Adevar Labs

Prepared for:

Edel Finance

March 26, 2026

Edel Deployment Script Audit Report

Version: before fix review

Table of Contents

1. Executive Summary

About Edel Finance

The Edel Finance deployment protocol is an Aave V3-based multi-market lending deployment framework, with market-specific configurations for Ethereum, Base, and Robinhood environments. The system manages reserve initialization, oracle wiring, incentives ownership handoff, admin role configuration, and post-deploy operational tasks.

Audit Summary

The table below summarizes identified vulnerabilities by risk level and remediation status.

Risk Level	Count	Fixed	Acknowledged
Critical	0	0	0
High	0	0	0
Medium	2	0	0
Low	4	0	0

Enhancement opportunities: 2

Overall Security Posture

The Edel Finance team shows solid familiarity with deployment workflows and market configuration patterns, but the findings indicate recurring gaps in configuration consistency, ownership handoff safeguards, and task robustness across environments. The codebase appears maintainable, yet stronger validation is needed before enabling borrowing or expanding the reserve set.

Before Fix Review Summary

The audit identified 2 Medium severity issues related to market configuration correctness, protocol ownership handoff, oracle safety, and 4 Low severity issues related to tooling resilience, configuration hygiene, and maintainability. 2 enhancement opportunities were also identified in this review.

Engagement Scope Limitations

This review was conducted under a constrained 3-day, single-researcher engagement. The following areas were assessed at reduced depth or deferred entirely and carry residual uncleared risk:

- Reserve risk parameters (LTV, liquidation threshold, supply/borrow caps) were not independently validated
- On-chain state verification was not performed – findings are based on source code and deployment artifacts, not live contract inspection
- Incentives and reward system (PullRewardsTransferStrategy, RewardOracle-EDEL, setupReward tasks) were not reviewed
- The Robinhood testnet configuration and deployment were not reviewed
- ACL role consolidation – all five core administrative roles (deployer, aclAdmin, emergencyAdmin, poolAdmin, addressesProviderRegistryOwner) resolve to the same address (**0xd0E0BFd1...**). Whether this address is a multisig or EOA was not verified on-chain within this engagement. This represents the highest-consequence unverified assumption in the deployment.

Launch Recommendations

I. Mandatory before launch:

- Resolve all Medium severity issues including market-specific reserve initialization, and EmissionManager ownership transfer correctness.
- Add pre-deployment and pre-handoff validation checks that fail fast on mismatched key maps, and skipped governance transfers.
- Re-run full deployment and operational task flows on each target market after fixes, with evidence that price, ownership, and admin role paths execute correctly.

II. Strongly recommended:

- Address Low severity findings affecting operational tooling, including missing network constants, hardcoded lookup keys, and unguarded optional artifact loading.
- Standardize naming and configuration patterns across markets to reduce drift and improve auditability.
- Add targeted tests for task-level behavior and environment-specific configuration loading, including negative-path checks.

III. Post-Launch Monitoring:

- Implement automated monitoring for oracle source validity and price update liveness across all listed assets.
- Track governance and admin role changes on-chain and alert on unexpected owner or role-holder deviations.
- Maintain periodic operational health checks for deployment tasks to detect regressions in market-specific scripts.

Audit Scope

The audit scope was performed on the diff between the forked commit hash and before-fix review commit hash.

- **Repository:** https://github.com/Edel-Finance/edel-deployment-script/
- **Before-Fix Review Commit Hash:** 3133bb341a85f7f8fa12761c8bc23684152ab9f4
- **Files/Modules in Scope:**
  - deploy/*, helpers/*, markets/*, scripts/*, tasks/*
- **Forked repository:** https://github.com/aave/aave-v3-deploy
- **Forked commit hash:** 27ccc6d24ef767a2b71946784a843526edbc9618

2. Severity Definitions

Each issue identified in this report is assigned a severity level based on two dimensions: **Impact** and **Likelihood**. These dimensions help project our team's understanding of both the potential consequences of a vulnerability and how likely a vulnerability is to be discovered and exploited in the real world.

Note: Enhancements represent non-blocking improvements–typically usability, observability, or defense-in-depth tweaks that do not pose an immediate asset risk but would improve the product's reliability and/or user experience if implemented.

Impact

Impact reflects the potential consequences of the issue–particularly on **project funds**, **user funds**, and the **availability or integrity** of the protocol.

- **High Impact**: Successful exploitation could result in a complete loss of user or protocol funds, disruption of core protocol functionality, or permanent loss of control over critical components.
- **Medium Impact**: Exploitation could cause significant disruption or partial loss of funds, but not a total compromise. May impact some users or non-core functionality.
- **Low Impact**: The issue has minor or negligible consequences. It may affect edge cases, expose metadata, or degrade performance slightly without putting funds or core logic at serious risk.

Likelihood

Likelihood reflects how easy a vulnerability is to discover and exploit by an attacker, as well as how economically attractive the exploit is to an attacker.

- **High Likelihood**: The vulnerability is trivially exploitable. This means it can be exploited by a wide range of actors without privileged access rights, with minimal capital requirements and low financial risks.
- **Medium Likelihood**: This type of vulnerability can be found and exploited with moderate effort. It might require a significant capital investment, but with manageable financial risk.
- **Low Likelihood**: Exploitation of these vulnerabilities is often technically unfeasible or requires highly specialized conditions. They may require extraordinary effort or a significant financial risk for an attacker, with a high chance of failure and minimal potential return.

