World Liberty Financial V2 Token Upgrade Audit Report

Prepared by Cyfrin

Version 2.1

Lead Auditors

0kage

Chinmay Farkya

September 5, 2025

Contents

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

5 Audit Scope

Scope of the current audit included following files:

- WorldLibertyFinancialRegistry.sol
- WorldLibertyFinancialV2.sol
- WorldLibertyFinancialVester.sol
- IWorldLibertyFinancialRegistry.sol
- IWorldLibertyFinancialV2.sol
- IWorldLibertyFinancialVester.sol

6 Executive Summary

Over the course of 3 days, the Cyfrin team conducted an audit on the World Liberty Financial V2 Token Upgrade smart contracts provided by World Liberty Financial. In this period, a total of 17 issues were found.

Summary

Project Name	World Liberty Financial V2 Token Upgrade
Repository	usd1-protocol
Commit	82149c77c1ed...
Audit Timeline	Aug 13th - Aug 15th
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	5
Low Risk	4
Informational	6
Gas Optimizations	2
Total Issues	17

Summary of Findings

[M-1] Unlimited token reallocation power creates centralization risk	Resolved
[M-2] Vester template misconfiguration can potentially block token claims	Resolved
[M-3] WLFI owner can DoS legacy users through direct vester activation	Resolved
[M-4] Addresses excluded from voting power can re-gain their voting power via a delegatee or by transfering tokens	Resolved
[M-5] Onchain governance integration breaks due to inconsistent implementation of voting power	Resolved
[L-1] Missing zero address validation for authorized signer in `WorldLibertyFinancialV2.initialize()`	Resolved
[L-2] No governance protection for `MAX_VOTING_POWER` changes	Acknowledged
[L-3] Weak signature validation in account activation	Resolved
[L-4] Guardian can override owner's emergency pause	Resolved
[I-1] Use `SafeCast` to safely downcast amounts	Resolved
[I-2] Remove obsolete `return` statements when using named returns	Resolved
[I-3] Incorrect error message in `_checkNotBlacklisted`	Resolved
[I-4] `renounceOwnership()` should be blocked in WLF V2 token contract	Resolved
[I-5] Missing `_disableInitializers()` in constructor	Resolved

[I-6] `ownerSetVotingPowerExcludedStatus()` applies onlyOwner modifier twice	Resolved
[G-1] Cheaper no to cache `calldata` array length	Resolved
[G-2] In Solidity don't initialize to default values	Resolved

7 Findings

7.1 Medium Risk

7.1.1 Unlimited token reallocation power creates centralization risk

Description: The `WorldLibertyFinancialV::ownerReallocateFrom` function provides the owner with unrestricted power to burn tokens from any address and mint them to any other address, completely bypassing all security mechanisms implemented throughout the contract.

While this may be intended for legal compliance scenarios, the function creates significant centralization risks and governance manipulation opportunities that undermine the decentralized nature of the token.

```
//WorldLibertyFinancialV2.sol
function ownerReallocateFrom(
    address _from,
    address _to,
    uint256 _value
) public onlyOwner {
    _burn(_from, _value);  // No approval, no checks
    _mint(_to, _value);    // No restrictions
}
```

The function circumvents ALL protective mechanisms:

- Can seize from and send to blacklisted addresses
- Can make transfers when contract is in paused state
- No timelocks - can instantly move tokens between accounts before/after voting deadline
- Minimal event emissions specific to `reallocation`
- No rate-limiting on reallocation - can reallocate any amount between accounts

Impact: Users don't truly "own" their tokens if owner can seize them arbitrarily.

Recommended Mitigation: Consider adding one or multiple safeguards for the use of this function:

- Clear documentation as to the circumstances when this function will be called (eg. court ordered seizures etc)
- Add specific event emissions when this function is called
- Add governance approval if reallocation is above a threshold amount
- Add time delay if reallocation is above a threshold amount

WLFI: Fixed in commit b567696

Cyfrin: Verified. Specific use case documentation added and additional safeguards implemented.

