// HALBORN

Staging Labs - Saferoot Contracts

Smart Contract Security Assessment

Prepared by: **Halborn**

Date of Engagement: **July 12th, 2023 - July 21st, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	07/16/2023
0.2	Draft Version	07/21/2023
0.3	Draft Review	07/21/2023
0.4	Draft Review	07/24/2023
1.0	Remediation Plan	08/18/2023
1.1	Remediation Plan Updates	08/25/2023
1.2	Remediation Plan Updates Review	08/25/2023
1.3	Remediation Plan Updates Review	08/26/2023

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Piotr Cielas	Halborn	Piotr.Cielas@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Saferoot is designed to provide additional safety features for the users of ERC-20, ERC-721, and ERC-1155 tokens. It provides a safety mechanism in case of a mistake, hack, or scam by transferring the users' tokens from their wallet to their backup address before the transaction that triggered the safeguard executes.

Staging Labs engaged Halborn to conduct a security assessment on their smart contracts beginning on July 12th, 2023 and ending on July 21st, 2023. The security assessment was scoped to the smart contracts provided in the Staging-Labs/Saferoot-Contract GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

1.2 ASSESSMENT SUMMARY

Halborn was provided one week for the engagement and assigned a team of one full-time security engineer to review the security of the smart contracts in scope. The security team consists of a blockchain and smart contract security experts with advanced penetration testing and smart contract hacking skills, and deep knowledge of multiple blockchain protocols.

The purpose of the assessment is to:

- Identify potential security issues within the smart contracts.
- Ensure that smart contract functionality operates as intended.

In summary, Halborn identified some improvements to reduce the likelihood and impact of risks that were mostly addressed by Staging Labs.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this assessment. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow the security best practices. The following phases and associated tools were used during the assessment:

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Scanning of solidity files for vulnerabilities, security hot-spots or bugs. (MythX)
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Foundry, Brownie)

2. RISK METHODOLOGY

Every vulnerability and issue observed by Halborn is ranked based on **two sets** of **Metrics** and a **Severity Coefficient**. This system is inspired by the industry standard Common Vulnerability Scoring System.

The two **Metric sets** are: **Exploitability** and **Impact**. **Exploitability** captures the ease and technical means by which vulnerabilities can be exploited and **Impact** describes the consequences of a successful exploit.

The **Severity Coefficients** is designed to further refine the accuracy of the ranking with two factors: **Reversibility** and **Scope**. These capture the impact of the vulnerability on the environment as well as the number of users and smart contracts affected.

The final score is a value between 0-10 rounded up to 1 decimal place and 10 corresponding to the highest security risk. This provides an objective and accurate rating of the severity of security vulnerabilities in smart contracts.

The system is designed to assist in identifying and prioritizing vulnerabilities based on their level of risk to address the most critical issues in a timely manner.

2.1 EXPLOITABILITY

Attack Origin (AO):

Captures whether the attack requires compromising a specific account.

Attack Cost (AC):

Captures the cost of exploiting the vulnerability incurred by the attacker relative to sending a single transaction on the relevant blockchain. Includes but is not limited to financial and computational cost.

Attack Complexity (AX):

Describes the conditions beyond the attacker's control that must exist in order to exploit the vulnerability. Includes but is not limited to macro situation, available third-party liquidity and regulatory challenges.

Metrics:

Exploitability Metric (m_E)	Metric Value	Numerical Value
Attack Origin (AO)	Arbitrary (AO:A) Specific (AO:S)	1 0.2
Attack Cost (AC)	Low (AC:L) Medium (AC:M) High (AC:H)	1 0.67 0.33
Attack Complexity (AX)	Low (AX:L) Medium (AX:M) High (AX:H)	1 0.67 0.33

Exploitability E is calculated using the following formula:

$$E = \prod m_e$$

2.2 IMPACT

Confidentiality (C):

Measures the impact to the confidentiality of the information resources managed by the contract due to a successfully exploited vulnerability. Confidentiality refers to limiting access to authorized users only.

Integrity (I):

Measures the impact to integrity of a successfully exploited vulnerability. Integrity refers to the trustworthiness and veracity of data stored and/or processed on-chain. Integrity impact directly affecting Deposit or Yield records is excluded.

Availability (A):

Measures the impact to the availability of the impacted component resulting from a successfully exploited vulnerability. This metric refers to smart contract features and functionality, not state. Availability impact directly affecting Deposit or Yield is excluded.

Deposit (D):

Measures the impact to the deposits made to the contract by either users or owners.

Yield (Y):

Measures the impact to the yield generated by the contract for either users or owners.

