Security Assessment Report

The Scrypto ecosystem of Caviar

Order Book, Order Book Factory and Fee Controller

October 25, 2023

Summary

The sec3 team (formerly Soteria) was engaged to do a security review of the order_book, the order_book_factory, and the fee_controller modules in the Scrypto ecosystem of Caviar.

The artifact was the source code (excluding tests) in the following folders in a private repository (commit 783977a8605c9e73ba7a6d12d5d2057cbf6d7971)

- scrypto-caviar/order_book
- scrypto-caviar/order_book_factory
- scrypto-caviar/fee_controller

The initial review identified 3 issues. The team responded with a second version for the post-audit review to see if the reported issues were resolved. The audit was concluded on commit 9b60b600a7b916b4b8ccc578b063038bf71b6333.

This report provides a comprehensive analysis of the findings and the resolutions.

Table of Contents

Result Overview

Issue	Impact	Status
[I-1] The stop_price may be a surprise for some users	Informational	Resolved
[I-2] Token names in the default OrderReceipt description are missing	Informational	Resolved
[I-3] Possible evasion of protocol fees	Informational	Resolved

Findings in Detail

[I-1] The stop_price may be a surprise for some users

```
/* order_book/src/order_book.rs */
747 | pub fn market_order(&mut self, tokens: Bucket, stop_price: Option<Decimal>)->(Bucket, Bucket) {
748 |     // Check parameters
749 |     assert!(tokens.amount() > Decimal::zero(), "Order size must be greater than zero.");
751 |     // Check if tokens_x or tokens_y
752 |     if tokens.resource_address() == self.tokens_x.resource_address() {
753 |         let stop_price: Price = match stop_price {
754 |             Some(price_dec) => price_dec.round_to_price_range().into(),
755 |             None => Price::MIN,
756 |         };
758 |         self.market_order_x_to_y(tokens, stop_price)
759 |     } else if tokens.resource_address() == self.tokens_y.resource_address() {
760 |         let stop_price: Price = match stop_price {
761 |             Some(price_dec) => price_dec.round_to_price_range().into(),
762 |             None => Price::MAX,
763 |         };
765 |         self.market_order_y_to_x(tokens, stop_price)
766 |     } else {
767 |         panic!("Invalid token address.");
768 |     }
769 | }
```

Users can provide an optional `stop_price` as a threshold to stop the trade.

In the current implementation, when the `stop_price` provided by the user is invalid (i.e., outside the predefined range), it is replaced with the nearest valid value, which may not be the expected behavior for the user.

Does it make sense to stop, say triggering a panic, when an invalid `stop_price` is entered by the user instead?

Resolution

The team believes it's not preferable to introduce a panic here, as it's better that the user does not have to understand the valid price range when doing a market order. The team also

thinks it would not be a surprise because it will never trigger when the users inputted stop price logically would not have and will trigger when it logically would have. We agree. This issue has been resolved.

[I-2] Token names in the default OrderReceipt description are missing

```
/* order_book/src/order_book.rs */
171 | // Create order receipt resource
172 | let order_receipt_manager = ResourceBuilder::new_integer_non_fungible::<OrderReceipt>(Owner-
Role::Updatable(owner_rule.clone()))
173 |     .metadata(metadata!(
174 |         init {
179 |             "name" => format!("Order Receipt {}/{}", symbol_x, symbol_y), updatable;
180 |             "description" => format!("Used to claim tokens from a limit order for pair {}/{}.",
                                        "", ""), updatable;
182 |         }
183 |     ))
184 |     .mint_roles(mint_roles!{
185 |             minter => rule!(require(global_caller(component_address)));
186 |             minter_updater => rule!(deny_all);
187 |         }
188 |     )
189 |     .burn_roles(burn_roles!{
190 |             burner => rule!(require(global_caller(component_address)));
191 |             burner_updater => rule!(deny_all);
192 |         }
193 |     )
194 |     .non_fungible_data_update_roles(non_fungible_data_update_roles!{
195 |             non_fungible_data_updater => rule!(require(global_caller(component_address)));
196 |             non_fungible_data_updater_updater => rule!(deny_all);
197 |         }
198 |     )
199 |     .create_with_no_initial_supply();
```

Users submitting limit orders who do not immediately have their trades fulfilled through market orders receive an `OrderReceipt` NFT. This NFT is for subsequent actions, such as retrieving tokens upon successful execution or canceling the transaction.

