INTERFI

SMART CONTRACT SECURITY ASSESSMENT

PREPARED FOR

BETTERBANK CONTRACTS

hello@interfi.network

interfinetwork

www.interfi.network

FOR BETTERBANK

Date of Report	October 30, 2025	

Audited Files	`Esteem.sol`
	`PulseMinter.sol`
	`Favor.sol`
	`FavorTreasury.sol`
	`Zapper.sol`
		`UniTWAPOracle.sol`
		`Epoch.sol`
		`LPTWAPOracle.sol`
		`Staking.sol`

Website	https://www.betterbank.io/	
Language	Solidity	
Methodology	Automated Review, Manual Review	
Auditor	InterFi	

Disclaimer: Smart contracts deployed on blockchains are inherently exposed to potential exploits, vulnerabilities, and security risks. Blockchain and cryptographic technologies are emerging and carry ongoing uncertainties. Please review the full audit report for detailed insights into risk severity, vulnerabilities, and audit scope limitations.

Centralization Warning: Centralized privileges—regardless of intent or access control—introduce elevated risks to contract security and user trust.

KYC Advisory: The project lacks verified third-party KYC of its owners, team, or deployers. Without independent KYC, transparency and accountability are reduced, increasing the risk of fraud or rug pulls.

Verification: Verify this report: https://www.github.com/interfinetwork

TABLE OF CONTENTS

FOR BETTERBANK 1

TABLE OF CONTENTS 3

1. SUMMARY 4
- 1.1 Summary of Findings 4
- 1.2 Resolution Status 4
- 1.3 System Overview 5
- 1.4 Files in Scope 5
- 1.5 Out-of-Scope Assumptions 6

2. METHODOLOGY 7
- 2.1 Audit Objectives 7
- 2.2 Methodologies 7
- 2.3 Risk Categorization 8
- 2.4 Resolution Status Definitions 9

3. FINDINGS 10

4. CENTRALIZATION 22
- 4.1 Privileged Functions 22

5. DISCLAIMER 26
- 5.1 Confidentiality 26
- 5.2 No Financial Advice 26
- 5.3 Technical Disclaimer 26
- 5.4 Timeliness & Accuracy 27
- 5.5 Third-Party Links 27

6. ABOUT 28
- 6.1 Connect with Us 28

1. SUMMARY

The audit resulted in the identification of issues across a range of severity levels, including logic flaws, access control oversights, and design inconsistencies. All high-impact findings were communicated to the development team with clear recommendations for remediation. Where applicable, the team has confirmed implementation of fixes or provided justifications for design choices.

1.1 Summary of Findings

Severity	Count
Critical	0
Major	2
Medium	3
Minor	6
Unknown	1
Centralization	1

1.2 Resolution Status

Status	Count
Fixed	7
Partially Fixed	3
Acknowledged	3
Pending	0

1.3 System Overview

This system is a decentralized protocol comprising a suite of smart contracts. These contracts collectively define the rules, permissions, and operational workflows for managing on-chain assets, executing user interactions, and enforcing protocol-level logic. Smart contracts in this context are self-executing code units that autonomously manage the state and behavior of digital assets based on predefined conditions. The protocol utilizes these contracts to enable key functionalities such as:

- Ownership and access control enforcement
- Permission and role-based actions
- Data storage and updates
- Event logging and auditability
- Batch processing and collection management

1.4 Files in Scope

InterFi was engaged by `BetterBank` to perform a security audit of the smart contracts. The audit scope was strictly limited to the files explicitly listed under the “Files in Scope” section. No other files or components were reviewed unless otherwise stated.

`Esteem.sol`	`PulseMinter.sol`
`Favor.sol`	`FavorTreasury.sol`
`Zapper.sol`	`UniTWAPOracle.sol`
`Epoch.sol`	`LPTWAPOracle.sol`
`Staking.sol`	

1.5 Out-of-Scope Assumptions

The following components and assumptions were explicitly excluded from this audit:

- Frontend or backend integration logic.
- Off-chain components, scripts, or oracles.
- External contracts or libraries unless explicitly stated.
- Compiler-level or EVM-specific behavior outside the contract's scope.
- Governance or tokenomics-related decisions not implemented in code.
- All third-party dependencies as discussed in findings.

