



# Pocket Network – Wrapped Pocket

Smart Contract Security  
Assessment

Prepared by: Halborn

Date of Engagement: August 21st, 2023 – August 23rd, 2023

Visit: [Halborn.com](https://Halborn.com)

DOCUMENT REVISION HISTORY	4
CONTACTS	4
1 EXECUTIVE OVERVIEW	5
1.1 INTRODUCTION	6
1.2 ASSESSMENT SUMMARY	6
1.3 TEST APPROACH & METHODOLOGY	6
2 RISK METHODOLOGY	8
2.1 EXPLOITABILITY	9
2.2 IMPACT	10
2.3 SEVERITY COEFFICIENT	12
2.4 SCOPE	14
3 ASSESSMENT SUMMARY & FINDINGS OVERVIEW	15
4 FINDINGS & TECH DETAILS	16
4.1 (HAL-01) SAFE SIGNER NUMBER THRESHOLD NOT ENFORCED - LOW(3.3)	18
Description	18
Code Location	18
BVSS	18
Recommendation	18
Remediation Plan	19
4.2 (HAL-02) THE WRAPPEDPOCKET CONTRACT CANNOT BE PAUSED - LOW(3.3)	20
Description	20
BVSS	22
Recommendation	22

	Remediation Plan	22
4.3	(HAL-03) ZERO ADDRESS CHECK MISSING - LOW(2.5)	23
	Description	23
	Code Location	23
	BVSS	23
	Recommendation	23
	Remediation Plan	23
4.4	(HAL-04) INVALID CONFIGURATION ALLOWED - LOW(2.0)	24
	Description	24
	Code Location	24
	BVSS	24
	Recommendation	25
	Remediation Plan	25
4.5	(HAL-05) INEFFICIENT FOR LOOPS - INFORMATIONAL(0.0)	26
	Description	26
	Code Location	26
	Proof of Concept	27
	BVSS	28
	Recommendation	28
	Remediation Plan	28
5	MANUAL TESTING	29
5.1	MintController	30
5.2	WrappedPocket	31
6	AUTOMATED TESTING	33
6.1	STATIC ANALYSIS REPORT	34
	Description	34

Results	34
6.2 AUTOMATED SECURITY SCAN	35
Description	35
Results	35

## DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE
0.1	Document Creation	08/21/2023
0.2	Draft Version	08/23/2023
0.3	Draft Review	08/24/2023
1.0	Remediation Plan	08/29/2023
1.1	Remediation Plan Review	08/29/2023

## CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	<a href="mailto:Rob.Behnke@halborn.com">Rob.Behnke@halborn.com</a>
Steven Walbroehl	Halborn	<a href="mailto:Steven.Walbroehl@halborn.com">Steven.Walbroehl@halborn.com</a>
Gabi Urrutia	Halborn	<a href="mailto:Gabi.Urrutia@halborn.com">Gabi.Urrutia@halborn.com</a>



# EXECUTIVE OVERVIEW



## 1.1 INTRODUCTION

The Wrapped Pocket contracts by Pocket Network include all the necessary functionality to enable a wrapped version of the POKT token to be seamlessly bridged from the Pocket Network Blockchain.

Pocket Network engaged Halborn to conduct a security assessment on their smart contracts beginning on August 21st, 2023 and ending on August 23rd, 2023. The security assessment was scoped to the smart contracts provided in the [wpokt](#) GitHub repository. Commit hashes and further details can be found in the Scope section of this report.

## 1.2 ASSESSMENT SUMMARY

Halborn was provided 3 days for the engagement and assigned 1 full-time security engineer to review the security of the smart contracts in scope. The engineer is 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 security risks, that were successfully addressed by Pocket Network.

## 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](#)).



## 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)	1
	Specific (AO:S)	0.2
Attack Cost (AC)	Low (AC:L)	1
	Medium (AC:M)	0.67
	High (AC:H)	0.33
Attack Complexity (AX)	Low (AX:L)	1
	Medium (AX:M)	0.67
	High (AX:H)	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)	1
	Partial (R:P)	0.5
	Full (R:F)	0.25
Scope ( $s$ )	Changed (S:C)	1.25
	Unchanged (S:U)	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

### Code repositories:

#### 1. Social

- Repository: [wpokt](#)
- Commit IDs:
  - Initial: [445b7a6](#)
  - Remediation Plan: [2d28b92](#)
- Smart contracts in scope:
  1. MintController ([src/MintController.sol](#))
  2. WrappedPocket ([src/WrappedPocket.sol](#))

### Out-of-scope

- Third-party libraries and dependencies.
- Economic attacks.

