HypurrFi Security Review

Pashov Audit Group

Conducted by: Hals, unforgiven, merlinboii

February 12th 2025 - February 18th 2025

Contents

1. About Pashov Audit Group

Pashov Audit Group consists of multiple teams of some of the best smart contract security researchers in the space. Having a combined reported security vulnerabilities count of over 1000, the group strives to create the absolute very best audit journey possible - although 100% security can never be guaranteed, we do guarantee the best efforts of our experienced researchers for your blockchain protocol. Check our previous work here or reach out on Twitter @pashovkrum.

2. 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.

3. Introduction

A time-boxed security review of the **lastdotnet/hypurrfi-deployments** repository was done by **Pashov Audit Group**, with a focus on the security aspects of the application's smart contracts implementation.

4. About HypurrFi

HypurrFi is a leveraged lending marketplace on Hyperliquid, enabling clean leverage loops while maintaining spot positions on native assets like HYPE and stHYPE. Its stablecoin, $USDXL, is backed by protocol revenue and a growing reserve of tokenized U.S. Treasuries.

5. Risk Classification

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

5.1. Impact

- High - leads to a significant material loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost (such as leakage of value) or a core functionality of the protocol is affected.
- Low - can lead to any kind of unexpected behavior with some of the protocol's functionalities that's not so critical.

5.2. Likelihood

- High - attack path is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount of funds that can be stolen or lost.
- Medium - only a conditionally incentivized attack vector, but still relatively likely.
- Low - has too many or too unlikely assumptions or requires a significant stake by the attacker with little or no incentive.

5.3. Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

6. Security Assessment Summary

review commit hashes:

- 2509ece7be02e22c1db54e2238ba4c1715ca2bae
- df0d50f3a37f3c199214b6c1e460e390a7a03e17

fixes review commit hashes:

- 2490d3ca12a081a8c49981935c2b11eddcb5d519
- 93b453dd146069946dc0f96e563d4a110eae3c26

Scope

The following smart contracts were in scope of the audit:

- `ConfigurrHyFiReservesMainnet`
- `ConfigurrHyFiReservesTestnet`
- `DeployCapAutomator`
- `DeployHyFi`
- `DeployWHYPE`
- `SupplyHyFi`
- `TransferOwnership`
- `USDfSilo`
- `HyperTestnetReservesConfigs`
- `DeployHyFiUtils`
- `DeployUtils`
- `BorrowUsdxlHyperTestnet`
- `DeployUsdxlGsmHyperTestnet`
- `DeployUsdxlHyperTestnet`
- `RepayUsdxlHyperTestnet`
- `HyperTestnetReservesConfigs`
- `DeployUsdxlFileUtils`
- `DeployUsdxlUtils`

7. Executive Summary

Over the course of the security review, Hals, unforgiven, merlinboii engaged with HypurrFi to review HypurrFi. In this period of time a total of **24** issues were uncovered.

Protocol Summary

Protocol Name	HypurrFi
Repository	https://github.com/lastdotnet/hypurrfi-deployments
Date	February 12th 2025 - February 18th 2025
Protocol Type	Lending

Findings Count

Severity	Amount
High	3
Medium	8
Low	13
Total Findings	**24**

Summary of Findings

ID	Title	Severity	Status
[H-01]	DeployUsdxlUtils does not transfer ownership of usdxlToken to admin	High	Resolved
[H-02]	Deployer does not transfer ownership of CapAutomator to admin	High	Resolved
[H-03]	Incorrect proxy address tracking misconfigures USDXL pool tokens	High	Resolved
[M-01]	DeployHyFiConfigEngine: double deployment of proxyAdmin	Medium	Resolved
[M-02]	DeployUsdxlUtils: Wrong setting for mint limits	Medium	Acknowledged
[M-03]	ConfigurrHyFiReserves deployer lacks access to set configurations	Medium	Acknowledged
[M-04]	Incomplete borrowing settings in ConfigurrHyFiReserves script	Medium	Acknowledged
[M-05]	Pool reserves should be initialized and supplied in same transaction	Medium	Resolved
[M-06]	Missing token transfer before supplying to the pool in SupplyHyFi script	Medium	Acknowledged
[M-07]	DeployHyFiConfigEngine script fails on HyperEVM due to missing key	Medium	Resolved
[M-08]	Uncompilable DeployUsdxlHyperTestnet script	Medium	Resolved
[L-01]	SupplyHyFi: missing configuration for USDC and sUSDe	Low	Acknowledged

