Gmeme Smart Contract

SMART CONTRACT AUDIT REPORT

April 2025

ExVul

www.exvul.com

Table of Contents

1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by **Gmeme** 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.

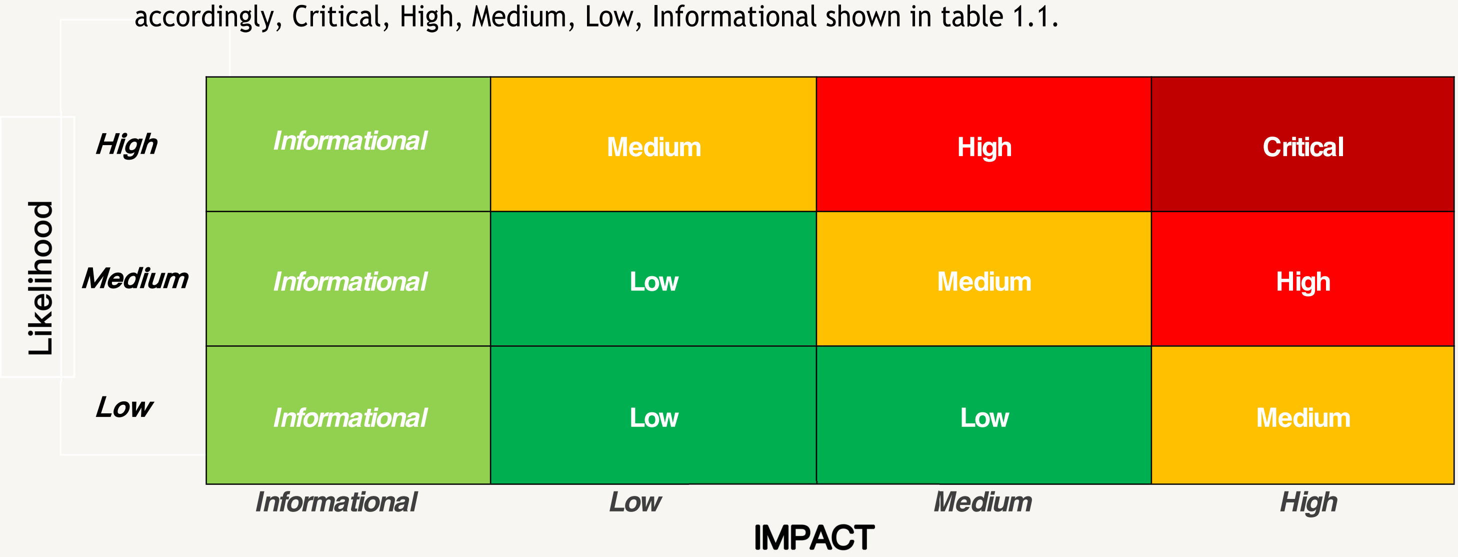
The outcome of the assessment outlined in chapter 3 provides the system's owners a full description of the vulnerabilities identified, the associated risk rating for each vulnerability, and detailed recommendations that will resolve the underlying technical issue.

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 \ IMPACT	Informational	Low	Medium	High
High	Informational	Medium	High	Critical
Medium	Informational	Low	Medium	High
Low	Informational	Low	Low	Medium

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: Gmeme

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

Language: move

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

2.2 Summary

Severity	Found	
Critical	0	
High	0	
Medium	3	🟨🟨🟨
Low	2	🟦🟦
Informational	2	🟩🟩

2.3 Key Findings

ID	Severity	Findings Title	Status	Confirm
NVE- 001	Medium	Invalid Collateral and Token Amounts Due to Missing Input Validation	Fixed	Confirmed
NVE- 002	Medium	Invalid Transaction Execution Due to Missing Input Validation	Fixed	Confirmed
NVE- 003	Medium	Insufficient Protocol Fee Calculation Due to Low Collateral Amount	Fixed	Confirmed
NVE- 004	Low	Mismatched Vector Lengths in Private Minting Function	Acknowledge	Confirmed
NVE- 005	Low	Option Extraction Without Graceful Error Handling	Acknowledge	Confirmed
NVE- 006	Info	Missing Log on Migration Failure	Acknowledge	Confirmed
NVE- 007	Info	Incorrect Word Spelling	Fixed	Confirmed