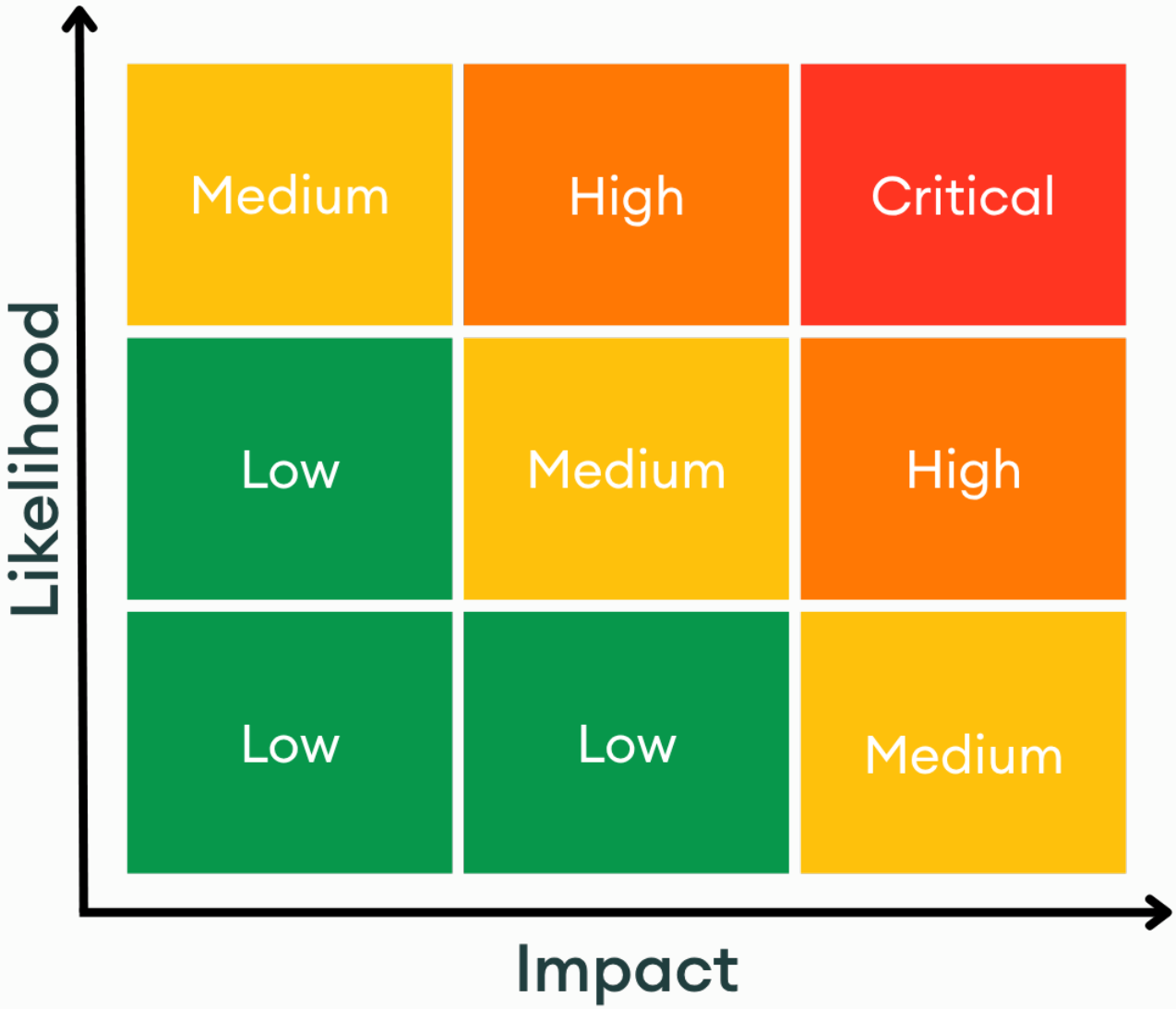
Severity Classification Matrix

By combining **Impact** and **Likelihood**, we assign a severity level using the matrix below:

- **Critical**: High impact + high likelihood (e.g. a bug that could allow anyone to drain a substantial amount of protocol funds with minimal effort)
- **High**: High impact with medium likelihood, or medium impact with high likelihood
- **Medium**: Moderate impact and/or discoverability

- **Low**: Minimal impact or unlikely to be exploited

This structured approach helps teams prioritize fixes and mitigate the most dangerous threats first.



Severity Matrix

3. Findings

M01: Case mismatch between ReserveAssets & ChainlinkAggregator keys breaks ethereum market oracle configuration

LOCATION

edel-deployment-script/main/markets/ethereum/index.ts:L26-L45

RELEVANT SNIPPET

>_ index.ts — TYPESCRIPT

```
24:  VariableDebtTokenNamePrefix: "",
25:  SymbolPrefix: "",
26:  ReserveAssets: {
27:    [eEthereumNetwork.main]: {
28:      TSLAX: "0x8aD3c73F833d3F9A523aB01476625F269aEB7Cf0",
29:      SPYX: "0x90A2a4c76b5D8c0bc892A69EA28Aa775a8f2dD48",
30:      QQQX: "0xa753A7395cAe905Cd615Da0B82A53E0560f250af",
31:      NVDAX: "0xc845b2894dBddd03858fd2D643B4eF725fE0849d",
32:      MSTRX: "0xAE2f842EF90C0d5213259Ab82639D5BBF649b08E",
33:      USDC: "0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48"
34:    }
35:  },
36:  ChainlinkAggregator: {
37:    [eEthereumNetwork.main]: {
38:      TSLAx: "0x13536966973F5F463918cEf7CD6E1545cD790116",
39:      SPYx: "0x52A8A8362aC83D22A52Cf01C822a94504630C7bB",
40:      QQQx: "0xf0EF3A07B7548C5BdF9e120dCFE7222416eb0af2",
41:      NVDAx: "0x52a8D617B63dB1235cc957C4B97439c2d0A6aAC2",
42:      MSTRx: "0x7ADe622936Fb71dce65a120E01C11A647304ac7c",
43:      USDC: "0x8fFfFfd4AfB6115b954Bd326cbe7B4BA576818f6"
44:    }
45:  },
46:  ReservesConfig: {
47:    TSLAX: strategyTSLAx,
```

DESCRIPTION

ReserveAssets in markets/ethereum/index.ts#L27-L34 uses all-caps keys (**TSLAX**, **SPYX**, **QQQX**, **NVDAX**, **MSTRX**), while **ChainlinkAggregator** at L37-L44 uses mixed-case keys (**TSLAx**, **SPYx**, **QQQx**, **NVDAx**, **MSTRx**).

>_ index.ts TYPESCRIPT

```typescript
SymbolPrefix: "",
ReserveAssets: {
  [eEthereumNetwork.main]: {
    TSLAX: "0x8aD3c73F833d3F9A523aB01476625F269aEB7Cf0",
    SPYX: "0x90A2a4c76b5D8c0bc892A69EA28Aa775a8f2dD48",
    QQQX: "0xa753A7395cAe905Cd615Da0B82A53E0560f250af",
    NVDAX: "0xc845b2894dBddd03858fd2D643B4eF725fE0849d",
    MSTRX: "0xAE2f842EF90C0d5213259Ab82639D5BBF649b08E",
    USDC: "0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48"
  }
},
ChainlinkAggregator: {
```

>_ index.ts TYPESCRIPT

```typescript
},
ChainlinkAggregator: {
  [eEthereumNetwork.main]: {
    TSLAx: "0x13536966973F5F463918cEf7CD6E1545cD790116",
    SPYx: "0x52A8A8362aC83D22A52Cf01C822a94504630C7bB",
    QQQx: "0xf0EF3A07B7548C5BdF9e120dCFE7222416eb0af2",
    NVDAx: "0x52a8D617B63dB1235cc957C4B97439c2d0A6aAC2",
    MSTRx: "0x7ADe622936Fb71dce65a120E01C11A647304ac7c",
    USDC: "0x8fFfFfd4AfB6115b954Bd326cbe7B4BA576818f6"
  }
},
ReservesConfig: {
```

The oracle deployment path in 04_deploy_oracles.ts#L43-L48 feeds both maps into **getPairsTokenAggregator** at init-helpers.ts#L246-L248, which does exact case-sensitive matching via **findIndex((value) => value === tokenSymbol)**. Since **"TSLAX" !== "TSLAx"**, **findIndex** returns **-1**, causing a TypeError on the undefined array access. A fresh redeploy with the current config would crash during oracle-pair construction rather than silently produce a partial deployment.

