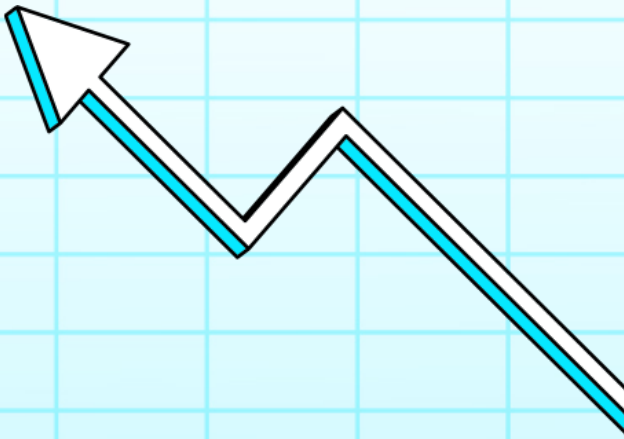
our shielding · Your smart contracts, our shielding · Your smart c

shieldify



Onchain Heroes

Maze of Gains (MoG)

SECURITY REVIEW

Date: 3 February 2026

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and has secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Vyper, Rust, Cairo, Move and Go.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Onchain Heroes - Maze of Gains (MoG)

Onchain Heroes is a fully on-chain, idle RPG with novel on-chain game mechanics.

Maze of Gains (MoG) is an on-chain turn-based roguelike dungeon crawler, playable on desktop and mobile, where every run competes for a shared weekly ETH prize pool.

Learn more about Maze of Gains (MoG): here

4. Risk Classification

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

4.1 Impact

- **High** - results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** - results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** - losses will be limited but bearable - and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** - almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** - still relatively likely, although only conditionally possible
- **Low** - requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review lasted 3 days, with a total of 48 hours dedicated by the Shieldify team.

Overall, the code is well-written. The audit report contributed by identifying two Low-severity issues. They're mainly related to gas safety and upgrade-related signature compatibility.

The Onchain Heroes team has been highly responsive to the Shieldify research team's inquiries and promptly implemented the recommendations.

5.1 Protocol Summary

Project Name	**Onchain Heroes - Maze of Gains (MoG)**
Repository	och-contracts
Type of Project	GameFi
Security Review Timeline	3 days
Review Commit Hash	a70f359ddf58543483e0faed52b8bc7a81c157d5

5.2 Scope

The following smart contracts were in the scope of the security review:

File	**nSLOC**
src/mog/KeyPurchase.sol	101
src/mog/ClaimVault.sol	142
Total	243

6. Findings Summary

The following issues have been identified, sorted by their severity:

- **Low** issues: **2**
- **Info** issues: **4**

ID	**Title**	**Severity**	**Status**
[L-01]	Unbounded Batch Claim Loop May Cause Out-of-Gas Reverts in `claimWeekly()` and `claimJackpot()`	Low	Acknowledged
[L-02]	EIP-712 Domain Version Is Static and May Cause Signature Compatibility Issues After Upgrades	Low	Acknowledged
[I-01]	Missing Signature Expiration Allows Indefinite Validity of Weekly and Jackpot Claims	Info	Acknowledged
[I-02]	The `buyKeys()` Performs External ETH Transfers Without Reentrancy Protection	Info	Acknowledged
[I-03]	Floating and Outdated Pragma	Info	Acknowledged
[I-04]	Suboptimal UUPS Implementation Initialization Pattern	Info	Acknowledged

7. Findings

[L-01] Unbounded Batch Claim Loop May Cause Out-of-Gas Reverts in `claimWeekly()` and `claimJackpot()`

Severity

Low Risk

Description

The `claimWeekly()` and `claimJackpot()` functions process user-provided arrays of claims using an unbounded for-loop. Since the contract does not enforce any maximum length on the claims array, a caller may submit an excessively large batch, causing the transaction to consume more gas than the block limit and revert.

While this primarily affects the caller submitting the oversized claim rather than enabling theft, it can still create a denial-of-service scenario at the user level, where legitimate users are unable to successfully claim rewards in a single transaction if they provide large batches.

Location of Affected Code

File: src/mog/ClaimVault.sol#L141

```
function claimWeekly(WeeklyClaim[] calldata claims) external nonReentrant
    whenNotPaused {
    uint256 length = claims.length;
    for (uint256 i; i < length; ) {
        // code
        unchecked { ++i; }
    }
}
```

File: src/mog/ClaimVault.sol#L180

```
function claimJackpot(JackpotClaim[] calldata claims) external
   nonReentrant whenNotPaused {
    uint256 length = claims.length;
    for (uint256 i; i < length; ) {
        // code
        unchecked { ++i; }
    }
}
```

Impact

If a user attempts to claim too many weekly or jackpot rewards in one call, the transaction may run out of gas and revert, preventing successful reward withdrawal.

Recommendation

Introduce a reasonable upper bound on the number of claims processed per call.

Team Response

Acknowledged.

[L-02] EIP-712 Domain Version Is Static and May Cause Signature Compatibility Issues After Upgrades

Severity

Low Risk

Description

The `ClaimVault` contract overrides `_domainNameAndVersion()` to return a fixed domain name and version string. Since the contract is deployed in an upgradeable (UUPS proxy) setup, future upgrades may modify claim validation logic or signed message semantics. However, because the EIP-712 domain version remains permanently set to "1" and `_domainNameAndVersionMayChange()` is not overridden, signatures issued before an upgrade may still remain valid under the same domain separator, potentially leading to unintended acceptance of stale authorizations.