### 3. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	4	1



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) SAFE SIGNER NUMBER THRESHOLD NOT ENFORCED	Low (3.3)	SOLVED - 08/27/2023
(HAL-02) THE WRAPPEDPOCKET CONTRACT CANNOT BE PAUSED	Low (3.3)	SOLVED - 08/27/2023
(HAL-03) ZERO ADDRESS CHECK MISSING	Low (2.5)	SOLVED - 08/27/2023
(HAL-04) INVALID CONFIGURATION ALLOWED	Low (2.0)	SOLVED - 08/27/2023
(HAL-05) INEFFICIENT FOR LOOPS	Informational (0.0)	SOLVED - 08/27/2023



# FINDINGS & TECH DETAILS



## 4.1 (HAL-01) SAFE SIGNER NUMBER THRESHOLD NOT ENFORCED – LOW (3.3)

### Description:

The `MintController` contract controls the minting of WPAKT tokens. One of the prerequisites for minting tokens is providing a set of correct validator signatures. Unless the identities of all validators are confirmed and enough validators signed the operation, tokens cannot be minted.

The contract, however, does not require the deployer to set the threshold on contract creation, nor does it verify if the threshold is sufficiently high on threshold update. This increases the chances of taking over the network in case validator keys are compromised.

### Code Location:

Listing 1: `src/MintController.sol` (Line 140)

```
139 function setSignerThreshold(uint256 signatureRatio) external
    ↳ onlyAdmin {
140     if (signatureRatio > validatorCount || signatureRatio == 0) {
141         revert InvalidSignatureRatio();
142     }
143     signerThreshold = signatureRatio;
144     emit SignerThresholdSet(signatureRatio);
145 }
```

### BVSS:

A0:A/AC:L/AX:H/C:N/I:C/A:N/D:N/Y:N/R:N/S:U (3.3)

### Recommendation:

Consider requiring the `signerThreshold` to be a majority or a super majority.

## Remediation Plan:





**SOLVED:** The Pocket Network team solved this issue in commit [2d28b92](#) by reverting the function call if `signatureRatio` is lower than 50% of the current `validatorCount`.

## 4.2 (HAL-02) THE WRAPPEDPOCKET CONTRACT CANNOT BE PAUSED - LOW (3.3)

### Description:

The `WrappedPocket` contract inherits from the `Pausable` contract by OpenZeppelin. This contract implements a set of modifiers, methods, and state variables that allow to pause some or all contract operations. No functions in the `WrappedPocket` contract, however, leverage those features, which means in case of a vulnerability or an attack the contract cannot be paused.

```

✓  ! burnAndBridge ( complex: 4 state: ☐ )
  # amount
  [🔗] poktAddress
>  ! pause ( complex: 2 state: ☒ )
>  ! unpause ( complex: 2 state: ☒ )
✓  ! mint ( complex: 10 state: ☒ )
  ✓ { } modifiers ... (1)
     @ onlyRole
    [🔗] to
    # amount
    # nonce
    # currentNonce
✓  ! batchMint ( complex: 12 state: ☒ )
  ✓ { } modifiers ... (1)
     @ onlyRole
    [🔗] to[]
    # amount[]
    # nonce[]
    #  adjustedAmounts[] memory
    # i
✓  ! setFee ( complex: 10 state: ☒ )
  ✓ { } modifiers ... (1)
     @ onlyRole

```

BVSS:

A0:A/AC:L/AX:H/C:N/I:N/A:C/D:N/Y:N/R:N/S:U (3.3)

Recommendation:

Decorate mission-critical contract functions with the `whenNotPaused` modifier.

Remediation Plan:

**SOLVED:** The Pocket Network team solved this issue in commit [2d28b92](#) by adding the `whenNotPaused` modifier to the `mint` and `burnAndBridge` functions.

## 4.3 (HAL-03) ZERO ADDRESS CHECK MISSING - LOW (2.5)

### Description:

The `MintController` contract constructor requires the deployer to provide the address of the `WrappedPocket` contract. This address is not verified to not be the zero address. Please note, the `WrappedPocket` contract address cannot be updated after the `MintController` contract is deployed.

### Code Location:

Listing 2: `contracts/contracts/VerifierList.sol` (Lines 58-61)

```
47 constructor(address _wPckt) EIP712("MintController", "1") {  
48     wPckt = IWPKT(_wPckt);  
49 }
```

### BVSS:

A0:A/AC:L/AX:L/C:N/I:L/A:N/D:N/Y:N/R:N/S:U (2.5)

### Recommendation:

Consider validating the user supplied `address` is not equal to the zero address.

### Remediation Plan:

**SOLVED:** The Pocket Network team solved this issue in commit [2d28b92](#) by reverting contract creation if the user-supplied `_wPckt` is the zero address.