>_ 04_deploy_oracles.ts TYPESCRIPT

```typescript
const fallbackOracleAddress = ZERO_ADDRESS;

const reserveAssets = await getReserveAddresses(poolConfig, network);
const chainlinkAggregators = await getChainlinkOracles(poolConfig, network);

const [assets, sources] = getPairsTokenAggregator(
  reserveAssets,
  chainlinkAggregators
);

```

>_ init-helpers.ts TYPESCRIPT

```typescript
const pairs = Object.entries(assetsAddressesWithoutEth).map(
  ([tokenSymbol, tokenAddress]) => {
    const aggregatorAddressIndex = Object.keys(
      aggregatorsAddresses
    ).findIndex((value) => value === tokenSymbol);
    const [, aggregatorAddress] = (
      Object.entries(aggregatorsAddresses) as [string, tEthereumAddress][]
```

The checked-in deployment artifact at AaveOracle-Ethereum.json#L345-L364 shows only 5 asset/source pairs in the constructor args, USDC is not among them. Since the fallback oracle is also **address(0)** at L361, a missing source hits the fallback path in AaveOracle.sol#L101, which returns empty data and reverts.

>_ AaveOracle-Ethereum.json JSON

```
  "byzantium": true
},
"args": [
  "0x7Ff6BAC9FedFA774060C880122bAd2b607Ba8A4d",
  [
    "0x8aD3c73F833d3F9A523aB01476625F269aEB7Cf0",
    "0x90A2a4c76b5D8c0bc892A69EA28Aa775a8f2dD48",
    "0xa753A7395cAe905Cd615Da0B82A53E0560f250af",
    "0xc845b2894dBddd03858fd2D643B4eF725fE0849d",
    "0xAE2f842EF90C0d5213259Ab82639D5BBF649b08E"
  ],
  [
    "0x13536966973F5F463918cEf7CD6E1545cD790116",
    "0x52A8A8362aC83D22A52Cf01C822a94504630C7bB",
    "0xf0EF3A07B7548C5BdF9e120dCFE7222416eb0af2",
    "0x52a8D617B63dB1235cc957C4B97439c2d0A6aAC2",
    "0x7ADe622936Fb71dce65a120E01C11A647304ac7c"
  ],
  "0x0000000000000000000000000000000000000000",
  "0x0000000000000000000000000000000000000000",
  "100000000"
],
"numDeployments": 1,
"solcInputHash": "f83ccc030186e5b58439131350431de3",
```

>_ AaveOracle-Ethereum.json JSON

```
    "0x7ADe622936Fb71dce65a120E01C11A647304ac7c"
],
"0x0000000000000000000000000000000000000000",
"0x0000000000000000000000000000000000000000",
"100000000"
```

RECOMMENDATION

Normalize the **ReserveAssets** keys at markets/ethereum/index.ts#L27-L34 to match the **ChainlinkAggregator** casing

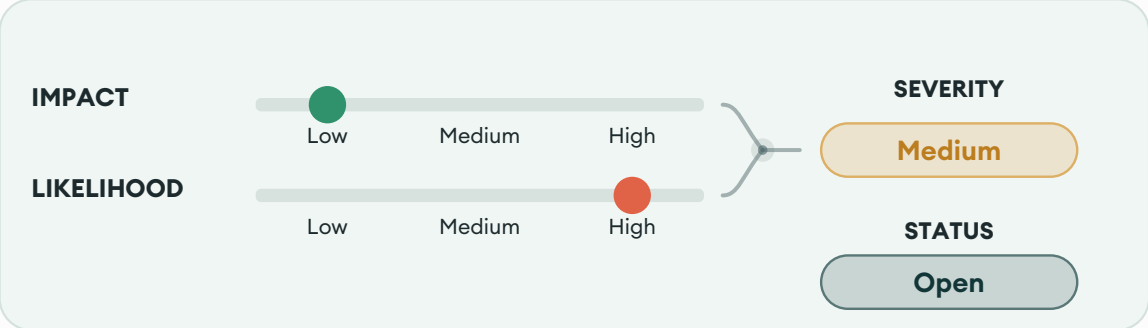
>_ index.ts TYPESCRIPT

```
25:  SymbolPrefix: "",
26:  ReserveAssets: {
27:    [eEthereumNetwork.main]: {
28:      TSLAX: "0x8aD3c73F833d3F9A523aB01476625F269aEB7Cf0",
29:      SPYX: "0x90A2a4c76b5D8c0bc892A69EA28Aa775a8f2dD48",
30:      QQQX: "0xa753A7395cAe905Cd615Da0B82A53E0560f250af",
31:      NVDAX: "0xc845b2894dBddd03858fd2D643B4eF725fE0849d",
32:      MSTRX: "0xAE2f842EF90C0d5213259Ab82639D5BBF649b08E",
33:      USDC: "0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48"
34:    }
35:  },
36:  ChainlinkAggregator: {
```

TYPESCRIPT

```
ReserveAssets: {
  [eEthereumNetwork.main]: {
    TSLAx: "0x8aD3c73F833d3F9A523aB01476625F269aEB7Cf0",
    SPYx: "0x90A2a4c76b5D8c0bc892A69EA28Aa775a8f2dD48",
    QQQx: "0xa753A7395cAe905Cd615Da0B82A53E0560f250af",
    NVDAx: "0xc845b2894dBddd03858fd2D643B4eF725fE0849d",
    MSTRx: "0xAE2f842EF90C0d5213259Ab82639D5BBF649b08E",
    USDC: "0xA0b86991c6218b36c1d19D4a2e9Eb0cE3606eB48"
  }
},
```

M02: Mismatched owner check in ownership handoff silently skips EmissionManager transfer to governance



LOCATION

edel-deployment-script/main/deploy/02_market/07_incentives.ts:L65-L68

RELEVANT SNIPPET

>_ 07_incentives.ts — TYPESCRIPT

```
63:
64: // Deploy EmissionManager
65: const emissionManagerArtifact = await deploy(EMISSION_MANAGER_ID, {
66:   from: deployer,
67:   contract: "EmissionManager",
68:   args: [deployer],
69:   ...COMMON_DEPLOY_PARAMS,
70: });
```

DESCRIPTION

The deploy script at 07_incentives.ts#L65-L68 deploys EmissionManager with deployer (named account index 0) as initial owner, then immediately transfers ownership to incentivesEmissionManager (named account index 1) at L167-L168.