Table 2.3: Key Audit Findings

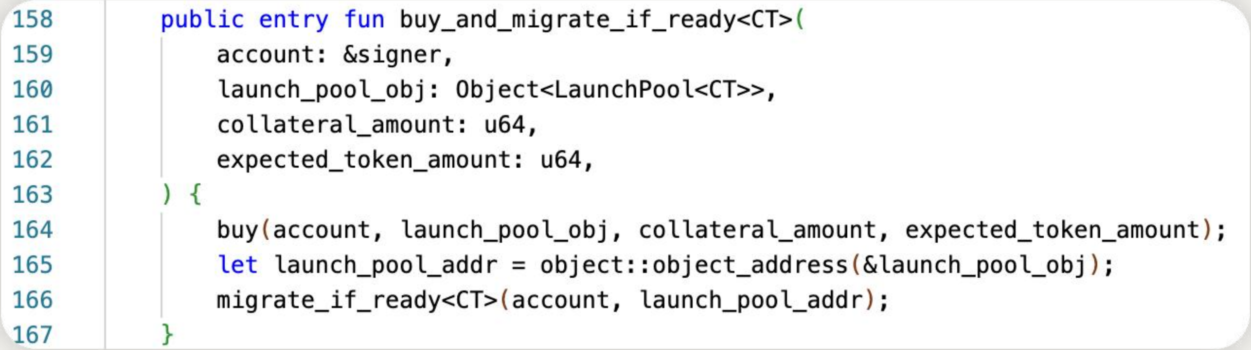
3. DETAILED DESCRIPTION OF FINDINGS

3.1 Invalid Collateral and Token Amounts Due to Missing Input Validation

ID:	NVE-001	**Location:**	launchpad_service.move
Severity:	Medium	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	Medium

Description:

The buy_and_migrate_if_ready function processes a purchase using collateral_amount and expected_token_amount but does not validate whether these values are greater than 0. This lack of validation could allow invalid inputs, leading to unexpected behavior or errors.



```
    public entry fun buy_and_migrate_if_ready<CT>(
        account: &signer,
        launch_pool_obj: Object<LaunchPool<CT>>,
        collateral_amount: u64,
        expected_token_amount: u64,
    ) {
        buy(account, launch_pool_obj, collateral_amount, expected_token_amount);
        let launch_pool_addr = object::object_address(&launch_pool_obj);
        migrate_if_ready<CT>(account, launch_pool_addr);
    }
```

Recommendations:

Implement validation to ensure both collateral_amount and expected_token_amount are greater than 0 before proceeding with the purchase and migration operations.

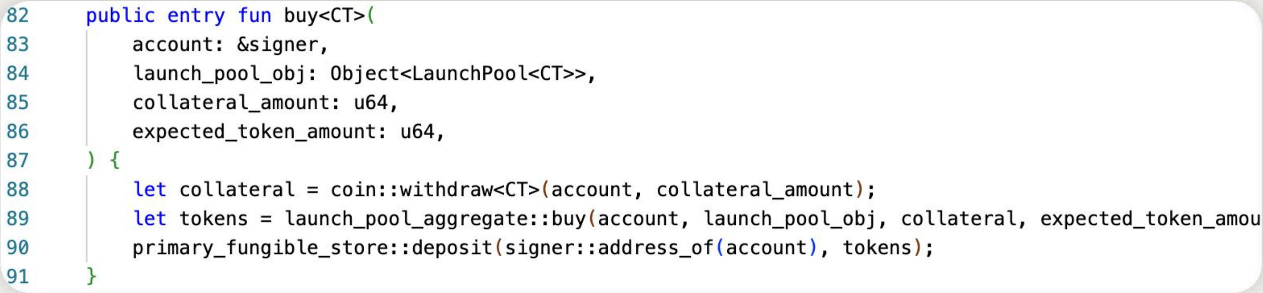
Result: Fixed

3.2 Invalid Transaction Execution Due to Missing Input Validation

ID:	NVE-002	**Location:**	launchpad_service.move
Severity:	Medium	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	Medium

