Stonewall
Smart Contract Security

iLemonati NFT Security Review

Introduction

A time-boxed security review of the **iLemonati NFT Minting Contract** was conducted by **Stonewall**, with a focus on the security aspects of the smart contract implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource and expertise bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs and on-chain monitoring are strongly recommended.

About Stonewall

Stonewall is an independent smart contract security firm delivering immovable protection for Web3 protocols. Our team brings deep expertise in DeFi security, having reviewed DEXs, yield farming protocols, gaming contracts, and complex financial systems.

About iLemonati

iLemonati is an NFT collection with a tiered minting system:

- **OG Phase**: Free mint for OG-listed wallets (1 per wallet, max 250 supply)
- **WL Phase**: Discounted mint at 450 MON for whitelisted wallets (1 per wallet, max 350 supply)
- **Public Phase**: Open mint at 650 MON for anyone

Privileged Roles & Actors

Role	Description
Owner	Can set admins, change funds wallet, set NFT contract address
Admin	Can modify OG/WL lists, change prices, adjust supplies, set start time, admin mint
fundsWallet	Receives all mint proceeds on withdrawal

Observations

- Contract uses OpenZeppelin's battle-tested Ownable and ReentrancyGuard
- Implements refund mechanism for overpayment
- Separates owner and admin privileges
- Tracks per-wallet mints for OG and WL phases

Risk Classification

	High Impact	Medium Impact	Low Impact
High Likelihood	Critical	High	Medium
Medium Likelihood	High	Medium	Low
Low Likelihood	Medium	Low	Low

Impact

- **High**: Leads to significant loss of user funds, protocol insolvency, or complete protocol failure
- **Medium**: Leads to partial loss of funds, temporary denial of service, or governance manipulation
- **Low**: Leads to minor issues, inconvenience, or suboptimal behavior

Likelihood

- **High**: Attack is easy to perform and likely to happen
- **Medium**: Attack requires specific conditions but is feasible
- **Low**: Attack requires significant effort, resources, or unlikely conditions

---

Security Assessment Summary

Review Details	
Protocol Name	iLemonati NFT
Repository	Private
Commit	Source provided directly
Review Date	January 2026
Methods	Manual review, static analysis
Network	Monad Mainnet (Chain ID: 143)

Deployed Contract Addresses

Contract	Address
iLemonati NFT	`0x026DA7F37262d801fa7afb62D0371705F4191DA2`
iLemonatiMint	`0x64E50DFeAd9F53A024F4D942DE8eb317f2ce1c79`
fundsWallet	`0xC7f34Ca5E0eA1033a40A50345F3C444020933174`

Project Links

Platform	Link
Mint	mint.lemonad.one
Twitter	@LeMONAD_Factory
Telegram	LeMONAD_Factory
Discord	Join Discord

Scope

Contract	SLOC
`iLemonatiMint.sol`	~95

Findings Summary

ID	Title	Severity	Status
[M-01]	User-controlled phase parameter allows bypassing OG/WL restrictions	Medium	Open
[M-02]	No maximum mint quantity enforced for public phase	Medium	Open
[L-01]	Default startTime set to far future date	Low	Open
[L-02]	Centralized admin control without timelock	Low	Open
[L-03]	No total supply cap enforcement in contract	Low	Open
[L-04]	Admin can mint unlimited NFTs bypassing all limits	Low	Open

Fixed Issues (Resolved During Audit)

ID	Title	Severity	Status
[H-01]	Missing reentrancy guard on mint function	High	**Fixed**
[M-03]	ETH not refunded on overpayment	Medium	**Fixed**
[L-05]	Missing zero quantity check	Low	**Fixed**

Fixed Findings

[H-01] Missing ReentrancyGuard on Mint Function - **FIXED**

Severity: High

Previous Issue: The mint function could be reentered via ERC721 callbacks during batch minting.

Resolution: Added `nonReentrant` modifier from OpenZeppelin.

[M-03] ETH Not Refunded on Overpayment - **FIXED**

Severity: Medium

Previous Issue: Users paying more than the required mint price would lose the excess ETH.

Resolution: Added refund logic: `if (msg.value > cost) payable(msg.sender).transfer(msg.value - cost);`

[L-05] Missing Zero Quantity Check - **FIXED**

Severity: Low

Previous Issue: Users could call mint with quantity=0, wasting gas.

Resolution: Added `require(quantity > 0, "Invalid quantity");`

Findings

[M-01] User-controlled phase parameter allows bypassing OG/WL restrictions

Severity: Medium

Impact: Medium - Users can bypass OG/WL verification by specifying an invalid phase number.