[L-02]	DeployUsdxlUtils: UsdxlInterestRateStrategy contract is deployed twice	Low	Acknowledged
[L-03]	Remove deprecated Göerli testnet files from deployment	Low	Acknowledged
[L-04]	Incorrect tokenName set during hyTokenImpl initialization	Low	Acknowledged
[L-05]	_deployUsdxl() doesn’t initialize usdxlVariableDebtToken	Low	Acknowledged
[L-06]	_deployUsdxl() doesn't initialize usdxlAToken	Low	Acknowledged
[L-07]	Missing functionalities in RepayUsdxlHyperTestnet and BorrowUsdxlHyperTestnet scripts	Low	Acknowledged
[L-08]	TODO resolution required for GSM proxy admin and unique interest rate strategy handling	Low	Acknowledged
[L-09]	DeployUsdxlUtils: _deployGsm() should use usdxlToken's proxy address instead of implementation	Low	Acknowledged
[L-10]	GSM contract is not deployed	Low	Acknowledged
[L-11]	DeployUsdxlUtils: usdxlAToken and usdxlVariableDebtToken contracts are deployed twice	Low	Acknowledged
[L-12]	CapAutomator does not have riskAdmin access in aclManager	Low	Acknowledged
[L-13]	Uncompilable DeployUsdxlGsmHyperTestnet script	Low	Acknowledged

8. Findings

8.1. High Findings

[H-01] `DeployUsdxlUtils` does not transfer ownership of usdxlToken to `admin`

Severity

Impact: Medium

Likelihood: High

Description

The `_deployUsdxl` function, deploys usdxlTokenProxy and sets `deployer` as the owner of `usdxlToken`

```
function _deployUsdxl(
      addressproxyAdmin,
      IDeployConfigTypes.HypurrDeployRegistrymemorydeployRegistry
    ) internal {
        --snip--
        // 1. Deploy USDXL token implementation and proxy
        UpgradeableUsdxlToken usdxlTokenImpl = new UpgradeableUsdxlToken();

        bytes memory initParams = abi.encodeWithSignature("initialize
          (address)", deployer);

        usdxlTokenProxy = address(new TransparentUpgradeableProxy(address
          (usdxlTokenImpl), proxyAdmin, initParams));

        usdxlToken = IUsdxlToken(usdxlTokenProxy);

        --snip--
     }
```

But it does not transfer the ownership (admin rights) from `deployer` to `admin`

Recommendations

Transfer ownership of `usdxlToken` to admin after deployment and config

[H-02] Deployer does not transfer ownership of `CapAutomator` to admin

Severity

Impact: Medium

Likelihood: High

Description

The function `DeployCapAutomator.run` deploys an instance of `CapAutomator`, assigning `msg.sender` (the `deployer`) as the initial owner. But it does not transfer the ownership to the designated `admin`.

```
function run() external {
        --snip--
        poolAddressesProvider = IPoolAddressesProvider
          (deployedContracts.readAddress(".poolAddressesProvider"));

        vm.startBroadcast(vm.envUint("PRIVATE_KEY"));

        capAutomator = new CapAutomator(address(poolAddressesProvider));

        vm.stopBroadcast();
        --snip--
    }
```

```
contract CapAutomator is ICapAutomator, Ownable {
    --snip--
    constructor(address poolAddressesProvider) Ownable(msg.sender) {
        pool              = IPool(IPoolAddressesProvider
          (poolAddressesProvider).getPool());
        poolConfigurator  = IPoolConfigurator(IPoolAddressesProvider
          (poolAddressesProvider).getPoolConfigurator());
    }
```

Recommendations

Transfer ownership of `capAutomator` to `admin` after deployment.

[H-03] Incorrect proxy address tracking misconfigures USDXL pool tokens

Severity

Impact: Medium

Likelihood: High

Description

The `DeployUsdxlUtils._getUsdxlATokenProxy()` and `DeployUsdxlUtils._getUsdxlVariableDebtTokenProxy()` functions incorrectly return implementation contract addresses instead of proxy addresses.

```
//File: src/deployments/utils/DeployUsdxlUtils.sol

function _getUsdxlATokenProxy() internal view returns (address) {
    return address(usdxlAToken);     // Returns implementation instead of proxy
}

function _getUsdxlVariableDebtTokenProxy() internal view returns (address) {
    return address
    //(usdxlVariableDebtToken); // Returns implementation instead of proxy
}
```

This causes four main issues:

1. Incorrect contract exports in deployment artifacts in the `_initializeUsdxlReserve()` function.
2. Incorrect token configurations in the `_setUsdxlAddresses()` function, leaving the USDXL pool's `AToken` and `VariableDebtToken` unconfigured.
3. Incorrect facilitator configurations for the USDXL token in the `_addUsdxlATokenAsEntity()` function.
4. Incorrect discount token and strategy configurations in the `_setDiscountTokenAndStrategy()` function.

Recommendation

Track the actual proxy addresses that are configured in the USDXL pool instead of using implementation addresses. This ensures that token configurations are applied to the correct contract instances that the pool interacts with.