Description:

The buy and sell functions allow users to execute transactions with collateral_amount, token_amount, or expected_collateral_amount set to 0. This lack of input validation can lead to unexpected behavior, such as successful transactions with zero amounts, which may violate business logic or system integrity.



```
    public entry fun buy<CT>(
        account: &signer,
        launch_pool_obj: Object<LaunchPool<CT>>,
        collateral_amount: u64,
        expected_token_amount: u64,
    ) {
        let collateral = coin::withdraw<CT>(account, collateral_amount);
        let tokens = launch_pool_aggregate::buy(account, launch_pool_obj, collateral, expected_token_amou
        primary_fungible_store::deposit(signer::address_of(account), tokens);
    }
```

Recommendations:

Implement validation to ensure that collateral_amount, token_amount, and expected collateral amount are greater than 0 before executing the transaction.

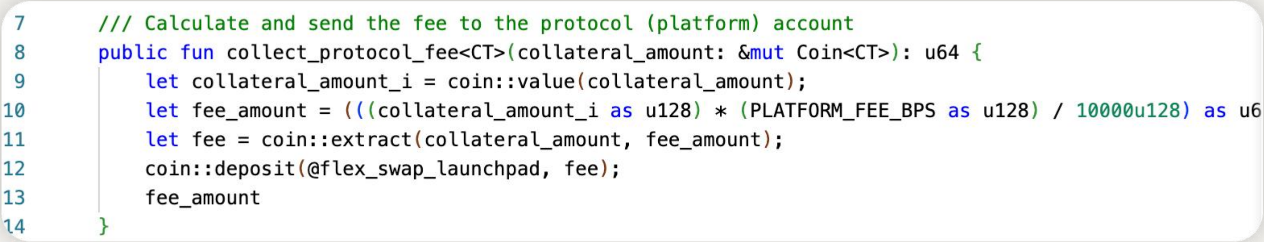
Result: Fixed

3.3 Insufficient Protocol Fee Calculation Due to Low Collateral Amount

ID:	NVE-003	**Location:**	launchpad_fee_util.move
Severity:	Medium	**Category:**	Business Issues
Likelihood:	Medium	**Impact:**	Medium

Description:

The collect_protocol_fee function calculates the protocol fee based on collateral_amount. However, if collateral_amount is smaller than 10000 / 30, the calculated fee_amount will be 0. This can lead to insufficient protocol fees, potentially violating the minimum fee requirements or leaving the protocol account underfunded.



```
/// Calculate and send the fee to the protocol (platform) account
public fun collect_protocol_fee<CT>(collateral_amount: &mut Coin<CT>): u64 {
    let collateral_amount_i = coin::value(collateral_amount);
    let fee_amount = (((collateral_amount_i as u128) * (PLATFORM_FEE_BPS as u128) / 10000u128) as u6
    let fee = coin::extract(collateral_amount, fee_amount);
    coin::deposit(@flex_swap_launchpad, fee);
    fee_amount
}
```

Recommendations:

Add a minimum fee amount to ensure that the protocol always receives a non-zero fee, even for small collateral_amount values.

