BLOCKSEC

Security Audit

Report for bgw-swap-aggregator-solana

Date: November 21, 2025 **Version:** 1.0

Contact: contact@blocksec.com

Contents

Report Manifest

Item	Description
Client	Bitget Wallet
Target	bgw-swap-aggregator-solana

Version History

Version	Date	Description
1.0	November 21, 2025	First release

Signature

Digitally signed by: BlockSec (https://blocksec.com)

SHA1 digest of public key: BAB106D3708BC6DE0963CE05E9478C24B326AE1C

Date: 2025-11-24 10:14:43+0000

About BlockSec BlockSec focuses on the security of the blockchain ecosystem and collaborates with leading DeFi projects to secure their products. BlockSec is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and successfully protected digital assets that are worth more than 14 million dollars by blocking multiple attacks. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Rust
Approach	Semi-automatic and manual verification

The target of this audit is the code repository [1] of bgw-swap-aggregator-solana of Bitget Wallet.

The BGW Swap Aggregator is a comprehensive, Solana-native DEX protocol designed for optimal crypto trading. Serving the Bitget Wallet Swap ecosystem, it achieves efficiency through intelligent liquidity aggregation and order splitting.The protocol analyzes liquidity across multiple integrated DEXs—including Raydium AMM/CLMM and Pumpswap—to calculate the best swap price and minimize slippage. By leveraging advanced routing algorithms and Solana's low fees, it provides a superior trading experience.

Note this audit only focuses on the smart contracts in the following directories/files. Code prior to and including the baseline version 0, where applicable, is outside the scope of this audit and assumes to be reliable and secure.

- programs/bgwaggregator/src/common/constants.rs
- programs/bgwaggregator/src/common/errors.rs
- programs/bgwaggregator/src/handler/handler_core.rs
- programs/bgwaggregator/src/swap/mod.rs
- programs/bgwaggregator/src/swap/swap_core.rs
- programs/bgwaggregator/src/swap/swap_sol_spl_free_v3.rs
- programs/bgwaggregator/src/swap/swap_sol_spl_v3.rs
- programs/bgwaggregator/src/swap/swap_spl_sol_free_v3.rs
- programs/bgwaggregator/src/swap/swap_spl_sol_v3.rs
- programs/bgwaggregator/src/swap/swap_spl_spl_free_v3.rs
- programs/bgwaggregator/src/swap/swap_spl_spl_v3.rs
- programs/bgwaggregator/src/swap/swap_spl_white_free_v3.rs
- programs/bgwaggregator/src/swap/swap_spl_white_v3.rs

Other files are not within the scope of the audit. Additionally, all dependencies of the smart contracts within the audit scope are considered reliable in terms of both functionality and security, and are therefore not included in the audit scope.

The auditing process is iterative. Specifically, we would audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The commit SHA values during the audit are shown in the following table. Our audit report is responsible for the code in the initial version (Version 1), as well as new code (in the following versions) to fix

[1] https://github.com/bitgetwallet/bgw-swap-aggregator-solana

issues in the audit report. Code prior to and including the baseline version (Version 0), where applicable, is outside the scope of this audit and assumes to be reliable and secure.

Project	Version	Commit Hash
bgw-swap-aggregator-solana	Version 0	d042a6340693a76a2cb4345f36b283863568b4d4
	Version 1	b4d803bfed1cb836448d68f079a1f1fc9dc941bc
	Version 2	d2d1a6d6e5399a601d8375ebc5f46f8acdc333d6

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Security Issues

* Access control
* Permission management
* Whitelist and blacklist mechanisms
* Initialization consistency

- Improper use of the proxy system
- Reentrancy
- Denial of Service (DoS)
- Untrusted external call and control flow
- Exception handling
- Data handling and flow
- Events operation
- Error-prone randomness
- Oracle security
- Business logic correctness
- Semantic and functional consistency
- Emergency mechanism
- Economic and incentive impact

1.3.2 Additional Recommendation

- Gas optimization
- Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [2] and Common Weakness Enumeration [3]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.1.

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following five categories:

- **Undetermined** No response yet.
- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Partially Fixed** The item has been confirmed and partially fixed by the client.
- **Fixed** The item has been confirmed and fixed by the client.

[2] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[3] https://cwe.mitre.org/

Table 1.1: Vulnerability Severity Classification

Impact		Likelihood: *High*	Likelihood: *Low*
	High	High	Medium
	Low	Medium	Low

Chapter 2 Findings

In total, we found **two** potential security issues. Besides, we have **two** recommendations and **four** notes.