To implement this:

1. Get and track proxy addresses from pool's reserve data after pool initialization:

```
function _initializeUsdxlReserve(
    address token,
    IDeployConfigTypes.HypurrDeployRegistry memory deployRegistry
)
    internal
{
    --- SNIPPED ---
    // set reserves configs
    _getPoolConfigurator(deployRegistry).initReserves(inputs);

+   IPoolAddressesProvider poolAddressesProvider = _getPoolAddressesProvider
+ (deployRegistry);
    //@audit DataTypes should be additional imported
+   DataTypes.ReserveData memory reserveData = IPool
+ (poolAddressesProvider.getPool()).getReserveData(token);

    //@audit Introduce new two state variables to track proxy addresses
+   usdxlATokenProxy = UsdxlAToken(reserveData.aTokenAddress);
+   usdxlVariableDebtTokenProxy = UsdxlVariableDebtToken
+ (reserveData.variableDebtTokenAddress);

    // export contract addresses
    DeployUsdxlFileUtils.exportContract
      (instanceId, "usdxlATokenProxy", _getUsdxlATokenProxy());
    DeployUsdxlFileUtils.exportContract(
      instanceId,
      "usdxlVariableDebtTokenProxy",
      _getUsdxlVariableDebtTokenProxy
    )
}
```

2. Update getter functions to return proxy addresses:

```
function _getUsdxlATokenProxy() internal view returns (address) {
-    return address(usdxlAToken);
+    return address(usdxlATokenProxy);
}

function _getUsdxlVariableDebtTokenProxy() internal view returns (address) {
-    return address(usdxlVariableDebtToken);
+    return address(usdxlVariableDebtTokenProxy);
}
```

3. Update treasury configuration to use proxy:

```
function _setUsdxlAddresses
  (IDeployConfigTypes.HypurrDeployRegistry memory deployRegistry)
    internal
{
-    usdxlAToken.updateUsdxlTreasury(deployRegistry.treasury);
+    UsdxlAToken(_getUsdxlATokenProxy()).updateUsdxlTreasury
+ (deployRegistry.treasury);

    UsdxlAToken(_getUsdxlATokenProxy()).setVariableDebtToken
      (_getUsdxlVariableDebtTokenProxy());
    UsdxlVariableDebtToken(_getUsdxlVariableDebtTokenProxy()).setAToken
      (_getUsdxlATokenProxy());
}
```

8.2. Medium Findings

[M-01] DeployHyFiConfigEngine: double deployment of `proxyAdmin`

Severity

Impact: Medium

Likelihood: Medium

Description

`DeployHyFiConfigEngine.run` creates a `ProxyAdmin` using `transparentProxyFactory` :

```
function run() external {
        --snip--
        transparentProxyFactory = new TransparentProxyFactory();
        proxyAdmin = ProxyAdmin(transparentProxyFactory.createProxyAdmin
          (admin));

        (ratesFactory,) = DeployRatesFactoryLib._createAndSetupRatesFactory(
             poolAddressesProvider, address(transparentProxyFactory), address
               (proxyAdmin), reservesToSkip);
       --snip--
     }
```

then calls `_createAndSetupRatesFactory` and passes the address of `proxyAdmin` as `ownerForFactory`:

```
function _createAndSetupRatesFactory(
        IPoolAddressesProvider addressesProvider,
        address transparentProxyFactory,
        address ownerForFactory,
        address[] memory reservesToSkip
    ) internal returns (V3RateStrategyFactory, address[] memory) {
        --snip--
        V3RateStrategyFactory ratesFactory = V3RateStrategyFactory(
            ITransparentProxyFactory(transparentProxyFactory).create(
                address(new V3RateStrategyFactory(addressesProvider)),
                ownerForFactory,
                abi.encodeWithSelector
                  (V3RateStrategyFactory.initialize.selector, uniqueStrategies)
            )
        );
       --snip--
}
```

It calls `ITransparentProxyFactory(transparentProxyFactory).create` and passes the address of `ownerForFactory` (already deployed `proxyAdmin`) as `initialOwner`: The problem is that `create` function expects the address of owner and deploys its own `adminProxy`:

https://github.com/bgd-labs/solidity-utils/blob/90266e46868fe61ed0b54496c10458c247acdb51/src/contracts/transparent-proxy/TransparentProxyFactoryBase.sol#L29

```
function create(
    address logic,
    address initialOwner,
    bytes calldata data
  ) external returns (address) {
    address proxy = address(new TransparentUpgradeableProxy
      (logic, initialOwner, data));
    _storeProxyInRegistry(proxy);

    emit ProxyCreated(proxy, logic, initialOwner);

    return proxy;
  }
```

So the pattern will be like: proxyAdmin(1) > proxyAdmin(2) > transparentProxy > Impl As a result, the admin will not be able to upgrade the contract.

Note: `import {ITransparentProxyFactory} from "solidity-utils/contracts/transparent-proxy/interfaces/ITransparentProxyFactory.sol";` The code for the above interface is here: https://github.com/bgd-labs/solidity-utils/blob/main/src/contracts/transparent-proxy/interfaces/ITransparentProxyFactory.sol

Recommendations

Dont deploy a separate `proxyAdmin` and just pass address of `admin` to `create` function.

