// HALBORN

Moonwell Finance - Token Sale & Comptroller Updates

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **April 5th, 2022 - June 17th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	04/05/2022	Ataberk Yavuzer
0.2	Document Edits	04/10/2022	Ataberk Yavuzer
0.3	Draft Review	04/11/2022	Gabi Urrutia
1.0	Remediation Plan	06/17/2022	Gokberk Gulgun
1.1	Remediation Plan Review	06/17/2022	Gabi Urrutia

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
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Moonwell Finance engaged Halborn to conduct a security audit on their smart contracts beginning on April 5th, 2022 and ending on June 17th, 2022 . The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided a week for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended.
- Identify potential security issues with the smart contracts.

In summary, Halborn identified some security risks that were addressed by the team.

1.3 TEST APPROACH & METHODOLOGY

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

- 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.
- Dynamic Analysis (ganache-cli, brownie, hardhat).

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating

a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

1. Moonwell Finance Token Sale Contracts
    (a) Repository: Token Sale
    (b) Commit ID: 726dcbaef18670d344fa5621c23c4db0e403583a
    (c) New PR : PR
    (d) New Commit : New Commit
2. Out-of-Scope
    (a) test/*.sol

Out-of-scope: External contract, libraries and financial related attacks.

FIX Commit ID : 762cdc4cd9a8d09f29765f9e143b25af0ebe9720

TAG : artemis-v1

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	3	3	12

LIKELIHOOD

IMPACT

	(HAL-02)			
(HAL-05) (HAL-06)		(HAL-01) (HAL-03)		
	(HAL-04)			
(HAL-07) (HAL-08) (HAL-09) (HAL-10) (HAL-11) (HAL-13) (HAL-14) (HAL-15) (HAL-16) (HAL-17) (HAL-18)	(HAL-12)			

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
(HAL-01) OLD TOKENS ARE NOT RECOVERABLE WHEN THE NEW TOKEN IS SET	Medium	SOLVED - 04/14/2022
(HAL-02) EXPIRED TOKENS ARE NOT CONSIDERED IN THE VOTING POWER	Medium	SOLVED - 06/17/2022
(HAL-03) OWNER CAN RESET ALLOCATIONS - DELEGATIONS	Medium	RISK ACCEPTED
(HAL-04) MISSING ZERO ADDRESS CHECK	Low	SOLVED - 04/14/2022
(HAL-05) MISSING EVENTS FOR OWNER ONLY FUNCTIONS THAT CHANGE CRITICAL PARAMETERS	Low	SOLVED - 04/14/2022
(HAL-06) UNNEEDED INITIALIZATION OF UINT256 VARIABLES TO 0	Informational	SOLVED - 04/14/2022
(HAL-07) USING ++I CONSUMES LESS GAS THAN I+=1 IN LOOPS	Informational	SOLVED - 04/14/2022
(HAL-08) CACHING THE LENGTH IN THE FOR LOOPS	Informational	SOLVED - 04/14/2022
(HAL-09) REVERT STRING SIZE OPTIMIZATION	Informational	ACKNOWLEDGED
(HAL-10) MISSING CHECKS FOR NON-ZERO TRANSFER VALUE CALLS	Informational	SOLVED - 04/14/2022
(HAL-11) BLOCK WITH GAS LIMIT	Informational	ACKNOWLEDGED
(HAL-12) EXPERIMENTAL KEYWORD USAGE	Informational	SOLVED - 04/14/2022
(HAL-13) UPGRADE AT LEAST PRAGMA 0.8.10	Informational	SOLVED - 04/14/2022
(HAL-14) OPEN TODOS	Informational	SOLVED - 06/17/2022
(HAL-15) CHANGING FUNCTION VISIBILITY FROM PUBLIC TO EXTERNAL CAN SAVE GAS	Informational	SOLVED - 06/17/2022
(HAL-16) DIRECT USAGE OF ECRECOVER ALLOWS SIGNATURE MALLEABILITY	Informational	SOLVED - 06/17/2022

(HAL-17) MISSING EVENTS	Informational	SOLVED - 06/17/2022
(HAL-18) OPTIMIZE UNSIGNED INTEGER COMPARISON	Informational	SOLVED - 06/17/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) OLD TOKENS ARE NOT RECOVERABLE WHEN THE NEW TOKEN IS SET - MEDIUM

Description:

The privileged address can set the token. However, when the token is set to another address, the old tokens cannot be retrieved using the contract.

Scenario:

- Admin sets allocation via setAllocations function.
- After the time, the user claims allocations.
- Admin sets new token on the contract.
- Old tokens are not recoverable by the contract. The withdraw function only takes an amount of argument.

