Security Assessment Report

CMTAT Coupon Bond

September 12th, 2023

Summary

The sec3 team (formerly Soteria) was engaged to do a thorough security analysis of the CMTAT Coupon Bond Contracts. The artifact of the audit was the source code of the smart contracts (excluding tests) in a private repository.

The initial audit was done on commit 6d9d4c42c0e9b34dd09490bbe8e084a36a028245. The audit revealed 8 issues or questions. The team responded with a second version for the post-audit review to confirm if the reported issues have been resolved. The audit concludes on commit 3b9118d1b231de5db7cf157743c2dc652cbb9ef3.

This report describes the findings and resolutions in detail.

Table of Contents

Result Overview

Issue	Impact	Status
[L-1] fundedUnits is not updated appropriately	Low	Resolved
[L-2] issuedUnits is not updated appropriately	Low	Resolved
[L-3] allocatedUnits cannot be updated to a smaller value	Low	Resolved
[I-1] DEBT_ROLE is too powerful	Informational	Resolved
[I-2] Missing zero address check	Informational	Resolved
[I-3] Missing zero address check	Informational	Resolved
[I-4] Missing zero address check	Informational	Resolved
[I-5] Unnecessary type conversion	Informational	Resolved

Findings in Detail

[L-1] fundedUnits is not updated appropriately

When calculating the remaining _allocatedUnits, instead of setting it to _allocatedUnits, the fundedUnits is updated to the difference between the current _allocatedUnits and the fundOrder. As a result, the access control at line 271 is always true and becomes unfunctional.

```
/* contracts/modules/ISSUANCE_PROGRAM_BASE.sol */
269 | uint256 amount = _allocatedUnits -
270 |     fundedUnits[_issuanceToken][_msgSender()];
271 | require(amount > 0, "Already funded");
272 |
273 | unchecked {
274 |     fundedUnits[_issuanceToken][_msgSender()] = amount;
275 | }
```

Possible repairs

Consider assigning _allocatedUnits to fundedUnits[_issuanceToken][_msgSender()] at line 274.

Resolution

This issue has been fixed by commit b966f1d.

[L-2] issuedUnits is not updated appropriately

Similar to L-1, at line 329, `issuedUnits` is set to the difference between `_fundedUnits` and `issuedUnits`. As a result, the access control at line 327 becomes unfunctional since the difference between `_fundedUnits` and `issuedUnits` is always not 0.

```
/* contracts/modules/ISSUANCE_PROGRAM_BASE.sol */
326 | uint256 amount = _fundedUnits - issuedUnits[_issuanceToken][_account];
327 | require(amount > 0, "No redeemable units");
328 | unchecked {
329 |     issuedUnits[_issuanceToken][_account] = amount;
330 | }
```

Possible repairs

Consider assigning `_fundedUnits` to `issuedUnits[_issuanceToken][_account]` at line 329.

Resolution

This issue has been fixed by commit `b966f1d`.

[L-3] allocatedUnits cannot be updated to a smaller value

At line 197, if the new amount is less than the old allocatedUnits, `_amount - allocatedUnits[_issuanceToken][_account]` will underflow and cause the update to fail.

```
/* contracts/modules/ISSUANCE_PROGRAM_BASE.sol */
196 | issuance.totalAllocatedUnits +=
197 |     _amount -
198 |     allocatedUnits[_issuanceToken][_account];
```

Possible repairs

Rewrite the assignment and avoid the underflow.

```
issuance.totalAllocatedUnits = issuance.totalAllocatedUnits + _amount -
                               allocatedUnits[_issuanceToken][_account];
```

Resolution

This issue has been fixed by commit `b966f1d`.

[I-1] DEBT_ROLE is too powerful

```
/* contracts/modules/wrapper/optional/DebtModule/DebtBaseModule.sol */
151 | function setDebtAdditionalInfo(
152 |     string memory issuerName_,
153 |     string memory issuerInfo_,
154 |     IERC20 currency_,
155 |     uint8[] memory labels_
156 | ) public onlyRole(DEBT_ROLE) {
157 |     _checkLabels(labels_);
158 |     debtAdditionalInfo = (
159 |         DebtAdditionalInfo(
160 |             issuerName_,
161 |             issuerInfo_,
162 |             currency_,
163 |             labels_
164 |         )
165 |     );
170 | }

/* contracts/modules/wrapper/optional/DebtModule/DistributionModule.sol */
216 | function repay(uint256 paymentIndex) public {
230 |     debtAdditionalInfo.currency.transferFrom(
231 |         _msgSender(),
232 |         address(this),
233 |         payments[paymentIndex].amount * totalSupply()
234 |     );
245 | }
246 |
247 | function revertRepayment(
248 |     uint256 paymentIndex
249 | ) public onlyRole(ISSUER_ROLE) {
271 |     debtAdditionalInfo.currency.transfer(
272 |         _msgSender(),
273 |         payments[paymentIndex].amount * totalSupply()
274 |     );
283 | }
284 |
285 | function claimPayment(uint256 index) public {
310 |     if (!debtAdditionalInfo.currency.transfer(_msgSender(), paymentAmount)) {
311 |         revert Errors.TransferFailed();
312 |     }
314 | }
```

`currency` is fully controlled by the `DEBT_ROLE` role. If `DEBT_ROLE` does something evil or the private key is stolen, users who call the `repay`, `revertRepayment`, `claimPayment` functions may suffer losses.