[M-02] `DeployUsdxlUtils`: Wrong setting for mint limits

Severity

Impact: Medium

Likelihood: Medium

Description

Functions `_addUsdxlATokenAsEntity()` and `_addUsdxlFlashMinterAsEntity()` set mint limit as 1B instead of 100mil:

```
function _addUsdxlATokenAsEntity
     (IDeployConfigTypes.HypurrDeployRegistry memory deployRegistry)
        internal
    {
        // pull aToken proxy from reserves config
        _getUsdxlToken().addFacilitator(
          address(_getUsdxlATokenProxy()),
          'HypurrFi Market Loans', // entity label
          1e27 // entity mint limit (100mil)
        );
    }

    function _addUsdxlFlashMinterAsEntity
      (IDeployConfigTypes.HypurrDeployRegistry memory deployRegistry)
        internal
    {
      _getUsdxlToken().addFacilitator(
        address(flashMinter),
        'HypurrFi Market Flash Loans', // entity label
        1e27 // entity mint limit (100mil)
      );
    }
```

Recommendations

Use 1e26 instead of 1e27 to set it to 100mil

[M-03] ConfigurrHyFiReserves deployer lacks access to set configurations

Severity

Impact: Low

Likelihood: High

Description

`ConfigurrHyFiReserves.run` calls different functions to set configurations in `poolConfigurator` but `deployer` no longer has access to make these changes:

```
function run() external {
        --snip--
        // set oracles
        _getAaveOracle().setAssetSources(tokens, oracles);

        // set reserve config
        _initReserves(tokens);

        // disable stable debt
        _disableStableDebt(tokens);

        // enable collateral
        _enableCollateral(tokens);

        // enable borrowing
        _enableBorrowing(tokens);

        vm.stopBroadcast();
    }
```

For example it calls `_initReserves` to set reserve config:

```
function _initReserves(address[] memory tokens) internal {
         --snip--
        // set reserves configs
        _getPoolConfigurator().initReserves(inputs);
    }
```

It calls `poolConfigurator.initReserves` which is restricted to `onlyAssetListingOrPoolAdmins`:

```
function initReserves(
    ConfiguratorInputTypes.InitReserveInput[] calldata input
  ) external override onlyAssetListingOrPoolAdmins {
    IPool cachedPool = _pool;
    for (uint256 i = 0; i < input.length; i++) {
      ConfiguratorLogic.executeInitReserve(cachedPool, input[i]);
    }
  }

  function _onlyAssetListingOrPoolAdmins() internal view {
    IACLManager aclManager = IACLManager(_addressesProvider.getACLManager());
    require(
      aclManager.isAssetListingAdmin(msg.sender) || aclManager.isPoolAdmin
        (msg.sender),
      Errors.CALLER_NOT_ASSET_LISTING_OR_POOL_ADMIN
    );
  }
```

The problem is that the deployer no longer has `poolAdmin` access, because at the end of `DeployHyFi._deployHyFi` it transfers ownerships to `admin`:

```
function _deployHyFi() internal {
        --snip--
        // 16. Transfer all ownership from deployer to admin

        aclManager.addEmergencyAdmin(admin);
        aclManager.addPoolAdmin(admin);
        if (admin != deployer) {
            aclManager.removePoolAdmin(deployer);
        }
        aclManager.grantRole(aclManager.DEFAULT_ADMIN_ROLE(), admin);
        if (admin != deployer) {
            aclManager.revokeRole(aclManager.DEFAULT_ADMIN_ROLE(), deployer);
        }

        poolAddressesProvider.setACLAdmin(admin);
        poolAddressesProvider.transferOwnership(admin);

        registry.transferOwnership(admin);
        emissionManager.transferOwnership(admin);
```

Note: Same is true with other functions: `_disableStableDebt`, `_enableCollateral`, `_enableBorrowing` also in `_getAaveOracle().setAssetSources(tokens, oracles);` which needs admin access in the oracle.

Recommendations

Revoke the deployer's admin privileges after completing all deployments and configurations.

[M-04] Incomplete borrowing settings in `ConfigurrHyFiReserves` script

Severity

Impact: Low

Likelihood: High

Description

In the `ConfigurrHyFiReserves` script, when borrowing is enabled for an asset, the script only activates the reserve borrowing by calling `setReserveBorrowing(token, true)`. However, it doesn't address the following essential settings:

- Borrowing cap: The script does not set the borrowing cap via `_getPoolConfigurator().setBorrowCap()`.
- Supply cap: The script does not set the supply cap via `_getPoolConfigurator().setSupplyCap()`.
- Reserve flash loaning: The script doesn't enable reserve flash loaning via `_getPoolConfigurator().setReserveFlashLoaning()`.
- Reserve factor: The script doesn't set the reserve factor via `_getPoolConfigurator().setReserveFactor()`.

As a result, the borrowing configuration is incomplete, leaving the pool with missing settings, which could lead to inconsistencies in how the borrowing mechanism functions.

Recommendations

Update `ConfigurrHyFiReserves` script to include calls to set all the aforementioned missing sets to fully enable asset borrowing.

