



# VRust

## **Security Assessment**

O2Lab VRust Team

10/29/2022 21:52:23

## Contents

|  |           |
|--|-----------|
| <b>Summary</b>                           | <b>4</b>  |
| <b>Overview</b>                          | <b>5</b>  |
| Project Summary . . . . .                | 5         |
| Audit Summary . . . . .                  | 5         |
| Vulnerability Summary . . . . .          | 5         |
| <b>Findings</b>                          | <b>6</b>  |
| <b>Finding Statistic</b>                 | <b>7</b>  |
| <b>Issue: 0: IntegerFlow</b>             | <b>8</b>  |
| <b>Issue: 1: IntegerFlow</b>             | <b>10</b> |
| <b>Issue: 2: IntegerFlow</b>             | <b>12</b> |
| <b>Issue: 3: IntegerFlow</b>             | <b>14</b> |
| <b>Issue: 4: IntegerFlow</b>             | <b>16</b> |
| <b>Issue: 5: MissingKeyCheck</b>         | <b>18</b> |
| <b>Issue: 6: MissingKeyCheck</b>         | <b>20</b> |
| <b>Issue: 7: CrossProgramInvocation</b>  | <b>22</b> |
| <b>Issue: 8: CrossProgramInvocation</b>  | <b>27</b> |
| <b>Issue: 9: CrossProgramInvocation</b>  | <b>32</b> |
| <b>Issue: 10: CrossProgramInvocation</b> | <b>35</b> |
| <b>Issue: 11: CrossProgramInvocation</b> | <b>41</b> |
| <b>Appendix</b>                          | <b>47</b> |
| Finding Categories . . . . .             | 47        |
| Gas Optimization . . . . .               | 47        |
| Mathematical Operations . . . . .        | 47        |
| Logical Issue . . . . .                  | 47        |
| Language Specific . . . . .              | 47        |

|                                       |           |
|---------------------------------------|-----------|
| Coding Style . . . . .                | 47        |
| Checksum Calculation Method . . . . . | 47        |
| <b>Disclaimer</b>                     | <b>49</b> |

## Summary

This report has been prepared for O2Lab VRust Team to discover issues and vulnerabilities in the source code of the O2Lab VRust Team project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques. The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

## Overview

### Project Summary

|                 |   |
|-----------------|---|
| Project Name    | O2Lab VRust Team  |
| Platform        | Ethereum  |
| Language        | Solana  |
| Crate           | sol_payment_processor   |
| GitHub Location | <a href="https://github.com/parasol-aser/vrust">https://github.com/parasol-aser/vrust</a> |
| sha256          | Unknown   |

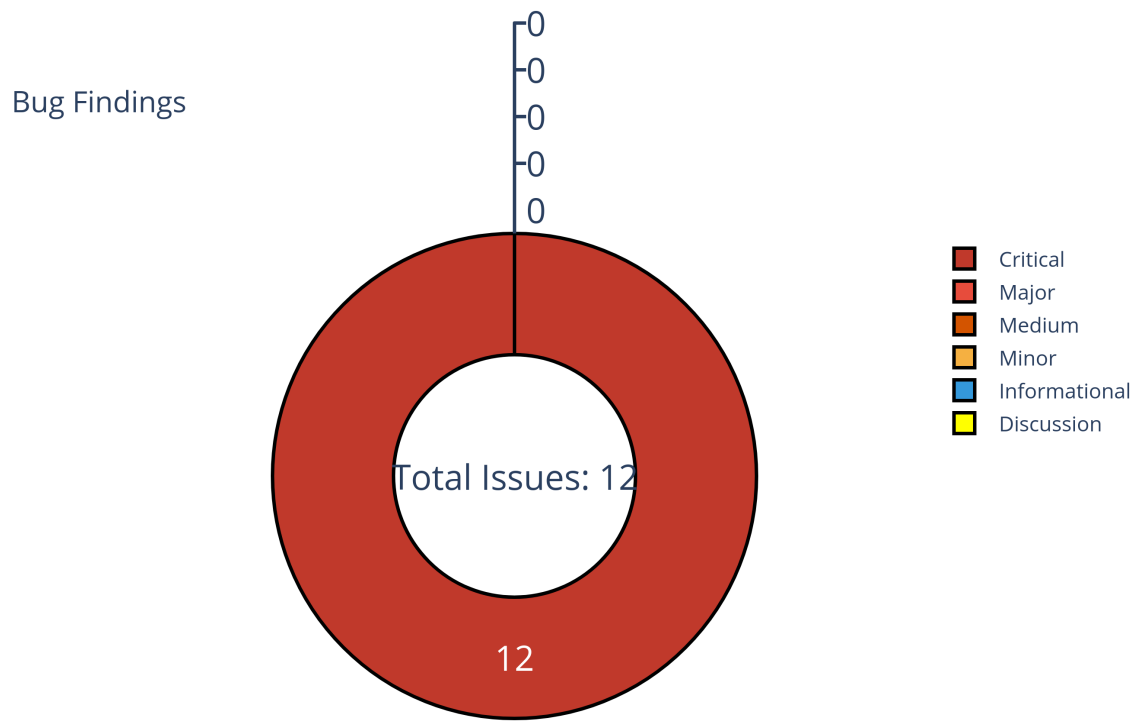
### Audit Summary

|                   |                 |
|-------------------|-----------------|
| Delivery Date     | 10/29/2022      |
| Audit Methodology | Static Analysis |
| Key Components    |                 |

### Vulnerability Summary

| Vulnerability Level | Total |
|---------------------|-------|
| Critical            | 12    |
| Major               | 0     |
| Medium              | 0     |
| Minor               | 0     |
| Informational       | 0     |
| Discussion          | 0     |