However, during the initialization, in the `description` field of the `OrderReceipt` NFT (line 180), empty strings were provided for the two token names.

Resolution

This issue has been fixed by the commit `1052530`.

[I-3] Possible evasion of protocol fees

```
/* order_book/src/order_book.rs */
1126 | fn market_order_x_to_y(&mut self, mut tokens_x: Bucket, stop_price: Price)->(Bucket, Bucket) {
1127 |     // Take fee
1128 |     let protocol_fee: Decimal =  FEE_CONTROLLER.get_protocol_fee(Runtime::package_address());
1129 |     let mut tokens_fee: Bucket = tokens_x.take(tokens_x.amount() * protocol_fee);
1130 |     let amount_x_input: Decimal = tokens_x.amount();
1131 |
1132 |     // Initialize amounts
1133 |     let mut amount_y_bought: Decimal = Decimal::zero();
1134 |     let mut amount_x_order: Decimal = tokens_x.amount();
1135 |     let mut fills: Vec<(Decimal, Decimal)> = Vec::new();

/* radixdlt-scrypto-1833d92590086ce0/302e040/radix-engine-common/src/math/decimal.rs */
408 | impl Mul<Decimal> for Decimal {
409 |     type Output = Self;
410 |
411 |     #[inline]
412 |     fn mul(self, other: Self) -> Self::Output {
413 |         self.checked_mul(other).expect("Overflow")
414 |     }
415 | }

/* radixdlt-scrypto-1833d92590086ce0/302e040/radix-engine-common/src/math/decimal.rs */
349 | impl CheckedMul<Decimal> for Decimal {
350 |     type Output = Self;
351 |
352 |     #[inline]
353 |     fn checked_mul(self, other: Self) -> Option<Self> {
354 |         // Use I256 (BInt<4>) to not overflow.
355 |         let a = I256::from(self.0);
356 |         let b = I256::from(other.0);
357 |         let mut c = a.checked_mul(b)?;
358 |         c = c.checked_div(I256::from(Self::ONE.0))?;
359 |
360 |         let c_192 = I192::try_from(c).ok();
361 |         c_192.map(Self)
362 |     }
363 | }
```

The protocol fees for market orders are deducted directly from the quantity of `token_x` used for the exchange. Due to the scale of `tokens_x` being 1E18, when the quantity of tokens used for the transaction is extremely small, for instance, less than 1E-17, it may result in a

protocol fee of 0. Using the `limit_order_batch` for batch trading may further relax the restrictions on token quantities.

However, due to the already high precision, fee loss is virtually negligible. Nevertheless, it could still be considered to "round it up", say by adding 1E-18 to the protocol fee.

Resolution

The team acknowledged the finding and believes it is not a concern.

Appendix: Methodology and Scope of Work

The sec3 (formerly Soteria) audit team, which consists of Computer Science professors, industrial researchers and CTF players with extensive experience in web3 security, program analysis, testing and formal verification, performed a comprehensive manual code review, software static analysis, and penetration testing.

Assisted by the sec3 Scanner developed in-house, the audit team particularly focused on the following work items:

- Check common security issues.
- Check program logic implementation against available design specifications.
- Check poor coding practices and unsafe behavior.

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a sec3 (the "Company") and CAVIAR LABS PTE. LTD. (the "Client"). This Report solely includes the results of a technical assessment of a specific build and/or version of the Client's code specified in the Report ("Assessed Code") by the Company. The sole purpose of the Report is to provide the Client with the results of the technical assessment of the Assessed Code. The Report does not apply to any other version and/or build of the Assessed Code. Regardless of the contents of the Report, the Report does not (and should not be interpreted to) provide any warranty, representation or covenant that the Assessed Code: (i) is error and/or bug free, (ii) has no security vulnerabilities, and/or (iii) does not infringe any third-party rights. Moreover, the Report is not, and should not be considered, an endorsement by the Company of the Assessed Code and/or of the Client. Finally, the Report should not be considered investment advice or a recommendation to invest in the Assessed Code and/or the Client.

This Report is considered null and void if the Report (or any portion thereof) is altered in any manner.

ABOUT

Founded by leading academics in the field of software security and senior industrial veterans, sec3 (formerly Soteria) is a leading blockchain security company. We are also building sophisticated security tools that incorporate static analysis, penetration testing, and formal verification.

At sec3, we identify and eliminate security vulnerabilities through the most rigorous process and aided by the most advanced analysis tools.

For more information, check out our website and follow us on twitter.