SECURITY AUDIT REPORT

for

WarpGate DEX

Prepared By: Xiaomi Huang

PeckShield

February 7, 2025

Document Properties

Client	WarpGate
Title	Security Audit Report
Target	WarpGate DEX
Version	1.0
Author	Daisy Cao
Auditors	Daisy Cao, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	February 7, 2025	Daisy Cao	Final Release
1.0-rc	Decemeber 19, 2024	Daisy Cao	Release Candidate #1

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Xiaomi Huang
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1 Introduction

Given the opportunity to review the design document and related smart contract source code of the WarpGate DEX protocol, we outline in the report our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About WarpGate DEX

WarpGate DEX is a decentralized exchange built on Aptos. It allows users to trade and swap Aptos tokens. In the meantime, it also allows liquidity providers to create trading pairs and add liquidity in a trustless manner. The basic information of audited contracts is as follows:

Table 1.1: Basic Information of WarpGate DEX

Item	Description
Name	WarpGate DEX
Type	Aptos
Language	Move
Audit Method	Whitebox
Latest Audit Report	February 7, 2025

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit.

- https://github.com/hatchy-fun/warpgate-swap (d875480)

And here is the commit ID after all fixes for the issues found in the audit have been checked in:

- https://github.com/warpgate-pro/warpgate-dex (b4b5630)

1.2 About PeckShield

PeckShield Inc. [7] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of current blockchain ecosystems by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/peckshield), Twitter (http://twitter.com/peckshield), or Email (contact@peckshield.com).

Table 1.2: Vulnerability Severity Classification

Impact \ Likelihood	*High*	*Medium*	*Low*
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [6]:

- 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 demonstrates the overall criticality of the risk.

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact, and can be accordingly classified into four categories, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

To evaluate the risk, we go through a list of check items and each would be labeled with a severity category. For one check item, if our tool or analysis does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further deploy contracts on our private testnet and run tests to confirm the findings. If necessary, we would

Table 1.3: The Full List of Check Items

Category	Check Item
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use Of Untrusted Libraries
	(Unsafe) Use Of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

In particular, we perform the audit according to the following procedure:

- 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.

- Semantic Consistency Checks: We then manually check the logic of implemented smart contracts and compare with the description in the white paper.

- Advanced DeFi Scrutiny: 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.

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [5], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development. Though some categories used in CWE-699 may not be relevant in smart contracts, we use the CWE categories in Table 1.4 to classify our findings. Moreover, in case there is an issue that may affect an active protocol that has been deployed, the public version of this report may omit such issue, but will be amended with full details right after the affected protocol is upgraded with respective fixes.

1.4 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

Table 1.4: Common Weakness Enumeration (CWE) Classifications Used in This Audit

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logics	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

2 | Findings

2.1 Summary

Here is a summary of our findings after analyzing the `WarpGate DEX` implementations. During the first phase of our audit, we study the smart contract source code and run our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tool. We further manually review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	0	
High	0	
Medium	1	■
Low	3	■ ■ ■
Informational	0	
Total	4	

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple contracts. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities need to be brought up and paid more attention to, which are categorized in the above table. More information can be found in the next subsection, and the detailed discussions of each of them are in Section 3.

2.2 Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.1), including 1 medium-severity vulnerability and 3 low-severity vulnerabilities.

Table 2.1: Key Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Low	Suggested Use of Immutable References	Business Logic	Resolved
PVE-002	Low	Revisited LP Token Symbol Name in create_pair()	Business Logic	Resolved
PVE-003	Low	Fee Mismatch Between Pair Creation and Swap Calculation	Business Logic	Resolved
PVE-004	Medium	Trust Issue of Admin Keys	Security Features	Confirmed

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to Section 3 for details.

3 | Detailed Results

3.1 Suggested Use of Immutable References

- ID: PVE-001
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: swap.move
- Category: Business Logic [4]
- CWE subcategory: CWE-837 [2]

Description

In Aptos, borrow_global_mut is used to borrow a mutable reference for modifying the global resource. While examining the WarpGate DEX protocol, we noticed that some functions use borrow_global_mut for read-only purposes.

In the following, we show the code snippet of the related fee_to() function. This function only needs to read the value of swap_info.fee_to and does not modify the SwapInfo resource (line 11). Therefore, borrowing an immutable reference to the resource would be more appropriate.

```
10     public fun fee_to(): address acquires SwapInfo {
11         let swap_info = borrow_global_mut<SwapInfo>(RESOURCE_ACCOUNT);
12         swap_info.fee_to
13     }
```

Listing 3.1: fee_to()

Note the same issue is also applicable to the admin() routine.

Recommendation Replace borrow_global_mut with borrow_global in above-mentioned functions.

Status This issue has been resolved in the following commit: b4b5630.