Likelihood: High - Easy to exploit, requires only passing a different parameter value.

Location: `iLemonatiMint.sol` - `mint()` function

Description:

The `mint()` function accepts a user-controlled `phase` parameter that determines which minting path is taken. A user who is not on the OG or WL lists can simply pass any value other than 1 or 2 to access the public mint path, even if public minting is not intended to be open:

```
function mint(uint256 quantity, uint256 phase) external payable nonReentrant {
    require(block.timestamp >= startTime, "Mint not started");
    require(quantity > 0, "Quantity must be > 0");

    uint256 cost = 0;

    if (phase == 1) {
        require(og[msg.sender], "Not OG");
        // OG logic
    } else if (phase == 2) {
```

```
        require(wl[msg.sender], "Not WL");
        // WL logic
    } else {
        cost = publicPrice * quantity;  // Anyone can reach this branch
    }
    // ...
}
```

Attack Scenario:

1. Protocol intends to run OG phase only (phase 1)
2. Non-OG user calls `mint(5, 99)` with 3250 MON
3. User bypasses OG requirement and mints at public price
4. This can happen before public mint is intended to open

Recommendation:

Remove user control over phase selection. Use admin-controlled phase state:

```
uint256 public currentPhase; // 0 = closed, 1 = OG, 2 = WL, 3 = public

function setCurrentPhase(uint256 _phase) external onlyAdminOrOwner {
    require(_phase <= 3, "Invalid phase");
    currentPhase = _phase;
}

function mint(uint256 quantity) external payable nonReentrant {
    require(block.timestamp >= startTime, "Mint not started");
    require(quantity > 0, "Quantity must be > 0");
    require(currentPhase > 0, "Minting not active");

    uint256 cost = 0;

    if (currentPhase == 1) {
        require(og[msg.sender], "Not OG");
        require(ogMinted[msg.sender] + quantity <= 1, "OG limit exceeded");
        require(totalOgMinted + quantity <= maxOgSupply, "OG supply exceeded");
        ogMinted[msg.sender] += quantity;
        totalOgMinted += quantity;
    } else if (currentPhase == 2) {
        require(wl[msg.sender], "Not WL");
        require(wlMinted[msg.sender] + quantity <= 1, "WL limit exceeded");
        require(totalWlMinted + quantity <= maxWlSupply, "WL supply exceeded");
        wlMinted[msg.sender] += quantity;
        cost = wlPrice * quantity;
        totalWlMinted += quantity;
    } else if (currentPhase == 3) {
        cost = publicPrice * quantity;
```

```
    }
    // ...
}
```

[M-02] No maximum mint quantity enforced for public phase

Severity: Medium

Impact: Medium - A single wallet can mint unlimited NFTs during public phase, potentially exhausting supply.

Likelihood: Medium - Requires public phase to be active and sufficient funds.

Location: `iLemonatiMint.sol` - `mint()` function, public phase branch

Description:

While OG and WL phases enforce a limit of 1 NFT per wallet, the public phase has no per-wallet or per-transaction limit:

```
if (phase == 1) {
    require(ogMinted[msg.sender] + quantity <= 1, "OG limit exceeded");
    require(totalOgMinted + quantity <= maxOgSupply, "OG supply exceeded");
    // ...
} else if (phase == 2) {
    require(wlMinted[msg.sender] + quantity <= 1, "WL limit exceeded");
    require(totalWlMinted + quantity <= maxWlSupply, "WL supply exceeded");
    // ...
} else {
    cost = publicPrice * quantity;  // NO QUANTITY OR PER-WALLET LIMIT
}
```

Attack Scenario:

1. Public mint opens
2. Whale or bot calls `mint(1000, 3)` with 650,000 MON
3. Single wallet mints entire remaining supply
4. Other users cannot participate

Recommendation:

Add public phase limits:

```
mapping(address => uint256) public publicMinted;
uint256 public maxPublicPerWallet = 5;
uint256 public maxPublicSupply = 400;
uint256 public totalPublicMinted;

// In the public mint branch:
} else {
    require(publicMinted[msg.sender] + quantity <= maxPublicPerWallet, "Public wallet l
    require(totalPublicMinted + quantity <= maxPublicSupply, "Public supply exceeded");
    publicMinted[msg.sender] += quantity;
    totalPublicMinted += quantity;
    cost = publicPrice * quantity;
}
```

[L-01] Default startTime set to far future date

Severity: Low

Location: `iLemonatiMint.sol:10`

Description:

The default `startTime` is hardcoded to `1766842200`, which corresponds to January 27, 2026. If not updated before deployment or shortly after, the mint will be inaccessible:

```
uint256 public startTime = 1766842200;
```

Recommendation:

Either:

1. Set a more reasonable default (e.g., `type(uint256).max` to explicitly indicate "not set")
2. Add a deployment script that sets the correct start time
3. Document that `setStartTime()` must be called before launch

[L-02] Centralized admin control without timelock

Severity: Low

Location: Multiple admin functions

Description:

Admins have significant control over the contract without any delay mechanism:

- `setOg()` / `setWl()` - Can add/remove addresses from lists instantly
- `setPrices()` - Can change mint prices without notice
- `setSupplies()` - Can modify supply caps
- `setStartTime()` - Can change or delay mint start
- `adminMint()` - Can mint NFTs without limits or payment

While this flexibility is often needed, it creates trust assumptions for users.

Recommendation:

Consider implementing:

1. Timelock for sensitive operations (price changes, supply changes)
2. Multi-sig requirement for admin actions
3. Event emissions for all admin actions (for transparency)

[L-03] No total supply cap enforcement in contract

Severity: Low

Location: `iLemonatiMint.sol`

Description:

The contract tracks `maxOgSupply` (250) and `maxWlSupply` (350) but has no overall collection cap. Combined with unlimited public mints and `adminMint()`, the total supply is effectively unlimited.

Recommendation:

Add a total supply cap:

```
uint256 public maxTotalSupply = 1000;
uint256 public totalMinted;

// In mint() before nftContract.mint():
require(totalMinted + quantity <= maxTotalSupply, "Max supply exceeded");
totalMinted += quantity;

// In adminMint():
```

```
require(totalMinted + quantity <= maxTotalSupply, "Max supply exceeded");
totalMinted += quantity;
```

[L-04] Admin can mint unlimited NFTs bypassing all limits

Severity: Low

Location: `iLemonatiMint.sol` - `adminMint()` function

Description:

The `adminMint()` function bypasses all restrictions:

```
function adminMint(address to, uint256 quantity) external onlyAdminOrOwner {
    nftContract.mint(to, quantity);
}
```

This allows admins to:

- Mint without payment
- Mint beyond supply caps
- Mint to any address
- Mint any quantity

Recommendation:

If admin minting is necessary, consider:

1. Adding it to `totalMinted` tracking
2. Emitting events for transparency
3. Capping admin mints or requiring multi-sig

```
event AdminMint(address indexed to, uint256 quantity, address indexed admin);

function adminMint(address to, uint256 quantity) external onlyAdminOrOwner {
    require(totalMinted + quantity <= maxTotalSupply, "Max supply exceeded");
    totalMinted += quantity;
    nftContract.mint(to, quantity);
    emit AdminMint(to, quantity, msg.sender);
}
```

Informational Findings

[I-01] Consider adding events for state changes

Admin functions like `setOg()`, `setWl()`, `setPrices()`, etc. do not emit events. Adding events improves transparency and allows off-chain tracking:

```
event OgStatusUpdated(address[] wallets, bool status);
event WlStatusUpdated(address[] wallets, bool status);
event PricesUpdated(uint256 publicPrice, uint256 wlPrice);
event SuppliesUpdated(uint256 maxOgSupply, uint256 maxWlSupply);
```

[I-02] Gas optimization in batch setters

The `setOg()` and `setWl()` functions could use `unchecked` for the loop increment since overflow is impossible with reasonable array sizes:

```
function setOg(address[] calldata wallets, bool status) external onlyAdminOrOwner {
    for (uint256 i = 0; i < wallets.length;) {
        og[wallets[i]] = status;
        unchecked { ++i; }
    }
}
```

Security Patterns Observed

Positive

- Solidity ^0.8.20 with automatic overflow protection
- ReentrancyGuard on mint function prevents reentrancy attacks
- Proper refund mechanism with success check
- Separation of owner and admin roles
- Uses well-audited OpenZeppelin contracts

Concerns

- User-controlled phase parameter
- No limits on public minting
- Centralized admin control
- No total supply enforcement

Conclusion

The iLemonatiMint contract is a straightforward NFT minting contract with reasonable security practices. The main concerns are:

1. **User-controlled phase parameter** - Allows bypassing intended phase restrictions
2. **No public mint limits** - Single wallet can mint unlimited NFTs

These issues should be addressed before deployment to ensure fair distribution and prevent gaming of the mint phases.

Overall Risk Assessment: Low-Medium

This security review was conducted by Stonewall. For questions or clarifications, contact our team.