Code Location:

Listing 1: TokenSaleDistributor.sol

```
197     function setTokenAddress(address newTokenAddress) external
 ↳ adminOnly {
198         require(tokenAddress == address(0), "Address already set")
 ↳ ;
199         tokenAddress = newTokenAddress;
200     }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Consider adding the following function for the accidental **setTokenAddress** function call.

Listing 2

```
    function withdraw(address token, uint amount) external
↳ adminOnly {
        IERC20(token).safeTransfer(admin, amount);
    }
```

Remediation Plan:

SOLVED: The Moonwell Team solved this issue by implementing the withdraw function in the TokenSaleDistributor.sol contract.

Commit ID: Commit ID

3.2 (HAL-02) EXPIRED TOKENS ARE NOT CONSIDERED IN THE VOTING POWER - MEDIUM

Description:

During the code review, It has been observed that expired tokens are not included in the voting power. Although the expired tokens are not used in the contract depends on the protocol behavior that can directly affect voting. The voting power should be carefully designed with expired tokens.

Code Location:

Listing 3: TokenSaleDistributor.sol

```
195     function totalVotingPower(address user) public view returns (
 ↳ uint) {
196         uint totalAllocatedToUser = totalAllocated(user);
197         uint totalClaimedByUser = totalClaimed(user);
198         return totalAllocatedToUser - totalClaimedByUser;
199     }
```

Risk Level:

Likelihood - 2
Impact - 4

Recommendation:

It is recommended to check all token features in the voting power. If the expired tokens are not used, the related code section should be removed from the code base.

Remediation Plan:

SOLVED: The Moonwell Team solved this issue by deleting expired tokens from the code base.

Commit ID: Commit ID

3.3 (HAL-03) OWNER CAN RESET ALLOCATIONS - DELEGATIONS - MEDIUM

Description:

During the code review, It has been noticed that the owner can delete all delegations from any account.

Code Location:

Listing 4: TokenSaleDistributor.sol

```
195     function resetAllocationsByUser(address[] memory recipients)
 ↳ external adminOnly {
196         uint length = recipients.length;
197         for (uint i; i < length; ++i) {
198             uint votingPower = totalVotingPower(recipients[i]);
199             _moveDelegates(delegates[recipients[i]], address(0),
 ↳ votingPower);
200             delete allocations[recipients[i]];
201         }
202     }
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Consider, not deleting, voting power and allocations users with the **resetAllocationsByUser** function, however If It is not possible, It is recommended to carefully manage the owner account's private key to avoid any potential risks of being hacked.

In terms of short-term and long-term goal:

- Time-lock with reasonable latency, e.g. 48 hours, for awareness on privileged operations.
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key.

Remediation Plan:

RISK ACCEPTED: The Moonwell Team states that they will do this if they found a critical vulnerability after deployment, and they needed to claw back all tokens to prevent them from being stolen. In that case, they will deploy a new contract and re-add the claims as well. On the other hand, The Moonwell Team commented the feature on the code with the following PR.

Commit ID: Commit ID

3.4 (HAL-04) MISSING ZERO ADDRESS CHECKS - LOW

Description:

The Token Sale contract has address fields in multiple functions. These functions are missing address validations. Each address should be validated and checked to be non-zero. This is also considered a best practice.

During testing, it has been found that some of these inputs are not protected against using the address(0) as the target address.

Code Location:

Listing 5: TokenSaleDistributorProxy.sol (Lines 18,37)

```
18      function setPendingAdmin(address newAdmin) public adminOnly {
19          pendingAdmin = newAdmin;
20      }
21
22      /**
23       * Accept admin transfer from the current admin to the new.
24       */
25      function acceptPendingAdmin() public {
26          require(msg.sender == pendingAdmin && pendingAdmin !=
↳ address(0), "Caller must be the pending admin");
27
28          admin = pendingAdmin;
29          pendingAdmin = address(0);
30      }
31
32      /**
33       * Request a new implementation to be set for the contract.
34       *
35       * @param newImplementation New contract implementation
↳ contract address
36       */
37      function setPendingImplementation(address newImplementation)
↳ public adminOnly {
```

```
38          pendingImplementation = newImplementation;
39      }
40
```

Recommendation:

It is recommended to validate that each address input is non-zero.

Remediation Plan:

SOLVED: The Moonwell Team solved this issue by implementing zero address checks.

Commit ID: Commit ID

3.5 (HAL-05) MISSING EVENTS FOR ADMIN ONLY FUNCTIONS THAT CHANGE CRITICAL PARAMETERS - LOW

Description:

Admin-only functions that change critical parameters should emit events. Events allow you to capture changed parameters so that tools/interfaces off-chain can register those changes.

