



Secur3

Smart Contract Security Audit

Prepared by ShellBoxes

May 26th, 2022 – May 30th, 2022

[Shellboxes.com](https://shellboxes.com)

contact@shellboxes.com

Document Properties

Client	Secur3
Version	1.0
Classification	Public

Scope

The Secur3 Contract in the Secur3 Repository

Repo	Commit Hash
https://github.com/lowkeycoders/secur3	545b1099afa2d5b2a5b410c398dc022daa452d18
https://github.com/lowkeycoders/secur3/tree/audited	0e95bbfde27fd9585baafecd8014d967dfe9474e

Files	MD5 Hash
contracts/Ownable.sol	dd71db3f99a7946125f4f4b70839ff68
contracts/TwoFactor.sol	86b10451ecf839b21549582d21466d52
contracts/TwoFactorFactory.sol	b9fd292210b5ff95e77373121d0e5e16

Contacts

COMPANY	EMAIL
ShellBoxes	contact@shellboxes.com

Contents

- 1 Introduction 5
 - 1.1 About Secur3 5
 - 1.2 Approach & Methodology 5
 - 1.2.1 Risk Methodology 6
- 2 Findings Overview 7
 - 2.1 Summary 7
 - 2.2 Key Findings 7
- 3 Finding Details 8
 - A TwoFactor.sol 8
 - A.1 Possible Front-Run On The Withdraw Process if private keys have already been compromised [MEDIUM] 8
 - A.2 Missing Transfer Verification [MEDIUM] 9
 - A.3 Floating Pragma [LOW] 11
 - B TwoFactorFactory.sol 11
 - B.1 Missing Address Verification [LOW] 11
 - B.2 Floating Pragma [LOW] 12
- 4 Best Practices 14
 - BP.1 Unnecessary Initializations 14
- 5 Static Analysis (Slither) 15
- 6 Conclusion 23

1 Introduction

Secur3 engaged ShellBoxes to conduct a security assessment on the Secur3 beginning on May 26th, 2022 and ending May 30th, 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 Secur3

Secur3 is the world's first decentralised 2FA solution for your self custody wallets. It provides an added authentication layer for the crypto & NFT assets.

Issuer	Secur3
Website	www.secur3.xyz
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 Secur3 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 , 2 medium-severity, 3 low-severity vulnerabilities.

Vulnerabilities	Severity	Status
Possible Front-Run On The Withdraw Process if private keys have already been compromised	MEDIUM	Mitigated
Missing Transfer Verification	MEDIUM	Fixed
Floating Pragma	LOW	Fixed
Missing Address Verification	LOW	Fixed
Floating Pragma	LOW	Fixed

3 Finding Details

A TwoFactor.sol

A.1 Possible Front-Run On The Withdraw Process if private keys have already been compromised [MEDIUM]

Description:

The contract provides a vault to the users where they can send and withdraw assets from it using a [One-Time Password](#). However, this mechanism does not provide an extra layer of security due to the transparency of the blockchain, anyone can front-run the transaction and extract the password before it gets changed.

Code:

Listing 1: TwoFactor.sol

```
97 modifier passwordMatchAndNewUpdated(  
98     string memory _oldSignedPassword,  
99     bytes32 _newEncryptedPassword  
100 ) {  
101     //TODO: string _oldPassword = "Fetch from signed of only owner";  
102     bytes32 _passwordSent = keccak256(abi.encodePacked(_oldSignedPassword));  
103     require(  
104         _passwordSent != _newEncryptedPassword,  
105         "New password should be different"  
106     );  
107     require(_passwordSent == encryptedPassword, "Passwords don't match");  
108     _;  
109 }
```


Risk Level:

Likelihood – 4

Impact – 2

Recommendation:

Consider removing the **One-Time Password** implementation, as it does not provide any additional value to the contract.

Status – Mitigated

The Secur3 team has mitigated the risk by adding methods that allow the user to transfer his assets to a backup wallet in case the user forgot his password.