[M-05] Pool reserves should be initialized and supplied in same transaction

Severity

Impact: High

Likelihood: Low

Description

Currently, for the **HyperEVM testnet**, the pool is initialized with the reserve tokens in the `ConfigurrHyFiReserves` script, and the tokens are supplied in a different script, `SupplyHyFi`. This approach leaves the system vulnerable to a **inflation attack** by the first depositor on an empty reserve. Ideally, both actions (initializing and supplying reserves) should happen in the same transaction to ensure that the system is correctly configured and cannot be exploited by an attacker who may manipulate the pool before the liquidity is added.

Instances:

- **USDC** and **sUSDe** tokens are supplied to the pool, but their respective reserves are not initialized by any of the deployment scripts as the `ConfigurrHyFiReserves` script only initializes the **KHYPE** token reserves.
- The **KHYPE reserve** is initialized by the `ConfigurrHyFiReserves` script but not supplied with liquidity.

Recommendations

Update the deployment process so that the pool reserves are both initialized and supplied with a minimum liquidity (seed amount) in the same transaction.

[M-06] Missing token transfer before supplying to the pool in `SupplyHyFi` script

Severity

Impact: Low

Likelihood: High

Description

In the current implementation of the `SupplyHyFi` script, tokens are not transferred to the script contract before being supplied to the pool. Since the script is using `vm.startBroadcast(vm.envUint("PRIVATE_KEY"))`, tokens should be transferred from the `PRIVATE_KEY` associated address to the contract first. Without transferring the tokens, the script will revert when it attempts to supply tokens to the pool, as no tokens are available in the contract.

Recommendations

Before supplying tokens to the pool, ensure that tokens are transferred from the sender (the address associated with the `PRIVATE_KEY`) to the script contract. However, this requires restricting the deployment script to an authorized address to prevent draining the holder of the `PRIVATE_KEY`. Alternatively, update the `SupplyHyFi` script to use `vm.startBroadcast()` with the correct caller to prevent the draining issue, so that tokens are pulled from the caller instead.

[M-07] `DeployHyFiConfigEngine` script fails on HyperEVM due to missing key

Severity

Impact: Low

Likelihood: High

Description

In the `DeployHyFiConfigEngine` script, when the script is run on the **HyperEVM testnet**, it attempts to read the `daiToken` address using the path `config.readAddress(".daiToken")` from the `hypurrfi-testnet.json` file. However, the `daiToken` key is not present in the JSON file, which causes Foundry's parser to fail with the error message:

```
FAIL: vm.parseJsonAddress: path ".daiToken" must return exactly one JSON value
```

This error occurs because the requested key (`.daiToken`) does not exist in the `hypurrfi-testnet.json` file, and as a result, Foundry is unable to find the expected value, which prevent the script from running successfully on the **HyperEVM testnet**.

```
function run() external {
        instanceId = vm.envOr("INSTANCE_ID", string("primary"));
       //...
        config = DeployUtils.readInput(instanceId);
       //...
        reservesToSkip[0] = config.readAddress(".daiToken");
        //...
 }
```

Recommendations

Ensure that the `daiToken` key is added to the `hypurrfi-testnet.json` file with a valid address or adjust the script to handle missing keys more gracefully.

[M-08] Uncompilable `DeployUsdxlHyperTestnet` script

Severity

Impact: Low

Likelihood: High

Description

The `DeployUsdxlHyperTestnet` script attempts to use `usdxlConfig` for deployment configuration but fails to declare it as a state variable. This causes compilation failures and renders the deployment script unusable.

```
//File: script/DeployUsdxlHyperTestnet.sol

function _deploy() internal {
    vm.setEnv('FOUNDRY_ROOT_CHAINID', vm.toString(block.chainid));
    instanceId = 'hypurrfi-testnet';

    config = DeployUsdxlFileUtils.readInput(instanceId);
@>  usdxlConfig = DeployUsdxlFileUtils.readUsdxlInput
//(instanceId);  // @audit usdxlConfig not declared
    --- SNIPPED ---

    _deployUsdxl(usdxlConfig.readAddress
    //('.usdxlAdmin'), deployRegistry);  // Fails: usdxlConfig not declared
}
```

Recommendation

Declare the `usdxlConfig` state variable in the `DeployUsdxlHyperTestnet`.

8.3. Low Findings

[L-01] SupplyHyFi: missing configuration for `USDC` and `sUSDe`

In `SupplyHyFi.run` function, supplying `USDC` and `sUSDe` would revert because `ConfigurrHyFiReserves()` didn't add them as collateral tokens because function `_fetchTestnetTokens()` doesn't return them:

```
function run() external {

    --snip--

    tokens[0] = 0x6fDbAF3102eFC67ceE53EeFA4197BE36c8E1A094; // USDC
    tokens[1] = 0x2222C34A8dd4Ea29743bf8eC4fF165E059839782; // sUSDe

     --snip--

    _supplyPool(
        tokens,
        amounts,
        vm.envAddress('SENDER')
    );
}
```

[L-02] `DeployUsdxlUtils`: `UsdxlInterestRateStrategy` contract is deployed twice

