Upgradeable Staker Review

September 10, 2025

Prepared for DappHero Corp

Conducted by:

Richie Humphrey (devtooligan)

About the DappHero Corp Upgradeable Staker Review

Tally provides comprehensive governance solutions for decentralized organizations. The Upgradeable Staker protocol enables token holders to stake their assets while maintaining control over their voting rights.

This review focuses on the upgradeable implementation of the ZKStaker contract and its associated deployment infrastructure, examining both the core staking functionality and the deployment scripts.

About Offbeat Security

Offbeat Security is a boutique security company providing unique security solutions for complex and novel crypto projects. Our mission is to elevate the blockchain security landscape through invention and collaboration.

Summary & Scope

The staker repository was reviewed at commit 126860b and the ZKstaker repository was reviewed at commit bee7270.

The following files and folders were in scope:

- staker repo: src/UpgradeableStaker.sol
- zkstaker repo: script/, src/

The review identified no vulnerabilities in the Solidity contract code. Two informational-level findings were identified in the deployment infrastructure, focusing on configuration security and operational best practices. Additional code quality recommendations address minor issues including variable naming and placeholder value management.

Summary of Findings

Identifier	Title	Severity	Fixed
I-01	Inconsistent private key handling pattern across deployment scripts	Informational	Fixed in PR#40
I-02	Default network configuration can lead to unintended mainnet deployment	Informational	Fixed in PR#40 TODO comments will be addressed prior to deployment.

Additional Recommendations

Code Quality

1. **Spelling error in variable name**

   Typo in variable name across deployment scripts (ZkStaker line 48, Sepolia line 46):

```
- const earningPowerCalculaterContractAddress = await earningPowerCalculator.
+ const earningPowerCalculatorContractAddress = await earningPowerCalculator.
```

2. **Production-critical constants marked as TODO/placeholders**

TODO comments indicate placeholder values in both deployment scripts:

- `ZK_CAPPED_MINTER` (line 23): TODO verify address
- `REWARD_AMOUNT` (line 16): placeholder value "1000000000000000000"
- `REWARD_INTERVAL` (line 17): placeholder value
- `INITIAL_TOTAL_STAKE_CAP` (line 26): placeholder value

Tests also hardcode these values (DeployZkStaker.test.ts lines 94-117), making drift likely. Wrong minter address or parameters could halt rewards or disable staking. Replace all placeholders with production values before mainnet deployment.

Detailed Findings

Informational Findings

[I-01] Inconsistent private key handling pattern across deployment scripts

Description

The deployment scripts exhibit inconsistent approaches to private key management. While `DeployZkStakerSepolia.ts` correctly retrieves the private key from environment variables (`process.env.DEPLOYER_PRIVATE_KEY`), the main deployment script `DeployZkStaker.ts` uses a hardcoded value with the environment variable approach commented out.

This inconsistency represents a deviation from security best practices, even when using development keys. Maintaining a consistent pattern of using environment variables for all sensitive configuration helps prevent accidents when transitioning from development to production environments.

Recommendation

Consider standardizing all deployment scripts to use environment variables for private key management. This ensures a consistent security pattern across the codebase and reduces the risk of accidentally committing sensitive keys when the scripts are modified for production use.

For local development environments, the development keys can be set in `.env.local` files rather than being hardcoded in scripts, maintaining the separation between code and configuration.

[I-02] Default network configuration can lead to unintended mainnet deployment

Description

The deployment infrastructure contains a configuration issue where the default network (`zkSyncLocal`) is not truly local but instead derives its RPC URL from an environment variable that defaults to mainnet. This creates a scenario where running deployment scripts without explicitly specifying a network can inadvertently target mainnet.

The issue stems from several interconnected configuration choices:

1. **Default network misconfiguration**: The `hardhat.config.ts` sets `defaultNetwork: "zkSyncLocal"`, but `zkSyncLocal` pulls its URL from `process.env.ZKSYNC_RPC_URL` without a fallback to a true local endpoint.

2. **Template points to mainnet**: The `.env.template` file sets `ZKSYNC_RPC_URL=https://mainnet.era.zksync.io`, meaning developers who copy the template as-is will have their "local" network pointing to mainnet.

3. **Script runner omits network flag**: The helper script `RunScript.js` executes Hardhat without the `--network` flag, causing it to use the misconfigured default network.

4. **Deployment scripts perform writes**: The deployment scripts execute state-changing transactions including `deployProxy()`, `setRewardNotifier()`, and `setAdmin()`. These are not read-only operations.

This combination means that developers or CI systems running the deployment script without explicitly specifying `--network` could accidentally broadcast transactions to mainnet when they believe they're deploying locally.

Recommendation

Consider implementing the following safeguards to prevent accidental mainnet deployments:

1. **Add explicit network validation in deployment scripts**:

```
// Add at the beginning of main()
const expectedChainId = 260; // zkSync local testnet chain ID
const network = await hre.ethers.provider.getNetwork();
if (network.chainId !== expectedChainId) {
  throw new Error(`Wrong network! Expected chainId ${expectedChainId}, got ${net
}
```

2. **Provide a true local default for zkSyncLocal**:

```
zkSyncLocal: {
-  url: process.env.ZKSYNC_RPC_URL || "http://localhost:8011",
```

```
+   url: "http://localhost:8011", // Always local
    ...
}
```

3. **Create a separate network configuration for mainnet**:

```
zkSyncMainnet: {
  url: process.env.ZKSYNC_MAINNET_RPC_URL || "https://mainnet.era.zksync.io",
  ...
}
```

4. **Make network flag mandatory in the script runner**:

```
// In RunScript.js
+ if (process.argv.length < 4) {
+   console.error("Usage: node RunScript.js <script-name> <network>");
+   process.exit(1);
+ }
const scriptName = process.argv[2];
+ const network = process.argv[3];
- execSync(`npx hardhat run script/${scriptName}`, { stdio: 'inherit' });
+ execSync(`npx hardhat run script/${scriptName} --network ${network}`, { stdio:
```

These changes ensure that local development truly uses local endpoints and that any mainnet deployment requires explicit network specification, reducing the risk of accidental production deployments.