

Scanned Code Report

AUDITAGENT

Code Info

Deep Scan

#	Scan ID 6	+	Date December 02, 2025
📦	Organization igor53627	📦	Repository iconregistry.eth
📖	Branch main	📄	Commit Hash 77c2e94e...461f1fcd

Contracts in scope

contracts/IIconRegistry.sol contracts/IIconRegistry.sol

Code Statistics

🔍	Findings 9	📄	Contracts Scanned 2	☰	Lines of Code 660
---	---------------	---	------------------------	---	----------------------

Findings Summary



Total Findings

- High Risk (0)
- Medium Risk (0)
- Low Risk (6)
- Info (2)
- Best Practices (1)

Code Summary

The Icon Registry protocol is an on-chain, upgradeable repository for PNG icons, designed to provide a canonical source of visual assets for dApps and wallets. It utilizes the SSTORE2 library to store icon data gas-efficiently, with each icon being written as an immutable data blob.

The system is managed by a privileged owner who is solely responsible for adding, updating, and mapping icons. Icons are identified by a human-readable `slug` (e.g., "protocols/uniswap"), which is hashed for on-chain operations. The registry supports versioning, allowing for icon updates while preserving access to all historical versions. The owner can map these icons to specific token addresses on different chains or directly to chain IDs, enabling easy lookups for front-end integrations.

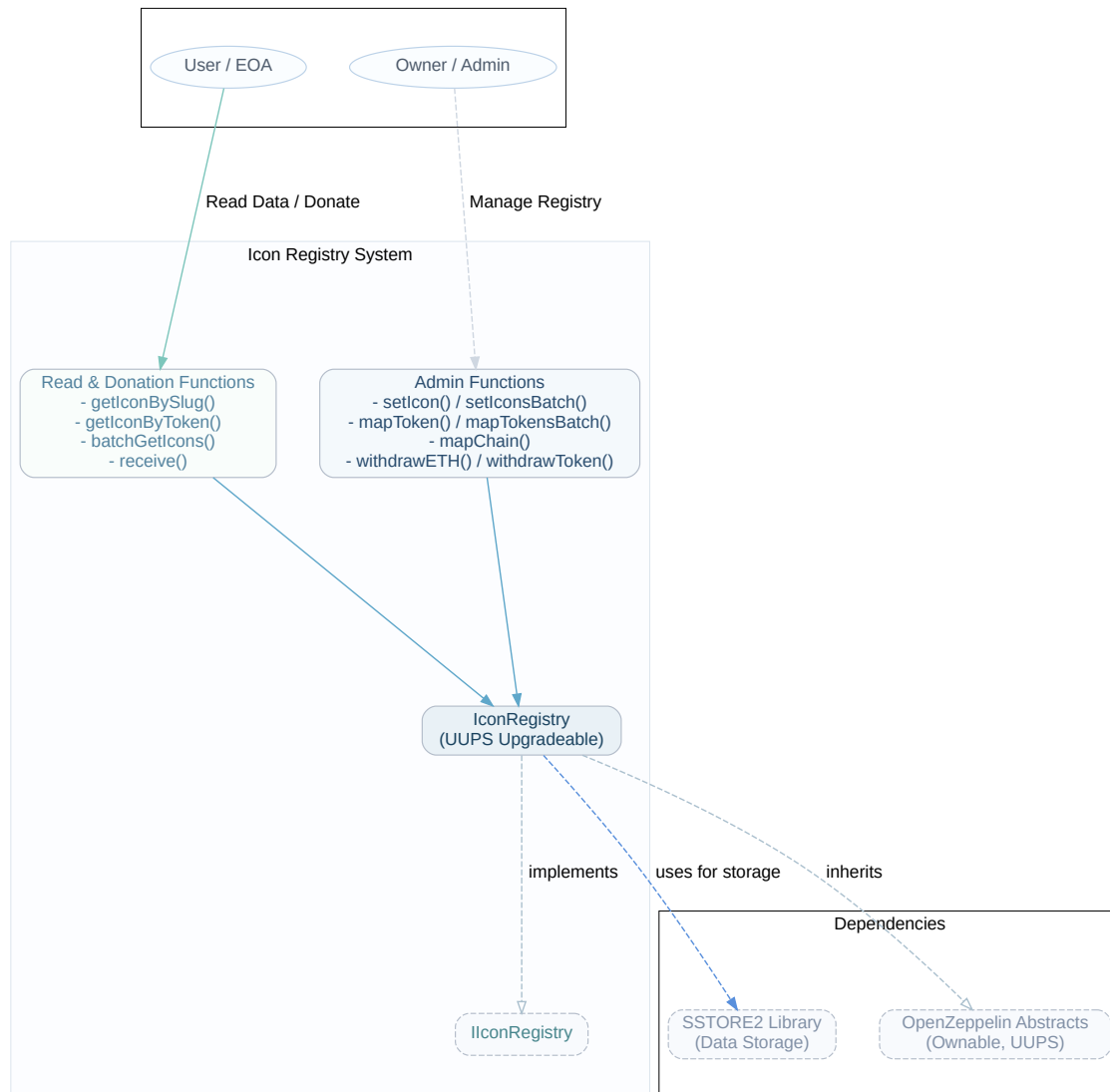
The contract is built using the UUPS upgradeable proxy pattern, allowing for future logic changes by the owner. It also includes a mechanism to accept ETH donations to support the project's maintenance.