- Low Risk: 2
- Recommendation: 2
- Note: 4

ID	Severity	Description	Category	Status
1	Low	Incorrect update in the function `handle_action_token_out()`	Security Issue	Confirmed
2	Low	Lack of handling pumpfun sell operation in `swap_v3()` function	Security Issue	Confirmed
3	-	Lack of validation between `proxy_pda_index` and `current_proxy_pda`	Recommendation	Fixed
4	-	Default security setting for signature verification	Recommendation	Confirmed
5	-	`fee_params` validation responsibility	Note	-
6	-	Potential Centralization Risks	Note	-
7	-	Potential dust funds locked in proxy pda token accounts	Note	-
8	-	The parameter `order_id` may be inaccurate	Note	-

The details are provided in the following sections.

2.1 Security Issue

2.1.1 Incorrect update in the function `handle_action_token_out()`

Severity Low

Status Confirmed

Introduced by `Version 1`

Description The function `handle_action_token_out()` updates the `total_amount` field when processing output tokens but fails to maintain the corresponding `used_percent` field, creating a critical state inconsistency in the token tracking mechanism.

Consider the following scenario:

1.Initially, a token has `total_amount` 100, `used_amount` 0, and `used_percent` 0.

2.After calling `handle_action_token_in()`, the values update to `total_amount` 100, `used_amount` 50, and `used_percent` 5,000.

3.In another hop, the function `handle_action_token_out()` modifies the parameters to `total_amount` 150, `used_amount` 50, but incorrectly retains `used_percent` 5,000.

At this stage, if a new action uses the token as input and assumes there are 100 available with `action.percent` 6,666. As a result, the `used_percent` 11666 exceeds 10,000, causing a DoS.

```rust
fn handle_action_token_out<'info>(
    action_token_states: &mut [ActionTokenState],
    output_token_mint: Pubkey,
    swapped_amount: u64,
    action_token_count: u8,
    max_action_token_count: u8,
) -> Result<u8> {
    if let Some(i) =
        find_token_state_index_by_mint(action_token_states, &output_token_mint, action_token_count)
    {
        action_token_states[i].total_amount = action_token_states[i]
            .total_amount
            .checked_add(swapped_amount)
            .ok_or(ErrorCode::CalcErrorActionTokenOut)?;
        return Ok(action_token_count);
    }

    require!(
        action_token_count < max_action_token_count,
        ErrorCode::ActionTokenCountOverflow
    );

    action_token_states[action_token_count as usize] = ActionTokenState {
        token_mint: output_token_mint,
        total_amount: swapped_amount,
        used_amount: 0,
        used_percent: 0,
    };

    Ok(action_token_count + 1)
}

fn find_token_state_index_by_mint(
    action_token_states: &[ActionTokenState],
    token_mint: &Pubkey,
    action_token_count: u8,
) -> Option<usize> {
    for i in 0..action_token_count as usize {
        if action_token_states[i].token_mint == *token_mint {
            return Some(i);
        }
    }
    None
}
```

Listing 2.1: programs/bgwaggregator/src/swap/swap_core.rs

Impact This will lead to DoS.

Suggestion Revise the logic accordingly.

Feedback from the project The project team stated that the swap path is based on a directed acyclic graph (DAG), which prevents loops, and the paths are guaranteed by the backend.

Additionally, the rule for constructing actions is that a token can only start flowing out after all of its inflows are completed.