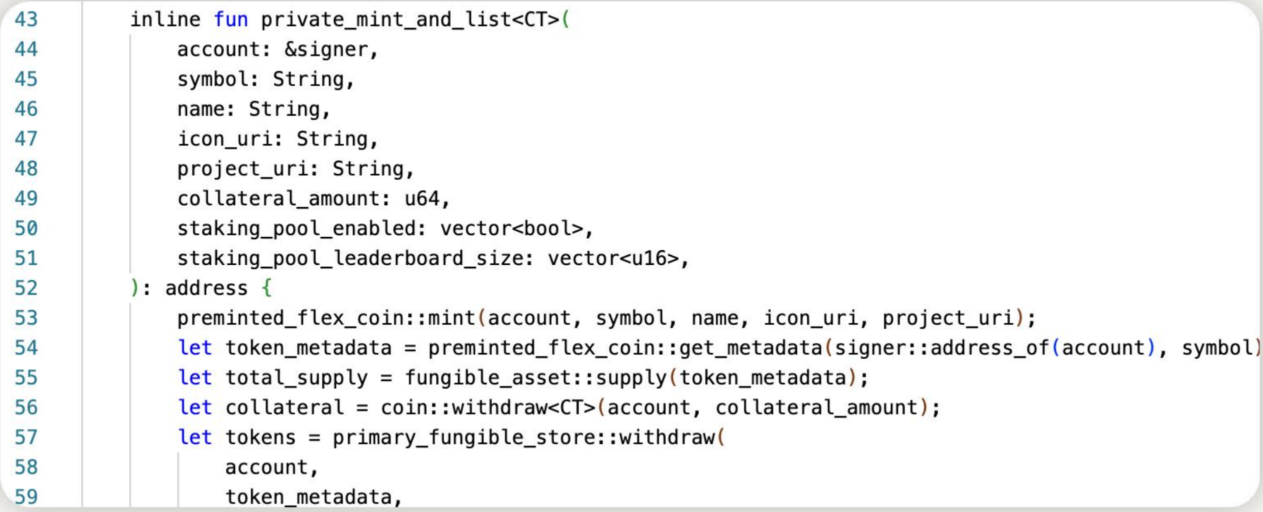
Result: Fixed

3.4 Mismatched Vector Lengths in Private Minting Function

ID:	NVE-004	**Location:**	launchpad_service.move
Severity:	Low	**Category:**	Business Issues
Likelihood:	Medium	**Impact:**	Low

Description:

The private_mint_and_list function accepts two parameters, staking pool enabled and staking pool leaderboard size, which are vectors. However, the function does not validate whether these vectors have the same length.



```
    inline fun private_mint_and_list<CT>(
        account: &signer,
        symbol: String,
        name: String,
        icon_uri: String,
        project_uri: String,
        collateral_amount: u64,
        staking_pool_enabled: vector<bool>,
        staking_pool_leaderboard_size: vector<u16>,
    ): address {
        preminted_flex_coin::mint(account, symbol, name, icon_uri, project_uri);
        let token_metadata = preminted_flex_coin::get_metadata(signer::address_of(account), symbol)
        let total_supply = fungible_asset::supply(token_metadata);
        let collateral = coin::withdraw<CT>(account, collateral_amount);
        let tokens = primary_fungible_store::withdraw(
            account,
            token_metadata,
```

Recommendations:

Add a validation check to ensure that staking_pool_enabled and staking_pool_leaderboard_size have the same length before proceeding.

