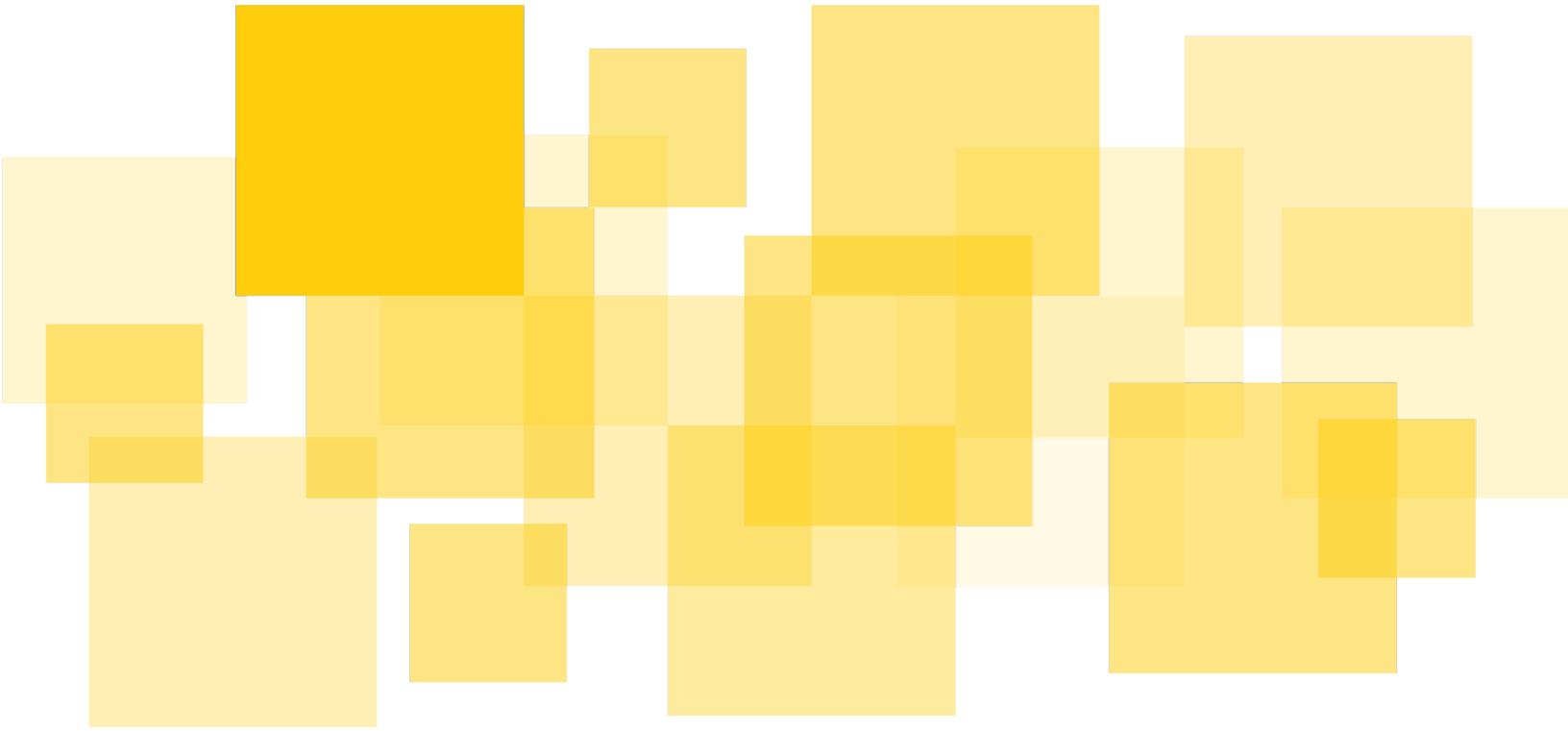
Security Audit Report

EXA Finance Smart Contracts

Delivered: March 1st, 2022



Prepared for EXA Finance by

runtime verification

Table of Contents

Executive Summary

EXA Finance engaged Runtime Verification Inc. to conduct a security audit of their smart contracts. The objective was to review the contracts' business logic and implementations in TEAL and identify any issues that could potentially cause the system to malfunction or be exploited.

The audit was conducted over the course of 7 calendar weeks (January 11, 2022 through March 1, 2022) focused on analyzing the security of the code related to the exchange of different types of assets through instances of contracts, allowing the usage of different peer-to-peer trading models for swapping groups of assets.

