ExVul

SMART CONTRACT AUDIT REPORT

BWSolanaSwap Smart Contract

JUNE 2025

Contents

1. EXECUTIVE SUMMARY

Exvul Web3 Security was engaged by **BWSolanaSwap** 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	INFO	MEDIUM	HIGH	CRITICAL
	Medium	INFO	LOW	MEDIUM	HIGH
	Low	INFO	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
	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

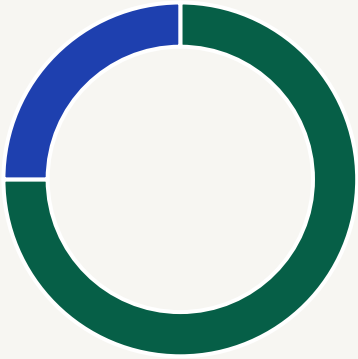
2.1 Project Info And Contract Address

Project Name	Audit Time	Language
BWSolanaSwap	June 23 2025 – June 27 2025	Rust

Soure code	Link
BWSolanaSwap	https://github.com/bitgetwallet/bgw-swap-aggregator-solana/commits/main/
Commit Hash	7f4b4af1eefcf563502e4df2b28de2cd9235cd8c

2.2 Summary

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



2.3 Key Findings

Severity	Findings Title	Status
LOW	Missing Zero Value Check in PDA SOL Transfer Function	Fixed
LOW	Missing Input-Output Token Identity Validation	Acknowledge
LOW	Missing Zero Weight Validation in Swap Router	Fixed
INFO	Unused Order ID Parameter Creates Security Risk	Acknowledge

Table 2.3: Key Audit Findings

3. DETAILED DESCRIPTION OF FINDINGS

3.1 Missing Zero Value Check in PDA SOL Transfer Function

Location	Severity	Category
transfer_helper.rs	LOW	Input Validation

Description:

The transfer_sol_from_proxy_pda function in BGW Swap Aggregator lacks proper validation for zero-value transfers before executing transaction logic. -It creates potential issues if called directly from other contexts.

```rust
pub fn transfer_sol_from_proxy_pda<'a>(
    proxy_pda: AccountInfo<'a>,
    to_pubkey: AccountInfo<'a>,
    system_program: AccountInfo<'a>,
    amount: u64,
    proxy_pda_index: u8,
) -> Result<()> {
    let signer_seeds: &[&[&[u8]]] = &[&[
        PROXY_PDA_SEEDS[proxy_pda_index as usize],
        &[PROXY_PDA_BUMPS[proxy_pda_index as usize]],
    ]];

    let cpi_context = CpiContext::new(
        system_program,
        Transfer {
            from: proxy_pda,
            to: to_pubkey,
        },
    )
    .with_signer(signer_seeds);

    transfer(cpi_context, amount)?;

    Ok(())
}
```

Impact

1. Unnecessary Computational Resource Consumption: Zero-value transfers waste computational resources by executing transfers that have no economic impact.

2. Transaction Fee Waste: Users may inadvertently pay transaction fees for zero-value transfers if called from contexts that bypass the pre_check validation.

Recommendations:

Implement a redundant zero check within transfer_sol_from_proxy_pda:

```rust
pub fn transfer_sol_from_proxy_pda(
    from_account: AccountInfo,
    to_account: AccountInfo,
    system_program: AccountInfo,
    amount: u64,
    proxy_pda_index: u8,
) -> Result<()> {
    // Early return if amount is zero
    if amount == 0 {
        return Ok(());
    }

    // Proceed with existing transfer logic
    // ...
}
```

Result	FixResult
Confirmed	Fixed

3.2 Missing Input-Output Token Identity Validation

Location	Severity	Category
swap_core.rs	LOW	Business Logic

Description:

The swap aggregator lacks validation to prevent users from swapping a token to itself. In swap_core.rs, the pre_check function only validates basic parameters but never checks if input and output tokens are identical:

```
fn pre_check<'info>(
    admin_info_account: &Account<'info, AdminInfo>,
    payer: &AccountInfo<'info>,
    sender: &AccountInfo<'info>,
    receiver: &AccountInfo<'info>,
    swap_params: &SwapParams,
    _order_id: u128,
) -> Result<()> {
    // Only validates basic parameters
    require!(!admin_info_account.is_paused, ErrorCode::ProtocolPaused);
    require!(swap_params.amount_in > 0, ErrorCode::AmountErrorAmountIn);
    // ... other basic checks
    // Missing: input_token_mint != output_token_mint check
    Ok(())
}
```

Impact

Users pay aggregator fees and gas costs without receiving any token exchange.

Recommendations:

Add token identity validation in the pre_check function.

Result	FixResult
Confirmed	Acknowledge

3.3 Missing Zero Weight Validation in Swap Rout

Location	Severity	Category
swap_core.rs	LOW	Input Validation

Description:

While the codebase correctly verifies that the sum of weights equals the TOTAL_WEIGHT constant (presumably 100), it fails to check if individual weights in the amount_in_weights array are zero. A weight of zero could lead to inefficiencies.

```
// Current validation only checks the sum of weights
        fn check_total_weights(weights: &[u8]) -> Result<()> {
    let total_weight: u8 = weights.iter().try_fold(0u8, |acc, &x| {
        acc.checked_add(x).ok_or(ErrorCode::CalcErrorTotalWeight)
    })?;
    require!(total_weight == TOTAL_WEIGHT, ErrorCode::InvalidTotalWeight);
    Ok(())
}

```

Recommendations:

```
fn check_total_weights(weights: &[u8]) -> Result<()> {

    require!(
        !weights.iter().any(|&weight| weight == 0),
        ErrorCode::ZeroWeightNotAllowed
    );

    let total_weight: u8 = weights.iter().try_fold(0u8, |acc, &x| {
        acc.checked_add(x).ok_or(ErrorCode::CalcErrorTotalWeight)
    })?;
    require!(total_weight == TOTAL_WEIGHT, ErrorCode::InvalidTotalWeight);
    Ok(())
}
```

Result	FixResult
Confirmed	Fixed

3.4 Unused Order ID Parameter Creates Security Risk

Location	Severity	Category
swap_core.rs	INFO	Code Quality

Description:

The swap_core.rs module pre_check function contains an unused order_id parameter that is passed through the protocol but never utilized for validation, creating significant security and reliability concerns:

```
fn pre_check<'info>(
    // ...other parameters...
    _order_id: u128,  // Underscore prefix indicates unused parameter
) -> Result<()> {
    // No usage of order_id in function body
}

pub fn swap<'info>(
    // ...other parameters...
    order_id: u128,  // Passed but never meaningfully used
    // ...other parameters...
) -> Result<u64> {
    pre_check(admin_info, payer, sender, receiver, &swap_params, order_id)?;
    // No other usage of order_id
}
```

Recommendations:

Adding a storage mechanism for previously processed order IDs or delete the parameter.

```
#[account]
pub struct OrderRegistry {
    pub processed_orders: Vec<u128>,
    // Alternative: Add a timestamp-based expiration mechanism
}
```

Result	FixResult
Confirmed	Acknowledge

4. CONCLUSION

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