7.1.2 Vester template misconfiguration can potentially block token claims

Description: The `WorldLibertyFinancialVester` contract can make user tokens temporarily inaccessible when template `capPerUser` values don't sum to the user's total allocation.

Users transfer their full allocation to the vester during activation, but can only claim back the portion covered by template caps until the owner adds additional templates or modifies existing ones.

The contract's design allows for:

- User allocation can be any amount (set in `_activateVest`)
- Template caps define maximum unlockable amounts per user
- No validation ensures template caps cover the full user allocation

If the contract owner incorrectly configures/modifies the template user cap, users could potentially have a portion of their tokens unclaimable inside the Vester contract.

```
function _unlockedTotal(uint8 _category, uint112 _allocation) internal view returns (uint256) {
    uint256 totalUnlocked = 0;
    uint256 remainingCap = _allocation;  // Start with full allocation

    for (uint8 i; i < count; ) {
        uint256 segmentCap = t.capPerUser < remainingCap ? t.capPerUser : remainingCap;
        uint256 unlocked = _segmentUnlocked(t, segmentCap);
        totalUnlocked += unlocked;

        // @audit remainingCap reduced by template cap, not allocation
        remainingCap -= segmentCap;

        unchecked { ++i; }
    }

    // @audit Any remaining allocation is ignored and becomes inaccessible
    return totalUnlocked;  // Missing: + remainingCap
}
```

Impact: Misconfigured templates can lead to unclaimable tokens for users even after completing full vesting. Users cannot claim portion of their tokens until template coverage is increased.

Recommended Mitigation: Consider documenting clearly that template caps should cover expected user allocations. Add both inline and interface comments that will prevent misconfiguration scenarios by admins. Alternatively consider making it mandatory to add a `remainder` template as the last template for every category.

WLFI: Fixed in commit b567696

Cyfrin: Verified. Moved to percentage allocation from a fixed cap per user.

7.1.3 WLFI owner can DoS legacy users through direct vester activation

Description: The `WorldLibertyFinancialVester::ownerActivateVest` function can be used to bypass the normal activation flow, potentially causing a denial-of-service for legacy users. When the owner directly activates a user in the vester with incorrect parameters, the user is unable to complete their normal activation flow and gets stuck with wrong vesting parameters.

The contract has two independent activation paths that don't coordinate with each other:

Normal path: `WLFI V2 → Registry::wlfiActivateAccount → Vester::wlfiActivateVest` (coordinated, uses registry data for allocation and category) Bypass path: `Owner → Vester::ownerActivateVest` (direct, uses owner-specified parameters as inputs)

The vester contract prevents double initialization but doesn't validate parameter consistency between paths.

Here is a normal activation

```
// WorldLibertyFinancialV2.sol
function _activateAccount(address _account) internal {
    REGISTRY.wlfiActivateAccount(_account);                    // @note -> Mark as activated in registry
    uint8 category = REGISTRY.getLegacyUserCategory(_account);
    uint112 allocation = REGISTRY.getLegacyUserAllocation(_account);

    _approve(_account, address(VESTER), 0);
    _approve(_account, address(VESTER), allocation);           // @note -> Set allowance

    VESTER.wlfiActivateVest(_account, category, allocation);   // @note -> Activate vesting
    assert(allowance(_account, address(VESTER)) == 0);
}
```

Here is a bypassed vesting route:

```
// WorldLibertyFinancialVester.sol
function ownerActivateVest(address _user, uint8 _category, uint112 _amount)
    external
    onlyWorldLibertyOwner(msg.sender)
{
    _activateVest(_user, _category, _amount);  // @audit No coordination with Registry/WLFI V2
    // @audit any amount that is approved by user can be taken -> not registry allocation
   // @audit vesting can be in any category -> not necessarily category in registry
}

function _activateVest(address _user, uint8 _category, uint112 _amount) internal {
    UserInfo storage userInfo = $.users[_user];
    if (userInfo.initialized) {
        revert AlreadyInitialized(_user);      // @audit if user tries to activate later, he will be
        ↪  DOS'ed here     }
    // ... activation logic
}
```