2. METHODOLOGY

2.1 Audit Objectives

This audit aims to ensure that the smart contract system is predictable, and behaves as intended under normal conditions. Primary audit objectives are to:

- Identify potential vulnerabilities or logic errors in the implementation.
- Evaluate adherence to best practices in smart contract development.
- Assess the correctness of access controls and permission systems.
- Recommend remediations or enhancements for improved security and performance.

2.2 Methodologies

The audit follows a layered security approach using both automated tools and manual techniques. We review the contracts for functional correctness, exploitability, and adherence to smart contract best practices:

Type	Tools & Techniques
Manual Code Review	Line-by-line analysis to check logic, permissions, and edge cases
Automated Analysis	Tools like Slither, MythX, or custom linters to catch known patterns
Static Analysis	Identification of bugs without executing the code (compile-time checks)
Unit Test Inspection	Evaluation of existing test coverage, assumptions, and potential false positives/negatives (if applicable)
Architecture Review	Mapping of privileged roles, callable paths, and contract interdependencies (if applicable)

2.3 Risk Categorization

Each issue identified during the audit is assigned a severity level based on its potential impact, exploitability, and likelihood of real-world abuse. These categories help prioritize remediation efforts:

Risk Severity	Definition
Critical	Represents a severe vulnerability that may result in complete contract compromise, such as asset theft, permanent loss of functionality, or unrestricted access. These issues are often easily exploitable and require immediate resolution.
Major	Indicates significant risk that can affect core contract behavior, enable unauthorized operations, or create unintended financial exposure. While not as urgent as critical risks, they should be remediated promptly.
Medium	These are moderate-level risks that may become exploitable under specific conditions. They often relate to logic errors, insufficient validation, or architectural oversights that could escalate over time.
Minor	Denotes issues that have low security impact but may degrade code quality, performance, or maintainability. These include inefficiencies, style violations, or redundant logic. Fixes are recommended for robustness.
Unknown	Risks where the severity cannot be confidently determined due to limited context, external dependencies, or ambiguous design intent. It is advised to treat these conservatively and address them proactively.
Centralization	Any function controlled by a single privileged role is treated as a critical risk, regardless of its purpose, due to the potential for misuse, override, or total asset control.

2.4 Resolution Status Definitions

All identified issues are also assigned a resolution status, indicating the current handling and response from the development team:

Status	Definition
Fixed	The issue has been remediated and verified as resolved during the re-audit or final check.
Partially Fixed	The issue has been partially mitigated, but remnants or related concerns may still exist. Further attention may be required.
Acknowledged	The development team has accepted the finding but opted not to implement a fix.
Pending	The issue remains unresolved at the time of publication. It poses a potential risk and should be addressed.

3. FINDINGS

01	**Favor taxation incorrectly applies to mints (inflation on mint)**
Severity	Major
Where	`Favor.sol` (`_update` override)
Description	Tax logic runs inside `_update`, which is invoked for all balance changes including `mint (from == address(0))`. When minting to a contract address, `destinationIsContract == true` and the code sends `taxAmount` via `super._update(_from, treasury, taxAmount)` where `_from` is `address(0)`, effectively minting additional tokens to treasury (on top of the intended mint). This silently inflates supply and skews accounting/emissions.
Recommendation	Supply corruption; inconsistent tokenomics; overpayment to treasury on any mint to a contract, e.g., Staking, Treasuries, Wrappers
Status	Fixed
Update	`Favor.sol` reverts on any transfer to contract addresses unless tax-exempt. All protocol contracts (minters, liquidity, zap) are pre-marked tax-exempt. No further mint inflation occurs.

