Computing Will – Security Review

Version 1.1

7 January 2024

Conducted by: 0xFlint_

Table of Contents

1 About Rocksolid 3
2 Disclaimer 3
3 Risk Classification 3
3.1 Impact 3
3.2 Likelihood 3
3.3 Actions required by severity level 3
4 Executive Summary 4
5 Findings 5
5.1 Critical 5
5.1.1 No access control on checkAfterExecution 5
5.2 High 5
5.2.1 No support for ERC721 or ERC1155 tokens 5
5.2.2 Failed transfers will be treated as successful transfers 6
5.2.3 Some valid transactions do not update lastTimestampTxs 7
5.3 Medium 7
5.3.1 Allowing non-EOA beneficiaries can break distribution 7
5.3.2 Static salt can lead to permanent Denial-Of-Service 8
5.3.3 Module transactions are not recorded as signs of life 8
5.3.4 Blacklisted beneficiaries will break the distribution 9

1 About Rocksolid

RockSolid Security aims to provide the best possible smart contract auditing services to protocols. 0xFlint_ has extensive experience in web2 cyber-security as well as web3 auditing, with a proven track record of over 25 audits performed on public and private codebases.

2 Disclaimer

Audits are a time, resource, and expertise bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can show the presence of vulnerabilities but **not their absence**.

3 Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - only a small amount of funds can be lost, or a functionality of the protocol is affected.
- **Low** - any kind of unexpected behaviour that is an inconvenience to users.

3.2 Likelihood

- **High** – almost certain to occur, either due to an inevitable logic flaw or very low costs and complexity to exploit the protocol.
- **Medium** – requires certain conditions for it to occur, but still relatively likely.
- **Low** – requires extreme conditions or little to no incentive to exploit.

3.3 Actions required by severity level

- **Critical** - client **must** fix the issue.
- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive Summary

Overview

Project Name	Computing Will
Repository	Public
Commit hash	c969eb2556db4d69b37935fb695d02c55d80111d
Resolution	87ea35b411885e7607cd4b7c6df3564e89c53e8f
Documentation	https://docs.10102.io/digital-inheritance
Methods	Manual review

Scope

forwarding/ForwardingWill.sol	common/WillFactory.sol
forwarding/ForwardingEOAWillRouter.sol	interfaces/IForwardingWill.sol
forwarding/ForwardingEOAWill.sol	interfaces/IERC20Whitelist.sol
forwarding/ForwardingWillRouter.sol	interfaces/IInheritanceWill.sol
inheritance/InheritanceWill.sol	interfaces/ISafeWallet.sol
SafeGuard.sol	interfaces/IERC20.sol
access/AccessGuard.sol	interfaces/ISafeGuard.sol
InheritanceWillRouter.sol	interfaces/IForwardingEOAWill.sol
common/EOAWillFactory.sol	libraries/InheritanceWillStruct.sol
common/GenericWill.sol	libraries/Enum.sol
common/WillRouter.sol	libraries/ForwardingWillStruct.sol

Issues Found

Critical risk	1
High risk	3
Medium risk	4

5 Findings

5.1 Critical

5.1.1 No access control on checkAfterExecution

Severity: Critical Risk

Context: SafeGuard.sol#L62

Description:

The lastTimestampTxs variable plays a critical role in determining if a Will can be activated or not. If sufficient time has passed since the last update of the timestamp, the distribution and inheritance functions become callable.

The issue is that currently checkAfterExecution is an external function with no access control. Which means that anyone can call this function as many times as they like and by doing so, permanently prevent a Will from being activated.

Recommandation:

Access control needs to be implemented so that only valid signed transactions originating from the linked safeWallet are accepted as valid signs of life and update the lastTimestampTxs.

Resolution: Fixed.

5.2 High

5.2.1 No support for ERC721 or ERC1155 tokens

Severity: High Risk

Context: ForwardingWill.sol#L13

Description:

The ForwardingWill.sol contract is supposed to distribute all assets from a safeWallet to a list of beneficiaries, but the contract only has functionality for native and ERC20 assets. There is no support for ERC721 or ERC1155 assets.

This is a major issue since any safeWallet can have ERC20, ERC721 and ERC1155 assets, as detailed by the TokenCallbackHandler.sol contract which can be implemented in Safe.sol through the fallbackhandler.

If a Will is activated and the safeWallet has any assets such as ERC721 and/or ERC1155, these assets cannot be distributed and remain locked in the safe indefinitely.

Recommandation:

A check needs to be implemented which prevents the creation of a ForwardingWill for any safeWallet that has implemented support for ERC721 or ERC1155 tokens. Users should be directed to creating an InheritanceWill.

```
// In createWill()
// Check if safe wallet supports ERC721 or ERC1155 callbacks
if (supportsTokenCallbacks(safeWallet)) revert TokenCallbacksNotSupported();


function supportsTokenCallbacks(address safe_) internal view returns (bool) {
  IERC165 safe = IERC165(safe_);
  return safe.supportsInterface(0x150b7a02) || // IERC721Receiver
         safe.supportsInterface(0x4e2312e0);    // IERC1155Receiver
}
```

Resolution: A pop-up window has been added to the front-end that will inform users of the potential effects whenever a wallet supporting ERC721 & ERC1155 has been detected.