2.1.2 Lack of handling pumpfun sell operation in `swap_v3()` function

Severity Low

Status Confirmed

Introduced by `Version 1`

Description Function `swap_v3()` processes complex multi-hop swap operations. However, it does not implement the specialized handling logic for pumpfun sell operations. In this case, the received native SOL from pumpfun sell operations cannot be properly processed.

```
141pub fn swap<'info>(
142    proxy_pda_input_token_account: &mut InterfaceAccount<'info, TokenAccount>,
143    proxy_pda_output_token_account: &mut InterfaceAccount<'info, TokenAccount>,
144    remaining_accounts: &'info [AccountInfo<'info>],
145    swap_params: SwapParams,
146    order_id: u128,
147    admin_info: &Account<'info, AdminInfo>,
148    payer: &AccountInfo<'info>,
149    sender: &AccountInfo<'info>,
150    receiver: &AccountInfo<'info>,
151) -> Result<u64> {
152    pre_check(admin_info, payer, sender, receiver, &swap_params, order_id)?;
153
154    let SwapParams {
155        proxy_pda_index,
156        amount_in,
157        min_amount_out,
158        quote_amount: _,
159        amount_in_weights,
160        amount_in_routes,
161    } = &swap_params;
162
163    // check case whether the swap is 'PumpfunSell' single hop
164    let proxy_pda = remaining_accounts.get(1).unwrap();
165    let is_single_pumpfun_sell = amount_in_routes.len() == 1 // one route
166        && amount_in_routes[0].len() == 1 // one swapStep
167        && amount_in_routes[0][0].protocols.len() == 1 // only one dex is PumpfunSell
168        && amount_in_routes[0][0].protocols[0] == ProtocolType::PumpfunSell;
169
170    let before_output_balance = if is_single_pumpfun_sell {
171        msg!("pfi_sell get bal");
172        proxy_pda.lamports()
173    } else {
174        proxy_pda_output_token_account.reload()?;
175        proxy_pda_output_token_account.amount
176    };
```

Listing 2.2: programs/bgwaggregator/src/swap/swap_core.rs

```
327pub fn swap_v3<'info>(
328    proxy_pda_input_token_account: &mut InterfaceAccount<'info, TokenAccount>,
329    proxy_pda_output_token_account: &mut InterfaceAccount<'info, TokenAccount>,
330    remaining_accounts: &'info [AccountInfo<'info>],
331    swap_params: SwapParamsV3,
332    order_id: u128,
333    admin_info: &Account<'info, AdminInfo>,
334    payer: &AccountInfo<'info>,
335    sender: &AccountInfo<'info>,
336    receiver: &AccountInfo<'info>,
337) -> Result<u64> {
338    pre_check_v3(admin_info, payer, sender, receiver, &swap_params)?;
339
340    let SwapParamsV3 {
341        proxy_pda_index,
342        amount_in,
343        min_amount_out,
344        quote_amount: _,
345        total_token_count,
346        actions,
347    } = &swap_params;
348
349    proxy_pda_output_token_account.reload()?;
350    let before_output_balance = proxy_pda_output_token_account.amount;
```

Listing 2.3: programs/bgwaggregator/src/swap/swap_core.rs

Impact This omission may result in the failure of function `swap_v3()`.

Suggestion Revise the logic accordingly.

Feedback from the project The project team stated that the pumpfun handler has been deprecated, so the `swap_v3()` function no longer needs to handle the pumpfun sell operation.

2.2 Recommendation

2.2.1 Lack of validation between `proxy_pda_index` and `current_proxy_pda`

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The function `invoke_protocol()` utilizes `proxy_pda_index` to generate signatures for token transfers through `invoke_signed()`, but does not verify that this index corresponds to the actual `current_proxy_pda` address that owns the token accounts.

```
73pub fn invoke_protocol<'a, 'info>(
74    account_infos: &[AccountInfo],
75    instruction: Instruction,
76    offset: &mut usize,
77    accounts_len: usize,
78    proxy_pda_index: u8,
79    current_proxy_pda_input_token_account: &InterfaceAccount<'info, TokenAccount>,
```

```
    current_proxy_pda_output_token_account: &mut InterfaceAccount<'info, TokenAccount>,
    current_proxy_pda: Pubkey,
    invoke_context: InvokeContext<'a>,
) -> Result<u64> {

    let before_output_balance: u64 = current_proxy_pda_output_token_account.amount;

    pre_check(
        &invoke_context,
        *offset,
        accounts_len,
        current_proxy_pda_input_token_account,
        current_proxy_pda_output_token_account,
        current_proxy_pda,
    )?;

    invoke_signed(
        &instruction,
        account_infos,
        &[&[
            PROXY_PDA_SEEDS[proxy_pda_index as usize],
            &[PROXY_PDA_BUMPS[proxy_pda_index as usize]],
        ]],
    )?;
```

Listing 2.4: programs/bgwaggregator/src/handler/invoke_core.rs

Suggestion Revise the logic accordingly.

