listing 39: Seed.sol

```
13 Period[] private vestingPeriods;
```

Risk Level:

Likelihood – 3
Impact - 5

Recommendation:

Consider requiring the sum of the pemil attribute of all elements to be equal to 1000.

Status - Fixed

The Metavill team has fixed the issue by requiring the sum of pemil to be equal to 1000.

F.4 Missing Address Verification [LOW]

Description:

Certain functions lack a safety check in the address, the address-type arguments should include a zero-address test, otherwise, the contract's functionality may become inaccessible. In the constructor, the contract must ensure that the tokenAddress_ is different from address(0).

Code:

Listing 40: Seed.sol

```
19 constructor(address tokenAddress_) {
20     mv = IERC20(tokenAddress_);
21 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you make sure the addresses provided in the arguments are different from the address(0).

Status - Fixed

The Metavill team has fixed the issue by adding require statements to ensure the addresses provided in the arguments are different from the address(0).

F.5 Owner Can Renounce Ownership [LOW]

Description:

Typically, the account that deploys the contract is also its owner. Consequently, the owner is able to engage in certain privileged activities in his own name. In smart contracts, the renounceOwnership function is used to renounce ownership, which means that if the contract's ownership has never been transferred, it will never have an Owner, rendering some owner-exclusive functionality unavailable.

Code:

Listing 41: Seed.sol

```
6 contract Seed is Ownable {
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you prevent the owner from calling renounceOwnership without first transferring ownership to a different address. Additionally, if you decide to use a multi-signature wallet, then the execution of the renounceOwnership will require for at least two or more users to be confirmed. Alternatively, you can disable Renounce Ownership functionality by overriding it.

Status - Fixed

The Metavill team has fixed the issue by overriding the renounce ownership functionality in order to disable it.

F.6 Usage of block.timestamp [LOW]

Description:

Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all what is guaranteed is that it is higher than the timestamp of the previous block.

Code:

Listing 42: Seed.sol

```
39      function _precheckPeriod(uint permil) private view returns (bool) {
40          uint total = 0;
41          for(uint i = 0; i < vestingPeriods.length; ++i) {
42              if (block.timestamp > vestingPeriods[i].timestamp) {
43                  total += vestingPeriods[i].permil;
44              }
45          }
46          require(total + permil <= 1000, 'Above 1');
47          return true;
48      }
```

Listing 43: Seed.sol

```
50      function claimable(address user) public view returns (uint) {
51          uint permil = 0;
52          for(uint i = 0; i < vestingPeriods.length; ++i) {
53              if (block.timestamp > vestingPeriods[i].timestamp) {
54                  permil += vestingPeriods[i].permil;
55              }
56          }
57          require(permil <= 1000, 'Above 1');
58          return totals[user] * permil / 1000 - claimed[user];
59      }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Verify that a delay of 900 seconds will not harm the logic of the contract.

Status - Acknowledged

The Metavill team has acknowledged the risk, stating that a 900 seconds delay will not harm the logic of the contract.

F.7 For Loop Over Dynamic Array [LOW]

Description:

When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial of Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Code:

Listing 44: Seed.sol

```
39      function _precheckPeriod(uint permil) private view returns (bool) {
40          uint total = 0;
41          for(uint i = 0; i < vestingPeriods.length; ++i) {
42              if (block.timestamp > vestingPeriods[i].timestamp) {
43                  total += vestingPeriods[i].permil;
44              }
45          }
```

```
46          require(total + permil <= 1000, 'Above 1');
47          return true;
48      }
```

Listing 45: Seed.sol

```
50      function claimable(address user) public view returns (uint) {
51          uint permil = 0;
52          for(uint i = 0; i < vestingPeriods.length; ++i) {
53              if (block.timestamp > vestingPeriods[i].timestamp) {
54                  permil += vestingPeriods[i].permil;
55              }
56          }
57          require(permil <= 1000, 'Above 1');
58          return totals[user] * permil / 1000 - claimed[user];
59      }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocs and thus multiple transactions.

Status - Fixed

The Metavill team has fixed the issue by requiring the length of the vestingPeriods array to be lower than 50.

