Flex Swap Smart Contract

SMART CONTRACT AUDIT REPORT

April 2025

ExVul

www.exvul.com

Table of Contents

1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by **Flex Swap** to review smart contract implementation. The assessment was conducted in accordance with our systematic approach to evaluate potential security issues based upon customer requirement. The report provides detailed recommendations to resolve the issue and provide additional suggestions or recommendations for improvement.

The outcome of the assessment outlined in chapter 3 provides the system's owners a full description of the vulnerabilities identified, the associated risk rating for each vulnerability, and detailed recommendations that will resolve the underlying technical issue.

1.1 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [10] which is the gold standard in risk assessment using the following risk models:

- Likelihood: represents how likely a particular vulnerability is to be uncovered and exploited in the wild.
- Impact: measures the technical loss and business damage of a successful attack.
- Severity: determine the overall criticality of the risk.

Likelihood can be: High, Medium and Low and impact are categorized into for: High, Medium, Low, Informational. Severity is determined by likelihood and impact and can be classified into five categories accordingly, Critical, High, Medium, Low, Informational shown in table 1.1.

Likelihood				
High	*Informational*	Medium	High	Critical
Medium	*Informational*	Low	Medium	High
Low	*Informational*	Low	Low	Medium
	Informational	*Low*	*Medium*	*High*
		IMPACT		

Table 1.1 Overall Risk Severity

To evaluate the risk, we will be going through a list of items, and each would be labelled with a severity category. The audit was performed with a systematic approach guided by a comprehensive assessment list carefully designed to identify known and impactful security issues. If our tool or analysis does not identify any issue, the contract can be considered safe regarding the assessed item. For any discovered issue, we might further deploy contracts on our private test environment and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.2.

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- Code and business security testing: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

Category	Assessment Item
Basic Coding Assessment	Apply Verification Control
	Authorization Access Control
	Forged Transfer Vulnerability
	Forged Transfer Notification
	Numeric Overflow
	Transaction Rollback Attack
	Transaction Block Stuffing Attack
	Soft Fail Attack
	Hard Fail Attack
	Abnormal Memo
	Abnormal Resource Consumption
	Secure Random Number
Advanced Source Code Scrutiny	Asset Security
	Cryptography Security
	Business Logic Review
	Source Code Functional Verification
	Account Authorization Control
	Sensitive Information Disclosure
	Circuit Breaker
	Blacklist Control
	System API Call Analysis
	Contract Deployment Consistency Check
Additional Recommendations	Semantic Consistency Checks
	Following Other Best Practices

Table 1.2: The Full List of Assessment Items

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [14], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development.

2. FINDINGS OVERVIEW

2.1 Project Info And Contract Address

Project Name: Flex Swap

Audit Time: March 23, 2025 - April 8, 2025

Language: move

Soure code	Link
Flex Swap	https://github.com/dddappp/aptos-flex-swap
Commit Hash	92533a4fea6bf008b860626cd6a15b2b3a7f4960

2.2 Summary

Severity	Found	
Critical	3	🟥🟥🟥
High	0	
Medium	0	
Low	0	
Informational	0	

2.3 Key Findings

ID	Severity	Findings Title	Status	Confirm
NVE- 001	Critical	Incorrect Liquidity Share Calculation When Adding/Removing Liquidity	Fixed	Confirmed
NVE- 002	Critical	Incorrect Liquidity Calculation After Burn	Acknowledge	Confirmed
NVE- 003	Critical	Incorrect Reference Check Leading to BurnRef Borrowing Failure	Fixed	Confirmed