## Findings



**Figure 1:** Findings

## Finding Statistic

| Category               | Count |
|------------------------|-------|
| IntegerFlow            | 5     |
| MissingKeyCheck        | 2     |
| CrossProgramInvocation | 5     |

| ID | Category               | Severity | Status     |
|----|------------------------|----------|------------|
| 0  | IntegerFlow            | Critical | UnResolved |
| 1  | IntegerFlow            | Critical | UnResolved |
| 2  | IntegerFlow            | Critical | UnResolved |
| 3  | IntegerFlow            | Critical | UnResolved |
| 4  | IntegerFlow            | Critical | UnResolved |
| 5  | MissingKeyCheck        | Critical | UnResolved |
| 6  | MissingKeyCheck        | Critical | UnResolved |
| 7  | CrossProgramInvocation | Critical | UnResolved |
| 8  | CrossProgramInvocation | Critical | UnResolved |
| 9  | CrossProgramInvocation | Critical | UnResolved |
| 10 | CrossProgramInvocation | Critical | UnResolved |
| 11 | CrossProgramInvocation | Critical | UnResolved |

**Issue: 0: IntegerFlow**

| Category    | Severity | Status     |
|-------------|----------|------------|
| IntegerFlow | Critical | UnResolved |

- Location

src/engine/cancel\_subscription.rs:101:21: 101:67

```
101 (subscription_account.joined + trial_duration)
102
```

- Code Context

Vulnerability at Line: 101

```

96 let trial_duration: i64 = match package.trial {
97     None => 0,
98     Some(value) => value,
99 };
100 // don't allow cancellation if trial period ended
101 if timestamp >= (subscription_account.joined + trial_duration) {
102     msg!("Info: Subscription amount not refunded because trial period
        ↳ has ended.");
103 } else {
104     // Transferring payment back to the payer...
105     invoke_signed(
106
```

- Call Stack

```

1 fn entrypoint::process_instruction() { // src/entrypoint.rs:11:1: 23:2 }
2     fn processor::<impl
        ↳ instruction::PaymentProcessorInstruction>::process() { //
        ↳ src/processor.rs:15:5: 61:6 }
3     fn engine::cancel_subscription::process_cancel_subscription() { //
        ↳ src/engine/cancel_subscription.rs:24:1: 165:2 }
4
```

- description:



- link:
- alleviation:

## Issue: 1: IntegerFlow

| Category    | Severity | Status     |
|-------------|----------|------------|
| IntegerFlow | Critical | UnResolved |

- Location

src/engine/renew.rs:52:27: 52:60

```
52 (quantity as u64) * package.price
53
```

- Code Context

Vulnerability at Line: 52

```
47     order_info,
48     subscription_info,
49     &subscription_account.name,
50 );?;
51 // ensure the amount paid is as expected
52 let expected_amount = (quantity as u64) * package.price;
53 if expected_amount > order_account.paid_amount {
54     return Err(PaymentProcessorError::NotFullyPaid.into());
55 }
56 // update subscription account
57
```

- Call Stack

```
1 fn entrypoint::process_instruction(){// src/entrypoint.rs:11:1: 23:2 }
2   fn processor::<impl
   ↳ instruction::PaymentProcessorInstruction>::process(){//
   ↳ src/processor.rs:15:5: 61:6 }
3   fn engine::renew::process_renew_subscription(){//
   ↳ src/engine/renew.rs:14:1: 74:2 }
4
```

- description:

- link:
- alleviation:

## Issue: 2: IntegerFlow

| Category    | Severity | Status     |
|-------------|----------|------------|
| IntegerFlow | Critical | UnResolved |

- Location

src/engine/subscribe.rs:115:21: 115:47

```
115 timestamp + trial_duration
116
```

- Code Context

Vulnerability at Line: 115

```
110         owner: signer_info.key.to_bytes(),
111         merchant: merchant_info.key.to_bytes(),
112         name,
113         joined: timestamp,
114         period_start: timestamp,
115         period_end: timestamp + trial_duration + package.duration,
116         data,
117     };
118     subscription.pack(&mut subscription_data);
119
120
```

- Call Stack

```
1 fn entrypoint::process_instruction() { // src/entrypoint.rs:11:1: 23:2 }
2     fn processor::<impl
   ↳ instruction::PaymentProcessorInstruction>::process() { //
   ↳ src/processor.rs:15:5: 61:6 }
3     fn engine::subscribe::process_subscribe() { //
   ↳ src/engine/subscribe.rs:16:1: 126:2 }
4
```

- description:

- link:
- alleviation:

### Issue: 3: IntegerFlow

| Category    | Severity | Status     |
|-------------|----------|------------|
| IntegerFlow | Critical | UnResolved |

- Location

src/engine/withdraw.rs:123:24: 123:70

```
123 (subscription_account.joined + trial_duration)
124
```

- Code Context

Vulnerability at Line: 123

```
118     let trial_duration: i64 = match package.trial {
119         None => 0,
120         Some(value) => value,
121     };
122     // don't allow withdrawal if still within trial period
123     if timestamp < (subscription_account.joined + trial_duration) {
124         return
125             ↪ Err(PaymentProcessorError::CantWithdrawDuringTrial.into());
126     }
127     // Transferring payment to the merchant...
128
```

- Call Stack

```
1 fn endpoint::process_instruction() { // src/endpoint.rs:11:1: 23:2 }
2     fn processor::<impl
3         ↪ instruction::PaymentProcessorInstruction>::process() { //
4         ↪ src/processor.rs:15:5: 61:6 }
5     fn engine::withdraw::process_withdraw_payment() { //
6         ↪ src/engine/withdraw.rs:23:1: 186:2 }
7
```

- description:

- link:
- alleviation:

## Issue: 4: IntegerFlow

| Category    | Severity | Status     |
|-------------|----------|------------|
| IntegerFlow | Critical | UnResolved |

- Location

src/utils.rs:12:41: 12:74

```
12 (amount as u128 * fee_percentage)
13
```

- Code Context

Vulnerability at Line: 12

```
7 pub fn get_amounts(amount: u64, fee_percentage: u128) -> (u64, u64) {
8     let mut fee_amount: u64 = 0;
9     let mut take_home_amount: u64 = amount;
10
11     if amount >= 100 {
12         let possible_fee_amount: u128 = (amount as u128 * fee_percentage) /
            ↳ 1000;
13         fee_amount = 1;
14         if possible_fee_amount > 0 {
15             fee_amount = possible_fee_amount as u64;
16         }
17     }
```