The audit led to identifying potentially critical issues, which have been identified as follows:

- Implementation flaws: A02, A08, A09, A11, A15, A16, A17
- Potential asset security vulnerabilities: A01, A02, A03, A04, A05, A06, A07

In addition, several informative findings and general recommendations also have been made, including:

- Input validations: A04, A10, A11, A12, A14
- Best practices: A01, A11, A12, A13, B03, B05, B06
- Optimization related particularities: A07, B02, B04
- Blockchain related particularities: A13, B01, B03

All the potentially critical issues have been addressed, and the great majority of the informative findings and general recommendations have been incorporated as well. All of the remaining findings have been acknowledged by the client and deemed non-threatening to the integrity of the platform, when not intended by design. The code is amply documented, modular and thoughtfully designed, following best practices.

Goal

The goal of the audit is twofold:

1. Review the high-level business logic (protocol design) of the EXA Finance system based on the provided documentation;
2. Review the low-level implementation of the system given as smart contracts in TEAL.

The audit focuses on trying to identify issues in the system's logic and its implementation that could potentially render the system vulnerable to attacks or cause it to malfunction. Furthermore, the audit highlights informative findings that could be used to improve safety and efficiency of the implementation.

Platform and Contract Features Description

EXA Finance is a platform built over the Algorand blockchain that allows users to trade assets with other users on a peer-to-peer model. Considering their features, a highlight is given to the *basket* functionality, which allows the exchange of fungible and/or non-fungible assets in groups of up to 4 *Algorand Standard Assets* (ASAs). A basket can be exchanged for an amount of another ASA chosen by the seller or calculated by the contract, according to its configuration.

The exchange between the assets in the basket and an amount of a price asset can be done following four different exchange models. These models, which are implemented in the audited contracts, are described below:

- **Simple sale:** baskets are sold for a fixed price specified by the seller;
- **Normal auction:** potential buyers bid in an auction in order to claim the assets in the basket, and the highest bidder wins;
- **Shopping cart:** a basket containing an amount of a single asset type, where different buyers can purchase fractions of the amount of the asset available;
- **Reverse/dutch auction:** the price of a basket is reduced according to the progression of time.

In the specification of the contract, it is possible to see that there are up to seven actors involved in the execution of the contract: these actors are the seller or the auctioneer, the buyer or the bidder, the fee receiver and the royalties receivers. The first two are

active actors in the execution of the contract, needing to initiate interactions with it in order to take part in the exchange of assets. The remaining are passive actors, only receiving assets from the contract. It should be noted that if the creator of the contract, which is the seller of the basket, does not specify a royalties address or the royalties amount per ten thousand assets is zero, no transfers will be made for the payment of royalties. Another particularity of the royalties payment is that an address for payment and an amount of royalties per ten thousand assets can be specified for each individual asset in the basket, hence there can be up to four actors representing the royalties addresses.

The EXA Finance platform also provides several functionalities for enabling the usage of baskets to facilitate the creation of offers directly between users.

Scope

The scope of the audit is limited to the artifacts available at git-commit-id **66e4d25caf7e661ec90f6091301d5688850b1ff1** for the branch **rv-audit** of a private repository provided by the client.

The comments provided in the code, a README.md file available in the project repository, the transaction groups description provided by the client, and the testnet documentation of the platform were used as reference material.

The analyzed functionalities consider the handling and exchange of groups of assets between users, which is implemented using two contracts:

1. **approval_program.teal**: implements the core functionalities of the contract, handling the application call transactions and controlling the assets according to the specified logic;
2. **clear_state_program.teal**: implements the actions of the contract when handling a clear state transaction.

The audit is limited in scope to the artifacts listed above. Off-chain and client-side portions of the codebase as well as deployment and upgrade scripts are not in the scope of this engagement.

Methodology

Although the manual code review cannot guarantee to find all possible security vulnerabilities as mentioned in our Disclaimer, we have followed the approaches described below to make our audit as thorough as possible.

First, we rigorously reasoned about the business logic of the contract, validating security-critical properties to ensure the absence of loopholes in the business logic. To this end, we carefully analyzed all the proposed features of the platform and actors involved in the lifetime of a deployed contract.