Code Location:

Listing 6: TokenSaleDistributor.sol

```
1     function setTokenAddress(address newTokenAddress) external
↳ adminOnly {
2         require(tokenAddress == address(0), "Address already set")
↳ ;
3         tokenAddress = newTokenAddress;
4     }
```

Listing 7: TokenSaleDistributorProxy.sol

```
1     function setPendingImplementation(address newImplementation)
↳ public adminOnly {
2         pendingImplementation = newImplementation;
3     }
4     /**
5      * Accept pending implementation change
6      */
7     function acceptPendingImplementation() public {
8         require(msg.sender == pendingImplementation &&
↳ pendingImplementation != address(0), "Only the pending
↳ implementation contract can call this");
9
10        implementation = pendingImplementation;
11        pendingImplementation = address(0);
12    }
```

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

Add events to all admin functions that change critical parameters.

Remediation plan:

SOLVED: The Moonwell team solved the issue by adding events following the above recommendation.

Commit ID: Commit ID

3.6 (HAL-06) UNNEEDED INITIALIZATION OF UINT256 VARIABLES TO 0 - LOW

Description:

Since `i` is an `uint256`, it is already initialized to 0. `uint256 i = 0` reassigns the 0 to `i` which wastes gas.

Code Location:

TokenSaleDistributor.sol

- Line 157: `for (uint i = 0; i < recipients.length; i += 1){`

Risk Level:

Likelihood - 1
Impact - 3

Recommendation:

It is recommended not to initialize uint256 variables to 0 to save gas. For example, use instead:
`for (uint256 i; i < recipients.length; ++i){`.

Remediation Plan:

SOLVED: The Moonwell Team solved this issue by removing initialization.

Commit ID: Commit ID

3.7 (HAL-07) USING ++I CONSUMES LESS GAS THAN I+=1 IN LOOPS - INFORMATIONAL

Description:

In the loop below, the variable `i` is incremented using `i+=`. It is known that, in loops, using `++i` costs less gas per iteration than `i+=1`.

Code Location:

Listing 8

```
1   TokenSaleDistributor.sol::31 => for (uint i; i < allocations[msg
↳ .sender].length; i += 1) {
2   TokenSaleDistributor.sol::59 => for (uint i; i < allocations[
↳ recipient].length; i += 1) {
3   TokenSaleDistributor.sol::71 => for (uint i; i < allocations[
↳ recipient].length; i += 1) {
4   TokenSaleDistributor.sol::83 => for (uint i; i < allocations[
↳ recipient].length; i += 1) {
5   TokenSaleDistributor.sol::95 => for (uint i; i < allocations[
↳ recipient].length; i += 1) {
6   TokenSaleDistributor.sol::107 => for (uint i; i < allocations[
↳ recipient].length; i += 1) {
7   TokenSaleDistributor.sol::157 => for (uint i = 0; i < recipients
↳ .length; i += 1) {
8   TokenSaleDistributor.sol::180 => for (uint i; i < recipients.
↳ length; i += 1) {
```

Risk Level:

Likelihood - 1
Impact - 1

Proof of Concept:

For example, based on the following test contract:

Listing 9: Test.sol

```
1 //SPDX-License-Identifier: MIT
2 pragma solidity 0.8.9;
3
4 contract test {
5     function postiincrement(uint256 iterations) public {
6         for (uint256 i = 0; i < iterations; i+=1) {
7         }
8     }
9     function preiincrement(uint256 iterations) public {
10        for (uint256 i = 0; i < iterations; ++i) {
11        }
12    }
13 }
```

We can see the difference in gas costs:

```
>>> test_contract.postiincrement(1)
Transaction sent: 0x1ecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 44
  test.postiincrement confirmed   Block: 13622335   Gas used: 21620 (0.32%)

<Transaction '0x1ecede6b109b707786d3685bd71dd9f22dc389957653036ca04c4cd2e72c5e0b'>
>>> test_contract.preiincrement(1)
Transaction sent: 0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 45
  test.preiincrement confirmed   Block: 13622336   Gas used: 21593 (0.32%)

<Transaction '0x205f09a4d2268de4c1a40f35bb2ec2847bf2ab8d584909b42c71a022b047614a'>
>>> test_contract.postiincrement(10)
Transaction sent: 0x98c04430526a59balcf947c114b62666a4417165947d31bf300cd6ae68328033
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 46
  test.postiincrement confirmed   Block: 13622337   Gas used: 22673 (0.34%)

<Transaction '0x98c04430526a59balcf947c114b62666a4417165947d31bf300cd6ae68328033'>
>>> test_contract.preiincrement(10)
Transaction sent: 0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20e1de05
  Gas price: 0.0 gwei   Gas limit: 6721975   Nonce: 47
  test.preiincrement confirmed   Block: 13622338   Gas used: 22601 (0.34%)

<Transaction '0xf060d04714eff8482a828342414d5a20be9958c822d42860e7992aba20e1de05'>
```

Risk Level:

Likelihood - 1

Impact - 1

Recommendation:

It is recommended to use `++i` instead of `i+=1` to increment the value of a `uint` variable within a loop. This does not just apply to the iterator variable. It also applies to increments made within the loop code block.