- Call Stack

```
1 fn entrypoint::process_instruction() { // src/entrypoint.rs:11:1: 23:2 }
2     fn processor::<impl
            ↳ instruction::PaymentProcessorInstruction>::process() { //
            ↳ src/processor.rs:15:5: 61:6 }
3     fn engine::pay::process_chain_checkout() { //
            ↳ src/engine/pay.rs:351:1: 368:2 }
4         fn engine::pay::process_order() { // src/engine/pay.rs:137:1:
            ↳ 329:2 }
```



```
5 fn utils::get_amounts(){// src/utils.rs:7:1: 21:2 }  
6
```

- description:
- link:
- alleviation:

## Issue: 5: MissingKeyCheck

| Category        | Severity | Status     |
|-----------------|----------|------------|
| MissingKeyCheck | Critical | UnResolved |

- Location

/home/yifei/.cargo/registry/src/github.com-1ecc6299db9ec823/solana-program-1.7.1/src/account\_info.rs:70:11: 70:33

```
70 self.lamports.borrow()
71
```

- Code Context

Vulnerability at Line: 70

```
69 pub fn lamports(&self) -> u64 {
70     **self.lamports.borrow()
71 }
72
```

- Call Stack

```
1 fn entrypoint::process_instruction() { // src/entrypoint.rs:11:1: 23:2 }
2   fn processor::<impl
   ↳ instruction::PaymentProcessorInstruction>::process() { //
   ↳ src/processor.rs:15:5: 61:6 }
3   fn engine::cancel_subscription::process_cancel_subscription() { //
   ↳ src/engine/cancel_subscription.rs:24:1: 165:2 }
4   fn
   ↳ solana_program::account_info::AccountInfo::<'a>::lamports() { //
   ↳ /home/yifei/.cargo/registry/src/github.com-
   ↳ 1ecc6299db9ec823/solana-program-
   ↳ 1.7.1/src/account_info.rs:69:5: 71:6
   ↳ }
5
```

- description:

- link:
- alleviation:

## Issue: 6: MissingKeyCheck

| Category        | Severity | Status     |
|-----------------|----------|------------|
| MissingKeyCheck | Critical | UnResolved |

- Location

src/engine/cancel\_subscription.rs:152:49: 152:77

```
152 order_info.data.borrow_mut()
153
```

- Code Context

Vulnerability at Line: 152

```
147         order_info.lamports(),
148     )?;
149     // Updating order account information...
150     order_account.status = OrderStatus::Cancelled as u8;
151     order_account.modified = timestamp;
152     OrderAccount::pack(&order_account, &mut
        ↳ order_info.data.borrow_mut());
153     // set period end to right now
154     subscription_account.period_end = timestamp;
155 }
156
157
```

- Call Stack

```
1 fn entrypoint::process_instruction() { // src/entrypoint.rs:11:1: 23:2 }
2     fn processor::<impl
        ↳ instruction::PaymentProcessorInstruction>::process() {
        ↳ src/processor.rs:15:5: 61:6 }
3     fn engine::cancel_subscription::process_cancel_subscription() { //
        ↳ src/engine/cancel_subscription.rs:24:1: 165:2 }
4
```

- description:

- link:
- alleviation:

## Issue: 7: CrossProgramInvocation

| Category               | Severity | Status     |
|------------------------|----------|------------|
| CrossProgramInvocation | Critical | UnResolved |

- Location

src/engine/cancel\_subscription.rs

- Code Context

```
24 pub fn process_cancel_subscription(program_id: &Pubkey, accounts:
    ↳ &[AccountInfo]) -> ProgramResult {
25     let account_info_iter = &mut accounts.iter();
26
27     let signer_info = next_account_info(account_info_iter)?;
28     let subscription_info = next_account_info(account_info_iter)?;
29     let merchant_info = next_account_info(account_info_iter)?;
30     let order_info = next_account_info(account_info_iter)?;
31     let order_token_info = next_account_info(account_info_iter)?;
32     let refund_token_info = next_account_info(account_info_iter)?;
33     let account_to_receive_sol_refund_info =
    ↳ next_account_info(account_info_iter)?;
34     let pda_info = next_account_info(account_info_iter)?;
35     let token_program_info = next_account_info(account_info_iter)?;
36
37     let timestamp = Clock::get()?.unix_timestamp;
38
39     // ensure signer can sign
40     if !signer_info.is_signer {
41         return Err(ProgramError::MissingRequiredSignature);
42     }
43     // ensure subscription account is owned by this program
44     if *subscription_info.owner != *program_id {
45         msg!("Error: Wrong owner for subscription account");
46         return Err(ProgramError::IncorrectProgramId);
47     }
48     // ensure token accounts are owned by token program
```

```
49     if *order_token_info.owner != spl_token::id() {
50         msg!("Error: Order token account must be owned by token program");
51         return Err(ProgramError::IncorrectProgramId);
52     }
53     if *refund_token_info.owner != spl_token::id() {
54         msg!("Error: Refund token account must be owned by token program");
55         return Err(ProgramError::IncorrectProgramId);
56     }
57     // check that provided pda is correct
58     let (pda, pda_nonce) = Pubkey::find_program_address(&[PDA_SEED],
59         ↪ &program_id);
60     if pda_info.key != &pda {
61         return Err(ProgramError::InvalidSeeds);
62     }
63     // get the subscription account
64     let mut subscription_account =
65         ↪ SubscriptionAccount::unpack(&subscription_info.data.borrow())?;
66     if !subscription_account.is_initialized() {
67         return Err(ProgramError::UninitializedAccount);
68     }
69     if subscription_account.is_closed() {
70         return Err(PaymentProcessorError::ClosedAccount.into());
71     }
72     if subscription_account.discriminator != Discriminator::Subscription as
73         ↪ u8 {
74         msg!("Error: Invalid subscription account");
75         return Err(ProgramError::InvalidAccountData);
76     }
77     let (mut order_account, package) = subscribe_checks(
78         program_id,
79         signer_info,
80         merchant_info,
81         order_info,
82         subscription_info,
83         &subscription_account.name,
84     )?;
85     // ensure the order payment token account is the right one
86     if order_token_info.key.to_bytes() != order_account.token {
87         msg!("Error: Incorrect order token account");
88         return Err(ProgramError::InvalidAccountData);
89     }
```

```
88     }
89     // ensure the signer is the order payer
90     if signer_info.key.to_bytes() != order_account.payer {
91         msg!("Error: One can only cancel their own subscription payment");
92         return Err(ProgramError::InvalidAccountData);
93     }
94
95     // get the trial period duration
96     let trial_duration: i64 = match package.trial {
97         None => 0,
98         Some(value) => value,
99     };
100    // don't allow cancellation if trial period ended
101    if timestamp >= (subscription_account.joined + trial_duration) {
102        msg!("Info: Subscription amount not refunded because trial period
        ↳ has ended.");
103    } else {
104        // Transferring payment back to the payer...
105        invoke_signed(
106            &spl_token::instruction::transfer(
107                token_program_info.key,
108                order_token_info.key,
109                refund_token_info.key,
110                &pda,
111                &[&pda],
112                order_account.paid_amount,
113            ),
114            .unwrap(),
115            &[
116                token_program_info.clone(),
117                pda_info.clone(),
118                order_token_info.clone(),
119                refund_token_info.clone(),
120            ],
121            &[&[&PDA_SEED, &[pda_nonce]]],
122        )?;
123        // Close the order token account since it will never be needed
        ↳ again
124        invoke_signed(
125            &spl_token::instruction::close_account(
126                token_program_info.key,
127                order_token_info.key,
```



```
128         account_to_receive_sol_refund_info.key,  
129         &pda,  
130         &[&pda],  
131     )  
132     .unwrap(),  
133     &[  
134         token_program_info.clone(),  
135         order_token_info.clone(),  
136         account_to_receive_sol_refund_info.clone(),  
137         pda_info.clone(),  
138     ],  
139     &[&[&PDA_SEED, &[pda_nonce]]],  
140 )?;  
141 // mark order account as closed  
142 order_account.discriminator = Discriminator::Closed as u8;  
143 // Transfer all the sol from the order account to the  
144 ↳ sol_destination.  
145 transfer_sol(  
146     order_info.clone(),  
147     account_to_receive_sol_refund_info.clone(),  
148     order_info.lamports(),  
149 )?;  
150 // Updating order account information...  
151 order_account.status = OrderStatus::Cancelled as u8;  
152 order_account.modified = timestamp;  
153 OrderAccount::pack(&order_account, &mut  
154 ↳ order_info.data.borrow_mut());  
155 // set period end to right now  
156 subscription_account.period_end = timestamp;  
157 }  
158 // Updating subscription account information...  
159 subscription_account.status = SubscriptionStatus::Cancelled as u8;  
160 SubscriptionAccount::pack(  
161 &subscription_account,  
162 &mut subscription_info.data.borrow_mut(),  
163 );  
164 Ok::<(),  
165 }  
166
```