G Airdrop.sol

G.1 The Owner Can Airdrop Any Amount To Any User [HIGH]

Description:

The airdrop function allows the owner to send a user any amount of mv tokens from the mvWalletAddress, this implementation cannot insure a good distribution of tokens for the users as the owner can just airdrop to one user with all the available amount, and leave the rest. This represents a significant centralization risk.

Code:

Listing 46: Airdrop.sol

```
18 function airdrop(address user, uint amount, string memory message)
       ↪ external onlyOwner {
19     require(user != address(0));
20     mv.transferFrom(mvWalletAddress, user, amount);
21 }
```

Risk Level:

Likelihood – 4
Impact - 5

Recommendation:

Consider constructing a Merkle tree that contains all the users who are eligible for the airdrop, and storing the Merkle root in the contract as a constant, then ensure that the user is eligible for the airdrop before sending him any amount. In addition to that, the users should be getting the same amount to assure a fair distribution of tokens.

Status - Acknowledged

The Metavill team has acknowledged the risk, stating that the users will get a random amount of tokens to increase the unpredictability & curiosity properties. In addition to that, the rewards in a period of time also depend on activities-point-collected, anti-fraud-suspected-lv, and the market (BTC price, MV price, MV TVL,...)

G.2 Missing Transfer Verification [MEDIUM]

Description:

The ERC20 standard token implementation functions return the transaction status as a boolean. It is a good practice to check for the return status of the function call to ensure that the transaction was executed successfully. It is the developer's responsibility to enclose these function calls with require() to ensure that, when the intended ERC20 function call returns false, the caller transaction also fails.

Code:

Listing 47: Airdrop.sol

```
18 function airdrop(address user, uint amount, string memory message)
       ↪ external onlyOwner {
19     require(user != address(0));
20     mv.transferFrom(mvWalletAddress, user, amount);
21 }
```

Risk Level:

Likelihood – 2
Impact - 4

Recommendation:

Use the safeTransfer function from the safeERC20 Implementation, or put the transfer call inside an assert or require verifying that it returned true.

Status - Fixed

The Metavill team has fixed the issue by using the safeTransferFrom function from the safeERC20 implementation.

G.3 Missing Address Verification [LOW]

Description:

Certain functions lack a safety check in the address, the address-type arguments should include a zero-address test, otherwise, the contract's functionality may become inaccessible. In the constructor, the contract must ensure that the mvTokenAddress_ is different from address(0).

Code:

Listing 48: Airdrop.sol

```
11 constructor(address mvTokenAddress_) {
12     mvWalletAddress = msg.sender;
13     mv = IERC20(mvTokenAddress_);
14 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you make sure the addresses provided in the arguments are different from the address(0).

Status - Fixed

The Metavill team has fixed the issue by adding require statements to ensure the addresses provided in the arguments are different from the address(0).

G.4 Owner Can Renounce Ownership [LOW]

Description:

Typically, the account that deploys the contract is also its owner. Consequently, the owner is able to engage in certain privileged activities in his own name. In smart contracts, the renounceOwnership function is used to renounce ownership, which means that if the contract's ownership has never been transferred, it will never have an Owner, rendering some owner-exclusive functionality unavailable.

Code:

Listing 49: Airdrop.sol

```
6 contract Airdrop is Ownable {
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you prevent the owner from calling renounceOwnership without first transferring ownership to a different address. Additionally, if you decide to use a multi-signature wallet, then the execution of the renounceOwnership will require for at least two or more users to be confirmed. Alternatively, you can disable Renounce Ownership functionality by overriding it.

Status - Fixed

The Metavill team has fixed the issue by overriding the renounce ownership functionality in order to disable it.

H Creative.sol

H.1 Owner Can Renounce Ownership [LOW]

Description:

Typically, the account that deploys the contract is also its owner. Consequently, the owner is able to engage in certain privileged activities in his own name. In smart contracts, the renounceOwnership function is used to renounce ownership, which means that if the contract's ownership has never been transferred, it will never have an Owner, rendering some owner-exclusive functionality unavailable.