5.2.2 Failed transfers will be treated as successful transfers

Severity: High Risk

Context: ForwardingWill.sol#237

Description:

The _transferErc20ToBeneficiary function in ForwardingWill distributes the ERC20 assets from the safeWallet to beneficiaries through an execTransactionFromModule call with transfer(address,uint256) as data.

The issue here is that the transferErc20Success bool is checking the success of the execTransactionFromModule call and **not** the success of the transfer call. This problematic since there are numerous ERC20 tokens which return false or nothing when the transfer fails.

As a result, the logic assumes the tokens have been transferred while in reality they remain in the safewallet contract.

Recommandation:

The bool used to check success needs to be based on the returndata from the transfer call.

Resolution: Fixed.

5.2.3 Some valid transactions do not update lastTimestampTxs

Severity: High

Context: SafeGuard.sol#62

Description:

The fundamental principle of Computing Will is that a Will should only be activated if the owner(s) has given no sign of life. Currently, the lastTimestampTxs is updated in checkAfterExecution **only** if the transaction was successful.

This is a major issue since any initiated transaction with sufficient valid signatures is a sign of life. The success or failure of the specific transaction is irrelevant. Both paths are accounted for in the Safe.sol code with corresponding events emitted.

Recommandation:

Remove the conditional nature of updating lastTimestampTxs or move it entirely into checkTransaction.

Resolution: Fixed.

5.3 Medium

5.3.1 Allowing non-EOA beneficiaries can break distribution

Severity: Medium

Context: ForwardingWill.sol#L218 & ForwardingEOAWill.sol#218

Description:

The only checks on beneficiaries are that they cannot be address(0) nor can they be the owner. This means that both EOA and contracts are allowed.

This is important since the asset distribution is based on a push mechanism where all assets are distributed in one transaction. If the contract is not set up to properly handle ETH or ERC20 assets, either by accident or malicious intent, this will cause a revert and lock the assets permanently in the safeWallet or EOA contract holding the assets.

Recommandation:

A pull system for distributing assets is the preferred mechanism since a beneficiary can then only cause harm to himself. Alternatively, if only EOA are allowed, then this scenario cannot occur.

Resolution: A check has been added to only allow EOA beneficiaries.

5.3.2 Static salt can lead to permanent Denial-Of-Service

Severity: Medium

Context: WillFactory.sol#43 & EOAWillFactory.sol#25

Description:

The create2 deployment process uses a salt which is based on address[sender] and noncebyUsers[sender_]. Once the transaction to deploy at the calculated address is submitted, it can be read from the mempool and a malicious actor can frontrun the transaction to deploy a contract at the address, which would make the deployment fail.

This should be a very minor issue since in most cases this would require an attacker to frontrun every transaction in order to cause a Denial-Of-Service, which is not realistic in the slightest.

However, due to the salt calculation used in the protocol, the calculated address will be the same on **every deployment**. Since the salt is based on the sender's address and a nonce which only increases after a successful deployment, the salt and resulting address will be the same whenever a deployment reverts due to frontrunning.

Thus, a malicious actor would only need to frontrun the initial create2 deployment to cause a permanent Denial-Of-Service.

Recommandation:

Include a third variable in the salt calculation which increases independently from the create2 deployment process.

Resolution: Fixed.

5.3.3 Module transactions are not recorded as signs of life

Severity: Medium

Context: ModuleManager.sol#L154

Description:

The fundamental premise of the protocol is the distribution of safeWallet assets once the owners has not given a sign of life after a certain pre-determined amount of time. This is being tracked through lastTimestampTxs, which is updated every time a valid execTransaction is submitted to the safeWallet.

However, an owner can add any number of safeModules to the safeWallet and the transactions that pass through these modules, through execTransactionFromModule, cannot be tracked by lastTimestampTxs.

Consequently, it is possible that an owner who is regularly updating his safeWallet through a safeModule will have his Will activated and his funds transferred or lose ownership of his safeWallet.

Recommandation:

There is no technical path to block owners from adding new safeModules after the creation of the Will and interacting with the safeWallet through the new module.

Resolution: The documentation will be updated to clearly communicate this to users.

5.3.4 Blacklisted beneficiaries will break the distribution

Severity: Medium

Context: ForwardingWill.sol#L184 & ForwardingEOAWill.sol#224

Description:

The _transferAssetToBeneficiaries function in both contracts distributes all assets to all beneficiaries through two for loops.

If any of the beneficiaries is blacklisted by an ERC20 token such as USDT or USDC, it will cause the entire transaction to revert and block distribution of assets to all parties until the blacklisted status has been resolved.

Recommandation:

Using a push mechanism is inherently problematic since any problem with any transaction for any beneficiary will have negative consequences for all beneficiaries.

If a pull mechanism where used, where every beneficiary can withdraw his part of the assets, a blacklisted beneficiary could only affect his own distribution.

Resolution: This will be re-designed in a next iteration.