- Call Stack

```
1 fn entrypoint::process_instruction() { // src/entrypoint.rs:11:1: 23:2 }
2   fn processor::<impl
   ↳ instruction::PaymentProcessorInstruction>::process() { //
   ↳ src/processor.rs:15:5: 61:6 }
3   fn engine::cancel_subscription::process_cancel_subscription() { //
   ↳ src/engine/cancel_subscription.rs:24:1: 165:2 }
4
```

- description:
- link:
- alleviation:

## Issue: 8: CrossProgramInvocation

| Category               | Severity | Status     |
|------------------------|----------|------------|
| CrossProgramInvocation | Critical | UnResolved |

- Location

src/engine/withdraw.rs

- Code Context

```
23 pub fn process_withdraw_payment(  
24     program_id: &Pubkey,  
25     accounts: &[AccountInfo],  
26     close_order_account: bool,  
27 ) -> ProgramResult {  
28     let account_info_iter = &mut accounts.iter();  
29     let signer_info = next_account_info(account_info_iter)?;  
30     let order_info = next_account_info(account_info_iter)?;  
31     let merchant_info = next_account_info(account_info_iter)?;  
32     let order_payment_token_info = next_account_info(account_info_iter)?;  
33     let merchant_token_info = next_account_info(account_info_iter)?;  
34     let account_to_receive_sol_refund_info =  
35         ↪ next_account_info(account_info_iter)?;  
36     let pda_info = next_account_info(account_info_iter)?;  
37     let token_program_info = next_account_info(account_info_iter)?;  
38  
39     let timestamp = Clock::get()?.unix_timestamp;  
40  
41     // ensure signer can sign  
42     if !signer_info.is_signer {  
43         return Err(ProgramError::MissingRequiredSignature);  
44     }  
45     // ensure merchant and order accounts are owned by this program  
46     if *merchant_info.owner != *program_id {  
47         msg!("Error: Wrong owner for merchant account");  
48         return Err(ProgramError::IncorrectProgramId);  
49     }
```

```
49     if *order_info.owner != *program_id {
50         msg!("Error: Wrong owner for order account");
51         return Err(ProgramError::IncorrectProgramId);
52     }
53     // ensure buyer token account is owned by token program
54     if *merchant_token_info.owner != spl_token::id() {
55         msg!("Error: Token account must be owned by token program");
56         return Err(ProgramError::IncorrectProgramId);
57     }
58     // check that provided pda is correct
59     let (pda, pda_nonce) = Pubkey::find_program_address(&[PDA_SEED],
60         ↪ &program_id);
61     if pda_info.key != &pda {
62         return Err(ProgramError::InvalidSeeds);
63     }
64     // get the merchant account
65     let merchant_account =
66         ↪ MerchantAccount::unpack(&merchant_info.data.borrow())?;
67     if merchant_account.is_closed() {
68         return Err(PaymentProcessorError::ClosedAccount.into());
69     }
70     if !merchant_account.is_initialized() {
71         return Err(ProgramError::UninitializedAccount);
72     }
73     // ensure that the token account that we will withdraw to is owned by
74     ↪ this
75     // merchant. This ensures that anyone can call the withdraw
76     ↪ instruction
77     // and the money will still go to the right place
78     let merchant_token_data =
79         ↪ TokenAccount::unpack(&merchant_token_info.data.borrow())?;
80     if merchant_token_data.owner !=
81         ↪ Pubkey::new_from_array(merchant_account.owner) {
82         return Err(PaymentProcessorError::WrongMerchant.into());
83     }
84     // get the order account
85     let mut order_account =
86         ↪ OrderAccount::unpack(&order_info.data.borrow())?;
87     if order_account.is_closed() {
88         return Err(PaymentProcessorError::ClosedAccount.into());
89     }
90     if !order_account.is_initialized() {
```

```
84     return Err(ProgramError::UninitializedAccount);
85 }
86 // ensure order belongs to this merchant
87 if merchant_info.key.to_bytes() != order_account.merchant {
88     return Err(ProgramError::InvalidAccountData);
89 }
90 // ensure the order payment token account is the right one
91 if order_payment_token_info.key.to_bytes() != order_account.token {
92     return Err(ProgramError::InvalidAccountData);
93 }
94 // ensure order is not already paid out
95 if order_account.status != OrderStatus::Paid as u8 {
96     return Err(PaymentProcessorError::AlreadyWithdrawn.into());
97 }
98 // check if this is for a subscription payment that has a trial period
99 if merchant_account.discriminator ==
100     ↪ Discriminator::MerchantSubscriptionWithTrial as u8 {
101     let subscription_info = next_account_info(account_info_iter)?;
102     // ensure subscription account is owned by this program
103     if *subscription_info.owner != *program_id {
104         msg!("Error: Wrong owner for subscription account");
105         return Err(ProgramError::IncorrectProgramId);
106     }
107     // ensure this order is for this subscription