Second, we thoroughly reviewed the contract source code to detect any unexpected (and possibly exploitable) behaviors. As TEAL is a relatively low-level language, we constructed different higher-level representations of the TEAL codebase, including:

1. Automatically generated control-flow graphs using an extended version of the Tealer Static Analyzer tool, coupled with internally developed tools to assist the code navigation and readability,
2. Modeled sequences of mathematical operations as equations and, considering the limitations of size and types of variables in the *Algorand Virtual Machine* (AVM), checking if all desired properties hold for any possible input value, and then

This approach enabled us to systematically check consistency between the logic and the low-level TEAL implementation.

Thirdly, we performed rounds of internal discussion with Algorand experts over the code and platform design, aiming to verify possible exploitation vectors and to identify improvements for the analyzed contracts.

Fourthly, we reviewed the TEAL guidelines published by Algorand to check for known issues.

Additionally, given the nascent TEAL development and auditing community, we reviewed this list of known Ethereum security vulnerabilities and attack vectors and checked whether they apply to TEAL smart contracts; if they apply, we checked whether the code is vulnerable to them.

Disclaimer

This report does not constitute legal or investment advice. The preparers of this report present it as an informational exercise documenting the due diligence involved in the secure development of the target contract only, and make no material claims or guarantees concerning the contract's operation post-deployment. The preparers of this report assume no liability for any and all potential consequences of the deployment or use of this contract.

Smart contracts are still a nascent software arena, and their deployment and public offering carries substantial risk. This report makes no claims that its analysis is fully comprehensive, and recommends always seeking multiple opinions and audits.

This report is also not comprehensive in scope, excluding a number of components critical to the correct operation of this system.

The possibility of human error in the manual review process is very real, and we recommend seeking multiple independent opinions on any claims which impact a large quantity of funds.

Findings

Findings presented in this section are issues that can cause the system to fail, malfunction, and/or be exploited, and should be properly addressed.

All findings have a severity level ranging and an execution difficulty level, ranging from low to high, as well as categories in which it fits.

A01: Addresses must be able to receive assets

[Severity: High | Difficulty: Low | Category: Asset security, Best practices]

Description

When purchasing assets in a basket or at the ending of auctions, the contract performs asset transfers to up to seven actors involved in the platform. Three transactions will always happen, which are the payment of fees to the platform, the transfer of the price asset to the seller, and the transfer of the basket assets to the buyer. One set of transactions are optional, which are the payment of royalties to user-specified addresses. Before creating internal transactions to transfer assets to these addresses, there are no checks to verify that the accounts have opted in to the assets that will be sent to them.

If an address opted out of an asset, or never opted in to it at all, the transfer of that ASA will fail.

This is especially dangerous for the normal auction exchange model, as all assets can be locked in the contract. In this specific scenario, after the end of an auction, if one of the inner transactions performed by the contract fails, the buyer cannot claim the auctioned assets. The seller also cannot delete an auction that has a bidder and is past its auction end time. The only way to recover the assets from the contract is to have all accounts opted in to the assets which will be transferred to them before a buyer claim the auctioned assets, which might not be a viable option as, for instance, the owner of one of the royalties addresses specified for that basket might not be aware of the need to opt-in to the asset in question.

Scenarios

- For the basic sale, shopping cart and reverse auction exchange models, a buyer will not be able to purchase the assets in a basket if the fee address, the seller address or one of the royalties addresses are unable to receive the price asset of that contract. The buyer also is not obliged to opt-in to the assets being sold when attempting to purchase the basket, which would imply in failure of the inner transactions transferring assets to the buyer;
- For the normal auction exchange model, using the royalties address to exemplify the point of failure, assets can be locked in the contract if:
  1. user A creates a auction for an asset with asset X as the price asset and B as the royalty address;
  2. address B, for instance, has not opted in to X;
  3. user C successfully bids last and wins the auction;
  4. user C attempts to claim its rightful assets, but fails to do so as this operation triggers the royalty payment transaction to B;
  5. user A cannot delete the auction, as it already has ended. The auctioned assets belonging to A and assets bid belonging to C are now frozen in the contract.

Recommendation

Before performing the asset transfers, the contract must ensure that the destination of the transfer will be able to receive the assets being sent, preventing a transfer from happening if an address has opted out from or never opted in to the asset in question. This guarantees that all users that can receive their rightful assets will be able to receive them.

To address this further, the basket claiming operation could be split into two, where the seller can claim his assets separately from the claim operation of the buyer. This way, the two active actors of this contract can receive their rightful assets without suffering any interference from each other, also preventing that a substantial part of the assets held by the contract stay locked.

