ExVul

SMART CONTRACT AUDIT REPORT

Aimonica Smart Contract

JULY 2025

Contents

1. EXECUTIVE SUMMARY

ExVul Web3 Security was engaged by **Aimonica** 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: 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.

		Informational	Low	Medium	High
Likelihood	*High*	INFO	MEDIUM	HIGH	CRITICAL
	Medium	INFO	LOW	MEDIUM	HIGH
	Low	INFO	LOW	LOW	MEDIUM

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 • Abnormal Resource Consumption
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	Audit Time	Language
Aimonica	25/07/2025 - 31/07/2025	Rust

Repository	Commit Hash
https://github.com/Aimonica-Brands/aimonica-core-solana	b821ba5498692601019e8cd9517978e0a34fb1ba

2.2 Summary

Severity	Found
CRITICAL	0
HIGH	0
MEDIUM	1
LOW	3
INFO	2

2.3 Key Findings

Severity	Findings Title	Status
MEDIUM	Emergency Unstake Time Validation Bypass	Fixed
LOW	Zero Amount Stake Validation Missing	Fixed
LOW	Platform Authority Mismatch in Project Configuration Updates	Fixed
LOW	UserStakeInfo Account Not Closed After Unstaking	Fixed
INFO	Fee Basis Points Validation Missing	Fixed
INFO	Duplicate Durations Validation Missing	Fixed

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Emergency Unstake Time Validation Bypass

SEVERITY: MEDIUM **STATUS:** Fixed

PATH:

`emergency_unstake instruction`

DESCRIPTION:

The emergency_unstake instruction omits time validation, enabling users to bypass lockup restrictions and incur high emergency fees after the lockup period has ended. Unlike the standard unstake function which properly validates lockup completion, emergency unstake proceeds without temporal checks.

```
/// Performs an emergency unstake, allowing withdrawal before the lock-up
    period ends.
pub fn emergency_unstake(ctx: Context<EmergencyUnstake>, _stake_id: u64)
    -> Result<()> {
    let stake_info = &mut ctx.accounts.stake_info;

    // Transfer tokens from vault back to user
    let project_id_bytes =
        ctx.accounts.project_config.project_id.to_le_bytes();
    let authority_seeds = &[
        b”vault-authority”.as_ref(),
        project_id_bytes.as_ref(),
        &[ctx.bumps.vault_authority],
    ];
    let signer_seeds = &[&authority_seeds[..]];
}
```

IMPACT:

Users can be charged excessive emergency unstake fees after lockup expiration, violating the intended emergency withdrawal design and causing unnecessary financial losses.

RECOMMENDATIONS:

Add lockup period validation to prevent emergency unstake after lockup expiration:

```
pub fn emergency_unstake(ctx: Context<EmergencyUnstake>, _stake_id: u64)
    -> Result<()> {
     let stake_info = &mut ctx.accounts.stake_info;
+
+      // Validate lockup period has not ended
+      let clock = Clock::get()?;
+      let lockup_seconds = (stake_info.duration_days as i64) * 24 * 60 *
    60;
+      if stake_info.stake_timestamp + lockup_seconds <=
    clock.unix_timestamp {
+          return err!(ErrorCode::LockupPeriodEnded);
+      }

     // Transfer tokens from vault back to user
```

3.2 Zero Amount Stake Validation Missing

SEVERITY: LOW **STATUS:** Fixed

PATH:

`stake instruction`

DESCRIPTION:

The stake instruction lacks validation for the amount parameter, allowing users to stake 0 tokens. This creates meaningless stake records and wastes account space and rent. When fee_bps > 0, these zero-amount stakes become permanently locked as unstake operations fail due to insufficient tokens for fee payment.

```rust
pub fn stake(ctx: Context<Stake>, amount: u64, duration_days: u32,
    stake_id: u64) -> Result<()> {
     // Validate duration
     if
         !ctx.accounts.project_config.allowed_durations.contains(&duration_days)
         {
          return err!(ErrorCode::InvalidDuration);
     }

     // Transfer tokens from user to vault
     let cpi_accounts = Transfer {
          from: ctx.accounts.user_token_account.to_account_info(),
          to: ctx.accounts.vault.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_interface::transfer(cpi_ctx, amount)?;
}
```

IMPACT:

Users can create stake records with 0 tokens, wasting SOL rent and creating meaningless accounts.

When fee_bps > 0, zero-amount stakes become permanently locked as stake_info.is_staked remains true forever, creating a DoS attack vector.

RECOMMENDATIONS:

Add validation to prevent zero amount stakes:

```
pub fn stake(ctx: Context<Stake>, amount: u64, duration_days: u32,
    stake_id: u64) -> Result<()> {
+     if amount == 0 {
+         return err!(ErrorCode::InvalidAmount);
+     }

    // Validate duration
    if
        !ctx.accounts.project_config.allowed_durations.contains(&duration_days)
        {
        return err!(ErrorCode::InvalidDuration);
    }

    // Transfer tokens from user to vault
```

3.3 Platform Authority Mismatch in Project Configuration Updates

SEVERITY: LOW **STATUS:** Fixed

PATH:

`update_allowed_durations and update_project_config instructions`

DESCRIPTION:

The update_allowed_durations and update_project_config instructions only require project authority validation, not platform authority validation as documented in README. This allows removed platform authorities to retain project-level administrative privileges.