Metrics:

Impact Metric (m_I)	Metric Value	Numerical Value
Confidentiality (C)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Integrity (I)	None (I:N)	0
	Low (I:L)	0.25
	Medium (I:M)	0.5
	High (I:H)	0.75
	Critical (I:C)	1
Availability (A)	None (A:N)	0
	Low (A:L)	0.25
	Medium (A:M)	0.5
	High (A:H)	0.75
	Critical	1
Deposit (D)	None (D:N)	0
	Low (D:L)	0.25
	Medium (D:M)	0.5
	High (D:H)	0.75
	Critical (D:C)	1
Yield (Y)	None (Y:N)	0
	Low (Y:L)	0.25
	Medium: (Y:M)	0.5
	High: (Y:H)	0.75
	Critical (Y:H)	1

Impact I is calculated using the following formula:

$$I = max(m_I) + \frac{\sum m_I - max(m_I)}{4}$$

2.3 SEVERITY COEFFICIENT

Reversibility (R):

Describes the share of the exploited vulnerability effects that can be reversed. For upgradeable contracts, assume the contract private key is available.

Scope (S):

Captures whether a vulnerability in one vulnerable contract impacts resources in other contracts.

Coefficient (C)	Coefficient Value	Numerical Value
Reversibility (r)	None (R:N) Partial (R:P) Full (R:F)	1 0.5 0.25
Scope (s)	Changed (S:C) Unchanged (S:U)	1.25 1

Severity Coefficient C is obtained by the following product:

$$C = rs$$

The Vulnerability Severity Score S is obtained by:

$$S = min(10, EIC * 10)$$

The score is rounded up to 1 decimal places.

Severity	Score Value Range
Critical	9 - 10
High	7 - 8.9
Medium	4.5 - 6.9
Low	2 - 4.4
Informational	0 - 1.9

2.4 SCOPE

1. Saferoot Contracts

- Repository: Staging-Labs/Saferoot-Contract
- Commit ID: 834455599dacda269ed222182c3576bf37d70038
- Smart contracts in scope:
  - contracts/ContractRegistry.sol
  - contracts/ErrorReporter.sol
  - contracts/Saferoot.sol
  - contracts/SaferootFactory.sol
- Final fix commit ID: 0884050

Out-of-scope:
- third-party libraries and dependencies
- economic attacks

3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	3	10

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) MISSING ZERO ADDRESS CHECKS	Low (3.1)	SOLVED - 08/07/2023
(HAL-02) REGISTRY CANNOT BE RE-ENABLED	Low (2.5)	SOLVED - 08/02/2023
(HAL-03) REGISTRY CANNOT BE CHANGED	Low (2.5)	SOLVED - 08/02/2023
(HAL-04) LACK OF SAFEGUARD DATA VALIDATION	Informational (1.7)	SOLVED - 08/08/2023
(HAL-05) LACK OF DISABLEINITIALIZERS IN THE IMPLEMENTATION CONTRACT	Informational (0.8)	SOLVED - 08/04/2023
(HAL-06) ARBITRARY AMOUNT OF SAFEGUARDS CAN BE ADDED	Informational (0.8)	ACKNOWLEDGED
(HAL-07) INCOMPATIBILITY WITH TOKENS NOT FOLLOWING THE STANDARDS	Informational (0.8)	ACKNOWLEDGED
(HAL-08) MULTIPLE SAFEGUARDS CAN BE ADDED TO THE SAME ASSET	Informational (0.8)	SOLVED - 08/23/2023
(HAL-09) ITERATING OVER A DYNAMIC ARRAY	Informational (0.8)	ACKNOWLEDGED
(HAL-10) MISTAKENLY SENT TOKENS AND ETHER CANNOT BE RECOVERED FROM THE CONTRACTS	Informational (0.8)	SOLVED - 08/08/2023
(HAL-11) FOR LOOPS CAN BE GAS OPTIMIZED	Informational (0.0)	SOLVED - 08/08/2023
(HAL-12) UNNECESSARY VALIDATION	Informational (0.0)	SOLVED - 08/07/2023
(HAL-13) NOT ALL EVM COMPATIBLE CHAIN SUPPORTS SOLIDITY 0.8.20	Informational (0.8)	ACKNOWLEDGED

FINDINGS & TECH DETAILS

4.1 (HAL-01) MISSING ZERO ADDRESS CHECKS - LOW (3.1)

Description:

The backup state variable is used to store the address where the user's tokens are transferred if safeguards are initiated. However, it was identified that this variable lacks zero address validation when configured. Setting an invalid backup address results in loss of funds.

The service state variable is used to store the address of the account that is authorized to initiate safeguards. This variable also lacks zero address validation. Configuring an invalid address prevents the Saferoot contract from initiating safeguards.

The contractRegistry state variable points to the contract's registry used for token verification. If it is set to zero when the registry is enabled, no safeguards can be added.

The user state variable points to the contract's user address. If set incorrectly, the contract's functions are inaccessible.