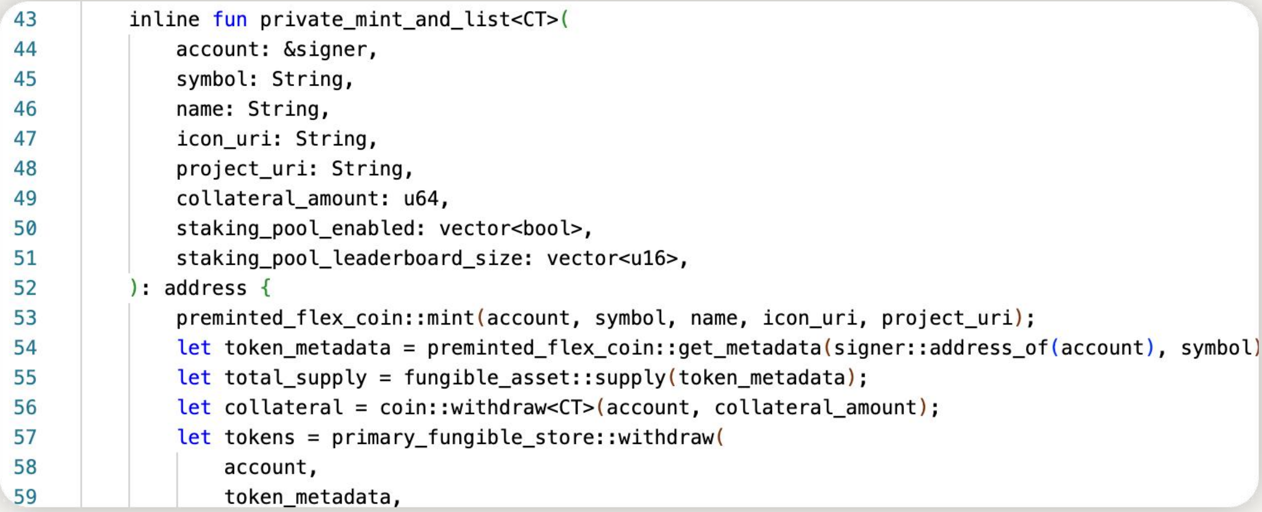
Result: Acknowledge. The client said there's no need to check here because the entry function doesn't support using Option.

3.5 Option Extraction Without Graceful Error Handling

ID:	NVE-005	Location:	launchpad_service.move
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

Description:

The private_mint_and_list function uses option::extract to retrieve the total supply from fungible_asset::supply. If total_supply is None, the transaction will abort without providing a meaningful error message.



```
inline fun private_mint_and_list<CT>(
    account: &signer,
    symbol: String,
    name: String,
    icon_uri: String,
    project_uri: String,
    collateral_amount: u64,
    staking_pool_enabled: vector<bool>,
    staking_pool_leaderboard_size: vector<u16>,
): address {
    preminted_flex_coin::mint(account, symbol, name, icon_uri, project_uri);
    let token_metadata = preminted_flex_coin::get_metadata(signer::address_of(account), symbol)
    let total_supply = fungible_asset::supply(token_metadata);
    let collateral = coin::withdraw<CT>(account, collateral_amount);
    let tokens = primary_fungible_store::withdraw(
        account,
        token_metadata,
```

Recommendations:

Replace option::extract with a check to ensure total_supply is Some before proceeding.

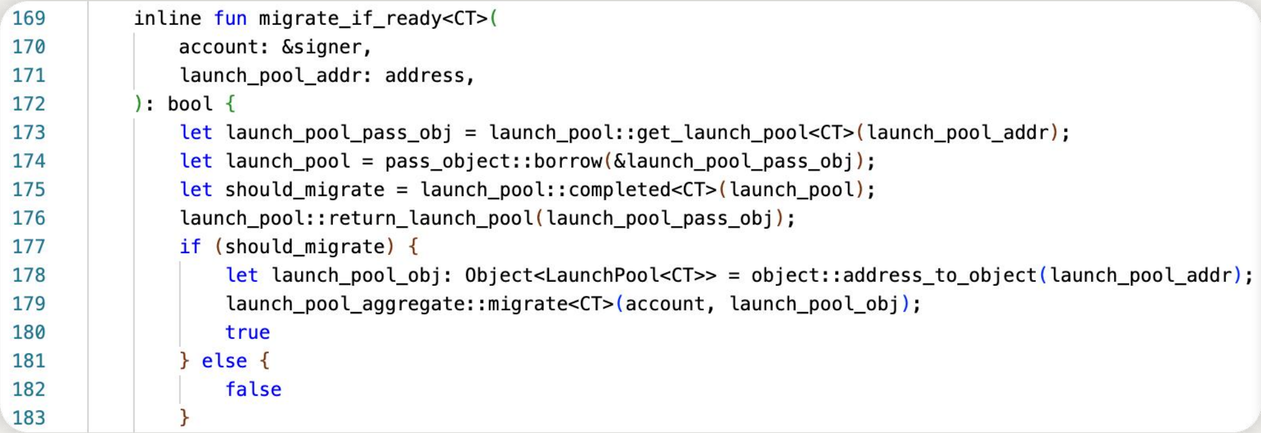
Result: Acknowledge. The client said it's a Meme Coin they minted themselves, so it shouldn't be None here.