Code:

Listing 50: Creative.sol

```
6 contract Creative is ERC721, Ownable {
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you prevent the owner from calling renounceOwnership without first transferring ownership to a different address. Additionally, if you decide to use a multi-signature wallet, then the execution of the renounceOwnership will require for at least two or more users to be confirmed. Alternatively, you can disable Renounce Ownership functionality by overriding it.

Status - Fixed

The Metavill team has fixed the issue by overriding the renounce ownership functionality in order to disable it.

H.2 The Contact Can End Up Without Operators [LOW]

Description:

The contract has a role named operator, which is the group of addresses allowed to execute the buy function. The _operators mapping is used to store the operators addresses. The owner of the contract is able to call the setOperator and remove all the operators, which will result in a denial of service.

Code:

Listing 51: Creative.sol

```
23 function setOperator(address operatorAddress, bool value) public
       ↪ onlyOwner {
24     _operators[operatorAddress] = value;
25     emit OperatorSetted(operatorAddress, value);
26 }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Consider adding a safety check in the setOperator function that will prevent the owner from removing all the operators from the contract.

Status - Not Fixed

I VestingSchedule.sol

I.1 vestingPeriods Elements permil Should Sum To 1000 [MEDIUM]

Description:

The vestingPeriods contains the vesting periods that will decide the amount that the user will be able to claim in each period. However, the sum of the pemil attribute of all elements should be equal to 1000 to assure that the user will be able to get all of his funds by the end of the vesting period.

Code:

Listing 52: VestingSchedule.sol

```
11 Period[] private vestingPeriods;
```

Risk Level:

Likelihood – 3
Impact - 5

Recommendation:

Consider requiring the sum of the pemil attribute of all elements to be equal to 1000.

Status - Fixed

The Metavill team has fixed the issue by requiring the sum of pemil to be equal to 1000.

I.2 Owner Can Renounce Ownership [LOW]

Description:

Typically, the account that deploys the contract is also its owner. Consequently, the owner is able to engage in certain privileged activities in his own name. In smart contracts, the renounceOwnership function is used to renounce ownership, which means that if the contract's ownership has never been transferred, it will never have an Owner, rendering some owner-exclusive functionality unavailable.

Code:

Listing 53: VestingSchedule.sol

```
6 contract VestingSchedule is Ownable {
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

We recommend that you prevent the owner from calling renounceOwnership without first transferring ownership to a different address. Additionally, if you decide to use a multi-signature wallet, then the execution of the renounceOwnership will require for at least two or more users to be confirmed. Alternatively, you can disable Renounce Ownership functionality by overriding it.

Status - Fixed

The Metavill team has fixed the issue by overriding the renounce ownership functionality in order to disable it.

I.3 Usage of block.timestamp [LOW]

Description:

Block.timestamp is used in the contract. The variable block is a set of variables. The timestamp does not always reflect the current time and may be inaccurate. The value of a block can be influenced by miners. Maximal Extractable Value attacks require a timestamp of up to 900 seconds. There is no guarantee that the value is right, all what is guaranteed is that it is higher than the timestamp of the previous block.

Code:

Listing 54: VestingSchedule.sol

```
25      function _precheckPeriod(uint permil) private view returns (bool) {
26          uint total = 0;
27          for(uint i = 0; i < vestingPeriods.length; ++i) {
28              if (block.timestamp > vestingPeriods[i].timestamp) {
29                  total += vestingPeriods[i].permil;
30              }
31          }
32          require(total + permil <= 1000, 'Above 1');
33          return true;
34      }
```

Listing 55: VestingSchedule.sol

```
36      function sumPermil() internal view returns (uint) {
37          uint permil = 0;
38          for(uint i = 0; i < vestingPeriods.length; ++i) {
39              if (block.timestamp > vestingPeriods[i].timestamp) {
40                  permil += vestingPeriods[i].permil;
41              }
42          }
43          return permil >= 1000 ? 1000 : permil;
44      }
45  }
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Verify that a delay of 900 seconds will not harm the logic of the contract.

Status - Acknowledged

The Metavill team has acknowledged the risk, stating that a 900 seconds delay will not harm the logic of the contract.