>_ 07_incentives.ts — TYPESCRIPT

```
63:
64: // Deploy EmissionManager
65: const emissionManagerArtifact = await deploy(EMISSION_MANAGER_ID, {
66:   from: deployer,
67:   contract: "EmissionManager",
68:   args: [deployer],
69:   ...COMMON_DEPLOY_PARAMS,
70: });
```

>_ 07_incentives.ts — TYPESCRIPT

```
165: // Transfer emission manager ownership
166:
167: await waitForTx(
168:   await emissionManager.transferOwnership(incentivesEmissionManager)
169: );
170:
```

Later, transfer-protocol-ownership.ts#L189-L190 attempts to hand EmissionManager to governance but guards the transfer with:

>_ transfer-protocol-ownership.ts TYPESCRIPT

```
187:
188: /** Start of EmissionManager transfer ownership */
189: const isDeployerEmissionManagerOwner =
190:   (await emissionManager.owner()) === deployer;
191: if (isDeployerEmissionManagerOwner) {
192:   await emissionManager.transferOwnership(desiredAdmin);
```

TYPESCRIPT

```
const isDeployerEmissionManagerOwner =

  (await emissionManager.owner()) === deployer;
```

At this point emissionManager.owner() returns incentivesEmissionManager (index 1, 0x8F5D... on mainnet per PullRewardsTransferStrategy.json#L166), not deployer (index 0, 0xd0E0... per EmissionManager.json constructor args). The condition evaluates to false and the transfer to governance is silently skipped.

>_ PullRewardsTransferStrategy.json JSON

```
164: "args": [
165:   "0x3885d7FE5745c2A94CaBA576c84463a3fbDe72Ba",
166:   "0x8F5Db2198DBC16Ac040Cbebd6B3c0842A4eebea3",
167:   "0x8F5Db2198DBC16Ac040Cbebd6B3c0842A4eebea3"
168: ],
```

The view-protocol-roles.ts#L219-L224 task confirms this is unintended, it asserts emissionManager.owner() === desiredAdmin (governance), which would report false.

>_ view-protocol-roles.ts TYPESCRIPT

```
217: },
218: {
219:   role: "EmissionManager controller contract Owner",
220:   address: emissionManager
221:     ? await emissionManager.owner()
222:     : "Missing contract address",
223:   assert: emissionManager
224:     ? (await emissionManager.owner()) === desiredAdmin
225:     : false,
226: },
```

The non-governance EOA (0x8F5D...) retains the ability to call setEmissionAdmin, setClaimer, and setRewardsController on the EmissionManager, as defined in the EmissionManager contract.

RECOMMENDATION

Change the guard at transfer-protocol-ownership.ts#L189-L190 to check against incentivesEmissionManager instead of deployer.

>_ transfer-protocol-ownership.ts TYPESCRIPT

```typescript
/** Start of EmissionManager transfer ownership */
const isDeployerEmissionManagerOwner =
  (await emissionManager.owner()) === deployer;
if (isDeployerEmissionManagerOwner) {
  await emissionManager.transferOwnership(desiredAdmin);
```

L01: Hardcoded StableEMode key in review-e-mode task crashes on all Edel markets

LOCATION

edel-deployment-script/main/tasks/misc/review-e-mode.ts:L40

RELEVANT SNIPPET

>_ review-e-mode.ts — TYPESCRIPT

```
38: const poolConfig = await loadPoolConfig(MARKET_NAME);
39:
40: const configEmodeCategory = poolConfig.EModes["StableEMode"];
41: const onChainCategory = await pool.getEModeCategoryData(id);
42:
```

DESCRIPTION

review-e-mode.ts#L40 hardcodes the lookup key

>_ review-e-mode.ts — TYPESCRIPT

```
38: const poolConfig = await loadPoolConfig(MARKET_NAME);
39:
40: const configEmodeCategory = poolConfig.EModes["StableEMode"];
41: const onChainCategory = await pool.getEModeCategoryData(id);
42:
```

TYPESCRIPT

```
const configEmodeCategory = poolConfig.EModes["StableEMode"];
```

Both Edel markets set **EModes: {}** ethereum/index.ts#L54, base/index.ts#L64. The lookup returns **undefined**, and accessing **.ltv** throws a **TypeError**, making the task unusable on any Edel deployment.

>_ index.ts — TYPESCRIPT

```
52:     USDC: strategyUSDC,
53:   },
54:   EModes: {},
55:   ReserveFactorTreasuryAddress: {},
56: };
```

>_ index.ts TYPESCRIPT

```
62:   },
63: },
64: EModes: {},
65: ChainlinkAggregator: {
66:   [eBaseNetwork.base]: {
```

RECOMMENDATION

Guard the lookup here and use the **--id** parameter to dynamically resolve the EMode key

>_ review-e-mode.ts TYPESCRIPT

```
38: const poolConfig = await loadPoolConfig(MARKET_NAME);
39:
40: const configEmodeCategory = poolConfig.EModes["StableEMode"];
41: const onChainCategory = await pool.getEModeCategoryData(id);
42:
```

TYPESCRIPT

```
const emodeKeys = Object.keys(poolConfig.EModes);
const configEmodeCategory = emodeKeys.length > 0
  ? poolConfig.EModes[emodeKeys.find((_, idx) => String(idx + 1) === id) || ""]
  : undefined;

if (!configEmodeCategory) {
  console.log(`No EMode category found in config for id ${id}`);
  return;
}
```

L02: Missing EMERGENCY_ADMIN entry for Base network crashes admin audit tooling

LOCATION

edel-deployment-script/main/helpers/constants.ts:L109-L112

RELEVANT SNIPPET

>_ constants.ts — TYPESCRIPT

```
107: };
108:
109: export const EMERGENCY_ADMIN: Record<string, string> = {
110:   [eEthereumNetwork.main]: "0xd0E0BFd1CD4f7A7A4c79A51aE60c5e9A1FB8A9AD", // Edel Finan
       ce Governance
111:   [eRobinhoodNetwork.robinhoodTestnet]: "0xd0E0BFd1CD4f7A7A4c79A51aE60c5e9A1FB8A9AD",
112: };
113:
114: export const DEFAULT_NAMED_ACCOUNTS = {
```

DESCRIPTION

EMERGENCY_ADMIN at constants.ts#L109-L112 only has entries for **main** and **robinhoodTestnet**. Base is absent, despite being present in **POOL_ADMIN** at L101-L107.