Code Location:

Listing 1: contracts/Saferoot.sol (Lines 127-130)

```
120 function initialize(
121     address _service,
122     address _user,
123     address _backup,
124     address _contractRegistry,
125     bool _registryEnabled
126 ) public initializer {
127     user = _user;
128     backup = _backup;
129     service = _service;
130     contractRegistry = _contractRegistry;
131     registryEnabled = _registryEnabled;
```

```
132     _grantRole(SERVICE_ROLE, _service);
133     _grantRole(USER_ROLE, _user);
134
135     emit SaferootCreated(_user, _service);
136 }
```

Listing 2: contracts/Saferoot.sol (Line 443)

```
442 function setBackupWallet(address _backup) external onlyUser {
443     backup = _backup;
444     emit BackupUpdated(_backup);
445 }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:L/Y:N/R:N/S:U (3.1)

Recommendation:

It is recommended to review the above functions and add zero address checks where applicable.

Remediation Plan:

SOLVED: The Staging Labs team solved the issue in commit 9e26e35 by adding zero address checks.

4.2 (HAL-02) REGISTRY CANNOT BE RE-ENABLED - LOW (2.5)

Description:

The ContractRegistry contract is responsible for storing the addresses of the token contracts that are Saferoot verified. If this contract is disabled in the Saferoot contract, it can no longer be re-enabled. As a result, users could no longer use it to verify the token contracts when adding safeguards.

Code Location:

If the registry is enabled in the Saferoot contract, it is used to verify the token contracts when adding safeguards:

Listing 3: contracts/Saferoot.sol

```
328     if (
329         registryEnabled &&
330         !registry.isContractSupported(incomingSafeguard.
 ↳ contractAddress)
331     ) {
332         revert InvalidContractAddress();
333     }
```

However, there is no enableRegistry function implemented in the Saferoot contract. The user can only disable the registry:

Listing 4: contracts/Saferoot.sol

```
450     function disableRegistry() external onlyUser {
451         registryEnabled = false;
452         emit RegistryFlagUpdated(false);
453     }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to add a function to the contract that allows users to enable the registry.

Remediation Plan:

SOLVED: The Staging Labs team concluded that the contract registry was unnecessary and removed it from the protocol in commit 5166c1d.

4.3 (HAL-03) REGISTRY CANNOT BE CHANGED - LOW (2.5)

Description:

The ContractRegistry contract is responsible for storing the addresses of the contracts that are Saferoot verified. Users may wish to use a different implementation, but once the Saferoot contract is initialized, it is no longer possible to update the registry address in it.

Code Location:

Listing 5: contracts/Saferoot.sol (Line 130)

```
120     function initialize(
121         address _service,
122         address _user,
123         address _backup,
124         address _contractRegistry,
125         bool _registryEnabled
126     ) public initializer {
127         user = _user;
128         backup = _backup;
129         service = _service;
130         contractRegistry = _contractRegistry;
131         registryEnabled = _registryEnabled;
132         _grantRole(SERVICE_ROLE, _service);
133         _grantRole(USER_ROLE, _user);
134 
135         emit SaferootCreated(_user, _service);
136     }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:L/D:N/Y:N/R:N/S:U (2.5)

Recommendation:

It is recommended to add a function to the contract that allows users to change the registry address.

Remediation Plan:

SOLVED: The Staging Labs team concluded that the contract registry was unnecessary and removed it from the protocol in commit 5166c1d.

4.4 (HAL-04) LACK OF SAFEGUARD DATA VALIDATION - INFORMATIONAL (1.7)

Description:

The safeguards are used to transfer the tokens from the user to the backup wallet address if they are initiated. It was identified that users can create safeguards with parameters that are not related to their token type. For example, it is possible to create an ERC20 token safeguard with a non-zero `tokenId` parameter. Using an invalid parameter indicates that the user may have accidentally switched the order of the values. Reverting these function makes it easier to detect such errors.

Code Location:

The `SafeguardEntry` and `SafeguardEditEntry` structs are used in the `addSafeguard` and `editSafeguard` functions:

Listing 6: contracts/Saferoot.sol

```
303     struct SafeguardEntry {
304         TokenType tokenType;
305         address contractAddress;
306         uint256 amount;
307         uint256 tokenId;
308     }
```

Listing 7: contracts/Saferoot.sol

```
377     struct SafeguardEditEntry {
378         bytes32 key;
379         uint256 newTokenId;
380         uint256 newAmount;
381     }
```

For example, the tokenId parameter is not checked in the ERC20 branch of the addSafeguard function:

Listing 8: contracts/Saferoot.sol (Lines 339-342)