Impact: Owner actions can cause denial of service for legacy user activation. Legacy users get stuck with incorrect vesting parameters and cannot self-correct

Recommended Mitigation: Consider validating vesting parameters and activating legacy user, if not activated.

```
function ownerActivateVest(address _user, uint8 _category, uint112 _amount) external {
    // For legacy users: validate parameters and sync registry
    if (REGISTRY.isLegacyUser(_user)) {
        // @audit Validate parameters match registry data
        require(_category == REGISTRY.getLegacyUserCategory(_user), "CATEGORY_MISMATCH");
        require(_amount == REGISTRY.getLegacyUserAllocation(_user), "ALLOCATION_MISMATCH");

        // @audit Auto-sync registry state to maintain consistency
        if (!REGISTRY.isLegacyUserAndIsActivated(_user)) {
            REGISTRY.ownerActivateAccount(_user);
        }
    }

    _activateVest(_user, _category, _amount);
}
```

WLFI: Fixed in commit b567696

Cyfrin: Verified. `ownerActivateVest` is removed.

7.1.4 Addresses excluded from voting power can re-gain their voting power via a delegatee or by transfering tokens

Description: WFI V2 token has a way to exclude a user's voting power by marking its `_excludedVotingPower` status to true. The intention is that after this, the user's balance will no longer be usable in voting as it delegates current balance to address(0) so that the current delegatee can no longer use it, and getVotes() returns 0 if isExcluded(account) == true.

But this can be bypassed in two ways :

- The account can call `delegate()` to re-delegate to any address X, and then address X will be able to use the account's voting power as a delegatee (getVotes retrieves delegation checkpoints).
- The account can transfer these tokens to another address Y that is controlled by him, and then Y will have the right to use that voting power

This issue occurs because the internal `_delegate()` function does not block an account from creating new delegations after the account was excluded, and `_update()` function does not block people from transferring out tokens when their address has been excluded from voting already.

Contrary to this, if the account was blacklisted, the process of removing the voting power is same, and re-delegation as well as transfers are prevented via `notBlacklisted(_account)` modifier.

```
function _delegate(
    address _account,
    address _delegatee
)
    notBlacklisted(_msgSender())
    notBlacklisted(_account)
    notBlacklisted(_delegatee)
    internal
    override
{
    super._delegate(_account, _delegatee);
}
```

As a result, even though the user's own voting power returns zero via getVotes() but the delegatee's voting power is measured via the `_delegateCheckpoints[account].latest()` which also includes the "user" voting power now after this new delegation. Similarly, transferring out tokens also transfers the related voting power to the new address which is not excluded, thus making it usable.

This bypasses the point of having an excludedVotingPower status for the user as their voting power is still in use.

Impact: Excluded voter can still make his voting power count by delegating votes/ transferring out tokens.

Recommended Mitigation: Consider reverting in `_delegate()` and `_update()` function if account's `excludedVotingPower` status is true. Note that this also blocks any kind of transfers/ burns from an excluded account.

WLFI: Fixed in commit b567696

Cyfrin: Verified.

7.1.5 Onchain governance integration breaks due to inconsistent implementation of voting power

Description: OpenZeppelin's GovernorVotesUpgradeable uses `getPastVotes` for all voting power calculations:

```
/**
 * Read the voting weight from the token's built in snapshot mechanism (see {Governor-_getVotes}).
 */
function _getVotes(
    address account,
    uint256 timepoint,
    bytes memory /*params*/
) internal view virtual override returns (uint256) {
    return token().getPastVotes(account, timepoint);
}
```