>_ constants.ts — TYPESCRIPT

```
107: };
108:
109: export const EMERGENCY_ADMIN: Record<string, string> = {
110:   [eEthereumNetwork.main]: "0xd0E0BFd1CD4f7A7A4c79A51aE60c5e9A1FB8A9AD", // Edel Finan
       ce Governance
111:   [eRobinhoodNetwork.robinhoodTestnet]: "0xd0E0BFd1CD4f7A7A4c79A51aE60c5e9A1FB8A9AD",
112: };
113:
114: export const DEFAULT_NAMED_ACCOUNTS = {
```

>_ constants.ts TYPESCRIPT

```
  "0x0000000000000000000000000000000000000000000000000000000000000000";

export const POOL_ADMIN: Record<string, string> = {
  [eEthereumNetwork.main]: "0xd0E0BFd1CD4f7A7A4c79A51aE60c5e9A1FB8A9AD", // Edel Finance Governance
  [eBaseNetwork.base]: "0xd0E0BFd1CD4f7A7A4c79A51aE60c5e9A1FB8A9AD",
  [eBaseNetwork.baseGoerli]: "0xA9F30e6ED4098e9439B2ac8aEA2d3fc26BcEbb45",
  [eEthereumNetwork.tenderly]: ETHEREUM_SHORT_EXECUTOR,
  [eRobinhoodNetwork.robinhoodTestnet]: "0xd0E0BFd1CD4f7A7A4c79A51aE60c5e9A1FB8A9AD",
};

export const EMERGENCY_ADMIN: Record<string, string> = {
```

view-protocol-roles.ts#L56-L61 calls **exit(403)** when **desiredEmergencyAdmin** is undefined, crashing the admin audit task for the Base deployment before any role data is displayed.

>_ view-protocol-roles.ts TYPESCRIPT

```
}

if (!desiredEmergencyAdmin) {
  console.error(
    "The constant desired EmergencyAdmin is undefined. Check missing Multisig at MULTISIG_ADDRESS constant"
  );
  exit(403);
}
const poolAddressesProvider = await getPoolAddressesProvider();
const rewardsController = await getIncentivesV2();
```

RECOMMENDATION

Add the Base network entry to **EMERGENCY_ADMIN** at constants.ts#L109-L112

>_ constants.ts TYPESCRIPT

```
};

export const EMERGENCY_ADMIN: Record<string, string> = {
  [eEthereumNetwork.main]: "0xd0E0BFd1CD4f7A7A4c79A51aE60c5e9A1FB8A9AD", // Edel Finance Governance
  [eRobinhoodNetwork.robinhoodTestnet]: "0xd0E0BFd1CD4f7A7A4c79A51aE60c5e9A1FB8A9AD",
};

export const DEFAULT_NAMED_ACCOUNTS = {
```

TYPESCRIPT

```
export const EMERGENCY_ADMIN: Record<string, string> = {
  [eEthereumNetwork.main]: "0xd0E0BFd1CD4f7A7A4c79A51aE60c5e9A1FB8A9AD",
  [eBaseNetwork.base]: "0xd0E0BFd1CD4f7A7A4c79A51aE60c5e9A1FB8A9AD",
  [eRobinhoodNetwork.robinhoodTestnet]: "0xd0E0BFd1CD4f7A7A4c79A51aE60c5e9A1FB8A9AD",
};
```

L03: Stale Base ERC20 token addresses used as Chainlink oracle entries in commons.ts base config

LOCATION

edel-deployment-script/main/markets/aave/commons.ts:L30-L38

RELEVANT SNIPPET

>_ commons.ts — TYPESCRIPT

```
28:  WrappedNativeTokenSymbol: "WETH",
29:  ChainlinkAggregator: {
30:    [eEthereumNetwork.main]: {
31:      USDT: "0x3E7d1eAB13ad0104d2750B8863b489D65364e32D",
32:      SPYon: "0x16Cbb13CcB1e3d951821Ad8CB67667111176B2C6",
33:      QQQon: "0x66E8D8E1ba5cfaDB32df6CC0B45eA05Cc3d7201E",
34:      GOOGLon: "0x07C6a25739Ffe02b1dae12502632126fFA7497c2",
35:      COINon: "0xA0Aa9Dd11c6a770cEbB4772728538648F2de0F82",
36:      TSLAon: "0x367A8A0A55f405AA6980e44f3920463ABC6BB132",
37:      NVDAon: "0xFBEfaE5034AA4cc7f3E9ac17E56d761a1bF211D4",
38:    },
39:  },
40:  ReserveFactorTreasuryAddress: {},
```

DESCRIPTION

Five of seven **ChainlinkAggregator** entries in commons.ts#L30-L38 point to Base network mock ERC20 token addresses, not Chainlink price feeds. For example, **QQQon** is set to **0x66E8...201E** which is the **mockUSDC** token on Base per base/index.ts#L45, and **GOOGLon** is set to **0x07C6...7c2** which is **mockSPY** on Base per L46.

>_ commons.ts — TYPESCRIPT

```
28:  WrappedNativeTokenSymbol: "WETH",
29:  ChainlinkAggregator: {
30:    [eEthereumNetwork.main]: {
31:      USDT: "0x3E7d1eAB13ad0104d2750B8863b489D65364e32D",
32:      SPYon: "0x16Cbb13CcB1e3d951821Ad8CB67667111176B2C6",
33:      QQQon: "0x66E8D8E1ba5cfaDB32df6CC0B45eA05Cc3d7201E",
34:      GOOGLon: "0x07C6a25739Ffe02b1dae12502632126fFA7497c2",
35:      COINon: "0xA0Aa9Dd11c6a770cEbB4772728538648F2de0F82",
36:      TSLAon: "0x367A8A0A55f405AA6980e44f3920463ABC6BB132",
37:      NVDAon: "0xFBEfaE5034AA4cc7f3E9ac17E56d761a1bF211D4",
38:    },
39:  },
40:  ReserveFactorTreasuryAddress: {},
```