108     verify_subscription_order(subscription_info, &order_account)?;
109     // get the subscription account
110     let subscription_account =
111         ↪ SubscriptionAccount::unpack(&subscription_info.data.borrow())?;
112     if subscription_account.is_closed() {
113         return Err(PaymentProcessorError::ClosedAccount.into());
114     }
115     if !subscription_account.is_initialized() {
116         return Err(ProgramError::UninitializedAccount);
117     }
118     let package = get_subscription_package(&subscription_account.name,
119         ↪ &merchant_account)?;
120     // get the trial period duration
121     let trial_duration: i64 = match package.trial {
122         None => 0,
123         Some(value) => value,
```

```

123         if timestamp < (subscription_account.joined + trial_duration) {
124             return
125             ↪ Err(PaymentProcessorError::CantWithdrawDuringTrial.into());
126         }
127     // Transferring payment to the merchant...
128     invoke_signed(
129         &spl_token::instruction::transfer(
130             token_program_info.key,
131             order_payment_token_info.key,
132             merchant_token_info.key,
133             &pda,
134             &[&pda],
135             order_account.paid_amount,
136         )
137         .unwrap(),
138         &[
139             token_program_info.clone(),
140             order_payment_token_info.clone(),
141             merchant_token_info.clone(),
142             pda_info.clone(),
143         ],
144         &[&[&PDA_SEED, &[pda_nonce]]],
145     )?;
146     // Close the order token account since it will never be needed again
147     invoke_signed(
148         &spl_token::instruction::close_account(
149             token_program_info.key,
150             order_payment_token_info.key,
151             account_to_receive_sol_refund_info.key,
152             &pda,
153             &[&pda],
154         )
155         .unwrap(),
156         &[
157             token_program_info.clone(),
158             order_payment_token_info.clone(),
159             account_to_receive_sol_refund_info.clone(),
160             pda_info.clone(),
161         ],
162         &[&[&PDA_SEED, &[pda_nonce]]],
163     )?;

```

```

164
165     if close_order_account {
166         if merchant_account.owner != signer_info.key.to_bytes() {
167             msg!("Error: Only merchant account owner can close order
168                 ↳ account");
169             return Err(ProgramError::MissingRequiredSignature);
170         }
171         // mark account as closed
172         order_account.discriminator = Discriminator::Closed as u8;
173         // Transfer all the sol from the order account to the
174         ↳ sol_destination.
175         transfer_sol(
176             order_info.clone(),
177             account_to_receive_sol_refund_info.clone(),
178             order_info.lamports(),
179             );
180     }
181
182     // Updating order account information...
183     order_account.status = OrderStatus::Withdrawn as u8;
184     order_account.modified = timestamp;
185     OrderAccount::pack(&order_account, &mut order_info.data.borrow_mut());
186
187     Ok(())
188 }

```

- Call Stack

```

1 fn entrypoint::process_instruction() { // src/entrypoint.rs:11:1: 23:2 }
2     fn processor::<impl
3         ↳ instruction::PaymentProcessorInstruction>::process() { //
4         ↳ src/processor.rs:15:5: 61:6 }
5     fn engine::withdraw::process_withdraw_payment() { //
6         ↳ src/engine/withdraw.rs:23:1: 186:2 }

```

- description:
- link:
- alleviation:

## Issue: 9: CrossProgramInvocation

| Category               | Severity | Status     |
|------------------------|----------|------------|
| CrossProgramInvocation | Critical | UnResolved |

- Location

src/engine/common.rs

- Code Context

```

136 pub fn create_program_owned_associated_token_account(
137     program_id: &Pubkey,
138     accounts: &[AccountInfo; 8],
139     rent: &Rent,
140 ) -> ProgramResult {
141     let signer_info = &accounts[0];
142     let base_account_info = &accounts[1];
143     let new_account_info = &accounts[2];
144     let mint_info = &accounts[3];
145     let pda_info = &accounts[4];
146     let token_program_info = &accounts[5];
147     let system_program_info = &accounts[6];
148     let rent_sysvar_info = &accounts[7];
149
150     let (associated_token_address, bump_seed) =
151         ↪ Pubkey::find_program_address(
152             &[
153                 &base_account_info.key.to_bytes(),
154                 &spl_token::id().to_bytes(),
155                 &mint_info.key.to_bytes(),
156             ],
157             program_id,
158         );
159     // assert that the derived address matches the one supplied
160     if associated_token_address != *new_account_info.key {
161         msg!("Error: Associated address does not match seed derivation");
162         return Err(ProgramError::InvalidSeeds);
163     }