3.2 Revisited LP Token Symbol Name in create_pair()

- ID: PVE-002
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: swap.move
- Category: Business Logic [4]
- CWE subcategory: CWE-837 [2]

Description

In WarpGate DEX, the create_pair() function is designed to initialize the LP token by calling coin:: initialize with the generated name and other parameters, such as the symbol and decimal. While examining the related initialization logic, we notice current implementation can be improved.

In the following, we show the code snippet of the related create_pair() function. Currently, the LP symbol name is hardcoded as Cake-LP (line 54) during the initialization. However, this name is not aligned with the protocol's branding. Therefore, it would be more appropriate to update the name to Warpgate-LP.

```
30      public(friend) fun create_pair<X, Y>(
31          sender: &signer,
32      ) acquires SwapInfo {
33          assert!(!is_pair_created<X, Y>(), ERROR_ALREADY_INITIALIZED);
34
35          let sender_addr = signer::address_of(sender);
36          let swap_info = borrow_global_mut<SwapInfo>(RESOURCE_ACCOUNT);
37          let resource_signer = account::create_signer_with_capability(&swap_info.
                signer_cap);
38
39          let lp_name: string::String = string::utf8(b"Warpgate-");
40          let name_x = coin::symbol<X>();
41          let name_y = coin::symbol<Y>();
42          string::append(&mut lp_name, name_x);
43          string::append_utf8(&mut lp_name, b"-");
44          string::append(&mut lp_name, name_y);
45          string::append_utf8(&mut lp_name, b"-LP");
46          if (string::length(&lp_name) > MAX_COIN_NAME_LENGTH) {
47              lp_name = string::utf8(b"Warpgate LPs");
48          };
49
50          // now we init the LP token
51          let (burn_cap, freeze_cap, mint_cap) = coin::initialize<LPToken<X, Y>>(
52              &resource_signer,
53              lp_name,
54              string::utf8(b"Cake-LP"),
55              8,
56              true
57          );
```

```
        move_to<TokenPairReserve<X, Y>>(
            &resource_signer,
            TokenPairReserve {
                reserve_x: 0,
                reserve_y: 0,
                block_timestamp_last: 0
            }
        );

        move_to<TokenPairMetadata<X, Y>>(
            &resource_signer,
            TokenPairMetadata {
                creator: sender_addr,
                fee_amount: coin::zero<LPToken<X, Y>>(),
                k_last: 0,
                balance_x: coin::zero<X>(),
                balance_y: coin::zero<Y>(),
                mint_cap,
                burn_cap,
                freeze_cap,
            }
        );

        move_to<PairEventHolder<X, Y>>(
            &resource_signer,
            PairEventHolder {
                add_liquidity: account::new_event_handle<AddLiquidityEvent<X, Y>>(&resource_signer),
                remove_liquidity: account::new_event_handle<RemoveLiquidityEvent<X, Y>>(&resource_signer),
                swap: account::new_event_handle<SwapEvent<X, Y>>(&resource_signer)
            }
        );

        // pair created event
        let token_x = type_info::type_name<X>();
        let token_y = type_info::type_name<Y>();

        event::emit_event<PairCreatedEvent>(
            &mut swap_info.pair_created,
            PairCreatedEvent {
                user: sender_addr,
                token_x,
                token_y
            }
        );
```

Listing 3.2: `create_pair()`

Recommendation Replace `Cake-LP` with `Warpgate-LP` in the above-mentioned function.

Status This issue has been resolved in the following commit: b4b5630.

3.3 Fee Mismatch Between Pair Creation and Swap Calculation

- ID: PVE-003
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `swap.move`
- Category: Business Logic [4]
- CWE subcategory: CWE-837 [2]

Description

In `WarpGate DEX`, each token pair in the project can define its own swap fee. The `swap` function calculates the adjusted `balances balance_x_adjusted` and `balance_y_adjusted` to check whether the pool invariant is maintained after the swap. While examining the swap related logic, we notice the associated implementation can be improved.