A.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 successful. 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. However, it is mostly missed by developers when they carry out checks in effect, the transaction would always succeed, even if the token transfer did not.

Code:

Listing 2: TwoFactor.sol

```
147 function _transferERC721FundsToAddress(  
148     address toAddress,  
149     address tokenAddress,  
150     uint256 tokenId  
151 ) private {
```

```
152     IERC721(tokenAddress).transferFrom(address(this), toAddress, tokenId);
153 }
```

Listing 3: TwoFactor.sol

```
175 function _transferERC20FundsToAddress(
176     address toAddress,
177     address[] memory tokenAddressList
178 ) private {
179     require(tokenAddressList.length != 0, "Assets list cannot be empty");
180     for (uint256 i = 0; i < tokenAddressList.length; i++) {
181         uint256 balance = IERC20(tokenAddressList[i]).balanceOf(
182             address(this)
183         );
184         if (balance > 0) {
185             IERC20(tokenAddressList[i]).transfer(toAddress, balance);
186         }
187     }
188 }
```

Recommendation:

It is recommended to use the [safeTransfer](#) function from the [safeERC20](#) implementation or put the transfer call inside an [assert](#) or [require](#) to verify that the transfer has passed successfully.

Status - Fixed

The Secur3 team has fixed the issue by using the [safeTransfer](#) function from the [safeERC20](#) implementation.

A.3 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma `0.8.0`. Contracts should be deployed using the same compiler version. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version, that may introduce issues in the contract system.

Code:

Listing 4: TwoFactor.sol

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity ^0.8.0;
```

Recommendation:

Consider locking the pragma version. It is advised that the floating pragma should not be used in production. Both `truffle-config.js` and `hardhat.config.js` support locking the pragma version.

Status - Fixed

The Secur3 team has fixed the issue by locking the pragma version to `0.8.7`.

B TwoFactorFactory.sol

B.1 Missing Address Verification [LOW]

Description:

Certain functions lack a safety check in the address, the address-type argument should include a zero-address test, otherwise, some of the contract's functionality may become inaccessible. The `_firstChildAddress` argument should be verified to be different from the `address(0)`.

Code:

Listing 5: TwoFactorFactory.sol

```
60 constructor(address _firstChildAddress) {  
61     firstChildAddress = _firstChildAddress;  
62 }
```

Recommendation:

It is recommended to verify that the addresses provided in the arguments are different from the `address(0)`.

Status - Fixed

The Secur3 team has fixed the issue by adding a `require` statement to make sure the address provided in the argument is different from the `address(0)`.

B.2 Floating Pragma [LOW]

Description:

The contract makes use of the floating-point pragma `0.8.0`. Contracts should be deployed using the same compiler version. Locking the pragma helps ensure that contracts are not unintentionally deployed using another pragma, such as an obsolete version, that may introduce issues in the contract system.

Code:

Listing 6: TwoFactorFactory.sol

```
1 // SPDX-License-Identifier: MIT  
2 pragma solidity ^0.8.0;
```

Recommendation:

Consider locking the pragma version. It is advised that the floating pragma should not be used in production. Both `truffle-config.js` and `hardhat.config.js` support locking the pragma version.

Status - Fixed

The Secur3 team has fixed the issue by locking the pragma version to `0.8.7`.

4 Best Practices

BP.1 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 7: TwoFactor.sol

```
15 bool private isInitialized = false;
```

5 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:

```
TwoFactor._transferNativeFundsToAddress(address) (TwoFactor.sol#190-195) sends eth to arbitrary user
```

Dangerous calls:

- toAddress.transfer(balance) (TwoFactor.sol#193)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#functions-that-send-ether-to-arbitrary-destinations>

```
TwoFactor._transferERC20FundsToAddress(address,address[]) (TwoFactor.sol#175-188) ignores return value by IERC20(tokenAddressList[i]).transfer(toAddress,balance) (TwoFactor.sol#185)
```

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#unchecked-transfer>