Remediation Plan:

SOLVED: The Moonwell Team solved this issue by following the above recommendation.

Commit ID: Commit ID

3.8 (HAL-08) CACHING THE LENGTH IN THE FOR LOOPS - INFORMATIONAL

Description:

The solidity compiler will always read the length of the array during each iteration.

- If it is a storage array, this is an additional sload operation (100 additional extra gas (EIP-2929) for each iteration except the first)
- If it is a memory array, this is an extra mload operation (3 additional gas for each iteration except the first),
- If it is a calldata array, it is an extra calldataload operation (3 additional gas for each iteration except the first).

Code Location:

Listing 10

```
1   TokenSaleDistributor.sol::31 => for (uint i; i < allocations[msg
↳ .sender].length; i += 1) {
2   TokenSaleDistributor.sol::59 => for (uint i; i < allocations[
↳ recipient].length; i += 1) {
3   TokenSaleDistributor.sol::71 => for (uint i; i < allocations[
↳ recipient].length; i += 1) {
4   TokenSaleDistributor.sol::83 => for (uint i; i < allocations[
↳ recipient].length; i += 1) {
5   TokenSaleDistributor.sol::95 => for (uint i; i < allocations[
↳ recipient].length; i += 1) {
6   TokenSaleDistributor.sol::107 => for (uint i; i < allocations[
↳ recipient].length; i += 1) {
7   TokenSaleDistributor.sol::157 => for (uint i = 0; i < recipients
↳ .length; i += 1) {
8   TokenSaleDistributor.sol::180 => for (uint i; i < recipients.
↳ length; i += 1) {
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Caching expensive state variables would prevent re-reading from storage.

Listing 11

```
uint length = arr.length;
for (uint i = 0; i < length; i++) {
    // do something that doesn't change arr.length
}
```

Remediation Plan:

SOLVED: The Moonwell Team solved this issue by caching arrays.

Commit ID: Commit ID

3.9 (HAL-09) REVERT STRING SIZE OPTIMIZATION - INFORMATIONAL

Description:

Shortening the revert strings to fit within 32 bytes will decrease deployment time gas and decrease runtime gas when the revert condition is met.

Revert strings that are longer than 32 bytes require at least one additional `mstore`, along with additional overhead to calculate memory offset, etc.

Code Location:

Listing 12: TokenSaleDistributor.sol

```
206     function becomeImplementation(TokenSaleDistributorProxy proxy)
 ↳  external {
207         require(msg.sender == proxy.admin(), "Only proxy admin can
 ↳  change the implementation");
208         proxy.acceptPendingImplementation();
209     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Shorten the revert strings to fit within 32 bytes. That will affect gas optimization.

Remediation Plan:

ACKNOWLEDGED: The Moonwell Team acknowledged this issue. This is a long-tail gas estimation issue, and the Moonwell Team would rather have clean error messages than fix gas on a revert the user could have detected.

3.10 (HAL-10) MISSING CHECKS FOR NON-ZERO TRANSFER VALUE CALLS - INFORMATIONAL

Description:

Checking for non-zero transfer values can prevent an external call to save gas.

Code Location:

TokenSaleDistributor.sol

- Line 189: `function withdraw(uint amount)external adminOnly {`

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Check if the transfer **amount > 0** before executing the transfer.

Remediation Plan:

SOLVED: The Moonwell Team solved this issue by implementing amount check.

Commit ID: Commit ID

3.11 (HAL-11) BLOCK WITH GAS LIMIT - INFORMATIONAL

Description:

There are several loops in the contract that can eventually grow large enough to make future operations of the contract cost too much gas to fit in one block.