2.2.2 Default security setting for signature verification

Status Confirmed

Introduced by `Version 1`

Description The program implements the field `is_need_check_signer` which controls whether `cosigner` validation is required for fee-exempt swap operations. However, this security-critical parameter is not initialized with a secure default value of true during contract deployment, creating a potential security gap where fee exemption could operate without proper authorization checks immediately after deployment.

```
pub fn set_is_need_check_signer(ctx: Context<SetAdminInfoByAuthority>, is_need_check_signer: bool)
    -> Result<()> {
    let account = &mut ctx.accounts.admin_info;
    require!(
        is_need_check_signer != account.is_need_check_signer,
        ErrorCode::ValueCannotBeEqual
    );
    msg!("old is_need_check_signer is {:?}", account.is_need_check_signer);
    account.is_need_check_signer = is_need_check_signer;
    msg!("new is_need_check_signer is {:?}", account.is_need_check_signer);

    Ok(())
```

```
163}
```

Listing 2.5: programs/bgwaggregator/src/manager/manager_authority.rs

```
26    #[account(
27        constraint =
28            !admin_info.is_need_check_signer ||
29            admin_info.free_cosigners_array01.iter().any(|k| *k == cosigner.key())
30            @ ErrorCode::InvalidFreeCosigner
31    )]
32    pub cosigner: Signer<'info>,
```

Listing 2.6: programs/bgwaggregator/src/swap/swap_sol_spl_free_v3.rs

Suggestion Explicitly set `is_need_check_signer` to true during initialization.

Feedback from the project The project team stated that since the `is_need_check_signer` field was added in a later upgrade, it was not set during initialization, but it would be set to true via the function `set_is_need_check_signer()`.

2.3 Note

2.3.1 `fee_params` validation responsibility

Introduced by `Version 1`

Description The function `swap_sol_spl_v3()` accepts `fee_params` as a parameter that includes financial components like `deduct_amount` and `fee_rate`. Advanced users can directly call the aggregator interface themselves and pass in smaller `fee_rate` and `deduct_amount` values.

```
81pub fn swap_sol_spl_v3<'a>(
82    ctx: Context<'_, '_, 'a, 'a, SwapSolToSplV3Accounts<'a>>,
83    swap_params: SwapParamsV3,
84    fee_params: FeeParams,
85    order_id: u128,
86) -> Result<u64> {
87    let mut amount_in_for_swap: u64 = swap_params.amount_in;
88    let min_amount_out = swap_params.min_amount_out;
89    let proxy_pda_index = swap_params.proxy_pda_index;
90
91    SwapCoreLog::swap_start(
92        order_id,
93        amount_in_for_swap,
94        swap_params.quote_amount,
95        min_amount_out,
96        fee_params.deduct_amount,
97        fee_params.fee_rate
98    );
```

Listing 2.7: programs/bgwaggregator/src/swap/swap_sol_spl_v3.rs

Feedback from the project The project team stated that the `fee_rate_min` represents their publicly expected fee. Some business scenarios use a higher fee, and the project sets a larger value when constructing transactions; if users call the aggregator directly with a lower fee, the project does not restrict it. The `deduct_amount` parameter is only used for gas sponsorship, which is not needed when users interact directly.

2.3.2 Potential Centralization Risks

Introduced by `Version 1`

Description In this project, several privileged roles (e.g., `admin_info.authority`) can conduct sensitive operations, which introduces potential centralization risks. For example, `admin_info.authority` can set the array of white lists of `fee_to`. If the private keys of the privileged accounts are lost or maliciously exploited, it could pose a significant risk to the protocol.

Feedback from the project The project team stated that they require a certain degree of control to meet the design requirements.

2.3.3 Potential dust funds locked in proxy pda token accounts

Introduced by `Version 1`

Description In this project, the function `swap_v3()` executes a series of `SwapAction` operations and tracks intermediate assets using the `action_token_states` array. All assets are held in proxy pda token accounts during the transaction execution. Since the swap process involves multiple hops with various intermediate tokens, small residual amounts (dust) may remain in the proxy pda token accounts after transaction completion.

Feedback from the project The project team stated that this does exist, but it does not affect business operations.

2.3.4 The parameter `order_id` may be inaccurate

Introduced by `Version 1`

Description The code in the project neither validates whether `order_id` is duplicated nor checks its validity. If off-chain logic relies on on-chain logs to track orders, the results may be inaccurate.

Feedback from the project The project team stated that the `order_id` is used in other handlers, which are outside the scope of this audit.


BOOST WEB3 THROUGH NEXT-GENERATION SECURITY & USABILITY INNOVATIONS