>_ index.ts TYPESCRIPT

```
43:  ReserveAssets: {
44:    [eBaseNetwork.base]: {
45:      mockUSDC: "0x66E8D8E1ba5cfaDB32df6CC0B45eA05Cc3d7201E",
46:      mockSPY: "0x07C6a25739Ffe02b1dae12502632126fFA7497c2",
47:      mockQQQ: "0xA0Aa9Dd11c6a770cEbB4772728538648F2de0F82",
```

>_ index.ts TYPESCRIPT

```
44:  [eBaseNetwork.base]: {
45:    mockUSDC: "0x66E8D8E1ba5cfaDB32df6CC0B45eA05Cc3d7201E",
46:    mockSPY: "0x07C6a25739Ffe02b1dae12502632126fFA7497c2",
47:    mockQQQ: "0xA0Aa9Dd11c6a770cEbB4772728538648F2de0F82",
48:    mockGOOGL: "0x367A8A0A55f405AA6980e44f3920463ABC6BB132",
```

All three active markets override **ChainlinkAggregator** entirely, ethereum/index.ts#L36-L45, base/index.ts#L65-L79, and robinhood/index.ts#L40-L49. The broken values only survive if someone deploys with **MARKET_NAME=Aave**, which returns **AaveMarket** directly from loadPoolConfig. **No current deployment uses this path.**

>_ index.ts TYPESCRIPT

```
34:    }
35:  },
36:  ChainlinkAggregator: {
37:    [eEthereumNetwork.main]: {
38:      TSLAx: "0x13536966973F5F463918cEf7CD6E1545cD790116",
39:      SPYx: "0x52A8A8362aC83D22A52Cf01C822a94504630C7bB",
40:      QQQx: "0xf0EF3A07B7548C5BdF9e120dCFE7222416eb0af2",
41:      NVDAx: "0x52a8D617B63dB1235cc957C4B97439c2d0A6aAC2",
42:      MSTRx: "0x7ADe622936Fb71dce65a120E01C11A647304ac7c",
43:      USDC: "0x8fFfFfd4AfB6115b954Bd326cbe7B4BA576818f6"
44:    }
45:  },
46:  ReservesConfig: {
47:    TSLAX: strategyTSLAx,
```

>_ index.ts TYPESCRIPT

```
63:    },
64:    EModes: {},
65:    ChainlinkAggregator: {
66:      [eBaseNetwork.base]: {
67:        mockUSDC: "0x7e860098F58bBFC8648a4311b374B1D669a2bc6B",
68:        mockSPY: "0xA703fFe45007c569bEB16B0230D41402E13A1b4c",
69:        mockQQQ: "0x70002Ee831a5e7F4229aa2C14E7a3B5cCf272cfC",
70:        mockGOOGL: "0x43e606Cc92194528faE829E2e23abf0fdC40BD41",
71:        mockAAPL: "0x97B917f5385c9585F886d066d8F0d4Cc0C22A28d",
72:        mockNVDA: "0x7dD14bF8A910e24Ce72A709AB735FB01065274AF",
73:        mockPLTR: "0x4Dac0ca4b4d7a62F39233A2D6DE0A9a760F44968",
74:        mockAMZN: "0xC8E52E81899F2972F8b6FB1BA376A2d526dc1D32",
75:        mockHOOD: "0x4c5d93E92f38ca68ceDa9eE2021Ad02EfF2FA298",
76:        mockCRCL: "0x00373B43BE5f9B1D82FaD81b8c3316340E25c88A",
77:        mockMETA: "0xE489B6709d5A07334349343D1c9119647e3B228B",
78:        mockTSLA: "0x54aAfA1AEA49E178AC12F80bd47AB6818aFCDbaf",
```

>_ index.ts TYPESCRIPT

```
    },
  },
};
```

>_ index.ts TYPESCRIPT

```
  },
  EModes: {},
  ChainlinkAggregator: {
    [eRobinhoodNetwork.robinhoodTestnet]: {
      WETH: "0x0000000000000000000000000000000000000000",
      TSLA: "0x0000000000000000000000000000000000000000",
      AMZN: "0x0000000000000000000000000000000000000000",
      PLTR: "0x0000000000000000000000000000000000000000",
      NFLX: "0x0000000000000000000000000000000000000000",
      AMD: "0x0000000000000000000000000000000000000000",
    }
  },
};

```

>_ market-config-helpers.ts TYPESCRIPT

```
export const loadPoolConfig = (configName: ConfigNames): PoolConfiguration => {
  switch (configName) {
    case ConfigNames.Aave:
      return AaveMarket;
    case ConfigNames.Ethereum:
      return EthereumV3Config;
```

If it were used, **latestRoundData()** calls against ERC20 contracts would revert, making the oracle non-functional.

RECOMMENDATION

Replace the stale addresses in commons.ts#L30-L38 with valid Chainlink feed addresses, or remove the entries entirely since all active markets override them.

>_ commons.ts TYPESCRIPT

```
WrappedNativeTokenSymbol: "WETH",
ChainlinkAggregator: {
  [eEthereumNetwork.main]: {
    USDT: "0x3E7d1eAB13ad0104d2750B8863b489D65364e32D",
    SPYon: "0x16Cbb13CcB1e3d951821Ad8CB67667111176B2C6",
    QQQon: "0x66E8D8E1ba5cfaDB32df6CC0B45eA05Cc3d7201E",
    GOOGLon: "0x07C6a25739Ffe02b1dae12502632126fFA7497c2",
    COINon: "0xA0Aa9Dd11c6a770cEbB4772728538648F2de0F82",
    TSLAon: "0x367A8A0A55f405AA6980e44f3920463ABC6BB132",
    NVDAon: "0xFBEfaE5034AA4cc7f3E9ac17E56d761a1bF211D4",
  },
},
ReserveFactorTreasuryAddress: {},
```

L04: Hardcoded ethereum config in add-reserve task will initialize wrong assets and oracles into non-ethereum markets

LOCATION

edel-deployment-script/main/tasks/misc/add-reserve.ts:L30

RELEVANT SNIPPET

>_ add-reserve.ts — TYPESCRIPT

```
28: const { deployer } = await hre.getNamedAccounts();
29:
30: const poolConfig = loadPoolConfig(ConfigNames.Ethereum) as IAaveConfiguration;
31: const {
32:   ATokenNamePrefix,
```

DESCRIPTION

add-reserve.ts#L30 hardcodes **ConfigNames.Ethereum** instead of reading **MARKET_NAME** from the environment

>_ add-reserve.ts — TYPESCRIPT

```
28: const { deployer } = await hre.getNamedAccounts();
29:
30: const poolConfig = loadPoolConfig(ConfigNames.Ethereum) as IAaveConfiguration;
31: const {
32:   ATokenNamePrefix,
```

TYPESCRIPT

```
const poolConfig = loadPoolConfig(ConfigNames.Ethereum) as IAaveConfiguration;
```

Every other task & deploy script in the repo uses loadPoolConfig(MARKET_NAME as ConfigNames) ... for example setup-debt-ceiling.ts#L17, & setup-isolation-mode.ts#L16.