`DeployUsdxlUtils`: The function `_deployUsdxl` deploys `UsdxlInterestRateStrategy` in step 3 :

```
function _deployUsdxl(
      addressproxyAdmin,
      IDeployConfigTypes.HypurrDeployRegistrymemorydeployRegistry
    ) internal {
        --snip--
          // 3. Deploy USDXL Interest Rate Strategy
        usdxlInterestRateStrategy = new UsdxlInterestRateStrategy(
            deployRegistry.poolAddressesProvider,
            0.02e27 // 2% base rate
        );
        --snip--
    }
```

But in step 10 of `_deployUsdxl`, calls `_updateUsdxlInterestRateStrategy()` which deploys `UsdxlInterestRateStrategy` for the second time:

```
function _updateUsdxlInterestRateStrategy
      (IDeployConfigTypes.HypurrDeployRegistry memory deployRegistry)
        internal
    {

          address(deployRegistry.poolAddressesProvider),
          0.02e27
        );

        _getPoolConfigurator(
          deployRegistry
        ).setReserveInterestRateStrategyAddress(address(_getUsdxlToken(
    }
```

Use the already deployed `UsdxlInterestRateStrategy` contract address, instead of deploying it again.

[L-03] Remove deprecated `Göerli testnet` files from deployment

The input folder contains a folder with `primary.json` for deployment on the **Göerli testnet**. However, the Göerli testnet has been deprecated and can no longer be used for test deployments. As a result, the presence of the `primary.json` file for Göerli is redundant and may cause confusion or lead to errors when attempting to deploy on this testnet.

Recommendation: remove the **Göerli testnet** related entries, files, and folders from the deployment process to avoid issues and ensure that only supported chains are used for deployments.

[L-04] Incorrect tokenName set during `hyTokenImpl` initialization

When the `hyTokenImpl` is initialized in the `DeployHyFiUtils` script, the `aTokenName` is set to `"SPTOKEN_IMPL"`, which is specific to the SparkLend protocol, while this should be set to the specific name corresponding to the **HypurrFi** protocol instead.

```
hyTokenImpl = new HyToken(pool);
        hyTokenImpl.initialize(
            pool, address(0), address(0), IHyFiIncentivesController(address
              (0)), 0, "SPTOKEN_IMPL", "SPTOKEN_IMPL", ""
        );
```

Recommendation: update the `DeployHyFiUtils` script to set the `aTokenName` to the appropriate name for the **HypurrFi** protocol during the initialization of `hyTokenImpl`.

[L-05] `_deployUsdxl()` doesn’t initialize `usdxlVariableDebtToken`

The `_deployUsdxl()` function is designed to deploy the **usdxl token** and the required contracts to initialize the **usdx reserve**, however, it was noticed that when the **`usdxlVariableDebtToken`** is deployed, it is not initialized in the script, which allows any malicious actor to initialize it with unintended, incorrect, or irrelevant parameters as the `usdxlVariableDebtToken.initialize()` function is unrestricted.

Recommendation: ensure that the **`usdxlVariableDebtToken`** is properly initialized within the script during the deployment process.

[L-06] `_deployUsdxl()` doesn't initialize `usdxlAToken`

The `_deployUsdxl()` function is designed to deploy the **usdxl token** and the required contracts to initialize the **usdx reserve**, however, it was noticed that when the **`usdxlAToken`** is deployed, it is not initialized in the script, which allows any malicious actor to initialize it with unintended, incorrect, or irrelevant parameters as the `usdxlAToken.initialize()` function is unrestrictred.

Recommendation: ensure that the **`usdxlAToken`** is properly initialized within the script during the deployment process.

[L-07] Missing functionalities in `RepayUsdxlHyperTestnet` and `BorrowUsdxlHyperTestnet` scripts

Description

The scripts `RepayUsdxlHyperTestnet` and `BorrowUsdxlHyperTestnet` are expected to implement functionalities for **repaying** and **borrowing** usdxl tokens, respectively, however,these scripts do not contain the necessary logic to perform these operations.

Recommendation: ensure that the scripts include the necessary functions to perform the actions they are named after.

[L-08] `TODO` resolution required for GSM proxy admin and unique interest rate strategy handling

The following unresolved `TODOs` introduce crucial issues in deployment and configuration logic:

1. Hardcoded `address(0)` as a proxy admin in `DeployUsdxlUtils._deployGsm()`. Currently, the proxy admin is hardcoded as `address(0)`, meaning no one can manage upgrades or administrative functions of the proxy.

```
//File: (usdxl-core) src/deployments/utils/DeployUsdxlUtils.sol

function _deployGsm() internal returns (address) {
    AdminUpgradeabilityProxy proxy = new AdminUpgradeabilityProxy(
        address(gsmImpl),
@>      address(0), // TODO: set admin to timelock
        ""
    );
    --- SNIPPED ---
}
```