Additionally, if the protocol intends to change the domain version in later upgrades, the current static configuration may create compatibility issues or require careful signer coordination.

Location of Affected Code

File: src/mog/ClaimVault.sol#L270-L273

```
function _domainNameAndVersion() internal pure override returns (string
    memory name, string memory version) {
     name = "ClaimVault";
     version = "1";
}
```

Impact

In an upgradeable environment, keeping a permanently static EIP-712 domain version can result in older signatures remaining valid across contract upgrades, increasing the risk of signature confusion if claim rules evolve. It may also complicate future upgrades that require domain separation changes, forcing manual signer resets or migration overhead.

Recommendation

Consider overriding `_domainNameAndVersionMayChange()` to return true if the protocol anticipates changing the domain name or version in future upgrades.

Team Response

Acknowledged.

[I-01] Missing Signature Expiration Allows Indefinite Validity of Weekly and Jackpot Claims

Severity

Informational Risk

Description

The `claimWeekly()` and `claimJackpot()` mechanisms rely on backend-signed EIP-712 messages to authorize reward payouts. However, the signed claim data does not include any deadline or expiration timestamp, meaning that once a signature is issued, it remains valid forever until either the signer is rotated or the contract is upgraded.

Location of Affected Code

File: src/mog/ClaimVault.sol#L40-L46

```
bytes32 private constant WEEKLY_CLAIM_TYPEHASH =
    keccak256("WeeklyClaim(address claimer,uint256 week,uint256 amount)")
        ;

bytes32 private constant JACKPOT_CLAIM_TYPEHASH =
    keccak256("JackpotClaim(address claimer,uint256 nonce,uint256 amount)
        ");
```

Impact

Signatures issued for weekly or jackpot rewards remain usable indefinitely, allowing claims to be executed far in the future even if the protocol's reward distribution rules or intended claim windows have changed.

Recommendation

Include an explicit deadline field in both `WeeklyClaim` and `JackpotClaim` signed structs, and require the contract to reject signatures past their expiry.

Team Response

Acknowledged.

[I-02] The `buyKeys()` Performs External ETH Transfers Without Reentrancy Protection

Severity

Informational Risk

Description

The `buyKeys()` function forwards ETH directly to the feeRecipient and claimsRecipient addresses using `SafeTransferLib.safeTransferETH()`. These transfers are external calls that may execute fallback logic if either recipient is a contract. Since `buyKeys()` does not use a `nonReentrant` guard, a malicious or misconfigured recipient contract could attempt to reenter the `KeyPurchase` contract during execution.

Although the current implementation does not modify internal state after the transfers and therefore does not present an immediate fund-stealing vector, adding reentrancy protection is considered a best-practice defensive measure for future upgrades or extensions.

Location of Affected Code

File: src/mog/KeyPurchase.sol#L102-L121

```
function buyKeys(uint256 quantity) external payable whenNotPaused {
    // code
    if (feeAmount > 0) {
        SafeTransferLib.safeTransferETH(feeRecipient, feeAmount);
    }
    if (claimsAmount > 0) {
        SafeTransferLib.safeTransferETH(claimsRecipient, claimsAmount);
    }
}
```

Impact

In the current version, reentrancy does not appear to directly compromise funds because no critical state updates occur after the transfers. However, the lack of a `nonReentrant` modifier reduces defensive safety and could introduce risk if future upgrades add additional state-dependent logic to `buyKeys()` or if recipients behave unexpectedly.

Recommendation

Add `ReentrancyGuard` and apply the `nonReentrant` modifier to buyKeys to follow industry-standard best practices when performing external ETH transfers, ensuring stronger protection against potential reentrancy in future contract iterations.

Team Response

Acknowledged.

[I-03] Floating and Outdated Pragma

Severity

Informational Risk

Description

Locking the pragma helps ensure that the contracts do not accidentally get deployed using an older version of the Solidity compiler affected by vulnerabilities. The contract allowed floating or unlocked pragma to be used, i.e., `>= 0.8.24` . This allows the contracts to be compiled with all the solidity compiler versions above the limit specified.

Location of Affected Code

File: src/mog/KeyPurchase.sol

File: src/mog/ClaimVault.sol

Impact

If the smart contract gets compiled and deployed with an older or too recent version of the Solidity compiler, there's a chance that it may get compromised due to the bugs present in the older versions or unidentified exploits in the new versions. Incompatibility issues may also arise if the contract code does not support features in other compiler versions, therefore, breaking the logic. The likelihood of exploitation is low.

Recommendation

Keep the compiler versions consistent in all the smart contract files. Do not allow floating pragmas anywhere. It is suggested to use the latest pragma version.

Team Response

Acknowledged.

[I-04] Suboptimal UUPS Implementation Initialization Pattern

Severity

Informational Risk

Description

Both contracts use the UUPS upgradeable pattern and currently initialize state in the constructor, which does protect against front-running attacks by setting the owner. However, this approach is not the recommended best practice and is less gas-efficient compared to using the standardised `_disableInitializers()` pattern that is already available in the codebase.

Location of Affected Code

File: src/mog/KeyPurchase.sol

File: src/mog/ClaimVault.sol

Recommendation

We recommend using `_disableInitializers()` in the constructors.

Team Response

Acknowledged.

shieldify

Thank you!