Table 2.3: Key Audit Findings

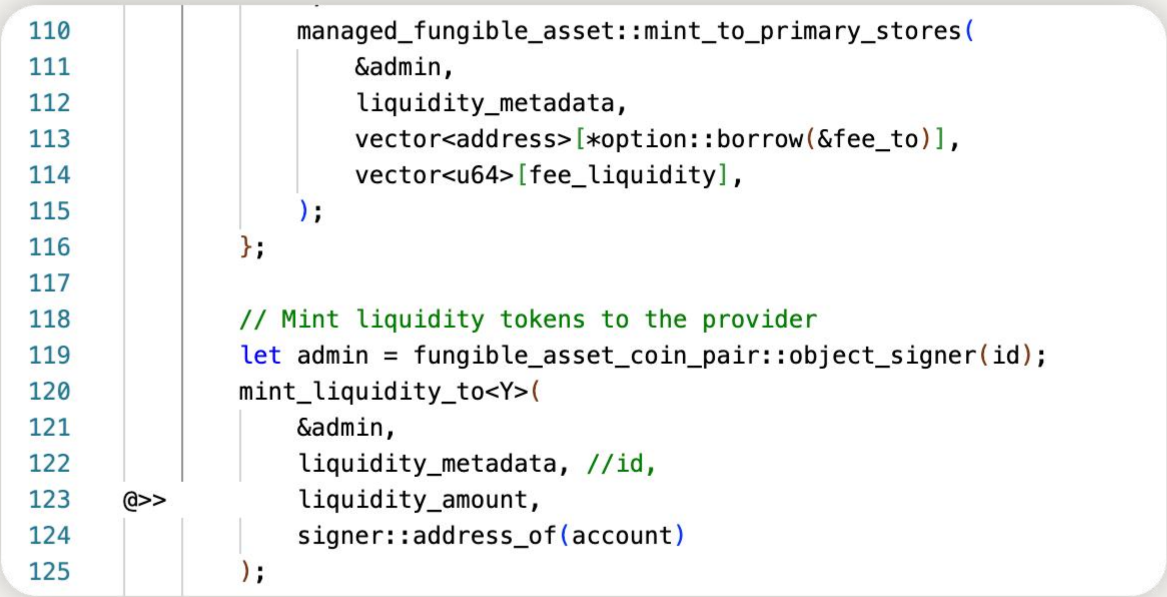
3. DETAILED DESCRIPTION OF FINDINGS

3.1 Incorrect Liquidity Share Calculation When Adding/Removing Liquidity

ID:	NVE-001	**Location:**	fungible_asset_coin_pair_add_liquidity_logic.move
Severity:	Critical	**Category:**	Business Issues
Likelihood:	High	**Impact:**	Critical

Description:

In the fungible_asset_coin_pair_add_liquidity_logic:mutate function, when adding liquidity to an existing pool, providers receive incorrect liquidity shares. When fee_on == true, the function first mints fee liquidity to fee_to, then calculates the provider's liquidity using the pre-fee total_liquidity from verify. This creates a discrepancy because the actual total liquidity should be total_liquidity + fee_liquidity, resulting in inaccurate share calculations for all users.



```
        managed_fungible_asset::mint_to_primary_stores(
            &admin,
            liquidity_metadata,
            vector<address>[*option::borrow(&fee_to)],
            vector<u64>[fee_liquidity],
        );
    };

    // Mint liquidity tokens to the provider
    let admin = fungible_asset_coin_pair::object_signer(id);
    mint_liquidity_to<Y>(
        &admin,
        liquidity_metadata, //id,
        liquidity_amount,
        signer::address_of(account)
    );
```

Recommendations:

Modify the fungible_asset_coin_pair_add_liquidity_logic:mutate function to update the total liquidity and recalculate shares after fee distribution. Apply similar changes to fungible_asset_coin_pair_remove_liquidity_logic:mutate.