02	**Zapper transactions have no slippage controls** **Sandwich attack possibility**
Severity	Major
Where	`Zapper.sol (_swap, buy*, sell*)`
Description	Swaps use `swapExactTokensForTokensSupportingFeeOnTransferTokens` with `amountOutMin = 0`. Attackers can sandwich these calls and force terrible execution prices; users (and treasury on tax swaps) can be drained of value even if the overall tx succeeds. This can be an easy target for MEV bots.
Recommendation	Accept user-provided `minOut` or slippage bps
Status	Partially Fixed Acknowledged
Update	`amountOutMin` parameter added to all public swap and manual LP-add functions. `zapToken()` remains internal without slippage guard. Team acknowledges that If user wants slippage protection, they can use `buy()` and then add liquidity and `_zapToken()` is internal.

03	Double-tax hazard if Zapper is not exempt
Severity	Medium
Where	`Zapper.sol (sell*) + Favor.sol (_update)`
Description	`Zapper.sell*` manually computes/collects tax. If `Favor.isTaxExempt(Zapper)` is not set, transfer from user to Zapper will itself trigger Favor's contract-recipient tax. Result: user taxed twice (once by Favor, once by Zapper).
Recommendation	Hard-require `isTaxExempt[Zapper] == true` for registered Favor tokens (assert in `addFavor`), or remove Zapper-side tax and centralize all tax logic in Favor.
Status	Fixed
Update	All contract transfers revert unless tax-exempt; `Zapper` is now tax-exempt and registered as a `BuyWrapper` in Favor. No double-tax path remains.

04	LPOracle constructor division-by-zero if pair has no supply
Severity	Medium
Where	`LPOracle.sol (constructor)`
Description	`initialSqrtK = ... / _pair.totalSupply()` without a non-zero check can revert when the LP was created but not yet minted. This blocks deployment on fresh pairs.
Recommendation	`require(_pair.totalSupply() > 0, "Oracle: zero LP supply"` Or delay seeding until first `update`
Status	Fixed
Update	A guard was added to require `pair.totalSupply() > 0` before seeding, preventing constructor revert on empty pairs.

05	**Unbounded epoch catch-up loop can revert gas and block updates**
Severity	Medium
Where	`Epoch.checkEpoch, LPOracle.update, Oracle.update.`
Description	If many periods elapse without calling `update`, the modifier loops once per missed epoch to increment epoch and `lastEpochTime`. This can exceed block gas and brick the first update, blocking further oracle progress.
Recommendation	Replace the loop with arithmetic catch-up
Status	Fixed
Update	Epoch logic replaced with a global `EpochKeeper` tracking time-based epochs. Each dependent contract references the global keeper and updates arithmetically.

06	MintRedeemer normalization reverts for tokens with >18 decimals
Severity	Minor
Where	`MintRedeemer._normalizeTokenAmount`
Description	For tokens with `decimals() > 18`, `10 ** (18 - tokenDecimals)` underflows and reverts. Blocks minting with such assets.

Recommendation	Change to:
	`if (tokenDecimals > 18) return amount / (10 ** (tokenDecimals - 18));`
	`if (tokenDecimals < 18) return amount * (10 ** (18 - tokenDecimals));`
	`return amount;`

Status	Acknowledged

07	**Oracle address can be set to zero causing hard reverts**
Severity	Minor
Where	`LPOracle.setMasterOracle`
Description	Owner can accidentally set `masterOracle` to `address(0)`, making `update()` and consultors revert.
Recommendation	Change to: `require(_oracle != address(0), "zero oracle");`
Status	Fixed
Update	Non-zero check is added

08	**Favor depends on external priceProvider with no initialization guard**
Severity	Minor
Where	`Favor.calculateFavorBonuses, logBuy`
Description	If `priceProvider` not set pre-use, calls to `logBuy` will revert on zero address. Breaks buys via wrappers until configured.
Recommendation	Change to: `priceProvider != address(0)` inside `calculateFavorBonuses`
Status	Fixed
Update	`calculateFavorBonuses` requires `priceProvider` to be set before use