Status

This issue has been acknowledged by the client.

A02: Reverse auction price calculation allows for purchase of assets with no payment

[Severity: High | Difficulty: Low | Category: Asset security, Implementation flaws]

Description

The equation used for price calculation in the reverse auction exchange model allows for users to buy the auctioned basket and pay with an asset transfer or payment of amount 0, where the only expense of the buyer would be the fees for the transaction group.

This happens due to a division between integers in the calculation of the basket price using the '/' opcode. From this division, the remainder is discarded as only the quotient is stored in a uint64 variable.

A mathematical model of the price calculation in the reverse auction exchange model can be seen below:

$$Price = PAA \cdot \left(\frac{TTP}{TRI} - \frac{TSS}{TRI} \right)$$

The descriptions of the variables used in the calculation are as follows:

- **PAA: Price Asset Amount**, or the minimum price defined by the seller;
- **TTP: Time to price**, or the time taken before using the base amount as price;
- **TSS: Time since start**, or the time since the beginning of the reverse auction;
- **TRI: Time reduction Interval**, or the interval duration between the reduction of the price.

Even if both divisions have the same quotient and different remainders (which are lost due to rounding), the subtraction returns 0, and, consequently, the multiplication by the price asset amount also returns 0.

Scenario

An example of how this exploit can be achieved uses the following variable values:

- (TRI) Time reduction interval = 4;
- (TSA) Time since start = 10;
- (TTP) Time to price = 11.

With these values, the price equation would equate to $PAA \cdot \left(\frac{11}{4} - \frac{10}{4}\right)$, which can also be written as $PAA \cdot (2.75 - 2.50)$; but the remainders are lost, as the AVM is dealing only with integers, which means that the contract actually computes this: $PAA \cdot (2 - 2)$, returning 0 as the price.

Recommendation

Round up the division of the time to price (TTP) by the time reduction interval (TRI), which prevents the division of the time since start (TSA) by the time reduction interval (TRI) from returning a value equal to the former.

Another possible, less flexible approach would be to ensure that the division of the time to price (TTP) by the time reduction interval (TRI) leaves no remainder.

Status

This issue has been fully addressed.

A03: The funds submitted by a previous bidder might not be refunded to their rightful owner

[Severity: Medium | Difficulty: Low | Category: Asset security]

Description

If a user bids in an auction and opts out from the asset that has been used to bid, it will not be possible to issue a refund if his bid is surpassed. The contract verifies if the address is able to receive the price asset before attempting to pay back the previous bidder and, if the address has opted out of the asset, the refund is skipped. While this is done to prevent the AVM from panicking, as the inner transaction for the refund would fail, implying that any other attempt to bid would fail as well, the asset that has been used to bid will stay locked within the contract until the last bidder claims the basket assets.

During the basket claim, the price asset is sent to the seller on a closing operation, which implies that all assets from users that should have received their refund will be sent to the seller.

Scenario

This issue can be triggered in scenarios with similar conditions to the one described below:

1. User A creates a normal auction of asset X, with Y as the price asset;
2. User B bids in the created auction with asset Y;
3. User B opts-out of asset Y;
4. User C bids in the same auction, and the user B does not receive a refund, as a transfer to its account would imply in failure of the contract execution;
5. User C claims the auctioned assets, triggering the transfer of all Y assets in the contract to user A.

Recommendation

Given that the expected behavior of the contract is to provide the highest bidder with the auctioned assets while providing the highest bid to the seller, it is not expected that the seller is going to receive more assets than what the winner of the auction paid. Furthermore, given that the claim of the auctioned assets by the winner implies in the

deletion of the contract, an approach to handling this issue is to transfer the non-reimbursed assets to another address (such as the fee address, given that it is able to receive the ASA in question) in order to be claimed through further interaction in the EXA Finance platform.

Still, similarly to the recommendation given for finding A01, fully addressing the issue would require enabling the previous bidders to claim their refunds at their own will, if the refund cannot be performed during the processing of the surpassing bid.

Status

This issue has been fully addressed.

A04: It is possible to fund baskets with no assets

[Severity: Medium | Difficulty: Low | Category: Asset security, Input Validation]

Description

There are no checks that prevent a seller from transferring 0 amounts of an asset when funding the basket. Malicious users can transfer zero amount of assets that they do not have. This can be threatening especially when dealing with a non-fractionary *non-fungible asset* (NFA), where there is only one token of its kind.