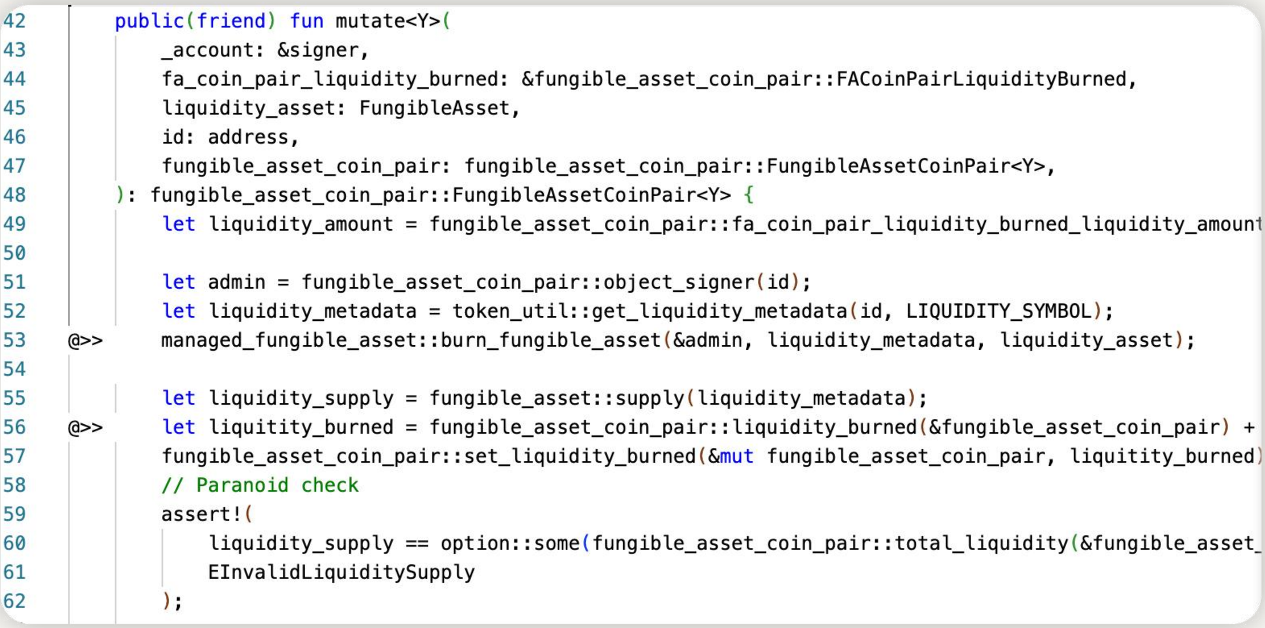
Result: Fixed

3.2 Incorrect Liquidity Calculation After Burn

ID:	NVE-002	**Location:**	fungible_asset_coin_pair_burn_liquidity_logic.move
Severity:	Critical	**Category:**	Business Issues
Likelihood:	High	**Impact:**	Critical

Description:

After burning liquidity, the system fails to deduct the burned amount from total_liquidity, instead tracking the total burned amount separately via burn_fungible_asset. When subsequently adding or removing liquidity, calculations still use the unadjusted total_liquidity without subtracting burn_fungible_asset, leading to inaccurate liquidity computations.



```
    public(friend) fun mutate<Y>(
        _account: &signer,
        fa_coin_pair_liquidity_burned: &fungible_asset_coin_pair::FACoinPairLiquidityBurned,
        liquidity_asset: FungibleAsset,
        id: address,
        fungible_asset_coin_pair: fungible_asset_coin_pair::FungibleAssetCoinPair<Y>,
    ): fungible_asset_coin_pair::FungibleAssetCoinPair<Y> {
        let liquidity_amount = fungible_asset_coin_pair::fa_coin_pair_liquidity_burned_liquidity_amount

        let admin = fungible_asset_coin_pair::object_signer(id);
        let liquidity_metadata = token_util::get_liquidity_metadata(id, LIQUIDITY_SYMBOL);
@>>     managed_fungible_asset::burn_fungible_asset(&admin, liquidity_metadata, liquidity_asset);

        let liquidity_supply = fungible_asset::supply(liquidity_metadata);
@>>     let liquitity_burned = fungible_asset_coin_pair::liquidity_burned(&fungible_asset_coin_pair) +
        fungible_asset_coin_pair::set_liquidity_burned(&mut fungible_asset_coin_pair, liquitity_burned)
        // Paranoid check
        assert!(
            liquidity_supply == option::some(fungible_asset_coin_pair::total_liquidity(&fungible_asset_
            EInvalidLiquiditySupply
        );
```

Recommendations:

When calculating total_liquidity for adding or removing liquidity, subtract the burn_fungible_asset amount first to ensure accurate liquidity calculations.

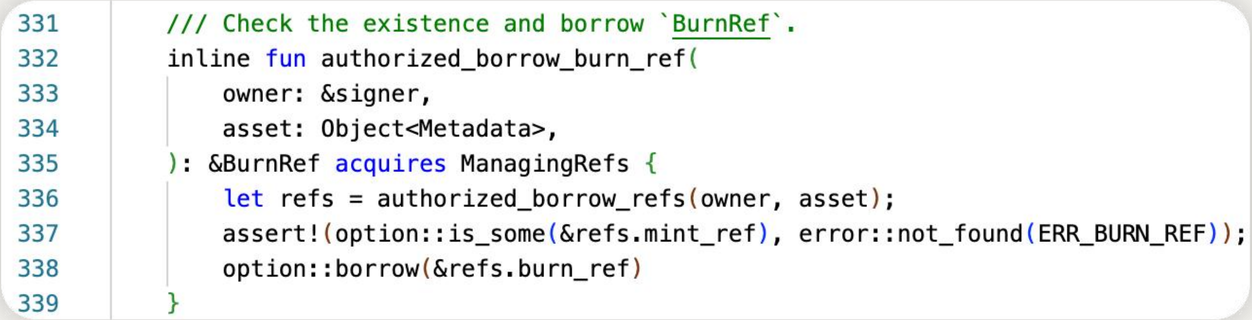
Result: Acknowledge. This aligns with the client's design intent of using burned LP tokens to increase the liquidity pool.