According to the README documentation: - `update_allowed_durations(new_durations: Vec<u32>)`: Updates the list of allowed staking durations for an existing project. - **Signer:** Platform Authority - `update_project_config(fee_wallet: Pubkey, unstake_fee_bps: u16, emergency_unstake_fee_bps: u16)`: Updates the fee configuration for an existing project. - **Signer:** Platform Authority

However, the current implementation only validates project authority:

```
#[derive(Accounts)]
pub struct UpdateAllowedDurations<'info> {
    #[account(
        mut,
        has_one = authority,
    )]
    pub project_config: Account<'info, ProjectConfig>,
    #[account(mut)]
    pub authority: Signer<'info>,
}
```

IMPACT:

Removed platform authorities can still modify project configurations (staking durations, fees) after being stripped of platform-level privileges, bypassing intended access controls and potentially causing unauthorized changes to project parameters.

RECOMMENDATIONS:

Add platform authority validation to both instructions to ensure proper access control alignment with documentation.

3.4 UserStakeInfo Account Not Closed After Unstaking

SEVERITY: LOW **STATUS:** Fixed

PATH:

`unstake and emergency_unstake instructions`

DESCRIPTION:

The unstake and emergency_unstake instructions do not close the UserStakeInfo account as documented, only setting is_staked to false. This contradicts the documentation which states the account should be closed and rent refunded.

According to the documentation:

```
/// Unstakes tokens after the lock-up period has ended.
///
/// This instruction checks if the lock-up duration has passed. If so, it
    transfers
/// the staked tokens back to the user, minus any applicable fees. The
    `UserStakeInfo`
/// account is closed, and the rent is refunded to the user.
pub fn unstake(ctx: Context<Unstake>, _stake_id: u64) -> Result<()> {
// ...
stake_info.is_staked = false;   // Only sets flag, doesn't close account
```

IMPACT:

Users continue paying rent for inactive stake accounts, wasting account space and SOL. This contradicts documented behavior and creates unnecessary costs for users.

RECOMMENDATIONS:

Add account closure logic to both unstake functions to properly close the UserStakeInfo account and refund rent to users as documented.

3.5 Fee Basis Points Validation Missing

SEVERITY: INFO **STATUS:** Fixed

PATH:

`update_project_config instruction`

DESCRIPTION:

The update_project_config instruction lacks validation for unstake_fee_bps and emergency_unstake_fee_bps parameters, allowing values exceeding 10000 basis points (100%). This can cause fee calculation errors in unstake operations.

```
pub fn update_project_config(
    ctx: Context<UpdateProjectConfig>,
    fee_wallet: Pubkey,
    unstake_fee_bps: u16,
    emergency_unstake_fee_bps: u16,
) -> Result<()> {
    let project_config = &mut ctx.accounts.project_config;
    project_config.fee_wallet = fee_wallet;
    project_config.unstake_fee_bps = unstake_fee_bps;
    project_config.emergency_unstake_fee_bps = emergency_unstake_fee_bps;
    Ok(())
}
```

IMPACT:

Setting fee basis points above 10000 (100%) could lead to arithmetic errors or unexpected behavior in fee calculations during unstake operations.

RECOMMENDATIONS:

Add validation to prevent fee_bps > 10000:

```
pub fn update_project_config(
    ctx: Context<UpdateProjectConfig>,
    fee_wallet: Pubkey,
    unstake_fee_bps: u16,
    emergency_unstake_fee_bps: u16,
) -> Result<()> {
+     if unstake_fee_bps > 10000 || emergency_unstake_fee_bps > 10000 {
+         return err!(ErrorCode::InvalidFeeBps);
+     }

    let project_config = &mut ctx.accounts.project_config;
    project_config.fee_wallet = fee_wallet;
    project_config.unstake_fee_bps = unstake_fee_bps;
    project_config.emergency_unstake_fee_bps = emergency_unstake_fee_bps;
    Ok(())
}
```

3.6 Duplicate Durations Validation Missing

SEVERITY: INFO **STATUS:** Fixed

PATH:

`update_allowed_durations instruction`

DESCRIPTION:

The update_allowed_durations instruction lacks validation to prevent duplicate values in the new_durations vector. This allows setting redundant duration values which wastes storage space and creates confusion in the allowed durations list.

```
pub fn update_allowed_durations(ctx: Context<UpdateAllowedDurations>,
    new_durations: Vec<u32>) -> Result<()> {
    if new_durations.len() > 10 {
        return err!(ErrorCode::TooManyDurations);
    }
    ctx.accounts.project_config.allowed_durations = new_durations;
    Ok(())
}
```

IMPACT:

Duplicate duration values waste storage space and can create confusion when users select staking durations, potentially leading to inconsistent user experience.

RECOMMENDATIONS:

Add validation to prevent duplicate durations by checking for uniqueness before setting the allowed_durations vector.

4. CONCLUSION

In this audit, we thoroughly analyzed **Aimonica** 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
Result	Not found
Severity	**HIGH**

5.2.2 Account Permission Control

Description	Examine permission control issue in the contract
Result	Not found
Severity	**MEDIUM**

5.2.3 Malicious Code Behavior

Description	Examine whether sensitive behavior present in the code
Result	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
Result	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

Contact

Website
www.exvul.com

Email
contact@exvul.com

Twitter
@EXVULSEC

Github
github.com/EXVUL-Sec

ExVul