The Governor's `_castVote` function retrieves voting weight using this `_getVotes` method:

```
/**
 * @dev Internal vote casting mechanism: Check that the vote is pending, that it has not been cast
 ↪  yet, retrieve
 * voting weight using {IGovernor-getVotes} and call the {_countVote} internal function.
 *
 * Emits a {IGovernor-VoteCast} event.
 */
function _castVote(
    uint256 proposalId,
    address account,
    uint8 support,
    string memory reason,
    bytes memory params
) internal virtual returns (uint256) {
    _validateStateBitmap(proposalId, _encodeStateBitmap(ProposalState.Active));
```

```
@>        uint256 totalWeight = _getVotes(account, proposalSnapshot(proposalId), params);
         uint256 votedWeight = _countVote(proposalId, account, support, totalWeight, params);

        // @more code

         return votedWeight;
    }
```

While WLFI's `getVotes` override correctly includes both balance and vesting tokens, and checks for blacklisted and excluded accounts, the `getPastVotes` function is not overridden.

Impact: While UI/frontend might show users full voting power (balance + vested) via `getVotes`, the actual voting outcomes use a different voting power. Additionally, blacklisted and excluded accounts also have valid voting power because `getPastVotes` does not account for such accounts.

Proof of Concept: Add the following test to `WorldLibertyFinancialV2.test.ts`

```
  describe('getVotes vs getPastVotes inconsistency', () => {
    it('should show discrepancy between current and historical voting power', async () => {
      // Setup: Give user1 1 ETH
      expect(await ctx.wlfi.balanceOf(ctx.core.hhUser1)).to.eq(ONE_ETH_BI);

      // User1 has NOT delegated yet (delegates returns address(0))
      expect(await ctx.wlfi.delegates(ctx.core.hhUser1)).to.eq(ADDRESS_ZERO);

      // Current voting power includes auto-delegation (balance added when delegates == address(0))
      expect(await ctx.wlfi.getVotes(ctx.core.hhUser1)).to.eq(ONE_ETH_BI);

      // Mine a block to create a checkpoint
      await mine();
      const checkpointBlock = await time.latestBlock();

      // Historical voting power at that block should be 0 (no delegation checkpoint exists)
      expect(await ctx.wlfi.getPastVotes(ctx.core.hhUser1, checkpointBlock - 1)).to.eq(ZERO_BI);

      // This shows the inconsistency:
      // - getVotes() returns 1 ETH (auto-includes balance)
      // - getPastVotes() returns 0 (no checkpoint)

      // Now test with vesting tokens
      await ctx.registry.connect(ctx.wlfiOwner).agentBulkInsertLegacyUsers(
        ZERO_BI,
        [ctx.core.hhUser1],
        [ONE_ETH_BI],
        [DEFAULT_CATEGORY],
      );

      // Setup vesting templates with immediate unlock portion
      await ctx.vester.connect(ctx.wlfiOwner).ownerSetCategoryTemplate(DEFAULT_CATEGORY, 0, TEMPLATE_1);
      await ctx.vester.connect(ctx.wlfiOwner).ownerSetCategoryTemplate(DEFAULT_CATEGORY, 1, TEMPLATE_2);
      await ctx.vester.connect(ctx.wlfiOwner).ownerSetCategoryEnabled(DEFAULT_CATEGORY, true);

      const signature = await signWlfiActivationMessage(ctx, ctx.core.hhUser1.address);
      await ctx.wlfi.connect(ctx.core.hhUser1).activateAccount(signature.serialized);

      // User1 now has 0 balance (all in vester) but 1 ETH vesting
      expect(await ctx.wlfi.balanceOf(ctx.core.hhUser1)).to.eq(ZERO_BI);
      expect(await ctx.vester.unclaimed(ctx.core.hhUser1)).to.eq(ONE_ETH_BI);

      // Current voting power includes vesting tokens
      expect(await ctx.wlfi.getVotes(ctx.core.hhUser1)).to.eq(ONE_ETH_BI);
```

```
      // Mine another block for checkpoint
      await mine();
      const vestingCheckpointBlock = await time.latestBlock();

      // Historical voting power still 0 (vesting not included in checkpoints)
      expect(await ctx.wlfi.getPastVotes(ctx.core.hhUser1, vestingCheckpointBlock - 1)).to.eq(ZERO_BI);

      // Now let's delegate to self to create a checkpoint
      await ctx.wlfi.connect(ctx.core.hhUser1).delegate(ctx.core.hhUser1);

      // Current voting power still includes vesting (but no more auto-balance since delegated)
      expect(await ctx.wlfi.getVotes(ctx.core.hhUser1)).to.eq(ONE_ETH_BI);

      await mine();
      const delegatedCheckpointBlock = await time.latestBlock();

      // Historical voting power after delegation only shows balance (0), not vesting
      expect(await ctx.wlfi.getPastVotes(ctx.core.hhUser1, delegatedCheckpointBlock -
      ↪  1)).to.eq(ZERO_BI);

      // Move to after start timestamp to allow claiming
      await advanceTimeToAfterStartTimestamp(ctx);

      // Claim the immediate portion (20% of 1 ETH = 0.2 ETH)
      await ctx.wlfi.connect(ctx.core.hhUser1).claimVest();

      const claimedAmount = parseEther('0.2'); // 20% immediate from TEMPLATE_1
      expect(await ctx.wlfi.balanceOf(ctx.core.hhUser1)).to.eq(claimedAmount);

      await mine();
      const finalCheckpointBlock = await time.latestBlock();

      // Now getPastVotes shows the claimed balance (0.2 ETH)
      expect(await ctx.wlfi.getPastVotes(ctx.core.hhUser1, finalCheckpointBlock -
      ↪  1)).to.eq(claimedAmount);

      // But getVotes shows balance + remaining vesting (0.2 + 0.8 = 1 ETH)
      const currentVotes = await ctx.wlfi.getVotes(ctx.core.hhUser1);
      const remainingVesting = await ctx.vester.unclaimed(ctx.core.hhUser1);

      expect(remainingVesting).to.eq(parseEther('0.8')); // 80% still vesting
      expect(currentVotes).to.eq(ONE_ETH_BI); // 0.2 claimed + 0.8 vesting = 1 ETH

      // Summary of the bug:
      // 1. Before delegation: getVotes returns 1 ETH, getPastVotes returns 0
      // 2. After delegation but before claim: getVotes returns 1 ETH (vesting), getPastVotes returns 0
      // 3. After claiming 0.2 ETH: getVotes returns 1 ETH (0.2 + 0.8 vesting), getPastVotes returns
      ↪  only 0.2 ETH
      // getPastVotes never includes vesting tokens or auto-delegation balance
    });
  });
```

