fliexible-staking Smart Contract

SMART CONTRACT AUDIT REPORT

September 2024

ExVul

www.exvul.com

Table of Contents

1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by Bitget 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: fliexible-staking

Audit Time: September 5, 2024 – September 9, 2024

Language: Rust

File Name	Link
bwb-fliexible-staking-solana	https://github.com/bitgetwallet/bwb-fliexible-staking-solana/515a45ecf274440e1733d3bdd355d3fb19809d91
bwb-fliexible-staking-solana	https://github.com/bitgetwallet/bwb-fliexible-staking-solana/b7d72f091744b16cb928ea2d7b5ee251c8d4159d

2.2 Summary

Severity	Found	
Critical	0	
High	0	
Medium	2	
Low	3	
Informational	0	

2.3 Key Findings

ID	Severity	Findings Title	Status	Confirm
NVE-001	Medium	After canceling the pledge, you can still claim the reward through the claim_reward method	Fixed	Confirmed
NVE-002	Medium	Privileged roles	Mitigated	Confirmed
NVE-003	Low	Errors in logic validation	Fixed	Confirmed
NVE-004	Low	Calculation accuracy issues	Ignored	Confirmed
NVE-005	Low	Converting from u128 to u64 may overflow	Ignored	Confirmed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 After canceling the pledge, you can still claim the reward through the claim_reward method

ID:	NVE-001	**Location:**	lib.rs
Severity:	Medium	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	High

Description:

Both the claim_reward method and the unstake method can claim rewards, but the difference is that the unstake method will take away the staked funds. Under normal circumstances, users can only claim rewards based on the amount of staked funds when the staked funds are still in the contract. If the user has canceled the stake, then the user should no longer be entitled to claim rewards. The problem here is:

When the user calls unstake during the staking period, they have already claimed part of the rewards, and the funds have been withdrawn. At this point, the user has actually no longer staked any funds, so they should be prohibited from continuing to call claim_reward. Since claim_reward only calculates rewards without checking the user's stake status or funds, rewards can still be claimed.

The main problems are as follows:

claim_reward does not determine whether the user has canceled the stake: When the user cancels the stake through unstake, the funds have been withdrawn from the contract, but claim_reward does not check whether the user has canceled the stake (that is, it does not check the status of order.is_unstake). This results in the user still being able to call claim_reward and claim the remaining rewards even after the stake is over.

claim_reward does not calculate the amount of staked funds: In claim_reward, the reward is calculated only based on the user's staked days (passed_days) and total rewards (reward_amount), without considering whether the user still has funds staked in the contract.

```rust
pub fn claim_reward(
    ctx: Context<ClaimReward>,
    order_id: u64,
    reward_amount: u64
) -> Result<()> {
    msg!("Instruction: Claim Reward");

    // check paused status
    let admin_info = &ctx.accounts.admin_info;
    require!(!admin_info.is_paused, ErrorCode::ProtocolPaused);
    let pool = &mut ctx.accounts.pool;
    require!(!pool.is_paused, ErrorCode::PoolPaused);

    // check order_id
    let user_info = &mut ctx.accounts.user_info;
    require!(order_id < user_info.next_order_id, ErrorCode::InvalidOrderId);

    // check order info
    let clock = Clock::get()?;
    let order = &mut ctx.accounts.order;
    require!(order.staker == ctx.accounts.user.key(), ErrorCode::InvalidUser);
    require!(clock.unix_timestamp >= order.start_time, ErrorCode::NotStartClaimReward);

    // claim reward
    let mut real_time: i64 = clock.unix_timestamp;
    if clock.unix_timestamp >= order.unstake_time {
        real_time = order.unstake_time;
    }
    let passed_days = ((real_time.checked_sub(order.start_time).ok_or(ErrorCode::ArithmeticError)?) as u64).checked_div(ONE_DAY).ok_or(ErrorCode::ArithmeticError)?;

    let period_days = pool.duration.checked_div(ONE_DAY).ok_or(ErrorCode::ArithmeticError)?;
    let reward = ((order.reward_amount.checked_mul(passed_days).ok_or(ErrorCode::ArithmeticError)?).checked_div(period_days).ok_or(ErrorCode::ArithmeticError)?).checked_sub(order.claimed_reward).ok_or(ErrorCode::ArithmeticError)?;
```

Figure 3.1.1 claim_reward() function

Recommendations:

Exvul Web3 Security recommends adding a check on the staking status in claim_reward to ensure that the user still has funds staked in the contract when calling claim_reward.