```



```
162     }
163     // get signer seeds
164     let associated_token_account_signer_seeds: &[_] = &[
165         &base_account_info.key.to_bytes(),
166         &spl_token::id().to_bytes(),
167         &mint_info.key.to_bytes(),
168         &bump_seed,
169     ];
170     // Fund the associated seller token account with the minimum balance to
171     ↳ be rent exempt
172     let required_lamports = rent
173         .minimum_balance(spl_token::state::Account::LEN)
174         .max(1)
175         .saturating_sub(new_account_info.lamports());
176     if required_lamports > 0 {
177         // Transfer lamports to the associated seller token account
178         invoke(
179             &system_instruction::transfer(
180                 &signer_info.key,
181                 new_account_info.key,
182                 required_lamports,
183             ),
184             &[
185                 signer_info.clone(),
186                 new_account_info.clone(),
187                 system_program_info.clone(),
188             ],
189         )?;
190     }
191     // Allocate space for the associated seller token account
192     invoke_signed(
193         &system_instruction::allocate(new_account_info.key,
194         ↳ spl_token::state::Account::LEN as u64),
195         &[new_account_info.clone(), system_program_info.clone()],
196         &[&associated_token_account_signer_seeds],
197     )?;
198     // Assign the associated seller token account to the SPL Token program
199     invoke_signed(
200         &system_instruction::assign(new_account_info.key,
201         ↳ &spl_token::id()),
202         &[new_account_info.clone(), system_program_info.clone()],
203         &[&associated_token_account_signer_seeds],
```

```

201     )?;
202     // Initialize the associated seller token account
203     invoke(
204         &spl_token::instruction::initialize_account(
205             &spl_token::id(),
206             new_account_info.key,
207             mint_info.key,
208             pda_info.key,
209         )?,
210         &[
211             new_account_info.clone(),
212             mint_info.clone(),
213             pda_info.clone(),
214             rent_sysvar_info.clone(),
215             token_program_info.clone(),
216         ],
217     )?;
218
219     Ok(())
220 }
221

```

- Call Stack

```

1  fn entrypoint::process_instruction() { // src/entrypoint.rs:11:1: 23:2 }
2      fn processor::<impl
   ↳ instruction::PaymentProcessorInstruction>::process() { //
   ↳ src/processor.rs:15:5: 61:6 }
3      fn engine::pay::process_chain_checkout() { //
   ↳ src/engine/pay.rs:351:1: 368:2 }
4      fn engine::pay::process_order() { // src/engine/pay.rs:137:1:
   ↳ 329:2 }
5      fn en-
   ↳ gine::common::create_program_owned_associated_token_account() { //
   ↳ src/engine/common.rs:136:1: 220:2 }
6

```

- description:
- link:
- alleviation:

## Issue: 10: CrossProgramInvocation

| Category               | Severity | Status     |
|------------------------|----------|------------|
| CrossProgramInvocation | Critical | UnResolved |