## 4.4 (HAL-04) INVALID CONFIGURATION ALLOWED - LOW (2.0)

### Description:

The `MintController` contract controls the minting of WPAKT tokens. The Pocket Network. team imposed certain restrictions on the amount of tokens that can be minted as a function of time. Once this limit is exhausted, it is reset at a pace defined by the `mintPerSecond` contract variable, up to the hard limit defined by another contract variable, `maxMintLimit`.

Those parameters are predefined, but they can be updated by the administrator with the `setMintCooldown` function. This function does not ensure, however, the new `maxMinLimit` is greater than the new `mintPerSecond`. This is a property the contract must hold at all times for WPAKT minting to be possible.

### Code Location:

#### Listing 3: `src/MintController.sol`

```
152 function setMintCooldown(uint256 newLimit, uint256
    ↳ newMintPerSecond) external onlyAdmin {
153     maxMintLimit = newLimit;
154     mintPerSecond = newMintPerSecond;
155
156     emit MintCooldownSet(newLimit, newMintPerSecond);
157 }
```

### BVSS:

A0:S/AC:L/AX:L/C:N/I:C/A:N/D:N/Y:N/R:N/S:U (2.0)

#### Recommendation:

In the `setMintCooldown` function, consider requiring the new `maxMinLimit` to always be greater than `mintPerSecond`.

#### Remediation Plan:

**SOLVED:** The Pocket Network team solved this issue in commit [2d28b92](#) by reverting the function call if `maxMinLimit` is less than `mintPerSecond`.

## 4.5 (HAL-05) INEFFICIENT FOR LOOPS - INFORMATIONAL (0.0)

### Description:

For loops are used in several functions in the contracts in scope. It was identified that all loops in the contract can be optimized to make the contracts more gas efficient:

- when iterating over an array, cache the array length outside of loops to avoid reading from memory/storage and pushing the length to stack in every iteration,
- when iterating from the 0 index, do not initialize the index to 0 because all numerical values in Solidity are initialized to zero by default,
- when incrementing the index, use preincrementation instead of postincrementation, and do it in an `unchecked` block.

### Code Location:

#### Listing 4: src/MintController.sol

```
201 for (uint256 i = 0; i < _signatures.length; i++) {
202     currentSigner = ECDSA.recover(digest, _signatures[i]);
203     if (validators[currentSigner] && currentSigner > lastSigner) {
204         validSignatures++;
205         lastSigner = currentSigner;
206     }
207 }
```

#### Listing 5: src/WrappedPocket.sol

```
118 if (feeFlag == true) {
119     uint256[] memory adjustedAmounts = _batchCollectFee(amount);
120
121     for (uint256 i = 0; i < to.length; i++) {
122         _mintBatch(to[i], adjustedAmounts[i], nonce[i]);
123     }
```

```

124 } else {
125     for (uint256 i = 0; i < to.length; i++) {
126         _mintBatch(to[i], amount[i], nonce[i]);
127     }
128 }

```

#### Listing 6: src/WrappedPocket.sol

```

200 for (uint256 i = 0; i < amounts.length; i++) {
201     if (amounts[i] % BASIS_POINTS != 0) {
202         revert FeeBasisDust();
203     }
204     fee = (amounts[i] * feeBasis) / BASIS_POINTS;
205     adjustedAmounts[i] = amounts[i] - fee;
206     totalFee += fee;
207 }

```

#### Proof of Concept:

#### Listing 7

```

1 function testGasConsumptionForUnoptimized() public pure {
2     uint256[5] memory array;
3
4     for (uint256 i = 0; i < array.length; i++) { }
5 }
6
7 function testGasConsumptionForOptimized() public pure {
8     uint256[5] memory array;
9     uint256 arrayLength = array.length;
10
11     for (uint256 i; i < arrayLength;) {
12         unchecked {
13             ++i;
14         }
15     }
16 }

```

BVSS:

A0:A/AC:L/AX:L/C:N/I:N/A:N/D:N/Y:N/R:N/S:U (0.0)

Recommendation:

Consider using the following pattern for `for` loops:

#### Listing 8

```
1 uint256 arrayLength = array.length;
2
3 for (uint256 i; i < arrayLength;) {
4     . . .
5     unchecked {
6         ++i
7     }
8 }
```

Remediation Plan:

**SOLVED:** The Pocket Network team solved this issue in commit [2d28b92](#) by applying the recommended gas optimization techniques.