>_ setup-debt-ceiling.ts — TYPESCRIPT

```
15: ).setAction(async (_, hre) => {
16:   const { poolAdmin } = await hre.getNamedAccounts();
17:   const config = await loadPoolConfig(MARKET_NAME as ConfigNames);
18:
19:   const poolConfigurator = (await getPoolConfiguratorProxy()).connect(
```

>_ setup-isolation-mode.ts TYPESCRIPT

```
14: ).setAction(async (_, hre) => {
15:   const { poolAdmin } = await hre.getNamedAccounts();
16:   const config = await loadPoolConfig(MARKET_NAME as ConfigNames);
17:   const poolConfigurator = (await getPoolConfiguratorProxy()).connect(
18:     await hre.ethers.getSigner(poolAdmin)
```

When an operator runs the task, the MARKET_NAME environment variable is ignored and loads markets/ethereum/index.ts instead of the correct market config (e.g Base). This sources the wrong ReservesConfig, ReserveAssets, ChainlinkAggregator, RateStrategies, and token name prefixes.

RECOMMENDATION

Replace the hardcoded config at **add-reserve.ts#L30** with the environment-driven pattern used everywhere else

TYPESCRIPT

```
import { MARKET_NAME } from "../../helpers/env";



const poolConfig = loadPoolConfig(MARKET_NAME as ConfigNames) as IAaveConfiguration;
```

4. Enhancement Opportunities

E01: Hardcoded "Edel" prefix in initReservesByHelper ignores market-specific token name parameters

LOCATION

edel-deployment-script/main/helpers/init-helpers.ts:L192-L197

RELEVANT SNIPPET

>_ init-helpers.ts — TYPESCRIPT

```
190:   incentivesController,
191:   underlyingAssetName: reserveSymbols[i],
192:   aTokenName: `Edel ${reserveSymbols[i]}`,
193:   aTokenSymbol: `e${reserveSymbols[i]}`,
194:   variableDebtTokenName: `Edel Variable Debt ${reserveSymbols[i]}`,
195:   variableDebtTokenSymbol: `variableDebt${reserveSymbols[i]}`,
196:   stableDebtTokenName: `Edel Stable Debt ${reserveSymbols[i]}`,
197:   stableDebtTokenSymbol: `stableDebt${reserveSymbols[i]}`,
198:   params: "0x10",
199: });
```

DESCRIPTION

initReservesByHelper accepts **aTokenNamePrefix**, **stableDebtTokenNamePrefix**, **variableDebtTokenNamePrefix**, and **symbolPrefix** as parameters, but L192-L197 ignores them all and hardcodes **"Edel"**:

>_ init-helpers.ts — TYPESCRIPT

```
42: declare var hre: HardhatRuntimeEnvironment;
43:
44: export const initReservesByHelper = async (
45:   reservesParams: iMultiPoolsAssets<IReserveParams>,
46:   tokenAddresses: { [symbol: string]: tEthereumAddress },
47:   aTokenNamePrefix: string,
48:   stableDebtTokenNamePrefix: string,
49:   variableDebtTokenNamePrefix: string,
50:   symbolPrefix: string,
51:   admin: tEthereumAddress,
52:   treasuryAddress: tEthereumAddress,
53:   incentivesController: tEthereumAddress
54: ) => {
55:   const poolConfig = (await loadPoolConfig(
```

>_ init-helpers.ts TYPESCRIPT

```
  incentivesController,
  underlyingAssetName: reserveSymbols[i],
  aTokenName: `Edel ${reserveSymbols[i]}`,
  aTokenSymbol: `e${reserveSymbols[i]}`,
  variableDebtTokenName: `Edel Variable Debt ${reserveSymbols[i]}`,
  variableDebtTokenSymbol: `variableDebt${reserveSymbols[i]}`,
  stableDebtTokenName: `Edel Stable Debt ${reserveSymbols[i]}`,
  stableDebtTokenSymbol: `stableDebt${reserveSymbols[i]}`,
  params: "0x10",
});
```

TYPESCRIPT

```
aTokenName: `Edel ${reserveSymbols[i]}`,
variableDebtTokenName: `Edel Variable Debt ${reserveSymbols[i]}`,
stableDebtTokenName: `Edel Stable Debt ${reserveSymbols[i]}`,
```

Base market sets **ATokenNamePrefix: "Base"** at base/index.ts#L24 and gets **"Edel"** prefixed tokens, making cross-market token identification ambiguous.

>_ index.ts TYPESCRIPT

```
...AaveMarket,
MarketId: "Base Edel Market",
ATokenNamePrefix: "Base",
StableDebtTokenNamePrefix: "Base",
VariableDebtTokenNamePrefix: "Base",
```

POTENTIAL BENEFIT

Prevent naming confusion.

RECOMMENDATION

Use the passed parameters here.

>_ init-helpers.ts TYPESCRIPT

```
  incentivesController,
  underlyingAssetName: reserveSymbols[i],
  aTokenName: `Edel ${reserveSymbols[i]}`,
  aTokenSymbol: `e${reserveSymbols[i]}`,
  variableDebtTokenName: `Edel Variable Debt ${reserveSymbols[i]}`,
  variableDebtTokenSymbol: `variableDebt${reserveSymbols[i]}`,
  stableDebtTokenName: `Edel Stable Debt ${reserveSymbols[i]}`,
  stableDebtTokenSymbol: `stableDebt${reserveSymbols[i]}`,
  params: "0x10",
});
```

```typescript
aTokenName: `${aTokenNamePrefix} ${reserveSymbols[i]}`,
aTokenSymbol: `${symbolPrefix}${reserveSymbols[i]}`,
variableDebtTokenName: `${variableDebtTokenNamePrefix} Variable Debt ${reserveSymbols[i]}`,
stableDebtTokenName: `${stableDebtTokenNamePrefix} Stable Debt ${reserveSymbols[i]}`,
```

E02: Unused L2Pool helper links EModeLogic as CalldataLogic, creating a landmine for future use