- Location

src/engine/pay.rs

- Code Context

```
137 pub fn process_order(  
138     program_id: &Pubkey,  
139     accounts: &[AccountInfo],  
140     amount: u64,  
141     order_id: String,  
142     secret: String,  
143     maybe_data: Option<String>,  
144     checkout_items: Option<OrderItems>,  
145 ) -> ProgramResult {  
146     let account_info_iter = &mut accounts.iter();  
147   
148     let signer_info = next_account_info(account_info_iter)?;  
149     let order_info = next_account_info(account_info_iter)?;  
150     let merchant_info = next_account_info(account_info_iter)?;  
151     let seller_token_info = next_account_info(account_info_iter)?;  
152     let buyer_token_info = next_account_info(account_info_iter)?;  
153     let program_owner_info = next_account_info(account_info_iter)?;  
154     let sponsor_info = next_account_info(account_info_iter)?;  
155     let mint_info = next_account_info(account_info_iter)?;  
156     let pda_info = next_account_info(account_info_iter)?;  
157     let token_program_info = next_account_info(account_info_iter)?;  
158     let system_program_info = next_account_info(account_info_iter)?;  
159     let rent_sysvar_info = next_account_info(account_info_iter)?;  
160   
161     let rent = &Rent::from_account_info(rent_sysvar_info)?;  
162     let timestamp = Clock::get()?.unix_timestamp;  
163 }
```

```
164     let merchant_account = order_checks(  
165         program_id,  
166         signer_info,  
167         merchant_info,  
168         buyer_token_info,  
169         mint_info,  
170         program_owner_info,  
171         sponsor_info,  
172     )?;  
173  
174     // get data  
175     let mut data = match maybe_data {  
176         None => String::from(DEFAULT_DATA),  
177         Some(value) => value,  
178     };  
179  
180     let mut order_account_type = Discriminator::OrderExpressCheckout as u8;  
181  
182     // process chain checkout  
183     if checkout_items.is_some() {  
184         order_account_type = Discriminator::OrderChainCheckout as u8;  
185         let order_items = checkout_items.unwrap();  
186         chain_checkout_checks(&merchant_account, &mint_info.clone(),  
187     ↪ &order_items, amount)?;  
188         if data == String::from(DEFAULT_DATA) {  
189             data = json!({ PAID: order_items }).to_string();  
190         } else {  
191             // let possible_json_data: Result<BTreeMap<&str, Value>,  
192     ↪ JSONError> = serde_json::from_str(&data);  
193             // let json_data = match possible_json_data {  
194         let json_data: Value = match serde_json::from_str(&data) {  
195             Err(_error) => return  
196     ↪ Err(PaymentProcessorError::InvalidOrderData.into()),  
197             Ok(data) => data,  
198         };  
199         data = json!({  
200             INITIAL: json_data,  
201             PAID: order_items  
202         })  
        .to_string();  
    }  
}
```

```
203
204 // create order account
205 let order_account_size = get_order_account_size(&order_id, &secret,
    ↪ &data);
206 // the order account amount includes the fee in SOL
207 let order_account_amount =
    ↪ Rent::().minimum_balance(order_account_size);
208 invoke(
209     &system_instruction::create_account(
210         signer_info.key,
211         order_info.key,
212         order_account_amount,
213         order_account_size as u64,
214         program_id,
215     ),
216     &[
217         signer_info.clone(),
218         order_info.clone(),
219         system_program_info.clone(),
220     ],
221 )?;
222
223 // next we are going to try and create a token account owned by the
    ↪ program
224 // but whose address is derived from the order account
225 // TODO: for subscriptions, should this use the subscription account as
    ↪ the base?
226 create_program_owned_associated_token_account(
227     program_id,
228     &[
229         signer_info.clone(),
230         order_info.clone(),
231         seller_token_info.clone(),
232         mint_info.clone(),
233         pda_info.clone(),
234         token_program_info.clone(),
235         system_program_info.clone(),
236         rent_sysvar_info.clone(),
237     ],
238     rent,
239 )?;
240
```

```
241 // Transfer payment amount to associated seller token account...
242 invoke(
243     &spl_token::instruction::transfer(
244         token_program_info.key,
245         buyer_token_info.key,
246         seller_token_info.key,
247         signer_info.key,
248         &&signer_info.key],
249         amount,
250     )
251     .unwrap(),
252     &[
253         buyer_token_info.clone(),
254         seller_token_info.clone(),
255         signer_info.clone(),
256         token_program_info.clone(),
257     ],
258 )?;
259
260 if Pubkey::new_from_array(merchant_account.sponsor) ==
261     ↪ Pubkey::from_str(PROGRAM_OWNER).unwrap()
262 {
263     // Transferring processing fee to the program owner...
264     invoke(
265         &system_instruction::transfer(
266             &signer_info.key,
267             program_owner_info.key,
268             merchant_account.fee,
269         ),
270         &[
271             signer_info.clone(),
272             program_owner_info.clone(),
273             system_program_info.clone(),
274         ],
275     )?;
276 } else {
277     // we need to pay both the program owner and the sponsor
278     let (program_owner_fee, sponsor_fee) =
279     ↪ get_amounts(merchant_account.fee, SPONSOR_FEE);
280     // Transferring processing fee to the program owner and sponsor...
281     invoke(
282         &system_instruction::transfer(
```

```
281         &signer_info.key,  
282         program_owner_info.key,  
283         program_owner_fee,  
284     ),  
285     &[  
286         signer_info.clone(),  
287         program_owner_info.clone(),  
288         system_program_info.clone(),  
289     ],  
290 )?;  
291 invoke(  
292     &system_instruction::transfer(&signer_info.key,  
↪ sponsor_info.key, sponsor_fee),  
293     &[  
294         signer_info.clone(),  
295         sponsor_info.clone(),  
296         system_program_info.clone(),  
297     ],  
298 )?;  
299 }  
300  
301 // get the order account  
302 // TODO: ensure this account is not already initialized  
303 let mut order_account_data = order_info.try_borrow_mut_data()?;  
304 // Saving order information...  
305 let order = OrderAccount {  
306     discriminator: order_account_type,  
307     status: OrderStatus::Paid as u8,  
308     created: timestamp,  
309     modified: timestamp,  
310     merchant: merchant_info.key.to_bytes(),  
311     mint: mint_info.key.to_bytes(),  
312     token: seller_token_info.key.to_bytes(),  
313     payer: signer_info.key.to_bytes(),  
314     expected_amount: amount,  
315     paid_amount: amount,  
316     order_id,  
317     secret,  
318     data,  
319 };  
320  
321 order.pack(&mut order_account_data);
```

```
322
323     // ensure order account is rent exempt
324     if !rent.is_exempt(order_info.lamports(), order_account_size) {
325         return Err(ProgramError::AccountNotRentExempt);
326     }
327
328     Ok(())
329 }
330
```

- Call Stack

```
1 fn entrypoint::process_instruction() { // src/entrypoint.rs:11:1: 23:2 }
2   fn processor::<impl
   ↳ instruction::PaymentProcessorInstruction>::process() { //
   ↳ src/processor.rs:15:5: 61:6 }
3   fn engine::pay::process_chain_checkout() { //
   ↳ src/engine/pay.rs:351:1: 368:2 }
4     fn engine::pay::process_order() { // src/engine/pay.rs:137:1:
   ↳ 329:2 }
5
```

- description:
- link:
- alleviation:



## Issue: 11: CrossProgramInvocation

| Category               | Severity | Status     |
|------------------------|----------|------------|
| CrossProgramInvocation | Critical | UnResolved |