I.4 For Loop Over Dynamic Array [LOW]

Description:

When smart contracts are deployed or their associated functions are invoked, the execution of these operations always consumes a certain quantity of gas, according to the amount of computation required to accomplish them. Modifying an unknown-size array that grows in size over time can result in a Denial of Service attack. Simply by having an excessively huge array, users can exceed the gas limit, therefore preventing the transaction from ever succeeding.

Code:

Listing 56: VestingSchedule.sol

```
25      function _precheckPeriod(uint permil) private view returns (bool) {
26          uint total = 0;
27          for(uint i = 0; i < vestingPeriods.length; ++i) {
28              if (block.timestamp > vestingPeriods[i].timestamp) {
29                  total += vestingPeriods[i].permil;
30              }
31          }
```

```
        require(total + permil <= 1000, 'Above 1');
        return true;
    }
```

Listing 57: VestingSchedule.sol

```
    function sumPermil() internal view returns (uint) {
        uint permil = 0;
        for(uint i = 0; i < vestingPeriods.length; ++i) {
            if (block.timestamp > vestingPeriods[i].timestamp) {
                permil += vestingPeriods[i].permil;
            }
        }
        return permil >= 1000 ? 1000 : permil;
    }
}
```

Risk Level:

Likelihood – 1
Impact - 3

Recommendation:

Avoid actions that involve looping across the entire data structure. If you really must loop over an array of unknown size, arrange for it to consume many blocs and thus multiple transactions.

Status - Fixed

The Metavill team has fixed the issue by requiring the length of the vestingPeriods array to be lower than 50.

J MVToken.sol

J.1 Approve Race Condition [LOW]

Description:

The standard ERC20 implementation contains a widely known racing condition in its approve function, wherein a spender can witness the token owner broadcast a transaction altering their approval and quickly sign and broadcast a transaction using transferFrom to move the current approved amount from the owner's balance to the spender. If the spender's transaction is validated before the owner's, the spender will be able to get both approval amounts of both transactions.

Code:

Listing 58: MVToken.sol

```
5 contract MVToken is ERC20 {
```

Risk Level:

Likelihood – 1
Impact – 2

Recommendation:

We recommend using increaseAllowance and decreaseAllowance functions to modify the approval amount instead of using the approve function to modify it.

Status – Acknowledged

The Metavill team has acknowledged the risk, stating that they do not use the approve function.

4 Best Practices

BP.1 Variable Not Used

Description:

A variable is declared and called busd, this variable is not used in any of the contract's functions. It is recommended to remove the variables that are not used in the logic of the contact.

Code:

Listing 59: CreativeMarket.sol

```
10 ERC20 private busd;
```

BP.2 Minimize The Amount Of Approvals

Description:

The buy function requires the buyerAddress to approve price + fee, and the market to approve price to the contract. The market approval can be removed using the following implementation:

Code:

Listing 60: CreativeMarket.sol

```
33 function buy(uint256 creativeId, address buyerAddress, address
       ↪ ownerAddress, uint price, uint fee) public onlyOperator {
34     mv.transferFrom(buyerAddress, market, fee);
35     mv.transferFrom(buyerAddress, ownerAddress, price);
36     creative.transferFrom(ownerAddress, buyerAddress, creativeId);
37
38     emit Bought(creativeId, ownerAddress, buyerAddress);
39 }
```

BP.3 Unnecessary Initializations

Description:

When a variable is declared in solidity, it gets initialized with its type's default value. Thus, there is no need to initialize a variable with the default value.

Code:

Listing 61: IDOPublic.sol

```
41 userCount = 0;
```

Listing 62: IDOWhitelist.sol

```
42 whitelistCount = 0;
```

Listing 63: Private.sol

```
48 uint total = 0
```

Listing 64: Private.sol

```
59 uint permil = 0;
```

Listing 65: Seed.sol

```
40 uint total = 0
```

Listing 66: Seed.sol

```
51 uint permil = 0;
```

BP.4 The message Argument Is Not Used

Description:

The airdrop function contains an argument that is not used, called message. As a best practice, it is recommended to remove the arguments that are not used inside the function.

Code:

Listing 67: IDOPublic.sol

```
18 function airdrop(address user, uint amount, string memory message)
       ↪ external onlyOwner {
19     require(user != address(0));
20     mv.transferFrom(mvWalletAddress, user, amount);
21 }
```

BP.5 Public Function Can Be Called External

Description:

Functions with a public scope that are not called inside the contract should be declared external to reduce the gas fees.