Code Location:

TokenSaleDistributor.sol

Listing 13

```
1   TokenSaleDistributor.sol::31 => for (uint i; i < allocations[msg
↳ .sender].length; i += 1) {
2   TokenSaleDistributor.sol::59 => for (uint i; i < allocations[
↳ recipient].length; i += 1) {
3   TokenSaleDistributor.sol::71 => for (uint i; i < allocations[
↳ recipient].length; i += 1) {
4   TokenSaleDistributor.sol::83 => for (uint i; i < allocations[
↳ recipient].length; i += 1) {
5   TokenSaleDistributor.sol::95 => for (uint i; i < allocations[
↳ recipient].length; i += 1) {
6   TokenSaleDistributor.sol::107 => for (uint i; i < allocations[
↳ recipient].length; i += 1) {
7   TokenSaleDistributor.sol::157 => for (uint i = 0; i < recipients
↳ .length; i += 1) {
8   TokenSaleDistributor.sol::180 => for (uint i; i < recipients.
↳ length; i += 1) {
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider introducing a reasonable upper limit based on block gas limits.

Remediation Plan:

ACKNOWLEDGED: The Moonwell Team acknowledged this issue.

3.12 (HAL-12) EXPERIMENTAL KEYWORD USAGE - INFORMATIONAL

Description:

ABIEncoderV2 is enabled and using experimental features could be dangerous in live deployments. The experimental ABI encoder does not correctly handle non-integer values less than 32 bytes. This applies to bytesNN types, bool, enum and other types when they are part of an array or a struct and encoded directly from storage. This means these storage references have to be used directly inside abi.encode(...) as arguments in external function calls or in event data without prior assignment to a local variable. The types **bytesNN** and **bool** will result in corrupted data, while enum could cause an invalid revert.

Code Location:

Listing 14: pragma experimental ABIEncoderV2

```
1 TokenSaleDistributor.sol - Line#4
```

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

When possible, do not use experimental features in the final live deployment.

Remediation Plan

SOLVED: The Moonwell Team solved this issue by deleting experimental check.

Commit ID: Commit ID

3.13 (HAL-13) UPGRADE AT LEAST PRAGMA 0.8.10 - INFORMATIONAL

Description:

Additional gas optimizations and security checks are available for free when using newer versions of the compiler and optimizer.

- Safemath by default since 0.8.0 (maybe more efficient on gas than library-based safemath.)
- Low level inliner : As of 0.8.2, leads to cheaper gas runtime. Especially relevant when the contract has small functions. For example, OpenZeppelin libraries typically have a lot of small helper functions and if they are not inlined, they cost an extra 20-40 gas because of the extra 2 jump instructions and additional stack operations needed for function calls.
- Optimizer improvements in packed structs: Before 0.8.3, storing packed structs, in some cases, used an additional storage read operation. After EIP-2929, if the slot was already cold, this means unnecessary stacking operations and extra deployment time costs. However, if the slot was already warm, this means additional cost of 100 gas along with the same unnecessary stack operations and extra deploy time costs.
- Custom errors from 0.8.4, leads to cheaper deploy time cost and run time cost. Note: the run time cost is only relevant when the revert condition is met. In short, replace revert strings with custom errors.

Code Location:

Listing 15: TokenSaleDistributor.sol

```
1 pragma solidity 0.6.12;
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Upgrade **pragma** to at least 0.8.10.

Remediation Plan:

SOLVED: The Moonwell Team solved this issue by updating pragma to **0.8.10**.

Commit ID: Commit ID

3.14 (HAL-14) OPEN TODOS - INFORMATIONAL

Description:

Open TODOs can point to architecture or programming issues that still need to be resolved.

Code Location:

Listing 16: TokenSaleDistributor.sol

```
1     /// @notice EIP-20 token name for this token
2     // TODO(lunar): validate this name.
3     string public constant name = "vWELL";
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider resolving the TODO before deploying.

Remediation Plan:

SOLVED: The Moonwell Team solved this issue by deleting TODOs check.

Commit ID: Commit ID

3.15 (HAL-15) CHANGING FUNCTION VISIBILITY FROM PUBLIC TO EXTERNAL CAN SAVE GAS - INFORMATIONAL

Description:

There is a function declared as public that is never called internally within the contract. It is best practice to mark such functions as external instead, as this saves gas (especially in the case where the function takes arguments, as external functions can read arguments directly from calldata instead of having to allocate memory).

Code Location:

Listing 17: TokenSaleDistributor.sol

```
1     function delegate(address delegatee) public {
2         return _delegate(msg.sender, delegatee);
3     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

All the public functions in the contract are not called internally, so access can be changed to external to reduce gas.

Remediation Plan:

SOLVED: The Moonwell Team solved this issue by marking function as an external.

Commit ID: Commit ID

3.16 (HAL-16) DIRECT USAGE OF ECRECOVER ALLOWS SIGNATURE MALLEABILITY - INFORMATIONAL

Description:

delegateBySig calls the Solidity ecrecover function directly to verify the given signatures. However, the ecrecover EVM opcode allows malleable (non-unique) signatures and thus is susceptible to replay attacks.

Although a replay attack seems not possible here since the nonce is increased each time, ensuring the signatures are not malleable is considered a best practice.