Contrary to fungible assets, it is easier for a malicious user to trick others into believing that the specific NFA is being sold, given that the basket has been already funded.

Scenario

This could be exploited in the following manner:

1. User A creates a basket and specifies its price;
2. User A funds the basket with 0 amounts of that asset. The basket is now considered funded;
3. User B interacts with the basket, purchasing it and expecting to receive the auctioned assets;
4. The contract transfer 0 amounts of the auctioned assets to the buyer, sending 0 amounts of fees and royalties to the respective addresses;
5. The seller receives the assets paid by the buyer.

Recommendation

This issue can be fixed by ensuring that the payment or asset transfer transactions responsible for funding the basket must have their amount greater than 0.

Status

This issue has been fully addressed.

A05: Assets stay locked in the contract if auction winner refuses or becomes unable to claim

[Severity: Medium | Difficulty: Low | Category: Asset security]

Description

In the normal auction exchange model, once the auction ends, the buyer has to claim the auctioned assets in order to transfer the fees to the fee address and royalties addresses, as well as the payment to the seller. Since the buyer is the only one that can trigger the claim operation, no other actor involved will be able to initiate the process of asset transfer to their respective owners. In case the auction winner refuses or becomes unable to initiate the claiming process, the assets remain in the contract.

The seller also is incapable of deleting the auction, given that, if the auction end time has been reached and there is a winning bid, only the address of the winning bidder is able to interact successfully with the contract.

Scenario

This issue can be triggered in a scenario where the following conditions are met:

1. User A initiates the basket in the normal auction exchange model;
2. User B bids last and wins the auction;
3. User B does not trigger the claim operation of the assets;
4. All of the basket items and the winning bid assets remain in the contract until the winning bidder can trigger the claim operation.

Recommendation

Enable the seller to trigger the claim operation. Since the seller and the buyer are the owners of all assets in the contracts, if both do not want to or are unable to trigger the claim operation, it is reasonable to assume that all assets should remain in the contract until one of the active actors in the scenario becomes capable of doing so.

Status

This issue has been addressed.

A06: It is possible to interact with non-funded baskets

[Severity: Medium | Difficulty: Low | Category: Asset security]

Description

The contract logic allows for the interaction of users with non-funded baskets. When buyers are trying to purchase the basket by sending an application call with "DeleteApplication" on its "OnCompletion" field, there are no checks to verify if the assets of the baskets have been supplied.

Upon an interaction of a buyer with the contract, the transaction will fail when comparing the zero address in bytes format with the value of a non-populated global which is supposed to indicate the address of the auctioned asset, triggering a panic behavior in the AVM.

Scenario

1. User A creates a basket in the basic sale or reverse auction exchange model and sets its price to a fixed amount of assets;
2. User A optionally does not triggers the funding operation in the contract, or fail to do so successfully;
3. User B interacts with the contract as a buyer, but the application call fails as the AVM panics.

Recommendation

Verify the state of the contract before allowing users to interact with it as buyers. If the contract state indicates that it has not been funded, fail the execution of the contract.

Status

This issue has been fully addressed.

A07: Fees accumulate in the contract if the royalties address is missing or no royalties fee are specified

[Severity: Low | Difficulty: Medium | Category: Asset security, Optimization of fees]

Description

In the shopping cart exchange model, the contract stays with the algos used for fee payments of the transfers and for holding assets even if there are no more assets to hold. The seller has to delete the contract in order to receive their algos back. Aggravating this issue, buyers have to send algos to pay for all the four transactions performed by the contract, and this amount is fixed for purchase of shopping carts. If the royalties address has not been specified or the royalties per ten thousand is zero, the royalties payment inner transactions will not happen. This means that algos will be accumulated in the contract and will be sent to the seller in the closing operation.

Furthermore, a malicious user that has knowledge of this issue can try to create baskets of low value assets while limiting the amount bought per transaction to a low amount to try and accumulate algos in the contract for his withdrawal, funding the user to repeatedly perform the operation with a higher amount of assets.