Result: Confirmed

Fix Result: Fixed

Customer response: Claim_reward function adds order.is_unstake check.

Fixed version: b7d72f091744b16cb928ea2d7b5ee251c8d4159d

```rust
require!(!order.is_unstake, ErrorCode::OrderAlreadyUnstake);
```

3.2 Privileged roles

ID:	NVE-002	**Location:**	lib.rs
Severity:	Medium	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	High

Description:

Many important operations in the code, such as staking pool updates, administrator updates, token withdrawals, etc., rely on the permission management of specific roles. It is recommended to use multi-signature management privileged roles.

```rust
pub fn withdraw_bwb_token(ctx: Context<WithdrawBWBToken>, amount: u64) -> Result<()> {

    // Transfer tokens from taker to initializer
    let bump = ctx.bumps.admin_info;
    let seeds = &[b"admin_info".as_ref(), &[bump]];

    token::transfer(
        CpiContext::new(
            ctx.accounts.token_program.to_account_info(),
            token::Transfer {
                from: ctx.accounts.from_token_account.to_account_info(),
                to: ctx.accounts.to_token_account.to_account_info(),
                authority: ctx.accounts.admin_info.to_account_info(),
            },
        )
        .with_signer(&[&seeds[..]]),
        amount,
    )?;

    Ok(())
}

pub fn withdraw_other_token(ctx: Context<WithdrawOtherToken>, amount: u64) -> Result<()> {

    token::transfer(
        CpiContext::new(
            ctx.accounts.token_program.to_account_info(),
            token::Transfer {
                from: ctx.accounts.from_token_account.to_account_info(),
                to: ctx.accounts.to_token_account.to_account_info(),
                authority: ctx.accounts.from_ata_owner.to_account_info(),
            }
        ),
        amount
    )?;

    Ok(())
}
```

Figure 3.2.1 withdraw_bwb_token () function

Recommendations:

Exvul Web3 Security recommends using multi-signature to manage privileged roles.

Result: Confirmed

Fix Result: Mitigated

Customer response: In terms of authority design, the authority to set contract upgrades and management addresses will be given to multi-signature wallets. Other management addresses are held by the project operator, and the receiver address of the contract is also held by the project operator.

3.3 Errors in logic validation

ID:	NVE-003	Location:	lib.rs
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

Description:

In the update_pool function, there is the following problem: 336 > 0 here is an unreasonable hard-coded logic. It should be reward_cap > 0, that is, it should check whether reward_cap is greater than zero. The hard-coded 336 has no practical meaning. It may fail to correctly check the rationality of reward_cap, thereby updating an unreasonable staking pool (for example, the reward cap is 0), which will affect the normal operation of the contract.

```
pub fn update_pool(
    ctx: Context<UpdatePool>,
    pool_id: u64,
    stake_cap: u64,
    reward_cap: u64,
    stake_start_at: i64,
    stake_end_at: i64,
    duration: u64,
    duration_days: u64
)-> Result<()> {
    msg!("Instruction: update_pool");
    // check paused status
    let admin_info = &ctx.accounts.admin_info;
    require!(!admin_info.is_paused, ErrorCode::ProtocolPaused);

    let clock = Clock::get()?;
    require!(stake_start_at >= clock.unix_timestamp, ErrorCode::StartTimeNeedGTENow);
    require!(stake_end_at > stake_start_at, ErrorCode::StartTimeNeedLTEndTime);

    require!(duration > 0, ErrorCode::DurationNeedGT0);
    require!(duration == duration_days.checked_mul(ONE_DAY)
        .ok_or(ErrorCode::ArithmeticError)?, ErrorCode::DurationMustBeMultiDays
    );

    require!(stake_cap > 0 && 336 > 0, ErrorCode::TwoCapsNeedGT0);

    let pool = &mut ctx.accounts.pool;
    let clock = Clock::get()?;
    require!(clock.unix_timestamp < pool.stake_start_at, ErrorCode::PoolAlreadyStartStake);
```