Code Location:

Listing 18: TokenSaleDistributor.sol

```
    function delegateBySig(address delegatee, uint nonce, uint
↳ expiry, uint8 v, bytes32 r, bytes32 s) public {
        bytes32 domainSeparator = keccak256(abi.encode(
↳ DOMAIN_TYPEHASH, keccak256(bytes(name)), getChainId(), address(
↳ this)));
        bytes32 structHash = keccak256(abi.encode(
↳ DELEGATION_TYPEHASH, delegatee, nonce, expiry));
        bytes32 digest = keccak256(abi.encodePacked("\x19\x01",
↳ domainSeparator, structHash));
        address signatory = ecrecover(digest, v, r, s);
        require(signatory != address(0), "VestingWell::
↳ delegateBySig: invalid signature");
        require(nonce == nonces[signatory]++, "VestingWell::
↳ delegateBySig: invalid nonce");
        require(block.timestamp <= expiry, "VestingWell::
↳ delegateBySig: signature expired");
        return _delegate(signatory, delegatee);
    }

```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Use the recover function from OpenZeppelin's ECDSA library for signature verification.

Remediation Plan:

SOLVED: The Moonwell Team solved this issue by changing implementation with ECDSA library.

Commit ID: Commit ID

3.17 (HAL-17) MISSING EVENTS - INFORMATIONAL

Description:

Owner/governor only functions that change critical parameters should emit events. Events allow capturing the changed parameters so that off-chain tools/interfaces can register such changes with timelocks that allow users to evaluate them and consider if they would like to engage/exit based on how they perceive the changes as affecting the trustworthiness of the protocol or profitability of the implemented financial services. The alternative of directly querying on-chain contract state for such changes is not considered practical for most users/usages.

Code Location:

Listing 19: TokenSaleDistributor.sol

```
1      function setVotingEnabled(bool enabled) external adminOnly {
2          votingEnabled = enabled;
3      }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Add events to all owner/governor functions that change critical parameters.

Remediation Plan:

SOLVED: The Moonwell Team solved this issue by adding event on the related function.

Commit ID: Commit ID

3.18 (HAL-18) OPTIMIZE UNSIGNED INTEGER COMPARISON - INFORMATIONAL

Description:

The check != 0 costs less gas compared to > 0 for unsigned integers in require statements with the optimizer enabled. While it may seem that > 0 is cheaper than !=0, this is only true without the optimizer enabled and outside a require statement. If the optimizer is enabled at 10k, and It is in a require statement, that would be more gas efficient.

Code Location:

Listing 20: TokenSaleDistributor.sol

```
1      function setVotingEnabled(bool enabled) external adminOnly {
2          votingEnabled = enabled;
3      }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Change > 0 comparison with != 0.

Remediation Plan:

SOLVED: The Moonwell Team solved this issue by changing “> 0” with “!=0” on the related functions.