3.3 Incorrect Reference Check Leading to BurnRef Borrowing Failure

ID:	NVE-003	**Location:**	managed_fungible_asset.move
Severity:	Critical	**Category:**	Business Issues
Likelihood:	High	**Impact:**	Critical

Description:

The provided code snippet contains a logic error in the authorized_borrow_burn_ref function. The function is intended to check the existence of mint_ref and then borrow from burn_ref. However, the code mistakenly checks for the existence of mint_ref but attempts to borrow from burn_ref, which is not the correct reference to check.



```
/// Check the existence and borrow `BurnRef`.
inline fun authorized_borrow_burn_ref(
    owner: &signer,
    asset: Object<Metadata>,
): &BurnRef acquires ManagingRefs {
    let refs = authorized_borrow_refs(owner, asset);
    assert!(option::is_some(&refs.mint_ref), error::not_found(ERR_BURN_REF));
    option::borrow(&refs.burn_ref)
}
```

Recommendations:

The code should check for the existence of burn_ref instead of mint_ref before borrowing from burn_ref.

Result: Fixed

4. CONCLUSION

In this audit, we thoroughly analyzed **Flex Swap** smart contract implementation. The problems found are described and explained in detail in Section 3. The problems found in the audit have been communicated to the project leader. We therefore consider the audit result to be PASSED. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

5. APPENDIX

5.1 Basic Coding Assessment

5.1.1 Apply Verification Control

- Description: The security of apply verification
- Result: Not found
- Severity: Critical

5.1.2 Authorization Access Control

- Description: Permission checks for external integral functions
- Result: Not found
- Severity: Critical

5.1.3 Forged Transfer Vulnerability

- Description: Assess whether there is a forged transfer notification vulnerability in the contract
- Result: Not found
- Severity: Critical

5.1.4 Transaction Rollback Attack

- Description: Assess whether there is transaction rollback attack vulnerability in the contract.
- Result: Not found
- Severity: Critical

5.1.5 Transaction Block Stuffing Attack

- Description: Assess whether there is transaction blocking attack vulnerability.
- Result: Not found
- Severity: Critical

5.1.6 Soft Fail Attack Assessment

- Description: Assess whether there is soft fail attack vulnerability.
- Result: Not found
- Severity: Critical

5.1.7 Hard Fail Attack Assessment

- Description: Examine for hard fail attack vulnerability
- Result: Not found
- Severity: Critical

5.1.8 Abnormal Memo Assessment

- Description: Assess whether there is abnormal memo vulnerability in the contract.
- Result: Not found
- Severity: Critical

5.1.9 Abnormal Resource Consumption

- Description: Examine whether abnormal resource consumption in contract processing.
- Result: Not found
- Severity: Critical

5.1.10 Random Number Security

- Description: Examine whether the code uses insecure random number.
- Result: Not found
- Severity: Critical

5.2 Advanced Code Scrutiny

5.2.1 Cryptography Security

- Description: Examine for weakness in cryptograph implementation.
- Results: Not Found
- Severity: High

5.2.2 Account Permission Control

- Description: Examine permission control issue in the contract
- Results: Not Found
- Severity: Medium

5.2.3 Malicious Code Behavior

- Description: Examine whether sensitive behavior present in the code
- Results: Not found
- Severity: Medium

5.2.4 Sensitive Information Disclosure

- Description: Examine whether sensitive information disclosure issue present in the code.
- Result: Not found
- Severity: Medium

5.2.5 System API

- Description: Examine whether system API application issue present in the code
- Results: Not found
- Severity: Low

6. DISCLAIMER

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without ExVul's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts ExVul to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. ExVul's position is that each company and individual are responsible for their own due diligence and continuous security. ExVul's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

7. REFERENCES

[1] MITRE. CWE- 191: Integer Underflow (Wrap or Wraparound).

https://cwe.mitre.org/data/ definitions/191.html.

[2] MITRE. CWE- 197: Numeric Truncation Error.

https://cwe.mitre.org/data/definitions/197. html.

[3] MITRE. CWE-400: Uncontrolled Resource Consumption.

https://cwe.mitre.org/data/ definitions/400.html.

[4] MITRE. CWE-440: Expected Behavior Violation.

https://cwe.mitre.org/data/definitions/440. html.

[5] MITRE. CWE-684: Protection Mechanism Failure.

https://cwe.mitre.org/data/definitions/ 693.html.

[6] MITRE. CWE CATEGORY: 7PK - Security Features.

https://cwe.mitre.org/data/definitions/ 254.html.

[7] MITRE. CWE CATEGORY: Behavioral Problems.

https://cwe.mitre.org/data/definitions/438. html.

[8] MITRE. CWE CATEGORY: Numeric Errors.

https://cwe.mitre.org/data/definitions/189.html.

[9] MITRE. CWE CATEGORY: Resource Management Errors.

https://cwe.mitre.org/data/ definitions/399.html.

[10] OWASP. Risk Rating Methodology.

https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

www.exvul.com

contact@exvul.com

@EXVULSEC

github.com/EXVUL-Sec

ExVul