```
TwoFactor._transferERC20FundsToAddress(address,address[]) (TwoFactor.sol#175-188) has external calls inside a loop: balance = IERC20(tokenAddressList[i]).balanceOf(address(this)) (TwoFactor.sol#181-183)
```

```
TwoFactor._transferERC20FundsToAddress(address,address[]) (TwoFactor.sol#175-188) has external calls inside a loop: IERC20(tokenAddressList[i]).transfer(toAddress,balance) (TwoFactor.sol#185)
```

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop>

`TwoFactor.init(address,bytes32)` (TwoFactor.sol#17-22) compares to a boolean constant:

```
-require(bool,string)(isInitialized == false,Contract already initialized) (TwoFactor.sol#18)
```

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality>

`Context._msgData()` (../../openzeppelin-contracts/contracts/utils/Context.sol#20-22) is never used and should be removed

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

`Pragma version^0.8.0` (Ownable.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6

`Pragma version^0.8.0` (TwoFactor.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6

`Pragma version^0.8.0` (../../openzeppelin-contracts/contracts/token/ERC1155/IERC1155.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6

`Pragma version^0.8.0` (../../openzeppelin-contracts/contracts/token/ERC1155/IERC1155Receiver.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6

`Pragma version^0.8.0` (../../openzeppelin-contracts/contracts/token/ERC1155/utils/ERC1155Holder.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6

`Pragma version^0.8.0` (../../openzeppelin-contracts/contracts/token/ERC1155/utils/ERC1155Receiver.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6

`Pragma version^0.8.0` (../../openzeppelin-contracts/contracts/token/ERC20/IERC20.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6

`Pragma version^0.8.0` (../../openzeppelin-contracts/contracts/token/ERC721/IERC721.sol#3) necessitates a version too recent to be trusted. Consider dep

loying with 0.6.12/0.7.6
Pragma version^0.8.0 (../../openzeppelin-**contracts/contracts**/token/ERC721/IERC721Receiver.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../../openzeppelin-**contracts/contracts**/token/ERC721/utils/ERC721Holder.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../../openzeppelin-**contracts/contracts**/utils/Context.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../../openzeppelin-**contracts/contracts**/utils/introspection/ERC165.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (../../openzeppelin-**contracts/contracts**/utils/introspection/IERC165.sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.6 is not recommended for deployment
Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#in-correct-versions-of-solidity>

Parameter TwoFactor.init(**address,bytes32**)._sender (TwoFactor.sol#17) is not in mixedCase
Parameter TwoFactor.init(**address,bytes32**)._encryptedPassword (TwoFactor.sol#17) is not in mixedCase
Parameter TwoFactor.transferERC20AssetsToWallet(**string,bytes32,address[]**)._oldSignedPassword (TwoFactor.sol#28) is not in mixedCase
Parameter TwoFactor.transferERC20AssetsToWallet(**string,bytes32,address[]**)._newEncryptedPassword (TwoFactor.sol#29) is not in mixedCase
Parameter TwoFactor.transferERC721AssetsToWallet(**string,bytes32,address,uint256**)._oldSignedPassword (TwoFactor.sol#42) is not in mixedCase
Parameter TwoFactor.transferERC721AssetsToWallet(**string,bytes32,address,uint256**)._newEncryptedPassword (TwoFactor.sol#43) is not in mixedCase
Parameter TwoFactor.transferERC1155AssetsToWallet(**string,bytes32,address,uint256**)._oldSignedPassword (TwoFactor.sol#57) is not in mixedCase

Parameter TwoFactor.transferERC1155AssetsToWallet(string,bytes32,address,uint256)._newEncryptedPassword (TwoFactor.sol#58) is not in mixedCase
Parameter TwoFactor.transferNativeAssetToWallet(string,bytes32)._oldSignedPassword (TwoFactor.sol#72) is not in mixedCase
Parameter TwoFactor.transferNativeAssetToWallet(string,bytes32)._newEncryptedPassword (TwoFactor.sol#73) is not in mixedCase