LOCATION

edel-deployment-script/main/helpers/contract-deployments.ts:L614

RELEVANT SNIPPET

>_ contract-deployments.ts — TYPESCRIPT

```
612:  ): Promise<L2Pool> => {
613:    const commonLibraries = await getPoolLibraries();
614:    const CalldataLogic = await (await hre.deployments.get("EModeLogic")).address;
615:
616:    return deployContract<L2Pool>("L2Pool", [addressesProviderAddress], {
```

DESCRIPTION

contract-deployments.ts#L614 fetches **EModeLogic**'s address and passes it as **CalldataLogic**

>_ contract-deployments.ts — TYPESCRIPT

```
612:  ): Promise<L2Pool> => {
613:    const commonLibraries = await getPoolLibraries();
614:    const CalldataLogic = await (await hre.deployments.get("EModeLogic")).address;
615:
616:    return deployContract<L2Pool>("L2Pool", [addressesProviderAddress], {
```

TYPESCRIPT

```
const CalldataLogic = await (await hre.deployments.get("EModeLogic")).address;
```

These are two entirely different libraries. Any code path using **deployL2DeployerImplementation()** would produce a broken L2Pool with mislinked logic.

This helper is currently never called, the actual Base L2Pool deployment uses 01b_l2_pool_implementation.ts#L44-L55, which correctly deploys and links **CalldataLogic**. No deployed contracts are affected.

POTENTIAL BENEFIT

Prevent future security risk.

RECOMMENDATION

Fix the library lookup at L614

>_ contract-deployments.ts — TYPESCRIPT

```
612:  ): Promise<L2Pool> => {
613:    const commonLibraries = await getPoolLibraries();
614:    const CalldataLogic = await (await hre.deployments.get("EModeLogic")).address;
615:
616:    return deployContract<L2Pool>("L2Pool", [addressesProviderAddress], {
```

```typescript
const CalldataLogic = await (await hre.deployments.get("CalldataLogic")).address;
```

About Us

About Adevar Labs

Adevar Labs is a boutique blockchain security firm specializing in web3 audits.

Built by a mix of experienced professionals in traditional enterprise and crypto natives who have contributed to some of the most critical projects in blockchain infrastructure.

Our team's background spans companies like Bitdefender, Asymmetric Research, Quantstamp, Chainproof, and Juicebox, and includes experience securing smart contracts, bridges, and L1 and L2 protocols across ecosystems like Solana, Ethereum, Polkadot, Cosmos, and MultiversX.

With over 100 audits completed and a portfolio that includes custom fuzzers, exploit modeling, and runtime testing frameworks, Adevar Labs brings both depth and precision to every engagement.

Our auditors have discovered critical vulnerabilities, built high-impact tooling, and placed in top positions in premier audit competitions including Code4rena and Sherlock.

Team members hold distinctions such as PhDs in software protection, and key roles at Fortune 500 companies, and leadership of flagship conferences like ETH Bucharest.

With team members having publications with over 1,100 academic citations and trusted by projects securing over $500M in on-chain value, Adevar Labs blends elite technical rigor with real-world security impact.

We also collaborate with some of the best independent security researchers in the web3 space.

Projects may optionally request specific contributors to be part of their audit.

Audit Methodology

Our audit methodology is specialized to provide thorough security assessments of Ethereum smart contracts. We focus explicitly on rigorous manual analysis, detailed threat modeling, and careful validation of implemented fixes to ensure your smart contracts operate securely and as intended.

1. Deployment Context and Architecture Analysis

Our auditors begin by deeply examining your deployment framework documentation and intended operational model. We meticulously map out deployment sequencing, script dependencies, environment-variable usage, and post-deploy execution paths. Special attention is given to ownership transfer flows, role assignment logic, oracle source wiring, and how market-specific parameters are propagated across networks to ensure the deployed system matches the intended architecture.

2. Threat Modeling

We conduct targeted threat modeling tailored specifically for deployment scripts and operational tooling in the EVM ecosystem. We carefully define attacker capabilities and identify potential vulnerabilities that may arise from deployment-time and configuration-time issues, including but not limited to:

- Misconfiguration of parameters that can make core protocol flows non-functional
- Incorrect ownership handoff and role-transfer logic for priveleged accounts
- Missing guards around modules and deployment artifacts, causing critical post-deploy tasks to fail
- Risks introduced by hardcoded values in scripts that reduce deployment reliability

3. In-depth Manual Security Review

Our experienced auditors perform an extensive manual review of the deployment scripts, market configuration files, and deployment artifacts. This involves a comprehensive inspection of Hardhat/TypeScript deployment logic, focusing on:

- Deployment script initialization ordering and privilege windows
- ACL role assignment correctness and ownership handoff completeness
- Oracle configuration mapping between reserve assets and price feed aggregators
- Market configuration consistency (reserve params, rate strategies, token symbol casing)
- Post-deploy operational task correctness and network guard coverage
- Deployment artifact cross-validation against script logic

During this stage, we clearly document any discovered vulnerabilities, including detailed descriptions, precise severity ratings, and recommendations for secure implementation. In situations where it is not clear how the vulnerability might be exploited we may also include a detailed proof-of-concept exploit including code snippets and instructions on how the exploit could be performed.

4. Detailed Fix Review and Validation

After the initial audit and your team's subsequent remediation efforts, we perform a comprehensive fix review to ensure the vulnerabilities identified have been effectively resolved. The fix review for this engagement is limited to the findings identified in this report and does not extend to areas explicitly deferred in the Scope Limitations section of the Executive Summary.

We verify each fix individually, confirming that:

- Corrections effectively eliminate the security risks
- Changes do not inadvertently introduce new vulnerabilities or regressions

Our detailed validation ensures that security improvements are robust, complete, and aligned with best practices specific to the EVM development ecosystem.

Confidentiality Notice

This report, including its content, data, and underlying methodologies, is subject to the confidentiality and feedback provisions in your agreement with Adevar Labs. These materials are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Adevar Labs.

Legal Disclaimer

The review and this report are provided by Adevar Labs on an as-is, where-is, and as-available basis. To the fullest extent permitted by law, Adevar Labs disclaims all warranties, expressed or implied, in connection with this report, its content, and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

You agree that access to and/or use of the report and other results of the review, including any associated services, products, protocols, platforms, content, and materials, will be at your sole risk.

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. This report is based on the scope of materials and documentation provided for a limited review at the time provided.

You acknowledge that blockchain technology remains under development and is subject to unknown risks and flaws. Adevar Labs does not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party, or any open-source or third-party software, code, libraries, materials, or information accessible through the report. As with the purchase or use of a product or service in any environment, you should use your best judgment and exercise caution where appropriate.