Recommended Mitigation: Consider implementing a snapshot that uses `getVotes` for a historical block and document the off-chain governance and execution process. Additionally, consider reverting on `getPastVotes` to disable any on-chain voting.

```
function getPastVotes(address account, uint256 timepoint)
    public view override returns (uint256) {
    revert("WLFI: Use getVotes() at historical block via RPC");
}

function getPastTotalSupply(uint256 timepoint)
    public view override returns (uint256) {
```

```
    revert("WLFI: Use totalSupply() at historical block via RPC");
}
```

WLFI: Fixed in commit 269f5c1.

Cyfrin: Verified.

7.2 Low Risk

7.2.1 Missing zero address validation for authorized signer in `WorldLibertyFinancialV2.initialize()`

Description: The `WorldLibertyFinancialV2::initialize()` function does not validate that the `_authorizedSigner` parameter is not the zero address.

This parameter is critical for the activateAccount() function, which allows legacy users to self-activate their accounts.

```
function initialize(address _authorizedSigner) external reinitializer(/* version = */ 2) {
    __EIP712_init(name(), "2");

    V2 storage $ = _getStorage();
    _ownerSetAuthorizedSigner($, _authorizedSigner); // @audit No zero address validation
}
```

Same issue also exists in the `ownerSetAuthorizedSigner`

Impact: The `activateAccount()` function will always revert with `InvalidSignature()` since `ECDSA.recover()` never returns the zero address for valid signatures

Recommended Mitigation: Consider adding a zero address validation in the `initialize()` function

WLFI: Fixed in commit b567696

Cyfrin: Verified.