Code:

Listing 68: Creative.sol

```
19 function mint(address to, uint256 id) public onlyOperator {
20     _safeMint(to, id);
21 }
```

Listing 69: Creative.sol

```
23 function setOperator(address operatorAddress, bool value) public
       ↪ onlyOwner {
24     _operators[operatorAddress] = value;
```

```
25      emit OperatorSetted(operatorAddress, value);
26  }
```

Listing 70: CreativeMarket.sol

```
33  function buy(uint256 creativeId, address buyerAddress, address
        ↪ ownerAddress, uint price, uint fee) public onlyOperator {
34      mv.transferFrom(buyerAddress, market, price + fee);
35      mv.transferFrom(market, ownerAddress, price);
36      creative.transferFrom(ownerAddress, buyerAddress, creativeId);
37
38      emit Bought(creativeId, ownerAddress, buyerAddress);
39  }
```

Listing 71: CreativeMarket.sol

```
42  function setOperator(address operatorAddress, bool value) public
        ↪ onlyOwner {
43      _operators[operatorAddress] = value;
44      emit OperatorSetted(operatorAddress, value);
45  }
```

5 Tests

Results:

```
IDO Public
[
  [
    BigNumber { value: "1" },
    BigNumber { value: "10" },
    timestamp: BigNumber { value: "1" },
    permil: BigNumber { value: "10" }
  ]
]
     Add (71ms)

  MV Token
    1) Total supply
     Transfer (612ms)

  Schedule
[
  [
    BigNumber { value: "1" },
    BigNumber { value: "10" },
    timestamp: BigNumber { value: "1" },
    permil: BigNumber { value: "10" }
  ]
]
     Add (47ms)

  Seed
BigNumber { value: "0" }
BigNumber { value: "1000" }
    2) Show success
```

```
3 passing (7s)
2 failing

1) MV Token
     Total supply:

    AssertionError: expected '100000000000000000000000000' to equal
       ↪ '60000000000000000000000000'
    + expected - actual

    -100000000000000000000000000
    +60000000000000000000000000

    at Context.<anonymous> (test/MV-test.js:25:67)
2) Seed
     Show success:
   Error: VM Exception while processing transaction: reverted with
      ↪ reason string 'Amount is zero'
  at Seed.claim (contracts/Seed.sol:64)
  at HardhatNode._mineBlockWithPendingTxs (node_modules/hardhat/src/
     ↪ internal/hardhat-network/provider/node.ts:1772:23)
  at HardhatNode.mineBlock (node_modules/hardhat/src/internal/hardhat-
     ↪ network/provider/node.ts:466:16)
  at EthModule._sendTransactionAndReturnHash (node_modules/hardhat/src
     ↪ /internal/hardhat-network/provider/modules/eth.ts:1496:18)
  at HardhatNetworkProvider.request (node_modules/hardhat/src/internal
     ↪ /hardhat-network/provider/provider.ts:118:18)
  at EthersProviderWrapper.send (node_modules/@nomiclabs/hardhat-
     ↪ ethers/src/internal/ethers-provider-wrapper.ts:13:20)
```

Conclusion:

We recommend fixing the errors encountered during the tests, also, consider adding more testing scenarios in order to guarantee the functionality of the contracts.

6 Static Analysis (Slither)

Description:

ShellBoxes expanded the coverage of the specific contract areas using automated testing methodologies. Slither, a Solidity static analysis framework, was one of the tools used. Slither was run on all-scoped contracts in both text and binary formats. This tool can be used to test mathematical relationships between Solidity instances statically and variables that allow for the detection of errors or inconsistent usage of the contracts' APIs throughout the entire codebase.