Figure 3.3.1 update_pool () function

Recommendations:

Exvul Web3 Security recommends checking if reward_cap is greater than zero.

Result: Confirmed

Fix Result: Fixed

Customer response: Remove the test code and add reward_cap value judgment.

Fixed version: b7d72f091744b16cb928ea2d7b5ee251c8d4159d

```
require!(stake_cap > 0 && reward_cap > 0, ErrorCode::TwoCapsNeedGT0);
```

3.4 Calculation accuracy issues

ID:	NVE-004	**Location:**	lib.rs
Severity:	Low	**Category:**	Business Issues
Likelihood:	Low	**Impact:**	Low

Description:

Since the division operation in the stake method involves calculating the proportion of the reward cap (reward_cap) to the stake cap (stake_cap), if the reward_cap and stake_cap values are large or the stake amount is small, the result after division may produce a loss of precision.

Example:

reward_cap = 1,000,000,000 (total reward pool).

stake_cap = 10,000,000,000 (total stake pool).

amount = 1 (the amount staked by the user).

In this example, the result of the calculation is 0.1, and since integer division is used, the result will be truncated to 0, so the user will not actually receive any rewards.

```rust
pub fn stake(ctx: Context<Stake>, pool_id: u64, amount: u64) -> Result<()> {
    msg!("Instruction: Stake");
    require!(amount > 0, ErrorCode::StakeAmountNeedGT0);
    // check paused status
    let admin_info = &ctx.accounts.admin_info;
    require!(!admin_info.is_paused, ErrorCode::ProtocolPaused);
    let pool = &mut ctx.accounts.pool;
    require!(!pool.is_paused, ErrorCode::PoolPaused);
    // check user token balance
    require!(amount  <= ctx.accounts.user_token_wallet.amount, ErrorCode::AmountOverBalance);

    // check pool limit
    require!(pool_id < admin_info.next_pool_id, ErrorCode::InvalidPoolId);
    require!(amount <= pool.stake_visible, ErrorCode::NotVisibleStakeAmount);
    // check stake time limit
    let clock = Clock::get()?;
    require!(clock.unix_timestamp > pool.stake_start_at, ErrorCode::NotStakeTime);
    require!(clock.unix_timestamp < pool.stake_end_at, ErrorCode::NotStakeTime);

    // Transfer BWB
    let cpi_accounts = Transfer {
        from: ctx.accounts.user_token_wallet.to_account_info(),
        to: ctx.accounts.vault_token_account.to_account_info(),
        authority: ctx.accounts.user.to_account_info(),
    };
    let cpi_program = ctx.accounts.token_program.to_account_info();
    let cpi_ctx = CpiContext::new(cpi_program, cpi_accounts);
    token::transfer(cpi_ctx, amount)?;

    // update order info
    let new_order = &mut ctx.accounts.new_order;
    new_order.order_id = ctx.accounts.user_info.next_order_id;
    new_order.pool_id = pool_id;
    new_order.staker = ctx.accounts.user.key();
    new_order.stake_amount = amount;

    let tmp: u128 = (pool.reward_cap as u128).checked_mul(amount as u128).ok_or(ErrorCode::ArithmeticError)?;
    new_order.reward_amount = (tmp.checked_div(pool.stake_cap as u128).ok_or(ErrorCode::ArithmeticError)?) as u64;

    new_order.start_time = clock.unix_timestamp;
    new_order.unstake_time = clock.unix_timestamp.checked_add(pool.duration as i64).ok_or(ErrorCode::ArithmeticError)?;
```

Figure 3.4.1 stake () function

Recommendations:

Exvul Web3 Security recommends introducing high-precision integers or floating-point numbers in calculations to retain decimal parts.