7.2.2 No governance protection for `MAX_VOTING_POWER` changes

Description: The `WorldLibertyFinancialV2::ownerSetMaxVotingPower()` allows the owner to change the maximum voting power cap at any time without restrictions. Since this is a governance token that inherits from `ERC20VotesUpgradeable` and is likely used with external governance systems (OZ Governor), the owner can manipulate voting outcomes by strategically timing changes to the voting power cap.

```
//WorldLibertyFinancialV2.sol
function ownerSetMaxVotingPower(uint256 _maxVotingPower) external onlyOwner {
    require(
        _maxVotingPower > 0 && _maxVotingPower <= (5_000_000_000 * 1 ether),
        "Invalid max voting power"
    );
    MAX_VOTING_POWER = _maxVotingPower; // @audit No governance protections
    emit SetMaxVotingPower(_maxVotingPower);
}
```

The `getVotes()` function immediately applies this cap:

```
if (votingPower > MAX_VOTING_POWER) {
    return MAX_VOTING_POWER; // @audit Immediate capping on voting power
}
```

Impact: Owner can reduce MAX_VOTING_POWER during active proposals to neutralize large holders who might vote against their interests in a specific proposal

Recommended Mitigation: Consider implementing a timelock mechanism to provide adequate notice before making max voting power change.

WLFI: Acknowledged. We're only using Snapshot for voting now and don't plan to do onchain voting as of now.

Cyfrin: Acknowledged.

7.2.3 Weak signature validation in account activation

Description: The `WorldLibertyFinancialV2::activateAccount` function uses a simple hash of the account address for signature validation instead of following EIP-712 standards.

While the contract initializes EIP-712 infrastructure during setup, the activation function bypasses this standard and uses a basic `keccak256(abi.encode(account))` hash. This deviates from established security best practices for signature hashing.

```
function activateAccount(bytes calldata _signature) external {
    address account = _msgSender();
    bytes32 hash = keccak256(abi.encode(account)); // @account simple hash, no EIP-712

    if (authorizedSigner() != ECDSA.recover(hash, _signature)) {
        revert InvalidSignature();
    }

    _activateAccount(account);
}
```

Impact: If WLFI expands to multiple chains in the future, signatures could be replayed across chains. Alternatively, if contract was ever migrated to a new proxy or implementation, signatures generated for current contract could work on new deployments.

Also, since the contract implements EIP712, off-chain systems expect EIP-712 structured data for security.

Recommended Mitigation: The practical risk is currently mitigated by:

- Double activation protection
- Assumed single-chain deployment

Nevertheless, consider implementing EIP-712 signature validation to follow security best practices and future-proof the contract,

WLFI: Fixed in commit b567696.

Cyfrin: Verified.

7.2.4 Guardian can override owner's emergency pause

Description: The contract implements symmetric `pause/unpause` powers between the owner and guardians, allowing guardians to unpause the contract even when the owner intentionally paused it for security or operational reasons. This creates an authority hierarchy conflict where guardians can override the owner's emergency decisions, potentially undermining security responses and operational control.

```
function guardianUnpause() external onlyGuardian whenPaused {
    // @audit - do you think only the owner should be able to unpause?
    _unpause();
}
```

Note: The comment in the code indicates the dev team flagged this design choice from a security viewpoint.

During periods of emergency or security breach, owner should have ultimate control over contract state. While pausing a contract is low-risk, unpausing it is higher-risk operation that needs to have a hierarchical access. Common security practice is:

```
Multiple parties can pause (defensive action, low risk)
Only highest authority can unpause (requires careful consideration)
```

Impact: Guardian override can undermine owner's authority on contract pause/unpause status.

Recommended Mitigation: Consider removing `unpause` option for guardians.

WLFI: Fixed in commit b567696

Cyfrin: Verified.