Main Entry Points and Actors

All core functions that modify the registry's content are restricted to the contract owner. The only state-changing interaction available to the general public is for making donations.

- **Any User:** Can send ETH to the contract via the `receive()` function to support the registry.

Code Diagram



✦ 1 of 9 Findings

contracts/IconRegistry.sol

Magic Numbers Instead Of Constants

• Low Risk

The contract uses magic number literals (e.g., `3`, `2`) directly in the code instead of defining them as named constants. Specifically, the modulo operation `len % 3 == 2` appears multiple times throughout the contract. Using named constants improves code readability, maintainability, and reduces the risk of errors when these values need to be updated. Define constants such as `uint256 private constant MODULO_DIVISOR = 3;` and `uint256 private constant MODULO_REMAINDER = 2;` and reference them throughout the contract.

✦ 2 of 9 Findings

contracts/IconRegistry.sol

Empty Code Block Detection

• Low Risk

The function `_authorizeUpgrade(address)` contains an empty code block. Empty functions should be removed or replaced with meaningful implementation. If the function is required by an interface or inheritance pattern, consider adding a comment explaining why it is intentionally empty, or implement the required logic.

✦ 3 of 9 Findings

contracts/IconRegistry.sol

Loop contains `require` / `revert` statements

• Low Risk

The contract contains loops with `require` or `revert` statements inside them. If any single iteration fails the require condition, the entire transaction will revert, causing all previous iterations to be rolled back. This can lead to denial of service scenarios where a single invalid item prevents processing of the entire batch. Consider implementing a fail-safe approach that collects failed items and returns them for post-processing, allowing the loop to continue processing valid items.

✦ 4 of 9 Findings

contracts/IconRegistry.sol contracts/IconRegistry.sol

Non-Specific Solidity Pragma Version

• Low Risk

The contracts use a wide pragma version range (`pragma solidity ^0.8.22;`) instead of a specific version. This can lead to unexpected behavior when compiled with different compiler versions. It is recommended to use a specific Solidity version (e.g., `pragma solidity 0.8.22;`) to ensure consistent compilation and predictable bytecode generation across different environments.

✦ 5 of 9 Findings

contracts/IconRegistry.sol contracts/IconRegistry.sol

PUSH0 Opcode Compatibility Issue

• Low Risk

The contracts use Solidity compiler version 0.8.22, which defaults to the Shanghai EVM version. This means the generated bytecode will include PUSH0 opcodes. If deployment is intended for chains other than Ethereum mainnet (such as Layer 2 solutions or other EVM-compatible chains), ensure that the target chain supports the PUSH0 opcode. Otherwise, contract deployment will fail. Consider explicitly specifying the EVM version in the compiler configuration to match your deployment target.