Results:

```
Airdrop.airdrop(address,uint256,string) (contracts/Airdrop.sol#18-21)
    ↪ ignores return value by mv.transferFrom(mvWalletAddress,user,
    ↪ amount) (contracts/Airdrop.sol#20)
Different versions of Solidity are used:
        - Version used: ['0.8.4', '^0.8.0']
        - 0.8.4 (contracts/MVToken.sol#2)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/ERC20.
          ↪ sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20
          ↪ .sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
          ↪ extensions/IERC20Metadata.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol
          ↪ #4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #different-pragma-directives-are-used

Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.
    ↪ sol#21-23) is never used and should be removed
ERC20._burn(address,uint256) (node_modules/@openzeppelin/contracts/token
    ↪ /ERC20/ERC20.sol#280-295) is never used and should be removed
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
    ↪ ERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
    ↪ IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
    ↪ extensions/IERC20Metadata.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context
    ↪ .sol#4) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #incorrect-versions-of-solidity

name() should be declared external:
        - ERC20.name() (node_modules/@openzeppelin/contracts/token/ERC20/
            ↪ ERC20.sol#62-64)
symbol() should be declared external:
        - ERC20.symbol() (node_modules/@openzeppelin/contracts/token/
            ↪ ERC20/ERC20.sol#70-72)
decimals() should be declared external:
        - ERC20.decimals() (node_modules/@openzeppelin/contracts/token/
            ↪ ERC20/ERC20.sol#87-89)
totalSupply() should be declared external:
        - ERC20.totalSupply() (node_modules/@openzeppelin/contracts/token
            ↪ /ERC20/ERC20.sol#94-96)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (node_modules/@openzeppelin/contracts/
            ↪ token/ERC20/ERC20.sol#101-103)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (node_modules/@openzeppelin/
            ↪ contracts/token/ERC20/ERC20.sol#113-117)
approve(address,uint256) should be declared external:
```

```
        - ERC20.approve(address,uint256) (node_modules/@openzeppelin/
            ↪ contracts/token/ERC20/ERC20.sol#136-140)
transferFrom(address,address,uint256) should be declared external:
        - ERC20.transferFrom(address,address,uint256) (node_modules/
            ↪ @openzeppelin/contracts/token/ERC20/ERC20.sol#158-167)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (node_modules/
            ↪ @openzeppelin/contracts/token/ERC20/ERC20.sol#181-185)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (node_modules/
            ↪ @openzeppelin/contracts/token/ERC20/ERC20.sol#201-210)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #public-function-that-could-be-declared-external
contracts/MVToken.sol analyzed (5 contracts with 78 detectors), 17
    ↪ result(s) found


Private.claim() (contracts/Private.sol#39-45) ignores return value by mv
    ↪ .transfer(msg.sender,amount) (contracts/Private.sol#44)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unchecked-transfer

Private._precheckPeriod(uint256) (contracts/Private.sol#47-56) uses
    ↪ timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > vestingPeriods[i].timestamp (contracts/
            ↪ Private.sol#50)
Private.claimable(address) (contracts/Private.sol#58-67) uses timestamp
    ↪ for comparisons
        Dangerous comparisons:
        - block.timestamp > vestingPeriods[i].timestamp (contracts/
            ↪ Private.sol#61)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #block-timestamp
```

```
Different versions of Solidity are used:
        - Version used: ['0.8.4', '^0.8.0']
        - 0.8.4 (contracts/Private.sol#2)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol
          ↪ #4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20
          ↪ .sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol
          ↪ #4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #different-pragma-directives-are-used

Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.
    ↪ sol#21-23) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/
    ↪ Ownable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
    ↪ IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context
    ↪ .sol#4) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #incorrect-versions-of-solidity

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/
          ↪ contracts/access/Ownable.sol#61-63)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/
          ↪ contracts/access/Ownable.sol#69-72)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #public-function-that-could-be-declared-external
contracts/Private.sol analyzed (4 contracts with 78 detectors), 10
    ↪ result(s) found



Seed.claim() (contracts/Seed.sol#61-67) ignores return value by mv.
    ↪ transfer(msg.sender,amount) (contracts/Seed.sol#66)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #unchecked-transfer

Seed._precheckPeriod(uint256) (contracts/Seed.sol#39-48) uses timestamp
    ↪ for comparisons
        Dangerous comparisons:
        - block.timestamp > vestingPeriods[i].timestamp (contracts/Seed.
            ↪ sol#42)
Seed.claimable(address) (contracts/Seed.sol#50-59) uses timestamp for
    ↪ comparisons
        Dangerous comparisons:
        - block.timestamp > vestingPeriods[i].timestamp (contracts/Seed.
            ↪ sol#53)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #block-timestamp

Different versions of Solidity are used:
        - Version used: ['0.8.4', '^0.8.0']
        - 0.8.4 (contracts/Seed.sol#2)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol
            ↪ #4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/IERC20
            ↪ .sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol
            ↪ #4)
```

```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #different-pragma-directives-are-used

Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.
    ↪ sol#21-23) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/
    ↪ Ownable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC20/
    ↪ IERC20.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context
    ↪ .sol#4) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #incorrect-versions-of-solidity

renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/
            ↪ contracts/access/Ownable.sol#61-63)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/
            ↪ contracts/access/Ownable.sol#69-72)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #public-function-that-could-be-declared-external
contracts/Seed.sol analyzed (4 contracts with 78 detectors), 10 result(s
    ↪ ) found



VestingSchedule._precheckPeriod(uint256) (contracts/VestingSchedule.sol
    ↪ #25-34) uses timestamp for comparisons
        Dangerous comparisons:
```

```
        - block.timestamp > vestingPeriods[i].timestamp (contracts/
            ↪ VestingSchedule.sol#28)
VestingSchedule.sumPermil() (contracts/VestingSchedule.sol#36-44) uses
    ↪ timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp > vestingPeriods[i].timestamp (contracts/
            ↪ VestingSchedule.sol#39)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #block-timestamp

Different versions of Solidity are used:
        - Version used: ['0.8.4', '^0.8.0']
        - 0.8.4 (contracts/VestingSchedule.sol#2)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol
            ↪ #4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol
            ↪ #4)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #different-pragma-directives-are-used

Context._msgData() (node_modules/@openzeppelin/contracts/utils/Context.
    ↪ sol#21-23) is never used and should be removed
VestingSchedule.sumPermil() (contracts/VestingSchedule.sol#36-44) is
    ↪ never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #dead-code

Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/access/
    ↪ Ownable.sol#4) allows old versions
Pragma version^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context
    ↪ .sol#4) allows old versions
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #incorrect-versions-of-solidity
```

```
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (node_modules/@openzeppelin/
            ↪ contracts/access/Ownable.sol#61-63)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (node_modules/@openzeppelin/
            ↪ contracts/access/Ownable.sol#69-72)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation
    ↪ #public-function-that-could-be-declared-external
contracts/VestingSchedule.sol analyzed (3 contracts with 78 detectors),
    ↪ 9 result(s) found
```