Scenario

An example of how the issue can be exploited:

1. User A creates a basket following a shopping cart exchange model without specifying a royalties address and X amount of an asset for sale;
2. The basket created by user A is configured to limit the amount bought per transaction to up to $\frac{X}{1000}$;
3. Buyers continually buy items from the created basket, always sending 4000 microAlgos to pay for the fees of the transactions per execution of a purchase. Only 3000 microAlgos are being used, while 1000 are accumulated to the contract;
4. Once all the X units of the sold asset have been depleted, user A deletes the basket, receiving all the accumulated algos. In this case, since at least 1000 transactions have been used to purchase all assets in the basket, user A will be collecting at least 1000000 microAlgos from accumulated fees.

Recommendation

If no royalties address is specified or if the royalties per ten thousand units is zero, require only the amount of algos to perform the three mandatory transactions.

Also, the seller should be notified that the amount of algos sent when funding the basket should be claimed back once all the items of the shopping cart basket have been sold. This avoids the possibility of keeping algos in contracts with no further use.

Status

This issue has been fully addressed.

A08: Bid price malfunction due to precision loss

[Severity: Low | Difficulty: Medium | Category: Implementation flaw]

Description

There is a possibility of loss due to rounding when calculating the minimum amount for bids, which is calculated in the excerpt of code shown below:

Minimum bid value calculation

```
1191   // if there is no bid, use the initial price, otherwise use
       the previous bid * 1.01
1192   load 6
1193   load 9
1194   dup
1195   int 100
1196   /
1197   +
1198   load 20
1199   select
1200   store 22
```

Given that the division performed in line 1196 discards the remainder of the operation, there will be loss in the precision of the values being handled.

If the last bid or the initial bid value relies on its two lowest decimal digits for the precise calculation of the minimum bid value, the check that evaluates if the attempted bid is at least 1% greater than the minimum bid price can be bypassed.

Recommendation

When performing the division of the last bid or the initial bid price by 100, round up the result to avoid the impacts of precision loss.

Status

This issue has been fully addressed.

A09: Royalties and fees loss due to rounding

[Severity: Low | Difficulty: Medium | Category: Implementation flaw]

Description

Both the fee and royalty amount calculations are subject to loss due to rounding. The calculations are performed as follow:

Sample royalties amount calculation

```
431   load 91
432   load 31
433   *
434   int 10000
435   /
436   store 25
```

Sample fee calculation

```
1191   load 33
       load 101
1192   *
1193   int 100
1194   /
1195   store 25
```

Similarly to the finding A08, the operations performed in lines 435 and 1194 do not take into account possible remainders in the division, which, for baskets of prices low enough, might imply in the complete loss of the fees and royalties, implying in 0 valued transfers.

Baskets where the fee percentage multiplied by the amount do not fit in a two digit value will issue payments or asset transfers of value greater than zero for the fee address, while payments or asset transfers for the royalties addresses will only be of a non-zero amount if the basket price multiplied by the royalties per ten thousand has five or more digits.

Recommendation

If it is desired to receive fees and royalties in such scenarios, a minimum fee value should be created or the calculation of the fees and royalties amount should be rounded up to the nearest integer.

Status

This issue has been acknowledged by the client.

A10: No start time and end time verification

[Severity: Low | Difficulty: Low | Category: Input Validation]

Description

There are no checks to verify if the start time, which is given in the arguments of the application call transaction, is smaller than the end time. This prevents users from bidding in an auction with such settings.

In case the bidder attempts to place a bid before the starting time of an auction, his request to bid will be denied. This prevents users from interacting with the contract before the beginning of the sale/auction, and is the intended behavior of the contract.

Even so, in case the bidder attempts to place a bid after the starting time, the application call still will fail, as the auction has already finished, according to its own settings.

In other words, this issue can be exploited by malicious users to prevent any legitimate user from buying an item that is for sale, or inducing users into interacting with an auction that has ended before it started, aiming at impacting the reputation of the platform.

Recommendation

During the creation of a basket, there must be a verification to ensure that the specified starting time is smaller than the end time.

Status

This issue has been fully addressed.

A11: No checks on the price reduction interval

[Severity: Low | Difficulty: Low | Category: Implementation flaw, Input validation]

Description

As shown in the finding A02, the time reduction interval (TRI - scratch space 37) is used as part of the calculation of the basket price for the reverse auction exchange model. Its value is used to divide the time to price (TTP) and time since start (TSS) in the price calculation equation. This implementation is shown below:

```
      Price calculation in the reverse auction exchange model
1590  load 36
1591  load 37
1592  / // nb of intervals in the time to price
1593  load 23
1594  load 37
1595  / // nb of intervals passed
1596  - // nb of intervals not passed
1597  load 6
1598  * // price at current interval
1599  store 33
```

There are no checks to ensure that the price reduction interval is greater than 0 in the reverse auction exchange model. In case such a value is used, the calculation of the basket price would always force panic behavior in the AVM at every attempt to purchase the basket in this exchange model. The root of this issue lies in the operation written in line 1592 of the above code excerpt, where prior verifications to prevent zero-valued dividends must always be performed.