7.3 Informational

7.3.1 Use `SafeCast` to safely downcast amounts

Description: Use SafeCast to safely downcast amounts. or add a comment indicate that this downcast is safe:

```
wlfi/WorldLibertyFinancialRegistry.sol
64:                 amount: uint112(_amounts[i]),
```

WLFI: Fixed in commit b567696

Cyfrin: Verified.

7.3.2 Remove obsolete `return` statements when using named returns

Description: Remove obsolete `return` statements when using named returns:

- `WorldLibertyFinancialV2::getVotes` - final `return votingPower;` at L260

WLFI: Fixed in commit b567696

Cyfrin: Fixed.

7.3.3 Incorrect error message in `_checkNotBlacklisted`

Description: The error message in following function says `WLFI: caller is blacklisted` even though the check is applicable on the input `account` address, not the caller address.

```
  function _checkNotBlacklisted(address _account) internal view {
        require(
            _account == address(0) || !_getStorage().blacklistStatus[_account],
            "WLFI: caller is blacklisted"
        );
    }
```

Recommended Mitigation: Consider changing the error message to `WLFI: account is blacklisted`

WLFI: Fixed in commit b567696

Cyfrin: Verified.

7.3.4 `renounceOwnership()` should be blocked in WLF V2 token contract

Description: The owner of the WLF token contract has many admin rights, including in VESTER and Registry contracts as well.

The WLF contract inherits ownership functionality from `Ownable2StepUpgradeable` which also has a `renounceOwnership()` function.

If this is called accidentally, the contract ownership will be lost forever.

Recommended Mitigation: It is a best practice to block renounceOnwership() from being called.

```
/// @notice Explicitly disallow renouncing ownership
function renounceOwnership() public payable override onlyOwner {
     revert OwnerRequired();
}
```

WLFI: Fixed.

Cyfrin: Verified.

7.3.5 Missing `_disableInitializers()` in constructor

Description: The `WorldLibertyFinancialV2`, `WorldLibertyFinancialVester` and `WorldLibertyFinancialRegistry` contracts are upgradeable but do not call `_disableInitializers()` in their constructor. In upgradeable contract patterns, this call is a best practice to prevent the implementation (logic) contract from being initialized directly.

While this doesn't affect the proxy's behavior, it helps protect against accidental or malicious use of the implementation contract in isolation, especially in environments where both proxy and implementation contracts are visible, like block explorers.

Recommended Mitigation: Consider adding the following line to the constructor of the `WorldLibertyFinancialV2` contract:

```
    _disableInitializers();
```

Add constructors with this line to `WorldLibertyFinancialVester` and `WorldLibertyFinancialVester` contracts.

This ensures that the implementation contracts cannot be initialized independently.

WLFI: Fixed.

Cyfrin: Verified.

7.3.6 `ownerSetVotingPowerExcludedStatus()` applies onlyOwner modifier twice

Description: In WLF V2 contract, the function `ownerSetVotingPowerExcludedStatus()` applies `onlyOwner` modifier twice :

- First in the external `ownerSetVotingPowerExcludedStatus()` function
- Again in the internal `_ownerSetVotingPowerExcludedStatus()` function in the same call flow.

The second onlyOwner modifier on `_ownerSetVotingPowerExcludedStatus()` is unnecessary.

Recommended Mitigation: Remove onlyOwner modifier from `_ownerSetVotingPowerExcludedStatus()` function.

WLFI: Fixed in commit b567696

Cyfrin: Verified.

7.4 Gas Optimization

7.4.1 Cheaper no to cache `calldata` array length

Description: It is cheaper not to cache `calldata` array length:

- `WorldLibertyFinancialRegistry::agentBulkInsertLegacyUsers`

```
54:         uint256 usersLength = _users.length;
55:         for (uint256 i; i < usersLength; ++i) {
```

WLFI: Fixed.

Cyfrin: Verified.

7.4.2 In Solidity don't initialize to default values

Description: In Solidity don't initialize to default values:

```
WorldLibertyFinancialVester.sol
228:         uint256 totalUnlocked = 0;
```

WLFI: Fixed.

Cyfrin: Verified.