Parameter TwoFactor.updatePassword(string,bytes32)._oldSignedPassword (TwoFactor.sol#86) is not in mixedCase
Parameter TwoFactor.updatePassword(string,bytes32)._newEncryptedPassword (TwoFactor.sol#87) is not in mixedCase
Variable TwoFactor.DAO_MULTI_SIG (TwoFactor.sol#14) is not in mixedCase
Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions>

TwoFactor.DAO_MULTI_SIG (TwoFactor.sol#14) is never used in TwoFactor (TwoFactor.sol#12-196)
Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable>

TwoFactor.DAO_MULTI_SIG (TwoFactor.sol#14) should be constant
Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant>

onERC1155Received(address,address,uint256,uint256,bytes) should be declared external:

- ERC1155Holder.onERC1155Received(address,address,uint256,uint256,bytes) (../../openzeppelin-contracts/contracts/token/ERC1155/utils/ERC1155Holder.sol#11-19)

onERC1155BatchReceived(address,address,uint256[],uint256[],bytes) should be declared external:

- ERC1155Holder.onERC1155BatchReceived(address,address,uint256[],uint256[],bytes) (../../openzeppelin-contracts/contracts/token/ERC1155/utils/ERC1155Holder.sol#21-29)

onERC721Received(address,address,uint256,bytes) should be declared external

:

- ERC721Holder.onERC721Received(address,address,uint256,bytes) (../openzeppelin-contracts/contracts/token/ERC721/utils/ERC721Holder.sol#19-26)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external>

Pragma version>=0.4.22<0.9.0 (Migrations.sol#2) is too complex
solc-0.8.6 is not recommended for deployment

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#in-correct-versions-of-solidity>

Variable Migrations.last_completed_migration (Migrations.sol#6) is not in mixedCase

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions>

setCompleted(uint256) should be declared external:

- Migrations.setCompleted(uint256) (Migrations.sol#16-18)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external>

Context._msgData() (../openzeppelin-contracts/contracts/utils/Context.sol#20-22) is never used and should be removed

Context._msgSender() (../openzeppelin-contracts/contracts/utils/Context.sol#16-18) is never used and should be removed

Ownable._transferOwnership(address) (Ownable.sol#45-49) is never used and should be removed

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

Pragma version^0.8.0 (Ownable.sol#2) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6

Pragma version^0.8.0 (../openzeppelin-contracts/contracts/utils/Context.

sol#3) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6

solc-0.8.6 is not recommended for deployment

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#in-correct-versions-of-solidity>

Reentrancy in TwoFactorFactory.createTwoFactor(bytes32) (TwoFactorFactory.sol#64-73):

External calls:

- ITwoFactor(clone).init(msg.sender, _encryptedPassword) (TwoFactorFactory.sol#70)

State variables written after the call(s):

- eoaToVaultMap[msg.sender] = clone (TwoFactorFactory.sol#71)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1>

TwoFactorFactory.constructor(address)._firstChildAddress (TwoFactorFactory.sol#60) lacks a zero-check on :

- firstChildAddress = _firstChildAddress (TwoFactorFactory.sol#61)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation>

Reentrancy in TwoFactorFactory.createTwoFactor(bytes32) (TwoFactorFactory.sol#64-73):

External calls:

- ITwoFactor(clone).init(msg.sender, _encryptedPassword) (TwoFactorFactory.sol#70)

Event emitted after the call(s):

- TwoFactorCreated(clone) (TwoFactorFactory.sol#72)

Reference: <https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3>

CloneFactory.createClone(address) (TwoFactorFactory.sol#5-20) uses assembly

6 Conclusion

In this audit, we examined the design and implementation of Secur3 contract and discovered several issues of varying severity. Secur3 team addressed all the issues raised in the initial report and implemented the necessary fixes.

The present code base is well-structured and ready for the mainnet.



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