09	LPZapper missing nonReentrant on user-facing flows
Severity	Minor
Description	`zapToken`, `sell`, `buy`, `addLiquidity*`, `requestFlashLoan`, `executeOperation` lack re-entrancy guards. Current state changes are limited, but adding `nonReentrant` is prudent given external router/pool calls.
Recommendation	Add `ReentrancyGuard` and mark sensitive externals
Status	Partially Fixed Acknowledged
Update	`nonReentrant` is added to `requestFlashLoan` and `executeOperation`. Other public/external paths flagged earlier (`buy`, `sell`, `addLiquidity*`, and possibly `zapToken` if externally reachable in the deployment) are still not explicitly guarded in this update. Recommendation to add `nonReentrant` to all swap/LP-entry points that make external calls (router/pool), or restructure to update state-before-call patterns where applicable. Team comments that re-entrancy guard is added where it is applicable, all external calls are covered. Buy/Sell simply call `buyTo` and `sellTo` which are protected and `_addLiquidity` is internal of the protected `zapToken` and `requestFlashLoan` functions. `ExecuteOperation` doesn't need re-entrancy guard as its only caller is from the designated AAVE pool itself and so all user interactions must go through `requestFlashLoan` which is protected.

10	**Repayment flow depends on allowance pull**
Severity	Minor
Where	`LPZapper.executeOperation`
Description	Aave V3 `flashLoanSimple` typically *pulls* funds via `transferFrom` after `approve`. It approves the pool, but it also borrows to the user (`POOL.borrow(...)`), leaving the contract with no asset to repay unless the pool pulls (and it still has balance). If the asset balance is insufficient (likely), the loan reverts.
Recommendation	Ensure the contract holds `amount + premium` before return
Status	Fixed
Update	Team fixed for `LPOracle.sol` by adding a `require(pair.totalSupply() > 0, "Zero LP supply");` to ensure oracle seed is valid. The flash loan repayment path issue remains only partially addressed, as it still relies on UI-flow user approval rather than in-contract enforcement.

11	Front-running possibilities
Severity	Minor
Description	▪ Oracle `update()` is time-gated; not value-dependent. ▪ `Favor._update` taxation is deterministic and not slippage-sensitive. ▪ `MintRedeemer` uses deadlines; prices come from oracles - no AMM reads used in mint/redeem math, so no direct sandwich vector available.
Recommendation	No real exploitable surface found for front-running beyond normal DEX interaction risks outside these contracts.
Status	Partially Fixed Acknowledged
Update	Team added `amountOutMin` parameters for swaps in the `LPZapper.sol` contract. That reduces the bare "`minOut = 0`" risk for many paths. According to the commit diff, team made internal functions non-reentrant and added the `nonReentrant` modifier to flash-loan entry points. However, internal `zapToken()` path still lacks explicit slippage. So, a risk remains for sandwich/MEV on that path. Team acknowledges that if user wants slippage protection, they can use `buy()` and then add liquidity and `_zapToken()` is internal.

12	Direct & indirect dependencies
Severity	Unknown
Description	▪ OpenZeppelin v5.x: `Ownable`, `ERC20`, `ERC20Burnable`, `SafeERC20`, `Pausable`, `ReentrancyGuard`. ▪ Uniswap V2: `IUniswapV2Pair`, router interfaces, `UniswapV2OracleLibrary`, `FixedPoint math lib`. ▪ Aave V3: `IPool` and `flash-loan behaviors`. ▪ Project Interfaces: `BBToken` (mint/burn authority), `IMasterOracle` (USD prices & TWAPs), `IBasisAsset` (mint), `IGrove` (seigniorage allocation), `IFavorToken` (tax & logging). ▪ Environment assumptions: `masterOracle.getLatestPrice/getTokenTWAP/esteemRate` return 18-decimals, non-manipulable. ▪ Favor/Esteem minter roles are correctly assigned to Treasury/Zapper/Minter as needed. If any of these assumptions fail, e.g., manipulable oracle, malicious pool, risk escalates significantly. ▪ Safety relies on `IMasterOracle` and `IPool` correctness and liveness; without the code we cannot rule out manipulation (oracle) or repayment semantics (pool).
Recommendation	Inspect third party dependencies regularly, and mitigate severe impacts whenever necessary
Status	Acknowledged