In the following, we show the code snippet of the related `swap()` function. The swap fee should be dynamic and depend on the specific token pair metadata. However, the current implementation applies a fixed swap fee during the calculation of `balances balance_x_adjusted` and `balances balance_y_adjusted` (line 225 and line 226).

```
200     fun swap<X, Y>(
201         amount_x_out: u64,
202         amount_y_out: u64
203     ): (coin::Coin<X>, coin::Coin<Y>) acquires TokenPairReserve, TokenPairMetadata {
204         assert!(amount_x_out > 0 || amount_y_out > 0, ERROR_INSUFFICIENT_OUTPUT_AMOUNT);
205         let reserves = borrow_global_mut<TokenPairReserve<X, Y>>(RESOURCE_ACCOUNT);
206         assert!(amount_x_out < reserves.reserve_x && amount_y_out < reserves.reserve_y,
                ERROR_INSUFFICIENT_LIQUIDITY);
207         let metadata = borrow_global_mut<TokenPairMetadata<X, Y>>(RESOURCE_ACCOUNT);
208         let fee = metadata.swap_fee;
209         let coins_x_out = coin::zero<X>();
210         let coins_y_out = coin::zero<Y>();
211         if (amount_x_out > 0) coin::merge(&mut coins_x_out, extract_x(amount_x_out,
                metadata));
212         if (amount_y_out > 0) coin::merge(&mut coins_y_out, extract_y(amount_y_out,
                metadata));
213         let (balance_x, balance_y) = token_balances<X, Y>();
214 
215         let amount_x_in = if (balance_x > reserves.reserve_x - amount_x_out) {
216             balance_x - (reserves.reserve_x - amount_x_out)
217         } else { 0 };
218         let amount_y_in = if (balance_y > reserves.reserve_y - amount_y_out) {
219             balance_y - (reserves.reserve_y - amount_y_out)
220         } else { 0 };
221         assert!(amount_x_in > 0 || amount_y_in > 0, ERROR_INSUFFICIENT_INPUT_AMOUNT);
```

```

        let prec = (PRECISION as u128);

        let balance_x_adjusted = (balance_x as u128) * prec - (amount_x_in as u128) * 25
            u128;
        let balance_y_adjusted = (balance_y as u128) * prec - (amount_y_in as u128) * 25
            u128;

        let reserve_x_adjusted = (reserves.reserve_x as u128) * prec;
        let reserve_y_adjusted = (reserves.reserve_y as u128) * prec;
        // No need to use u256 when balance_x_adjusted * balance_y_adjusted and
            reserve_x_adjusted * reserve_y_adjusted are less than MAX_U128.
        let compare_result = if(balance_x_adjusted > 0 && reserve_x_adjusted > 0 &&
            MAX_U128 / balance_x_adjusted > balance_y_adjusted && MAX_U128 /
            reserve_x_adjusted > reserve_y_adjusted){
            balance_x_adjusted * balance_y_adjusted >= reserve_x_adjusted *
                reserve_y_adjusted
        }else{
            let p: u256 = (balance_x_adjusted as u256) * (balance_y_adjusted as u256);
            let k: u256 = (reserve_x_adjusted as u256) * (reserve_y_adjusted as u256);
            p >= k
        };....}
```

Listing 3.3: `swap()`

Recommendation Ensures that the correct fees are applied based on the specific configuration of each token pair.

Status This issue has been resolved in the following commit: b4b5630.

3.4 Trust Issue of Admin Keys

- ID: PVE-004
- Severity: Medium
- Likelihood: Low
- Impact: High
- Target: `Multiple contracts`
- Category: Security Features [3]
- CWE subcategory: CWE-287 [1]

Description

In `WarpGate DEX`, there is a privileged account, i.e., `admin`. This account plays a critical role in governing and regulating the system-wide operations (e.g., set `fee_to`, update contract etc.). Our analysis shows that this privileged account needs to be scrutinized. In the following, we use the `upgrade_swap()` function as an example and show the representative functions potentially affected by the privileges of the `admin` account.

```
    public entry fun upgrade_swap(sender: &signer, metadata_serialized: vector<u8>, code: vector<vector<u8>>) acquires SwapInfo {
        let sender_addr = signer::address_of(sender);
        let swap_info = borrow_global<SwapInfo>(RESOURCE_ACCOUNT);
        assert!(sender_addr == swap_info.admin, ERROR_NOT_ADMIN);
        let resource_signer = account::create_signer_with_capability(&swap_info.signer_cap);
        code::publish_package_txn(&resource_signer, metadata_serialized, code);
    }
```

Listing 3.4: `upgrade_swap()`

We understand the need of the privileged functions for proper `WarpGate DEX` operations, but at the same time the extra power to the `admin` may also be a counter-party risk to the `WarpGate DEX` contract users. Therefore, we list this concern as an issue here from the audit perspective and highly recommend making these privileges explicit or raising necessary awareness among protocol users.

Recommendation Make the list of extra privileges granted to `WarpGate DEX` explicit to `WarpGate DEX` contract users.

Status This issue has been confirmed.

4 Conclusion

In this audit, we have analyzed the design and implementation of the `WarpGate DEX` protocol, is a decentralized exchange built on `Aptos`. It allows users to trade and swap `Aptos` tokens. In the meantime, it also allows liquidity providers to create trading pairs and add liquidity in a trustless manner. The current code base is well structured and neatly organized. Those identified issues are promptly confirmed and addressed.

Meanwhile, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

References

[1] MITRE. CWE-287: Improper Authentication. https://cwe.mitre.org/data/definitions/287.html.

[2] MITRE. CWE-837: Improper Enforcement of a Single, Unique Action. https://cwe.mitre.org/data/definitions/837.html.

[3] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

[4] MITRE. CWE CATEGORY: Business Logic Errors. https://cwe.mitre.org/data/definitions/840.html.

[5] MITRE. CWE VIEW: Development Concepts. https://cwe.mitre.org/data/definitions/699.html.

[6] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[7] PeckShield. PeckShield Inc. https://www.peckshield.com.