```
322             SafeguardEntry memory incomingSafeguard = _ercEntries[
 ↳ index];
323             if (incomingSafeguard.contractAddress == address(0)) {
324                 revert ZeroAddress();
325             }
326
327             // If registry is enabled and the contract is a safe +
 ↳  supported address
328             if (
329                 registryEnabled &&
330                 !registry.isContractSupported(incomingSafeguard.
 ↳ contractAddress)
331             ) {
332                 revert InvalidContractAddress();
333             }
334
335             bytes32 key = encodeKey(currentKey, incomingSafeguard.
 ↳ tokenType);
336             if (incomingSafeguard.tokenType == TokenType.ERC20) {
337                 ERC20SafeguardInfo storage safeguard =
 ↳ erc20Safeguards[key];
338                 if (safeguard.contractAddress == address(0)) {
339                     safeguard.contractAddress = incomingSafeguard
340                         .contractAddress;
341                     safeguard.amount = incomingSafeguard.amount;
342                     emit ERC20SafeguardAdded(key);
343                     unchecked {
344                         ++currentKey;
345                     }
346                 }
347             }
```

BVSS:

AO:A/AC:L/AX:M/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (1.7)

Recommendation:

Consider reverting the addSafeguard and editSafeguard functions if the safeguards contain inappropriate values.

Remediation Plan:

SOLVED: The Staging Labs team solved the issue in commits 6562223 and bbe3474 by reverting the related functions if the safeguards contain inappropriate values. The editSafeguard function was removed from the contract.

4.5 (HAL-05) LACK OF DISABLEINITIALIZERS IN THE IMPLEMENTATION CONTRACT - INFORMATIONAL (0.8)

Description:

The Saferoot contract uses the Initializable module from OpenZeppelin, and the implementations of this contract are not marked as initialized in the constructor. An uninitialized instance can be initialized by anyone to take over the contract. Even if it does not affect the instances deployed by the SaferootFactory, it is still a good practice to prevent the initialization of the implementation contracts to avoid misuse. In the latest versions, this is done by calling the _disableInitializers function in the constructor.

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (0.8)

Recommendation:

Consider including a constructor to automatically mark the contracts as initialized when they are deployed:

Listing 9: Initialization Example

```
1 * /// @custom:oz-upgrades-unsafe-allow constructor
2 * constructor() {
3 *     _disableInitializers();
4 * }
```

Remediation Plan:

SOLVED: The Staging Labs team solved the issue in commit d1df49e by including a constructor to automatically mark the contracts as initialized.

4.6 (HAL-06) ARBITRARY AMOUNT OF SAFEGUARDS CAN BE ADDED - INFORMATIONAL (0.8)

Description:

It was identified that there is no limit in the Saferoot contract on how many safeguards can be added by the user. Adding too many safeguards may prevent the service user to efficiently monitoring the system and call safeguards in time.

Note that the off-chain backend system was outside this security assessment's scope.

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (0.8)

Recommendation:

Consider limiting the number of safeguards that can be added to the Saferoot contract.

Remediation Plan:

ACKNOWLEDGED: The Staging Labs team acknowledged this finding and will perform tests to identify the limitations of the service and manage the number of active safeguards off-chain.

4.7 (HAL-07) INCOMPATIBILITY WITH TOKENS NOT FOLLOWING THE STANDARDS - INFORMATIONAL (0.8)

Description:

It was identified that the Saferoot contract may not be compatible with tokens that do not follow the ERC standards if these tokens do not implement the functions (e.g., getApproved, safeTransferFrom) that the Saferoot contract uses to verify permissions for token transfers or transfer tokens to the backup address (e.g., CryptoKitties and CryptoPunks NFTs).

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (0.8)

Recommendation:

Consider modifying the Saferoot contract to be more flexible and support tokens that do not follow the standards or notify the users about these limitations.

Remediation Plan:

ACKNOWLEDGED: The Staging Labs team acknowledged this finding. They are going to create an off-chain database with the supported tokens, and they will notify the users about these limitations.

4.8 (HAL-08) MULTIPLE SAFEGUARDS CAN BE ADDED TO THE SAME ASSET - INFORMATIONAL (0.8)

Description:

It was identified that multiple safeguards can be added to the same asset in the Saferoot contract. Adding multiple safeguards for the same asset may prevent the backend systems from correctly identifying which safeguards should be called.

Note that the backend systems were outside this security assessment's scope.

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (0.8)

Recommendation:

Consider limiting the number of safeguards to one per asset.

Remediation Plan:

SOLVED: The Staging Labs team solved the issue in commit 837c806 by modifying the design of the Saferoot contract.

4.9 (HAL-09) ITERATING OVER A DYNAMIC ARRAY - INFORMATIONAL (0.8)

Description:

The initiateSafeguard function transfers the tokens from the user's wallet to the backup address based on the added safeguards. The keys of the safeguards to use are given in the function's _safeguardKeys array parameter. The function iterates through these keys and gets the data of the safeguards from the associated mappings, transferring the specified tokens from the user to the backup address. However, this involves looping over multiple array items, querying the required data, calling external contracts, and emitting events.

The execution of these actions requires a certain amount of gas based on how much computation is needed to complete them. Adding too many keys to the _safeguardKeys parameter may max out the gas limit, reverting the initiateSafeguard function, which prevents transferring the user's tokens in time.

Code Location:

The initiateSafeguard function is looping over its dynamic array parameter:

Listing 10: contracts/Saferoot.sol