4. CENTRALIZATION

Centralization is one of the leading causes of smart contract-related asset losses. When a contract assigns critical powers to a privileged role—such as an `owner`, `admin`, or designated `controller`—the associated risk becomes elevated, especially if that role is tied to a single externally owned account (EOA). In many cases, privileged roles serve operational or safety functions, including:

- Emergency Controls: Ability to `pause()` the contract during active threats or bugs.
- Contract Configuration: Updating key addresses, thresholds, or operational variables post-deployment.

4.1 Noteworthy Privileged Functions

Epoch

`setPeriod, setEpoch, setApprovedUser`	`onlyOwner`

Esteem

`mint`	`onlyMinter`
`addMinter / removeMinter`	`onlyOwner`

Favor

`mint`	`isMinter`
`setTreasury / setPriceProvider / setEsteem`	`onlyOwner`
`setSellTax / setBonusRates`	`onlyOwner`
`setTaxExempt / setBuyWrapper`	`onlyOwner`
`addMinter / removeMinter`	`onlyOwner`

FavorTreasury

`initialize`	`onlyOwner`
`setGrove / setFavorOracle`	`onlyOwner`
`setMaxSupplyExpansionPercents / setMinSupplyExpansionPercents`	`onlyOwner`
`add/removeExcludedAddress`	`onlyOwner`
`add/removeLpPairToExclude`	`onlyOwner`
`setExtraFunds`	`onlyOwner`

pause / unpause	onlyOwner
governanceRecoverUnsupported	onlyOwner

LPOracle

update	onlyApproved
setMaxPriceCap / setMasterOracle	onlyOwner
setApprovedUser (via Epoch)	onlyOwner

UniTWAPOracle

update	onlyApproved
setMaxPriceCap	onlyOwner
setApprovedUser (via Epoch)	onlyOwner

MintRedeemer

updateEsteemRate	onlyApprovedUser
setApprovedUser / setDailyRateIncrease / setEsteemRate	onlyOwner
setPool / setRedeemRate / setTreasuryBonus / setTreasury	onlyOwner
setPriceOracle / setAllowedMintToken / setActiveFavorToken	onlyOwner
adminWithdraw / adminWithdrawPLS	onlyOwner
pause / unpause	onlyOwner

Staking

initialize	onlyOwner
allocateSeigniorage	owner or treasuryOperator
setTreasuryOperator	onlyOwner
governanceRecoverUnsupported	onlyOwner
pause / unpause	onlyOwner

LPZapper

setPool / setTreasury	onlyOwner
add/removeDustToken	onlyOwner
addFavor / removeFavorToken	onlyOwner
adminWithdraw / adminWithdrawPLS	onlyOwner

01	Centralized and controlled privileges
Severity	Centralization
Description	A single EOA with control can be compromised via phishing, private key leakage, or insider threats. Malicious or negligent use of privileges can lead to - token supply manipulation, disruption of trading via pausing, arbitrary fee changes or wallet exclusions, asset seizures or rerouting, etc. Review 4.1 Noteworthy Privileged Functions ▪ Arbitrary mint/burn or redirect of minted assets: o `Esteem.mint` by any `isMinter` (owner-controlled). o `Favor.mint` by `isMinter` (owner-controlled). o `FavorTreasury` mints Favor via `IBasisAsset.mint` (must be whitelisted as minter). ▪ Pause trading/claims or global transfer block: `Pausable` in `MintRedeemer, Staking, FavorTreasury`; Owner controlled. ▪ Blacklist/whitelist: `Favor.isTaxExempt, LPOracle / Oracle onlyApproved` controlled. ▪ Fee/tax control: Up to 50% sell tax; owner can set exemptions and receivers. ▪ Liquidity control: `LPZapper` routes deposits/taxes and can interact with pools; owner configures mappings. ▪ Privileged sweep: Admin withdraw functions across contracts. ▪ Router/pair/address setters: Multiple setters for oracle, treasury, pools.