Commit ID: Commit ID

AUTOMATED TESTING

4.1 STATIC ANALYSIS REPORT

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped contract. Among the tools used was Slither, a Solidity static analysis framework. After Halborn verified all the contracts in the repository and was able to compile them correctly into their ABI and binary formats. 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:

```
TokenSaleDistributorProxy.fallback() (tokensale/TokenSaleDistributorProxy.sol#51-63) uses delegatecall to a input-controlled function id
        - (success) = implementation.delegatecall(msg.data) (tokensale/TokenSaleDistributorProxy.sol#52)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#controlled-delegatecall

TokenSaleDistributorProxy.setPendingAdmin(address).newAdmin (tokensale/TokenSaleDistributorProxy.sol#18) lacks a zero-check on :
                - pendingAdmin = newAdmin (tokensale/TokenSaleDistributorProxy.sol#19)
TokenSaleDistributorProxy.setPendingImplementation(address).newImplementation (tokensale/TokenSaleDistributorProxy.sol#37) lacks a zero-check on :
                - pendingImplementation = newImplementation (tokensale/TokenSaleDistributorProxy.sol#38)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

TokenSaleDistributorProxy.fallback() (tokensale/TokenSaleDistributorProxy.sol#51-63) uses assembly
        - INLINE ASM (tokensale/TokenSaleDistributorProxy.sol#54-62)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage

Low level call in TokenSaleDistributorProxy.fallback() (tokensale/TokenSaleDistributorProxy.sol#51-63):
        - (success) = implementation.delegatecall(msg.data) (tokensale/TokenSaleDistributorProxy.sol#52)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

setPendingAdmin(address) should be declared external:
        - TokenSaleDistributorProxy.setPendingAdmin(address) (tokensale/TokenSaleDistributorProxy.sol#18-20)
acceptPendingAdmin() should be declared external:
        - TokenSaleDistributorProxy.acceptPendingAdmin() (tokensale/TokenSaleDistributorProxy.sol#25-30)
setPendingImplementation(address) should be declared external:
        - TokenSaleDistributorProxy.setPendingImplementation(address) (tokensale/TokenSaleDistributorProxy.sol#37-39)
acceptPendingImplementation() should be declared external:
        - TokenSaleDistributorProxy.acceptPendingImplementation() (tokensale/TokenSaleDistributorProxy.sol#44-49)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external

SafeMath.add(uint256,uint256) (tokensale/SafeMath.sol#29-34) is never used and should be removed
SafeMath.add(uint256,uint256,string) (tokensale/SafeMath.sol#44-49) is never used and should be removed
SafeMath.div(uint256,uint256) (tokensale/SafeMath.sol#133-135) is never used and should be removed
SafeMath.div(uint256,uint256,string) (tokensale/SafeMath.sol#148-155) is never used and should be removed
SafeMath.mod(uint256,uint256) (tokensale/SafeMath.sol#168-170) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (tokensale/SafeMath.sol#183-186) is never used and should be removed
SafeMath.mul(uint256,uint256) (tokensale/SafeMath.sol#86-98) is never used and should be removed
SafeMath.mul(uint256,uint256,string) (tokensale/SafeMath.sol#108-120) is never used and should be removed
SafeMath.sub(uint256,uint256) (tokensale/SafeMath.sol#59-61) is never used and should be removed
SafeMath.sub(uint256,uint256,string) (tokensale/SafeMath.sol#71-76) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

TokenSaleDistributorProxyStorage.admin (tokensale/TokenSaleDistributorProxyStorage.sol#7) should be constant
TokenSaleDistributorProxyStorage.implementation (tokensale/TokenSaleDistributorProxyStorage.sol#13) should be constant
TokenSaleDistributorProxyStorage.pendingAdmin (tokensale/TokenSaleDistributorProxyStorage.sol#10) should be constant
TokenSaleDistributorProxyStorage.pendingImplementation (tokensale/TokenSaleDistributorProxyStorage.sol#16) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
```

```
TokenSaleDistributor._vested(TokenSaleDistributorStorage.Allocation) (tokensale/TokenSaleDistributor.sol#222-247) performs a multiplication on the result of a division:
        -elapsedPeriods = elapsed.div(monthlyVestingInterval) (tokensale/TokenSaleDistributor.sol#239)
        -monthlyVestedAmount = postCliffAmount.div(allocation.vestingDuration) (tokensale/TokenSaleDistributor.sol#244)
        -initialAmount.add(monthlyVestedAmount.mul(elapsedPeriods)) (tokensale/TokenSaleDistributor.sol#246)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply

TokenSaleDistributor._claim(TokenSaleDistributorStorage.Allocation) (tokensale/TokenSaleDistributor.sol#285-295) uses a dangerous strict equality:
        - claimable == 0 (tokensale/TokenSaleDistributor.sol#287)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities

TokenSaleDistributor.totalVested(address).i (tokensale/TokenSaleDistributor.sol#71) is a local variable never initialized
TokenSaleDistributor.totalExpired(address).total (tokensale/TokenSaleDistributor.sol#106) is a local variable never initialized
TokenSaleDistributor.totalAllocated(address).i (tokensale/TokenSaleDistributor.sol#59) is a local variable never initialized
TokenSaleDistributor.totalClaimable(address).i (tokensale/TokenSaleDistributor.sol#95) is a local variable never initialized
TokenSaleDistributor.claim().i (tokensale/TokenSaleDistributor.sol#31) is a local variable never initialized
TokenSaleDistributor.resetAllocationsByUser(address[]).i (tokensale/TokenSaleDistributor.sol#180) is a local variable never initialized