```
243     function initiateSafeguard(bytes32[] calldata _safeguardKeys)
244         external
245         payable
246         onlyService
247         nonReentrant
248     {
249         for (uint256 index = 0; index < _safeguardKeys.length; ++
 ↳ index) {
250             bytes32 key = _safeguardKeys[index];
251             TokenType tokenType = decodeKeyTokenType(key);
252
```

```
253             if (tokenType == TokenType.ERC20) {
254                 ERC20SafeguardInfo memory safeguard =
 ↳ erc20Safeguards[key];
255                 address contractAddress = safeguard.
 ↳ contractAddress;
256
257                 if (contractAddress != address(0)) {
258                     // Transfer token
259                     if (_transfer20(key, contractAddress,
 ↳ safeguard.amount)) {
260                         emit SafeguardInitiated(key);
261                     }
262                     // If the contract address is already set,
 ↳ then skip
263                     continue;
264                 }
265             }
```

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (0.8)

Recommendation:

Consider performing tests on each chain to find out the limitations of the `initiateSafeguard` function. It may be appropriate to limit the size of the parameter array off-chain, adjust the default gas limit, and initiate large amounts of safeguards in multiple separate transactions.

Remediation Plan:

ACKNOWLEDGED: The Staging Labs team acknowledged this finding.

4.10 (HAL-10) MISTAKENLY SENT TOKENS AND ETHER CANNOT BE RECOVERED FROM THE CONTRACTS - INFORMATIONAL (0.8)

Description:

It was identified that the Saferoot, SaferootFactory, and ContractRegistry contracts are missing functions to sweep/recover accidental ERC20 token and Ether transfers. Mistakenly sent tokens and Ether are locked in the contracts indefinitely.

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (0.8)

Recommendation:

Consider adding a function to recover accidental token and Ether transfers.

Remediation Plan:

SOLVED: The Staging Labs team solved the issue in commit 910d794 by adding a function to enable the user to recover accidental ERC20 token transfers and removing the payable functions to prevent accidental Ether transfers. Note that only the user can withdraw accidental token transfers from the Saferoot contract.

4.11 (HAL-11) FOR LOOPS CAN BE GAS OPTIMIZED - INFORMATIONAL (0.0)

Description:

It was identified that the for loops employed in the contracts can be gas optimized by the following principles:

- Unnecessary reading of the array length on each iteration wastes gas.
- A postfix (e.g. `i++`) operator was used to increment the `i` variables. It is known that, in loops, using prefix operators (e.g. `++i`) costs less gas per iteration than postfix operators. It is also possible to further optimize loops by using unchecked loop index incrementing and decrementing.

Note that view or pure functions only cost gas if they are called from on-chain.

Code Location:

contracts/ContractRegistry.sol
- Line 52: `for (uint256 i = 0; i < _contracts.length; i++){`

contracts/Saferoot.sol
- Line 249: `for (uint256 index = 0; index < _safeguardKeys.length; ++index){`
- Line 321: `for (uint256 index; index < _ercEntries.length; ++index){`
- Line 392: `for (uint256 index = 0; index < _ercEditEntries.length; ++index){`

Gas Consumption Benchmark Tests:

The original and the optimized gas cost of the updateSupportedContracts function were compared to measure its gas efficiency:

Original: 162236

```
>>> registry.updateSupportedContracts([usdt,usdc,token721a,token721b,weth,token1155a],
[True,True,True,True,True,True], {'from': deployer})
Transaction sent: 0xb7e3a1e578d93b4b50a5ba21cc5b55b67c118bc55c8a6ee4e3195a4a92ff6577
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 4
  ContractRegistry.updateSupportedContracts confirmed   Block: 17740120   Gas used: 162236 (2.41%)

<Transaction '0xb7e3a1e578d93b4b50a5ba21cc5b55b67c118bc55c8a6ee4e3195a4a92ff6577'>
```

Optimized: 161833

```
>>> registry.updateSupportedContracts([usdt,usdc,token721a,token721b,weth,token1155a],
[True,True,True,True,True,True], {'from': deployer})
Transaction sent: 0xb7e3a1e578d93b4b50a5ba21cc5b55b67c118bc55c8a6ee4e3195a4a92ff6577
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 4
  ContractRegistry.updateSupportedContracts confirmed   Block: 17740097   Gas used: 161833 (2.41%)