Conclusion:

Most of the vulnerabilities found by the analysis have already been addressed by the smart contract code review.

7 Conclusion

In this audit, we examined the design and implementation of MetaVill contract and discovered several issues of varying severity. Metavill team addressed 39 issues raised in the initial report and implemented the necessary fixes, while classifying the rest as a risk with low-probability of occurrence. Shellboxes' auditors advised Metavill Team to maintain a high level of vigilance and to keep those findings in mind in order to avoid any future complications.

8 Disclaimer

Shellboxes reports should not be construed as "endorsements" or "disapprovals" of particular teams or projects. These reports do not reflect the economics or value of any "product" or "asset" produced by any team or project that engages Shellboxes to do a security evaluation, nor should they be regarded as such. Shellboxes Reports do not provide any warranty or guarantee regarding the absolute bug-free nature of the examined technology, nor do they provide any indication of the technology's proprietors, business model, business or legal compliance. Shellboxes Reports should not be used in any way to decide whether to invest in or take part in a certain project. These reports don't offer any kind of investing advice and shouldn't be used that way. Shellboxes Reports are the result of a thorough auditing process designed to assist our clients in improving the quality of their code while lowering the significant risk posed by blockchain technology. According to Shellboxes, each business and person is in charge of their own due diligence and ongoing security. Shellboxes does not guarantee the security or functionality of the technology we agree to research; instead, our purpose is to assist in limiting the attack vectors and the high degree of variation associated with using new and evolving technologies.

For a Contract Audit, contact us at contact@shellboxes.com