TokenSaleDistributor.totalClaimed(address).i (tokensale/TokenSaleDistributor.sol#83) is a local variable never initialized
TokenSaleDistributor.totalExpired(address).i (tokensale/TokenSaleDistributor.sol#107) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

TokenSaleDistributor.setTokenAddress(address).newTokenAddress (tokensale/TokenSaleDistributor.sol#197) lacks a zero-check on :
                - tokenAddress = newTokenAddress (tokensale/TokenSaleDistributor.sol#199)
TokenSaleDistributorProxy.setPendingAdmin(address).newAdmin (tokensale/TokenSaleDistributorProxy.sol#18) lacks a zero-check on :
                - pendingAdmin = newAdmin (tokensale/TokenSaleDistributorProxy.sol#19)
TokenSaleDistributorProxy.setPendingImplementation(address).newImplementation (tokensale/TokenSaleDistributorProxy.sol#37) lacks a zero-check on :
                - pendingImplementation = newImplementation (tokensale/TokenSaleDistributorProxy.sol#38)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation

TokenSaleDistributor.claim() (tokensale/TokenSaleDistributor.sol#29-38) uses timestamp for comparisons
        Dangerous comparisons:
        - claimed > 0 (tokensale/TokenSaleDistributor.sol#35)
TokenSaleDistributor._vested(TokenSaleDistributorStorage.Allocation) (tokensale/TokenSaleDistributor.sol#222-247) uses timestamp for comparisons
        Dangerous comparisons:
        - block.timestamp < allocation.epoch + allocation.cliff (tokensale/TokenSaleDistributor.sol#223)
        - elapsed >= allocation.vestingDuration (tokensale/TokenSaleDistributor.sol#232)
        - elapsedPeriods >= allocation.vestingDuration (tokensale/TokenSaleDistributor.sol#240)
TokenSaleDistributor._isExpired(TokenSaleDistributorStorage.Allocation) (tokensale/TokenSaleDistributor.sol#252-268) uses timestamp for comparisons
        Dangerous comparisons:
        - allocation.epoch.add(allocation.cliff).add(allocation.vestingDuration).add(allocation.gracePeriod) < block.timestamp (tokensale/TokenSaleDistributor.sol#258-261)
        - allocation.epoch.add(allocation.cliff).add(allocation.vestingDuration.mul(monthlyVestingInterval)).add(allocation.gracePeriod) < block.timestamp (tokensale/TokenSaleDistributor.sol#264-267)
TokenSaleDistributor._claim(TokenSaleDistributorStorage.Allocation) (tokensale/TokenSaleDistributor.sol#285-295) uses timestamp for comparisons
        Dangerous comparisons:
        - claimable == 0 (tokensale/TokenSaleDistributor.sol#287)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp

Address.isContract(address) (tokensale/Address.sol#34-43) uses assembly
        - INLINE ASM (tokensale/Address.sol#41)
Address.verifyCallResult(bool,bytes,string) (tokensale/Address.sol#202-222) uses assembly
        - INLINE ASM (tokensale/Address.sol#214-217)
TokenSaleDistributorProxy.fallback() (tokensale/TokenSaleDistributorProxy.sol#51-63) uses assembly
        - INLINE ASM (tokensale/TokenSaleDistributorProxy.sol#54-62)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
Low level call in Address.sendValue(address,uint256) (tokensale/Address.sol#61-66):
        - (success) = recipient.call{value: amount}() (tokensale/Address.sol#64)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (tokensale/Address.sol#129-140):
        - (success,returndata) = target.call{value: value}(data) (tokensale/Address.sol#138)
Low level call in Address.functionStaticCall(address,bytes,string) (tokensale/Address.sol#158-167):
        - (success,returndata) = target.staticcall(data) (tokensale/Address.sol#165)
Low level call in Address.functionDelegateCall(address,bytes,string) (tokensale/Address.sol#185-194):
        - (success,returndata) = target.delegatecall(data) (tokensale/Address.sol#192)
Low level call in TokenSaleDistributorProxy.fallback() (tokensale/TokenSaleDistributorProxy.sol#51-63):
        - (success) = implementation.delegatecall(msg.data) (tokensale/TokenSaleDistributorProxy.sol#52)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Constant TokenSaleDistributorStorage.monthlyVestingInterval (tokensale/TokenSaleDistributorStorage.sol#11) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

setPendingAdmin(address) should be declared external:
        - TokenSaleDistributorProxy.setPendingAdmin(address) (tokensale/TokenSaleDistributorProxy.sol#18-20)
acceptPendingAdmin() should be declared external:
        - TokenSaleDistributorProxy.acceptPendingAdmin() (tokensale/TokenSaleDistributorProxy.sol#25-30)
setPendingImplementation(address) should be declared external:
        - TokenSaleDistributorProxy.setPendingImplementation(address) (tokensale/TokenSaleDistributorProxy.sol#37-39)
acceptPendingImplementation() should be declared external:
        - TokenSaleDistributorProxy.acceptPendingImplementation() (tokensale/TokenSaleDistributorProxy.sol#44-49)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external

Address.isContract(address) (tokensale/Address.sol#34-43) uses assembly
        - INLINE ASM (tokensale/Address.sol#41)
Address.verifyCallResult(bool,bytes,string) (tokensale/Address.sol#202-222) uses assembly
        - INLINE ASM (tokensale/Address.sol#214-217)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

As a result of the tests carried out with the Slither tool, some results were obtained and these results were reviewed by Halborn. Based on the results reviewed, some vulnerabilities were determined to be false positives and these results were not included in the report. The actual vulnerabilities found by Slither are already included in the report findings.

THANK YOU FOR CHOOSING

// HALBORN