<Transaction '0xb7e3a1e578d93b4b50a5ba21cc5b55b67c118bc55c8a6ee4e3195a4a92ff6577'>
```

Difference: 403

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

It is recommended to cache array lengths outside of loops, as long the size is not changed during the loop.

It is recommended to use the unchecked ++i operation instead of i++ to increment the values of the uint variable inside the loop. It is noted that using unchecked operations requires particular caution to avoid overflows, and their use may impair code readability.

The following code is an example implementation of the above recommendations:

Listing 11: For Loop Optimization

```
        uint256 length = _contracts.length;
        for (uint256 i; i < length;) {
            supportedContracts[_contracts[i]] = _statuses[i];
            unchecked { ++i; }
        }
```

Remediation Plan:

SOLVED: The Staging Labs team solved the issue in commit 0914a6b by applying the above recommendations.

4.12 (HAL-12) UNNECESSARY VALIDATION - INFORMATIONAL (0.0)

Description:

It was identified that the addSafeguard function in the Saferoot contract employs unnecessary validation to check if the currentKey is already used or not. However, because the currentKey is increased every time a safeguard is added, this condition is always true.

Code Location:

Listing 12: contracts/Saferoot.sol (Lines 338,349,360)

```
335     bytes32 key = encodeKey(currentKey, incomingSafeguard.
 ↳ tokenType);
336     if (incomingSafeguard.tokenType == TokenType.ERC20) {
337         ERC20SafeguardInfo storage safeguard = erc20Safeguards[key
 ↳ ];
338         if (safeguard.contractAddress == address(0)) {
339             safeguard.contractAddress = incomingSafeguard
340                 .contractAddress;
341             safeguard.amount = incomingSafeguard.amount;
342             emit ERC20SafeguardAdded(key);
343             unchecked {
344                 ++currentKey;
345             }
346         }
347     } else if (incomingSafeguard.tokenType == TokenType.ERC721) {
348         ERC721SafeguardInfo storage safeguard = erc721Safeguards[
 ↳ key];
349         if (safeguard.contractAddress == address(0)) {
350             safeguard.contractAddress = incomingSafeguard
351                 .contractAddress;
352             safeguard.tokenId = incomingSafeguard.tokenId;
353             emit ERC721SafeguardAdded(key);
354             unchecked {
355                 ++currentKey;
356             }
357         }
```