2. Duplicate strategy contracts in
   `DeployHyFiConfigEngine._getUniqueStrategiesOnPool()`.

```
//File: (hypurrfi-deployment) script/DeployHyFiConfigEngine.s.sol

library DeployRatesFactoryLib {
@>   // TODO check also by param, potentially there could be different
// contracts, but with exactly same params
    function _getUniqueStrategiesOnPool
      (IPool pool, address[] memory reservesToSkip) {...}
```

The function currently checks for duplicate strategies only by contract address, but not by actual parameters. However, in `V3RateStrategyFactory.initialize()`, strategies are identified using a hash of their parameters. This means the same configuration can be registered multiple times under different contracts, leading to unnecessary duplication.

```
//File:
//(hypurrfi-deployment) lib/aave-helpers/src/v3-config-engine/V3RateStrategyFactory.so

function initialize
  (IDefaultInterestRateStrategy[] memory liveStrategies) external initializer {
for (uint256 i = 0; i < liveStrategies.length; i++) {
    RateStrategyParams memory params = getStrategyData(liveStrategies[i]);

    bytes32 hashedParams = strategyHashFromParams(params);

@>  _strategyByParamsHash[hashedParams] = address(liveStrategies[i]);
@>  _strategies.push(address(liveStrategies[i]));

    emit RateStrategyCreated(address(liveStrategies[i]), hashedParams, params);
}
}
```

Recommendation

- For `DeployUsdxlUtils._deployGsm()` function: If the proxy admin is meant to be a contract (such as a timelock contract), deploy it as part of the script and assign it properly. Otherwise, pass the proxy admin address as a parameter to `_deployGsm()` instead of hardcoding `address(0)`.
- For `DeployHyFiConfigEngine._getUniqueStrategiesOnPool()`: Before adding a new unique strategy, check if another strategy with the same parameters already exists.

[L-09] DeployUsdxlUtils: `_deployGsm()` should use `usdxlToken`'s proxy address instead of implementation

`DeployUsdxlUtils`: The `_deployGsm()` function should use proxy address (`_getUsdxlToken()`) instead of its implementation when deploying new Gsm:

```
function _deployGsm(
        address token,
        address gsmOwner,
        uint256 maxCapacity,
        IDeployConfigTypes.HypurrDeployRegistry memory deployRegistry
    ) internal returns (address gsmProxy) {
        --snip--

        // Deploy GSM implementation
        Gsm gsmImpl = new Gsm(address(usdxlToken), address(token), address
          (fixedPriceStrategy));

        --snip--
```

Recommendations:

Use `_getUsdxlToken()` instead of `address(usdxlToken)`.

[L-10] GSM contract is not deployed

Function `DeployUsdxUtils._deployGsm` is not called anywhere in deployment scripts to deploy GSM:

```
function _deployGsm(
        address token,
        address gsmOwner,
        uint256 maxCapacity,
        IDeployConfigTypes.HypurrDeployRegistry memory deployRegistry
    ) internal returns (address gsmProxy) {
        // Deploy price and fee strategies
        FixedPriceStrategy fixedPriceStrategy = new FixedPriceStrategy(
            1e8, // Default price of $1.00
            address(token),
            IERC20Metadata(token).decimals()
        );

        FixedFeeStrategy fixedFeeStrategy = new FixedFeeStrategy(
            0.02e4, // 2% for buys
            0 // 0% for sells
        );

        // Deploy GSM implementation
        Gsm gsmImpl = new Gsm(address(usdxlToken), address(token), address
          (fixedPriceStrategy));

        // Deploy and initialize GSM proxy
        AdminUpgradeabilityProxy proxy = new AdminUpgradeabilityProxy(
            address(gsmImpl),
            address(0), // TODO: set admin to timelock
            ""
        );

        Gsm(address(proxy)).initialize
          (gsmOwner, deployRegistry.treasury, uint128(maxCapacity));

        // Export contracts
        DeployUsdxlFileUtils.exportContract(instanceId, "gsmImpl", address
          (gsmImpl));
        DeployUsdxlFileUtils.exportContract(instanceId, "gsmProxy", address
          (proxy));
        DeployUsdxlFileUtils.exportContract(
          instanceId,
          "gsmFixedPriceStrategyImpl",
          address
        )
        DeployUsdxlFileUtils.exportContract
          (instanceId, "gsmFixedFeeStrategyImpl", address(fixedFeeStrategy));

        return address(proxy);
    }
```

Add a call to `_deployGsm` to setup GSM contract and fee strategy.