# 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 smart contracts for.

## 5.1 MintController

For the `MintController` contract, the following scenarios were tested:

Scenario	Expected Result	Result
add a duplicate validator	revert	PASS
add a validator as an anonymous user	revert	PASS
add a zero address validator	revert	PASS
mint WPOKT without providing the signature array	revert	PASS
mint WPOKT with an array of invalid signatures	revert	PASS
mint WPOKT with an array of valid signatures below the required threshold	revert	PASS
mint WPOKT with duplicate valid signatures	revert	PASS
mint WPOKT over the allowed limit	revert	PASS
mint WPOKT over when limit is zero	revert	PASS
set mint cooldown as an anonymous user	revert	PASS
set signer threshold as an anonymous user	revert	PASS
set signer threshold over the signer count	revert	PASS
set signer threshold to zero	revert	PASS

```

Running 15 tests for test/MintController.t.sol:MintControllerTest
[PASS] test_addDuplicateValidatorRevert(address) (runs: 10000, μ: 63429, ~: 63429)
[PASS] test_addValidatorAnonymousRevert(address,address) (runs: 10000, μ: 19125, ~: 19125)
[PASS] test_addZeroAddressValidatorRevert(address) (runs: 10000, μ: 14709, ~: 14709)
[PASS] test_mintWrappedPocketEmptySignaturesRevert() (gas: 211)
[PASS] test_mintWrappedPocketIncorrectSignaturesRevert() (gas: 187)
[PASS] test_mintWrappedPocketMintableDuplicateSigners() (gas: 166)
[PASS] test_mintWrappedPocketMintableOverMaxLimitRevert() (gas: 186)
[PASS] test_mintWrappedPocketMintableSignersBelowThresholdRevert() (gas: 232)
[PASS] test_mintWrappedPocketMintableZeroRevert() (gas: 253)
[PASS] test_removeValidatorAnonymousRevert(address,address) (runs: 10000, μ: 19137, ~: 19137)
[PASS] test_removeValidatorZeroAddressRevert() (gas: 208)
[PASS] test_setMintCooldownAnonymousRevert() (gas: 210)
[PASS] test_setSignerThresholdAnonymousRevert() (gas: 189)
[PASS] test_setSignerThresholdOverSignerCountRevert() (gas: 210)
[PASS] test_setSignerThresholdZeroRevert() (gas: 232)
Test result: ok. 15 passed; 0 failed; 0 skipped; finished in 1.34s
Ran 1 test suites: 15 tests passed, 0 failed, 0 skipped (15 total tests)

```

## 5.2 WrappedPocket

For the `WrappedPocket` contract, the following scenarios were tested:

Scenario	Expected Result	Result
batch mint as an anonymous user	revert	PASS
batch mint with stale nonces	revert	PASS
burn tokens	revert	PASS
mint as an anonymous user	revert	PASS
mint with a stale nonce	revert	PASS
set fee as an anonymous user	revert	PASS
set the zero address as the fee collector	revert	PASS
set the fee over MaxFeeBasis	revert	PASS



```
Running 15 tests for test/MintController.t.sol:MintControllerTest
[PASS] test_addDuplicateValidatorRevert(address) (runs: 10000,  $\mu$ : 63429,  $\sim$ : 63429)
[PASS] test_addValidatorAnonymousRevert(address,address) (runs: 10000,  $\mu$ : 19125,  $\sim$ : 19125)
[PASS] test_addZeroAddressValidatorRevert(address) (runs: 10000,  $\mu$ : 14709,  $\sim$ : 14709)
[PASS] test_mintWrappedPocketEmptySignaturesRevert() (gas: 211)
[PASS] test_mintWrappedPocketIncorrectSignaturesRevert() (gas: 187)
[PASS] test_mintWrappedPocketMintableDuplicateSigners() (gas: 166)
[PASS] test_mintWrappedPocketMintableOverMaxLimitRevert() (gas: 186)
[PASS] test_mintWrappedPocketMintableSignersBelowThresholdRevert() (gas: 232)
[PASS] test_mintWrappedPocketMintableZeroRevert() (gas: 253)
[PASS] test_removeValidatorAnonymousRevert(address,address) (runs: 10000,  $\mu$ : 19137,  $\sim$ : 19137)
[PASS] test_removeValidatorZeroAddressRevert() (gas: 208)
[PASS] test_setMintCooldownAnonymousRevert() (gas: 210)
[PASS] test_setSignerThresholdAnonymousRevert() (gas: 189)
[PASS] test_setSignerThresholdOverSignerCountRevert() (gas: 210)
[PASS] test_setSignerThresholdZeroRevert() (gas: 232)
Test result: ok. 15 passed; 0 failed; 0 skipped; finished in 1.34s
Ran 1 test suites: 15 tests passed, 0 failed, 0 skipped (15 total tests)
```



# 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.

The security team assessed all findings identified by the Slither software, however, findings with severity **Information** and **Optimization** are not included in the below results for the sake of report readability.

### Results:

No issues were detected by Slither.

## 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:

No issues were detected by MythX.



THANK YOU FOR CHOOSING

// HALBORN