- Location

src/engine/pay.rs

- Code Context

```
137 pub fn process_order(  
138     program_id: &Pubkey,  
139     accounts: &[AccountInfo],  
140     amount: u64,  
141     order_id: String,  
142     secret: String,  
143     maybe_data: Option<String>,  
144     checkout_items: Option<OrderItems>,  
145 ) -> ProgramResult {  
146     let account_info_iter = &mut accounts.iter();  
147   
148     let signer_info = next_account_info(account_info_iter)?;  
149     let order_info = next_account_info(account_info_iter)?;  
150     let merchant_info = next_account_info(account_info_iter)?;  
151     let seller_token_info = next_account_info(account_info_iter)?;  
152     let buyer_token_info = next_account_info(account_info_iter)?;  
153     let program_owner_info = next_account_info(account_info_iter)?;  
154     let sponsor_info = next_account_info(account_info_iter)?;  
155     let mint_info = next_account_info(account_info_iter)?;  
156     let pda_info = next_account_info(account_info_iter)?;  
157     let token_program_info = next_account_info(account_info_iter)?;  
158     let system_program_info = next_account_info(account_info_iter)?;  
159     let rent_sysvar_info = next_account_info(account_info_iter)?;  
160   
161     let rent = &Rent::from_account_info(rent_sysvar_info)?;  
162     let timestamp = Clock::get()?.unix_timestamp;  
163 }
```

```
164     let merchant_account = order_checks(  
165         program_id,  
166         signer_info,  
167         merchant_info,  
168         buyer_token_info,  
169         mint_info,  
170         program_owner_info,  
171         sponsor_info,  
172     )?;  
173  
174     // get data  
175     let mut data = match maybe_data {  
176         None => String::from(DEFAULT_DATA),  
177         Some(value) => value,  
178     };  
179  
180     let mut order_account_type = Discriminator::OrderExpressCheckout as u8;  
181  
182     // process chain checkout  
183     if checkout_items.is_some() {  
184         order_account_type = Discriminator::OrderChainCheckout as u8;  
185         let order_items = checkout_items.unwrap();  
186         chain_checkout_checks(&merchant_account, &mint_info.clone(),  
↪ &order_items, amount)?;  
187         if data == String::from(DEFAULT_DATA) {  
188             data = json!({ PAID: order_items }).to_string();  
189         } else {  
190             // let possible_json_data: Result<BTreeMap<&str, Value>,  
↪ JSONError> = serde_json::from_str(&data);  
191             // let json_data = match possible_json_data {  
192                 let json_data: Value = match serde_json::from_str(&data) {  
193                     Err(_error) => return  
↪ Err(PaymentProcessorError::InvalidOrderData.into()),  
194                     Ok(data) => data,  
195                 };  
196                 data = json!({  
197                     INITIAL: json_data,  
198                     PAID: order_items  
199                 })  
200                 .to_string();  
201             }  
202         }
```

```
203
204 // create order account
205 let order_account_size = get_order_account_size(&order_id, &secret,
    ↪ &data);
206 // the order account amount includes the fee in SOL
207 let order_account_amount =
    ↪ Rent::().minimum_balance(order_account_size);
208 invoke(
209     &system_instruction::create_account(
210         signer_info.key,
211         order_info.key,
212         order_account_amount,
213         order_account_size as u64,
214         program_id,
215     ),
216     &[
217         signer_info.clone(),
218         order_info.clone(),
219         system_program_info.clone(),
220     ],
221 )?;
222
223 // next we are going to try and create a token account owned by the
    ↪ program
224 // but whose address is derived from the order account
225 // TODO: for subscriptions, should this use the subscription account as
    ↪ the base?
226 create_program_owned_associated_token_account(
227     program_id,
228     &[
229         signer_info.clone(),
230         order_info.clone(),
231         seller_token_info.clone(),
232         mint_info.clone(),
233         pda_info.clone(),
234         token_program_info.clone(),
235         system_program_info.clone(),
236         rent_sysvar_info.clone(),
237     ],
238     rent,
239 )?;
240
```

```
241 // Transfer payment amount to associated seller token account...
242 invoke(
243     &spl_token::instruction::transfer(
244         token_program_info.key,
245         buyer_token_info.key,
246         seller_token_info.key,
247         signer_info.key,
248         &&signer_info.key],
249         amount,
250     )
251     .unwrap(),
252     &[
253         buyer_token_info.clone(),
254         seller_token_info.clone(),
255         signer_info.clone(),
256         token_program_info.clone(),
257     ],
258 )?;
259
260 if Pubkey::new_from_array(merchant_account.sponsor) ==
261     ↪ Pubkey::from_str(PROGRAM_OWNER).unwrap()
262 {
263     // Transferring processing fee to the program owner...
264     invoke(
265         &system_instruction::transfer(
266             &signer_info.key,
267             program_owner_info.key,
268             merchant_account.fee,
269         ),
270         &[
271             signer_info.clone(),
272             program_owner_info.clone(),
273             system_program_info.clone(),
274         ],
275     )?;
276 } else {
277     // we need to pay both the program owner and the sponsor
278     let (program_owner_fee, sponsor_fee) =
279     ↪ get_amounts(merchant_account.fee, SPONSOR_FEE);
280     // Transferring processing fee to the program owner and sponsor...
281     invoke(
282         &system_instruction::transfer(
```

```
281         &signer_info.key,  
282         program_owner_info.key,  
283         program_owner_fee,  
284     ),  
285     &[  
286         signer_info.clone(),  
287         program_owner_info.clone(),  
288         system_program_info.clone(),  
289     ],  
290 )?;  
291 invoke(  
292     &system_instruction::transfer(&signer_info.key,  
↪ sponsor_info.key, sponsor_fee),  
293     &[  
294         signer_info.clone(),  
295         sponsor_info.clone(),  
296         system_program_info.clone(),  
297     ],  
298 )?;  
299 }  
300  
301 // get the order account  
302 // TODO: ensure this account is not already initialized  
303 let mut order_account_data = order_info.try_borrow_mut_data()?;  
304 // Saving order information...  
305 let order = OrderAccount {  
306     discriminator: order_account_type,  
307     status: OrderStatus::Paid as u8,  
308     created: timestamp,  
309     modified: timestamp,  
310     merchant: merchant_info.key.to_bytes(),  
311     mint: mint_info.key.to_bytes(),  
312     token: seller_token_info.key.to_bytes(),  
313     payer: signer_info.key.to_bytes(),  
314     expected_amount: amount,  
315     paid_amount: amount,  
316     order_id,  
317     secret,  
318     data,  
319 };  
320  
321 order.pack(&mut order_account_data);
```

```
322
323     // ensure order account is rent exempt
324     if !rent.is_exempt(order_info.lamports(), order_account_size) {
325         return Err(ProgramError::AccountNotRentExempt);
326     }
327
328     Ok(())
329 }
330
```

- Call Stack

```
1 fn entrypoint::process_instruction() { // src/entrypoint.rs:11:1: 23:2 }
2     fn processor::<impl
   ↳ instruction::PaymentProcessorInstruction>::process() { //
   ↳ src/processor.rs:15:5: 61:6 }
3     fn engine::pay::process_express_checkout() { //
   ↳ src/engine/pay.rs:331:1: 349:2 }
4     fn engine::pay::process_order() { // src/engine/pay.rs:137:1:
   ↳ 329:2 }
5
```

- description:
- link:
- alleviation:

## Appendix

Copied from <https://leaderboard.certik.io/projects/aave>

### Finding Categories

#### Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

#### Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

#### Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how `block.timestamp` works.

#### Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

#### Coding Style

Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

#### Checksum Calculation Method

The “Checksum” field in the “Audit Scope” section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux “sha256sum” command against the target file.



## Disclaimer

Copied from <https://leaderboard.certik.io/projects/aave>

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 you (“Customer” or 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, nor may copies be delivered to any other person other than the Company, without CertiK’s prior written consent in each instance.

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 CertiK 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. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’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. The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.