Result: Confirmed

Fix Result: Ignored

Customer response: No processing. There is indeed a loss of minimum precision, but the token will not lose all precision.

3.5 Converting from u128 to u64 may overflow

ID:	NVE-005	Location:	lib.rs
Severity:	Low	Category:	Business Issues
Likelihood:	Low	Impact:	Low

Description:

In the stake method, when calculating reward_amount, the u128 type will be converted to u64. Since the value range allowed by the u128 type is much larger than that of u64, the value exceeding the u64 range may be converted to u64 during the reward calculation. If the result of tmp/pool.stake_cap exceeds the upper limit of u64 (18,446,744,073,709,551,615), the result will overflow, causing the user to receive an incorrect reward amount.

```rust
pub fn stake(ctx: Context<Stake>, pool_id: u64, amount: u64) -> Result<()> {
    msg!("Instruction: Stake");
    require!(amount > 0, ErrorCode::StakeAmountNeedGT0);
    // check paused status
    let admin_info = &ctx.accounts.admin_info;
    require!(!admin_info.is_paused, ErrorCode::ProtocolPaused);
    let pool = &mut ctx.accounts.pool;
    require!(!pool.is_paused, ErrorCode::PoolPaused);
    // check user token balance
    require!(amount  <= ctx.accounts.user_token_wallet.amount, ErrorCode::AmountOverBalance);

    // check pool limit
    require!(pool_id < admin_info.next_pool_id, ErrorCode::InvalidPoolId);
    require!(amount <= pool.stake_visible, ErrorCode::NotVisibleStakeAmount);
    // check stake time limit
    let clock = Clock::get()?;
    require!(clock.unix_timestamp > pool.stake_start_at, ErrorCode::NotStakeTime);
    require!(clock.unix_timestamp < pool.stake_end_at, ErrorCode::NotStakeTime);

    // Transfer BWB
    let cpi_accounts = Transfer {
        from: ctx.accounts.user_token_wallet.to_account_info(),
        to: ctx.accounts.vault_token_account.to_account_info(),
        authority: ctx.accounts.user.to_account_info(),
    };
    let cpi_program = ctx.accounts.token_program.to_account_info();
    let cpi_ctx = CpiContext::new(cpi_program, cpi_accounts);
    token::transfer(cpi_ctx, amount)?;

    // update order info
    let new_order = &mut ctx.accounts.new_order;
    new_order.order_id = ctx.accounts.user_info.next_order_id;
    new_order.pool_id = pool_id;
    new_order.staker = ctx.accounts.user.key();
    new_order.stake_amount = amount;

    let tmp: u128 = (pool.reward_cap as u128).checked_mul(amount as u128).ok_or(ErrorCode::ArithmeticError)?;
    new_order.reward_amount = (tmp.checked_div(pool.stake_cap as u128).ok_or(ErrorCode::ArithmeticError)?) as u64;

    new_order.start_time = clock.unix_timestamp;
    new_order.unstake_time = clock.unix_timestamp.checked_add(pool.duration as i64).ok_or(ErrorCode::ArithmeticError)?;
```

Figure 3.5.1 stake () function

Recommendations:

Exvul Web3 Security recommends that you keep using u128 during the calculation process until the final calculation result is confirmed to not exceed the u64 limit, and then perform type conversion.

Result: Confirmed

Fix Result: Ignored

Customer response: No processing. Theoretically, this is possible, but in actual business, pool.reward_cap, pool., stake_cap, amount are all u64-bit in size, and pool.reward_cap is 1 or 2 orders of magnitude smaller than pool.stake_cap. In the end, reward_amount ~= amount / 1eX, which is always smaller than u64_max. So no processing for now.

4. CONCLUSION

In this audit, we thoroughly analyzed **fliexible-staking** 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