Recommendation	Using Multi-Signature Wallets: Assign privileged roles to a multi-sig contract requiring signatures from multiple trusted parties. This reduces the impact of any single compromised key. Time-Locked Functions: Introduce delays before executing sensitive operations, allowing time for community review or cancellation. Role Revocation or Transfer: If privileges are no longer needed post-deployment, renounce them or migrate them to DAO governance. Secure Key Management: Any private keys associated with privileged roles must be protected using hardware wallets, secret sharing schemes, or offline signing protocols.
Status	Acknowledged
Update	Without time-lock or multi-sig, these controls constitute significant centralization risk. A malicious or compromised owner could revoke exemptions, redirect fees, or change routing.

5. DISCLAIMER

InterFi Network provides professional smart contract audits for blockchain-based codebases (commonly known as smart contracts). This audit assessed the reviewed contract(s) for common vulnerabilities, centralization risks, and logic flaws. However, no audit can guarantee the complete absence of bugs or vulnerabilities. This report does not constitute a security guarantee, endorsement, or assurance of business model soundness or legal compliance.

The review is limited strictly to the source code and its logic as provided, and does not extend to compiler behavior, off-chain components, or external integrations. Due to the evolving nature of blockchain technology and associated risks, users should understand that all materials, including this audit report, are provided strictly on an “as is”, “as available”, and “with all faults” basis.

5.1 Confidentiality

This report is confidential and intended solely for the client. It may not be disclosed, reproduced, or relied upon by third parties without prior written consent from InterFi Network. All terms, including confidentiality, liability limitations, and scope, are governed by the audit agreement.

5.2 No Financial Advice

This report is not financial, investment, tax, legal, or regulatory advice. It should not be relied upon for making investment decisions or assessing the value, viability, or safety of any token, product, or platform. No part of this document should be interpreted as an endorsement or recommendation. InterFi Network accepts no liability for any actions taken based on this report.

5.3 Technical Disclaimer

InterFi disclaims all warranties—express, implied, or statutory—including merchantability, fitness for a particular purpose, title, and non-infringement. We do not guarantee that the reviewed contracts are error-free, fully secure, or meet any specific requirements. Audit results may contain false positives or negatives, and findings are subject to the context and limitations of the review scope.

5.4 Timeliness & Accuracy

Audit results reflect the state of the code at the time of review. InterFi makes no commitment to update findings after publication. We do not warrant the accuracy, completeness, or timeliness of information delivered via this report.

5.5 Third-Party Links

This report may contain references or links to external websites and social media accounts. InterFi Network is not responsible for the content or operation of third-party platforms and assumes no liability for actions taken based on their content.

6. ABOUT

InterFi Network is a leading provider of intelligent blockchain solutions, offering secure, scalable, and production-ready smart contract services. Our team specializes in the development, testing, and auditing of smart contracts across a wide range of blockchain ecosystems.

We have delivered:

- 300+ smart contract systems developed
- 2,000+ smart contracts audited
- 500,000+ lines of code reviewed and analyzed

Our technical expertise spans multiple languages including:

- Solidity for EVM-compatible chains (Ethereum, BNB Chain, Polygon, Avalanche, Cronos, Fantom, Velas, Metis, and more)
- Move for next-generation platforms such as Sui and Aptos
- Rust for advanced ecosystems like Solana, Near, and Cosmos SDK-based chains

6.1 Connect with Us

InterFi Network is driven by a multidisciplinary team of engineers, developers, UI/UX specialists, and blockchain researchers. The core team consists of 3 senior members supported by 4+ expert contributors across code auditing, tooling, and protocol design.

- Website: interfi.network
- Email: hello@interfi.network
- GitHub: github.com/interfinetwork
- Telegram (Engineering): @interfiaudits
- Telegram (Onboarding): @interfisupport

THANK YOU