[L-11] `DeployUsdxlUtils`: `usdxlAToken` and `usdxlVariableDebtToken` contracts are deployed twice

Function `_deployUsdxl` deploys `usdxlAToken` and `usdxlVariableDebtToken` token contracts in step 4 and uses their address in different configurations:

```
function _deployUsdxl(
      addressproxyAdmin,
      IDeployConfigTypes.HypurrDeployRegistrymemorydeployRegistry
    ) internal {
        --snip--

        // 4. Deploy USDXL AToken and Variable Debt Token
        usdxlAToken = new UsdxlAToken(IPool(IPoolAddressesProvider
          (deployRegistry.poolAddressesProvider).getPool()));

        usdxlVariableDebtToken =
            new UsdxlVariableDebtToken(IPool(IPoolAddressesProvider
              (deployRegistry.poolAddressesProvider).getPool()));

        // 5. Deploy Flash Minter
        flashMinter = new UsdxlFlashMinter(
            address(usdxlToken),
            deployRegistry.treasury,
            0, // no fee
            deployRegistry.poolAddressesProvider
        );
```

But in step 8 of `_deployUsdxl`, calls `_initializeUsdxlReserve()` which deploys `usdxlAToken` and `usdxlVariableDebtToken` tokens and exports their address for the second time:

```
function _initializeUsdxlReserve(
      addresstoken,
      IDeployConfigTypes.HypurrDeployRegistrymemorydeployRegistry
    ) internal {


        usdxlAToken = new UsdxlAToken(_getPoolInstance(deployRegistry));

        usdxlVariableDebtToken = new UsdxlVariableDebtToken(_getPoolInstance
          (deployRegistry));

        DeployUsdxlFileUtils.exportContract
          (instanceId, "usdxlATokenImpl", address(usdxlAToken));
        DeployUsdxlFileUtils.exportContract(
          instanceId,
          "usdxlVariableDebtTokenImpl",
          address
        )

        --snip--
    }
```

Recommendations:

Use the already deployed `usdxlAToken` and `usdxlVariableDebtToken` contract addresses, instead of deploying them again.

[L-12] CapAutomator does not have riskAdmin access in aclManager

`DeployCapAutomator.run` deploys `capAutomator` but it doesn't grant the necessary admin permissions.

In CapAutomator contract, the `execSupply` function calls `_updateSupplyCap`, which tries to set a new supply cap in `poolConfigurator`:

```
function execSupply(address asset) external override returns (uint256) {
        return _updateSupplyCap(asset);
    }

    function _updateSupplyCap(address asset) internal returns (uint256) {
        --snip--
        poolConfigurator.setSupplyCap(asset, newSupplyCap);
        --snip--
    }
```

But this function is restricted to `onlyRiskOrPoolAdmins`:

```
function setSupplyCap(
    address asset,
    uint256 newSupplyCap
  ) external override onlyRiskOrPoolAdmins {

    uint256 oldSupplyCap = currentConfig.getSupplyCap();
    currentConfig.setSupplyCap(newSupplyCap);
    _pool.setConfiguration(asset, currentConfig);
    emit SupplyCapChanged(asset, oldSupplyCap, newSupplyCap);
  }
```

This means only `riskAdmin` or `poolAdmin` in `aclManager` can call it:

```
function _onlyRiskOrPoolAdmins() internal view {
    IACLManager aclManager = IACLManager(_addressesProvider.getACLManager());
    require(
      aclManager.isRiskAdmin(msg.sender) || aclManager.isPoolAdmin(msg.sender),
      Errors.CALLER_NOT_RISK_OR_POOL_ADMIN
    );
  }
```

So functions like `execSupply` and `execBorrow` in `capAutomator` will not work unless it is added as `riskAdmin` in `aclManager`

Recommendations:

Add `capAutomator` as `riskAdmin` in `aclManager`

[L-13] Uncompilable `DeployUsdxlGsmHyperTestnet` script

The `DeployUsdxlGsmHyperTestnet` deployment script contains multiple implementation issues that make the deployment process unusable. The issues include undefined function usage, incorrect import path and contracts.

```
//File: script/DeployUsdxlGsmHyperTestnet.sol

@1>
   import {ZeroDiscountRateStrategy} from 'src/contracts/facilitators/aave/interestStr
@2>
   import {HyperTestnetReservesConfig} from 'src/deployments/configs/HyperTestnetReser

@2> contract Default is HyperTestnetReservesConfig, Script {
    function run() external {
        --- SNIPPED ---
        vm.startBroadcast(deployerPrivateKey);
        _deploy(deployerAddress);
        vm.stopBroadcast();
    }

    function _deploy(address deployerAddress) internal {
        // launch USDXL token and oracle
        --- SNIPPED ---

@3>      _launchGsm(
            0x6fDbAF3102eFC67ceE53EeFA4197BE36c8E1A094, // USDC
            vm.envAddress('PUBLIC_KEY') // GSM owner
        );
    }
}
```

`@1>` and `@2>`: There are no target contracts that exist: `ZeroDiscountRateStrategy`, `HyperTestnetReservesConfig`

`@3>`: The `_launchGsm()` function does not exist in the codebase. The potential correct function to use for GSM deployment is `DeployUsdxlUtils._deployGsm()`.

Recommendation:

- Import and use the correct contracts and functions for the GSM deployment by replacing the existing imports and inheritance with `HyperTestnetUsdxlConfigs`, `DeployUsdxlUtils`, and updating the function call to `_deployGsm()`.

- The `ZeroDiscountRateStrategy` contract is not used in the GSM deployment script. Consider removing the unused import or update to the correct path if it is needed for deployment.