✦ 6 of 9 Findings

contracts/IconRegistry.sol

Unused Custom Error

• Low Risk

The custom error `VersionNotFound()` is defined but never used anywhere in the contract. Unused custom errors should be removed to reduce contract size and improve code clarity. If this error is intended for future use or external reference, consider documenting its purpose or removing it if it is no longer needed.

✦ 7 of 9 Findings

contracts/IconRegistry.sol

Unprotected initializer allows ownership takeover

• Info

```
function initialize(address owner_) external initializer {
    __Ownable_init(owner_);
}
```

The `initialize` function is external and only guarded by OpenZeppelin's `initializer` modifier, but not restricted to the deploying address. If the proxy is deployed without being initialized atomically, any third party can call `initialize(attackerAddress)` and become the contract owner. Once the attacker is owner, they gain full privileges: they can upgrade the implementation via UUPS, manage or overwrite all icons, remap token and chain icons, and withdraw any ETH or ERC20 tokens held by the contract. A realistic attack is to monitor the transaction pool for the proxy deployment, then front-run the legitimate initialization with a higher gas price to claim ownership.

✦ 8 of 9 Findings

contracts/IconRegistry.sol

withdrawToken function breaks invariant with fee-on-transfer tokens[Info](#)

The `withdrawToken` function violates its invariant when handling fee-on-transfer or deflationary ERC20 tokens. The invariant states that after withdrawal, the owner's balance must increase by exactly the amount that was in the contract. However, fee-on-transfer tokens deduct a percentage during transfers, causing the owner to receive less than the contract's original balance.

```
function withdrawToken(address token) external onlyOwner {  
    uint256 balance = IERC20(token).balanceOf(address(this));  
    IERC20(token).safeTransfer(owner(), balance);  
}
```

Scenario:

1. Contract holds 1000 fee-on-transfer tokens (5% fee)
2. `withdrawToken` is called, which transfers 1000 tokens
3. Token contract deducts 5% fee (50 tokens)
4. Owner receives only 950 tokens
5. Contract balance becomes 0 ✓
6. Owner balance increases by 950, not 1000 ✗

The invariant condition

`IERC20(token).balanceOf(owner()) == balance_before_owner + balance_before_contract` is violated because the owner receives `balance_before_contract - fee` instead of the full `balance_before_contract`.

Since the protocol accepts token donations via the `receive()` function and documentation, users could donate fee-on-transfer tokens. When the owner withdraws, they would receive less than expected, and the invariant would be broken.

✦ 9 of 9 Findings

contracts/IconRegistry.sol

Missing Events for Fund Withdrawals

• Best Practices

The `withdrawETH()` and `withdrawToken()` functions do not emit events when funds are withdrawn from the contract.

```
function withdrawETH() external onlyOwner {
    (bool success,) = payable(owner()).call{value: address(this).balance}("");
    if (!success) revert TransferFailed();
    // No event emitted
}

function withdrawToken(address token) external onlyOwner {
    uint256 balance = IERC20(token).balanceOf(address(this));
    IERC20(token).safeTransfer(owner(), balance);
    // No event emitted
}
```

This reduces transparency and makes it difficult to:

1. Track fund movements on-chain
2. Monitor donation withdrawals
3. Audit the contract's financial history
4. Set up alerts for fund movements

Disclaimer

Kindly note, the Audit Agent is currently in beta stage and no guarantee is being given as to the accuracy and/or completeness of any of the outputs the Audit Agent may generate, including without limitation this Report. The results set out in this Report may not be complete nor inclusive of all vulnerabilities. The Audit Agent is provided on an 'as is' basis, without warranties or conditions of any kind, either express or implied, including without limitation as to the outputs of the code scan and the security of any smart contract verified using the Audit Agent.

Blockchain technology remains under development and is subject to unknown risks and flaws. This Report does not indicate the endorsement of any particular project or team, nor guarantee its security. Neither you nor any third party should rely on this Report in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset.

To the fullest extent permitted by law, Nethermind disclaims any liability in connection with this Report, its content, and any related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. Nethermind does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and Nethermind will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.