Resolution

The team clarified that the `DEBT_ROLE` will not be held by anyone. The team will set the `debtinfo` using the `DEFAULT_ADMIN` role. Once completed, the team will renounce that role so there's no one with `DEBT_ROLE` once the token is issued. This issue has been resolved.

[I-2] Missing zero address check

```
/* contracts/modules/wrapper/optional/DebtModule/DistributionModule.sol */
107 | function __DistributionModule_init_unchained(address paymentRedemptionTokenFactory)
              public onlyInitializing {
108 |     paymentRedemptionTokenFactory =
          PAYMENT_REDEMPTION_TOKEN_FACTORY_BASE(paymentRedemptionTokenFactory_);
109 | }

/* contracts/modules/PAYMENT_REDEMPTION_TOKEN_FACTORY_BASE.sol */
020 | contract PAYMENT_REDEMPTION_TOKEN_FACTORY_BASE is Initializable, ContextUpgradeable {
021 |
022 |   address public paymentRedemptionTokenBeacon;
023 |
024 |   function initialize(address paymentRedemptionTokenBeacon_) public{
025 |     __PAYMENT_REDEMPTION_TOKEN_FACTORY_init(paymentRedemptionTokenBeacon_);
026 |   }
027 |
028 |   function __PAYMENT_REDEMPTION_TOKEN_FACTORY_init(
                        address paymentRedemptionTokenBeacon_) internal initializer {
029 |     __Context_init_unchained();
030 |     __PAYMENT_REDEMPTION_TOKEN_FACTORY_init_unchained(paymentRedemptionTokenBeacon_);
031 |   }
032 |
033 |   function __PAYMENT_REDEMPTION_TOKEN_FACTORY_init_unchained(
                        address paymentRedemptionTokenBeacon_) internal initializer {
034 |         paymentRedemptionTokenBeacon = paymentRedemptionTokenBeacon_;
035 |     }
072 | }
```

The zero address check is missing for `paymentTokenFactory_`
in `PAYMENT_TOKEN_FACTORY_BASE`

Resolution

This issue has been fixed by commit `b966f1d`.

[I-3] Missing zero address check

```
/* contracts/modules/wrapper/optional/DebtModule/DistributionModule.sol */
216 | function repay(uint256 paymentIndex) public {
230 |     debtAdditionalInfo.currency.transferFrom(
231 |         _msgSender(),
232 |         address(this),
233 |         payments[paymentIndex].amount * totalSupply()
234 |     );
245 | }
246 |
247 | function revertRepayment(
271 |     debtAdditionalInfo.currency.transfer(
272 |         _msgSender(),
273 |         payments[paymentIndex].amount * totalSupply()
274 |     );
283 | }

/* contracts/modules/wrapper/optional/DebtModule/DebtBaseModule.sol */
151 | function setDebtAdditionalInfo(
152 |     string memory issuerName_,
153 |     string memory issuerInfo_,
154 |     IERC20 currency_,
155 |     uint8[] memory labels_
156 | ) public onlyRole(DEBT_ROLE) {
157 |     _checkLabels(labels_);
158 |     debtAdditionalInfo = (
165 |     );
170 | }
```

The zero address check is done for `debtAdditionalInfo.currency` in `pushPayment`. However, it's missing in `repay` and `revertRepayment`. Consider adding the zero address check in the function `setDebtAdditionalInfo`.

Resolution

This issue has been fixed by commit `b966f1d`.

[I-4] Missing zero address check

```
/* contracts/modules/ISSUANCE_PROGRAM_BASE.sol */
134 | function createIssuance(
135 |     address _issuerSigningAddress,
136 |     address _issuerPaymentAddress,
137 |     uint _issuanceDate,
138 |     uint _issuancePricePerUnit,
139 |     IERC20 _currency,
140 |     uint256 invoiceAmount,
141 |     address invoiceRecipient,
142 |     MintModule _issuanceToken
143 | ) external onlyRole(DEFAULT_ADMIN_ROLE) whenNotPaused  {
168 |
169 |     if(invoiceAmount > 0)  {
170 |         issuance.invoice = Invoice(invoiceAmount, invoiceRecipient, false);
171 |     }
179 | }
180 |
181 | function setInvoice(
182 |     address _issuanceToken,
183 |     uint256 _amount,
184 |     address _recipient
185 | ) external onlyRole(DEFAULT_ADMIN_ROLE) whenNotPaused {
188 |     issuance.invoice = Invoice(_amount, _recipient, false);
189 | }
```

At line 170 and line 188, the zero address checks for `invoiceRecipient` and `_recipient` are missing, which may lead to invalid invoices.

Resolution

This issue has been fixed by commit `3b9118d`.

[I-5] Unnecessary type conversion

```
/* contracts/modules/wrapper/optional/DebtModule/DebtBaseModule.sol */
139 | function _checkLabels(uint8[] memory labels_) internal pure {
140 |     if(labels_.length > 8) revert Errors.OutOfBounds(labels_.length);
141 |     for (uint256 i = 0; i < labels_.length; i++) {
142 |         if(uint8(labels_[i]) > 7) revert Errors.OutOfBounds(uint256(labels_[i]));
149 | }
```

labels_ is already a uint8 array. There is no need to convert the elements to uint8 again.

Resolution

This issue has been fixed by commit b966f1d.

Appendix: Methodology and Scope of Work

The sec3 (formerly Soteria) audit team, which consists of Computer Science professors and industrial researchers with extensive experience in smart contract 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.
- The soundness of the economics design and algorithm is out of scope of this work

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a sec3 (the "Company") and FQX AG (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.