```
358     } else if (incomingSafeguard.tokenType == TokenType.ERC1155) {
359         ERC1155SafeguardInfo storage safeguard = erc1155Safeguards
 ↳ [key];
360         if (safeguard.contractAddress == address(0)) {
361             safeguard.contractAddress = incomingSafeguard
362                 .contractAddress;
363             safeguard.tokenId = incomingSafeguard.tokenId;
364             safeguard.amount = incomingSafeguard.amount;
365             emit ERC1155SafeguardAdded(key);
366             unchecked {
367                 ++currentKey;
368             }
369         }
370     }
```

BVSS:

AO:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider removing the unnecessary checks from the addSafeguard function to save gas.

Remediation Plan:

SOLVED: The Staging Labs team solved the issue in commit bc0e505 by removing the unnecessary checks from the addSafeguard function.

4.13 (HAL-13) NOT ALL EVM COMPATIBLE CHAIN SUPPORTS SOLIDITY 0.8.20 - INFORMATIONAL (0.8)

Description:

It was identified that the contracts are using Solidity version 0.8.20 with its default Shanghai EVM version. The Shanghai fork introduced the PUSH0 opcode and was only supported on Mainnet, Goerli and Sepolia at the time of the assessment.

BVSS:

AO:A/AC:L/AX:H/C:N/I:N/A:N/D:L/Y:N/R:N/S:U (0.8)

Recommendation:

Consider changing the EVM version from Shanghai to a widely supported version.

Remediation Plan:

ACKNOWLEDGED: The Staging Labs team acknowledged this finding. The Staging Labs team will adjust the EVM version when deploying to new ecosystems.

MANUAL TESTING

In the manual testing phase, the following scenarios were simulated. The scenarios listed below were selected based on the severity of the vulnerabilities Halborn was testing the program for.

5.1 ACCESS CONTROL AND ROLE MANAGEMENT

Description:

Proper access control on privileged functions was tested. All functions were reviewed to ensure that no sensitive functionality was left unprivileged.

Results:

- Only the configured user with the USER_ROLE can add or edit safeguards, modify the backup wallet address or disable the registry in the Saferoot contract. This role is configured during initialization, and it is not possible to change it or assign it to any other users afterward.
- Only the configured service account with the SERVICE_ROLE can initiate safeguards. This role is configured during initialization, and it is not possible to change it or assign it to any other users afterward.
- Only the contract's deployer can edit the supported contract list in the ContractRegistry contract.
- Anyone can use the SaferootFactory contract to create Saferoot contracts.

5.2 CONTRACT FUNCTIONALITY

Description:

It was tested that the smart contract functionalities operate as intended.

Results:

- The encodeKey, decodeKeyTokenType and decodeKeyID functions used in the Saferoot contract are working as intended.
- After initiating a safeguard by the service, the associated tokens are transferred from the user to the backup address. The Saferoot contract cannot transfer the assets to any other address.
- If the registry is disabled, the user no longer able to enable it.
- It is not possible to change the registry address.
- It is possible to add multiple safeguards for the same asset.

5.3 DEPLOYMENT

Description:

It was tested that the deployments of the Saferoot contracts with the SaferootFactory contract operate as intended.

Results:

- All contracts were deployed successfully, and the properties of the Saferoot contracts were configured properly.
- The Saferoot contracts were deployed and initialized in the same transaction, preventing any malicious user to front run the initialize transaction.

AUTOMATED TESTING

6.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance the coverage of certain areas of the smart contracts in scope. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified the smart contracts in the repository and was able to compile them correctly into their ABIs and binary format, Slither was run against the contracts. This tool can statically verify mathematical relationships between Solidity variables to detect invalid or inconsistent usage of the contracts' APIs across the entire code-base.

Results:

contracts/Saferoot.sol

Slither results for Saferoot.sol	
Finding	**Impact**
Saferoot._transfer20(bytes32,address,uint256) (contracts/Saferoot.sol#146-172) uses a dangerous strict equality: - balance == 0 || allowance == 0 || _transferAmount == 0 (contracts/Saferoot.sol#158)	Medium
Saferoot.addSafeguard(Saferoot.SafeguardEntry[]).index (contracts/Saferoot.sol#321) is a local variable never initialized	Medium
Saferoot.initialize(address,address,address,address,bool)._backup (contracts/Saferoot.sol#123) lacks a zero-check on : - backup = _backup (contracts/Saferoot.sol#128)	Low
Saferoot.initialize(address,address,address,address,bool)._user (contracts/Saferoot.sol#122) lacks a zero-check on : - user = _user (contracts/Saferoot.sol#127)	Low
Saferoot.initialize(address,address,address,address,bool)._contract Registry (contracts/Saferoot.sol#124) lacks a zero-check on : - contractRegistry = _contractRegistry (contracts/Saferoot.sol#130)	Low
Saferoot.initialize(address,address,address,address,bool)._service (contracts/Saferoot.sol#121) lacks a zero-check on : - service = _service (contracts/Saferoot.sol#129)	Low

Finding	Impact
Saferoot.setBackupWallet(address)._backup (contracts/Saferoot.sol#442) lacks a zero-check on : - backup = _backup (contracts/Saferoot.sol#443)	Low
Saferoot._transfer20(bytes32,address,uint256) (contracts/Saferoot.sol#146-172) has external calls inside a loop: balance = IERC20(_contractAddress).balanceOf(user) (contracts/Saferoot.sol#155)	Low
Saferoot._transfer721(bytes32,address,uint256) (contracts/Saferoot.sol#180-197) has external calls inside a loop: (token.getApproved(_tokenId) != address(this) && ! token.isApprovedForAll(owner,address(this))) || owner != user (contracts/Saferoot.sol#188-189)	Low