Similarly to the finding A10, this issue can be exploited by users aiming to perform griefing attacks.

Recommendation

From the parameters sent by the creator of the reverse auction, always check if the price reduction interval is greater than zero.

Status

This issue has been fully addressed.

A12: No checks on the maximum amount purchased per transaction in shopping carts

[Severity: Low | Difficulty: Medium | Category: Input validation]

Description

There are no checks to ensure that the maximum number of assets purchased in a transaction using the shopping cart exchange model, defined in the scratch space 35, is greater than 0. A malicious user can trick other users into trying to buy something that is available, but no amount of it can be bought.

This exploit does not provide any profit to the attacker, but might impact the experience of honest users that, besides not obtaining the desired asset, will also have small amounts of microAlgo losses in fees, since attempts to purchase any amount greater than 0 assets from the shopping cart will result in failure of the application call.

Recommendation

When creating a basket, if it uses the shopping cart exchange model, check if the maximum number of assets purchased per transaction is greater than 0.

Status

This issue has been fully addressed.

A13: No checks on specific Algorand fields and properties

[Severity: Low | Difficulty: High | Category: Blockchain related particularities, Best practices]

Description

Considering the available transaction fields and their functionalities, it is recommended to verify possible misconfigured fields, either by malformation of transactions or interference by a malicious third party, that may have a significant impact on the assets held by an account.

Also, ASAs have, by default, mutable properties that are capable of interfering with the proper execution of a smart contract. For the case of the smart contract in question, the functionalities of asset freezing and reclaiming of assets, also referred to as clawing back assets, might impose a problem for the successful execution of the approval program of the analyzed contract.

Considering the aforementioned, the following findings were identified:

- The RekeyTo field is not verified for any transaction handled by the contract;
- The AssetCloseTo and CloseRemainderTo fields are not verified in asset transfers and payments that are not opt-in transactions;
- There are no verifications to evaluate if assets were frozen before attempting to handle them;
- There are no verifications to evaluate if assets have been clawed back before attempting to handle them.

Recommendation

For the transaction fields mentioned above, it should be ensured that they have not been populated before being handled by the contract. For the mutable properties of the assets, it is recommended that the availability of assets should be verified before performing operations over them, preventing unrelated assets from being freezed in the contract as well.

Status

With regards to the verification of transaction fields, the issues have been addressed by the client, while the verification of the availability of the assets has been acknowledged.

A14: No boundaries in the royalty fees and platform fees leading to possible overflows and underflows

[Severity: Low | Difficulty: Medium | Category: Input Validation]

Description

There are no checks that limit the amount of royalties and fees to be paid. Royalties, which are specified as amounts per ten thousand, can exceed 10000, while the fee percentage can exceed 100.

The calculations of the fees and royalties amount are shown in finding A09, and the operation which might cause panic behaviors in the AVM can be seen in lines 433 and 1192.

Regardless of their values, when adding the fees amount with the royalties amount, the value should not exceed 100 percent of the price being paid for the basket, as any purchase transaction would fail due to lack of funds in the contract.

Scenario

The issue can be triggered in scenarios such as the following:

1. User A creates a basket of assets with a price X, 10 percent of fees 950 per ten thousand of royalties;
2. User B attempts to purchase the basket, sending the payment valued in X amounts of the desired assets;
3. The smart contract fails to execute the purchase, it would be required $X \cdot 1.05$ to fulfill the requirements to perform all necessary inner transactions;
4. Even if user B attempts to pay $X \cdot 1.05$ for the basket, the purchase still would fail as the value differs from the price amount.

Recommendation

Limit the sum of royalties and fees values paid to a number below the price amount.

Status

This issue has been fully addressed.

A15: Overflow possibility in reverse auctions

[Severity: Low | Difficulty: Medium | Category: Implementation flaw]

Description

In a reverse auction, if a user specifies a time to price that is, in orders of magnitude, greater than the time reduction interval, it is possible that no user can purchase the basket for some period of time.