3.6 Missing Log on Migration Failure

ID:	NVE-006	**Location:**	launchpad_service.move
Severity:	Info	**Category:**	Business Issues
Likelihood:	Info	**Impact:**	Info

Description:

The migrate_if_ready function checks whether migration should occur and returns true if migration is successful. However, if migration does not proceed (i.e., should_migrate is false), the function returns false without providing any logging or indication of why migration did not occur. This lack of logging makes it difficult to debug or monitor why migration might have failed.



```
inline fun migrate_if_ready<CT>(
    account: &signer,
    launch_pool_addr: address,
): bool {
    let launch_pool_pass_obj = launch_pool::get_launch_pool<CT>(launch_pool_addr);
    let launch_pool = pass_object::borrow(&launch_pool_pass_obj);
    let should_migrate = launch_pool::completed<CT>(launch_pool);
    launch_pool::return_launch_pool(launch_pool_pass_obj);
    if (should_migrate) {
        let launch_pool_obj: Object<LaunchPool<CT>> = object::address_to_object(launch_pool_addr);
        launch_pool_aggregate::migrate<CT>(account, launch_pool_obj);
        true
    } else {
        false
    }
```

Recommendations:

Add logging to the migrate_if_ready function to record when migration does not occur.

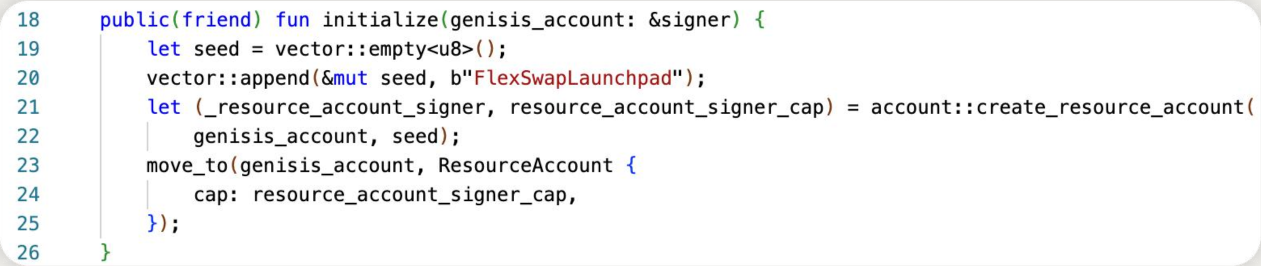
Result: Acknowledge. The client plans to use off-chain monitoring.

3.7 Incorrect Word Spelling

ID:	NVE-007	**Location:**	flex_swap_launchpad_resource_account.move
Severity:	Info	**Category:**	Business Issues
Likelihood:	Info	**Impact:**	Info

Description:

In the flex_swap_launchpad_resource_account:initialize function, there is a spelling mistake in the parameter name genisis_account.



```
public(friend) fun initialize(genisis_account: &signer) {
    let seed = vector::empty<u8>();
    vector::append(&mut seed, b"FlexSwapLaunchpad");
    let (_resource_account_signer, resource_account_signer_cap) = account::create_resource_account(
        genisis_account, seed);
    move_to(genisis_account, ResourceAccount {
        cap: resource_account_signer_cap,
    });
}
```

Recommendations:

The parameter name genisis_account should be corrected to genesis_account.

Result: Fixed

4. CONCLUSION

In this audit, we thoroughly analyzed **Gmeme** 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