Saferoot._transfer1155(bytes32,address,uint256,uint256) (contracts/Saferoot.sol#206-237) has external calls inside a loop: amount = IERC1155(_contractAddress).balanceOf(user,_tokenId) (contracts/Saferoot.sol#213)	Low
Saferoot._transfer1155(bytes32,address,uint256,uint256) (contracts/Saferoot.sol#206-237) has external calls inside a loop: IERC1155(_contractAddress).safeTransferFrom(user,backup,_tokenId,transferAmount,) (contracts/Saferoot.sol#228-234)	Low
Saferoot._transfer721(bytes32,address,uint256) (contracts/Saferoot.sol#180-197) has external calls inside a loop: owner = token.ownerOf(_tokenId) (contracts/Saferoot.sol#186)	Low
Saferoot._transfer20(bytes32,address,uint256) (contracts/Saferoot.sol#146-172) has external calls inside a loop: allowance = IERC20(_contractAddress).allowance(user,address(this)) (contracts/Saferoot.sol#151-154)	Low
Saferoot.addSafeguard(Saferoot.SafeguardEntry[]) (contracts/Saferoot.sol#314-374) has external calls inside a loop: registryEnabled && ! registry.isContractSupported(incomingSafeguard.contractAddress) (contracts/Saferoot.sol#329-330)	Low
Saferoot._transfer1155(bytes32,address,uint256,uint256) (contracts/Saferoot.sol#206-237) has external calls inside a loop: ! IERC1155(_contractAddress).isApprovedForAll(user,address(this)) || amount == 0 (contracts/Saferoot.sol#217-218)	Low

Finding	Impact
Saferoot._transfer721(bytes32,address,uint256) (contracts/Saferoot.sol#180-197) has external calls inside a loop: token.safeTransferFrom(user,backup,_tokenId) (contracts/Saferoot.sol#194)	Low
Reentrancy in Saferoot.initiateSafeguard(bytes32[]) (contracts/Saferoot.sol#243-298): External calls: - _transfer1155(key,contractAddress_scope_3,safeguard_scope_2.tokenId,safeguard_scope_2.amount) (contracts/Saferoot.sol#284-289) - IERC1155(_contractAddress).safeTransferFrom(user,backup,_tokenId,transferAmount,) (contracts/Saferoot.sol#228-234) Event emitted after the call(s): - SafeguardInitiated(key) (contracts/Saferoot.sol#291)	Low
Reentrancy in Saferoot.initiateSafeguard(bytes32[]) (contracts/Saferoot.sol#243-298): External calls: - _transfer721(key,contractAddress_scope_1,safeguard_scope_0.tokenId) (contracts/Saferoot.sol#271) - token.safeTransferFrom(user,backup,_tokenId) (contracts/Saferoot.sol#194) Event emitted after the call(s): - SafeguardInitiated(key) (contracts/Saferoot.sol#272)	Low
End of table for Saferoot.sol	

contracts/SaferootFactory.sol

Slither results for SaferootFactory.sol	
Finding	**Impact**
Reentrancy in SaferootFactory.createSaferoot(address,address,address,bool) (contracts/SaferootFactory.sol#40-63): External calls: - Saferoot(clone).initialize(_service,msg.sender,_backup,_contractRegistry,_registryEnabled) (contracts/SaferootFactory.sol#47-53) Event emitted after the call(s): - SaferootDeployed(clone,_service,msg.sender,_backup,_contractRegistry,_registryEnabled) (contracts/SaferootFactory.sol#54-61)	Low
End of table for SaferootFactory.sol	

contracts/ContractRegistry.sol

Slither did not identify any vulnerabilities in the contract.

contracts/ErrorReporter.sol

Slither did not identify any vulnerabilities in the contract.

The findings obtained as a result of the Slither scan were reviewed. The medium-risk and reentrancy vulnerabilities were not included in the report because they were determined false positives.

6.2 AUTOMATED SECURITY SCAN

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and to identify low-hanging fruits on the targets for this engagement. Among the tools used was MythX, a security analysis service for Ethereum smart contracts. MythX performed a scan on the smart contracts and sent the compiled results to the analyzers in order to locate any vulnerabilities.
Results

contracts/Saferoot.sol

Report for contracts/Saferoot.sol
https://dashboard.mythx.io/#/console/analyses/a0d425ee-8727-4e41-ad5c-4ed29937cd52
https://dashboard.mythx.io/#/console/analyses/2ef676c1-e21e-4107-8e18-324bf3af8d11

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
249	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
250	(SWC-110) Assert Violation	Unknown	Out of bounds array access
321	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
322	(SWC-110) Assert Violation	Unknown	Out of bounds array access
344	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
355	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
367	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
392	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
393	(SWC-110) Assert Violation	Unknown	Out of bounds array access
458	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "-" discovered

contracts/SaferootFactory.sol

Report for contracts/SaferootFactory.sol
https://dashboard.mythx.io/#/console/analyses/a0d425ee-8727-4e41-ad5c-4ed29937cd52

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.

contracts/ErrorReporter.sol

Report for contracts/ErrorReporter.sol
https://dashboard.mythx.io/#/console/analyses/ca2d798d-b77d-4bd2-bae2-9c2d6f8a02ab

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
10	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
15	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
21	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
27	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.
32	(SWC-108) State Variable Default Visibility	Low	State variable visibility is not set.

contracts/ContractRegistry.sol

Report for contracts/ContractRegistry.sol
https://dashboard.mythx.io/#/console/analyses/0bd7e487-8a3c-4acc-9051-e6bf19d3fe49

Line	SWC Title	Severity	Short Description
2	(SWC-103) Floating Pragma	Low	A floating pragma is set.
52	(SWC-101) Integer Overflow and Underflow	Unknown	Arithmetic operation "++" discovered
53	(SWC-110) Assert Violation	Unknown	Out of bounds array access

The findings obtained as a result of the MythX scan were examined, and they were not included in the report because they were determined false positives.

THANK YOU FOR CHOOSING

// HALBORN