The calculation of the basket price in the reverse auction exchange model is shown in finding A02.

Right after the start of a reverse auction configured with such settings, the time since the start divided by the time reduction interval will return a negligible value compared to the division of the time to price by the time reduction interval, which means that the price asset amount will be multiplied by an enormous number. In this scenario, it is possible to obtain a value greater than the uint64 range, which causes the transaction to fail. The basket can still be deleted by the seller.

Recommendation

The overflow possibility when multiplying the basket price with the quotient of the time to price (TTP) by the time reduction interval (TRI) must be verified in the contract initiation.

Status

This issue has been fully addressed.

A16: Overflow possibility in shopping cart purchase when multiplying the base price by the amount bought

[Severity: Low | Difficulty: Medium | Category: Implementation flaw]

Description

There is a possibility for overflow when purchasing from shopping carts. There are no checks to verify if there can be an overflow when multiplying the parameter given by the buyer (amount bought) with the price asset amount. The overflow happens during the processing of a purchase, specifically in the calculation of the total price paid by the buyer, as seen in the code excerpt below:

Price calculation in the shopping cart exchange model

```
1072   load 34
1073   load 6
1074   *
1075   store 33
```

Scenario

The issue can be triggered in the following steps:

1. User A creates a basket in the shopping cart exchange model containing asset X, and the price asset is Y;
2. The price of asset Y is significantly smaller than the asset X, resulting in a large integer I as the price of every individual asset in the basket;
3. User B tries to purchase $\left(\frac{2^{64}-1}{I} - 1\right)$ of the asset X from the basket, but the value surpasses the maximum uint64 value, forcing a panic behavior in the AVM.

Recommendation

The maximum amount purchased per transaction must be smaller than the value which, if multiplied by the basket price, is greater than the maximum uint64 value.

Status

This issue has been fully addressed.

Informative Findings

Findings presented in this section do not necessarily represent any flaw in the code itself. However, they indicate areas where the code may need some external support or areas where the code deviates from best practices. We have also included information on potential code size reductions and remarks on the operational perspective of the contract.

B01: Effects of the Algorand block generation time

[Severity: Low | Difficulty: - | Category: Implementation flaw]

Description

Since it takes roughly 4.5 seconds to generate a block in the Algorand blockchain, depending on the timing when the transaction is submitted, it is possible to bid/purchase assets up to 4.5 seconds after the end of the auction/sale. The same works on the protection time window in auctions, where users could bid without increasing the time to finish, and any other operations that involve operations related to time.

Status

The issue has been acknowledged by the client.

B02: Unnecessary conditional

[Severity: - | Difficulty: - | Category: Execution cost optimization]

Description

Given the following code excerpt:

Unnecessary conditional

```
1526  txn GroupIndex
1528  int 0
1529  ==
1530  assert
1531
1532  global GroupSize
1533  int 1
1534  ==
1535  assert
```

There is an unnecessary check starting at line 1526. First, there is a check to evaluate if the index of the transaction in the group is 0, and right after it there is another check to see if the transaction group has only 1 transaction. The first check can be removed, as it is less strict than the second.

Recommendation

The conditional should be removed.

Status

The issue has been fully addressed by the client.

B03: Transaction groups can include opt-ins unrelated to the platform

[Severity: Low | Difficulty: - | Category: Blockchain related particularities, Best practices]

Description

The function fn_check_is_optin_tx only checks if the transaction is an optin. In the basket initiation, for instance, the application call can be followed by multiple opt-ins not related to the platform itself.

There also are no checks to verify if a seller is opting in to the price asset or if a buyer is opting in to all the assets in the basket.

Recommendation

It should be checked whether the user should perform a specific opt-in transaction and ensure that, if needed, it is one of the transactions in the transaction group.

Status

The issue has been acknowledged by the client.

B04: Fee pooling functionality is not used

[Severity: Low | Difficulty: - | Category: Fees optimization]

Description

In order to avoid paying fees for the payment transactions that funds the contract with enough algos for paying the fees for transferring assets to seller/buyer/fees/royalties addresses, the contract can make use of the fee pooling functionality to send the fees in another mandatory transaction of the group, instead of having an exclusive transaction responsible for this.

This reduces the size of some of the platform's transaction groups, making the interaction with the contracts cheaper.

Status

The issue has been acknowledged by the client.

B05: Actions in the contract branch based on operation before branching on product type

[Severity: Low | Difficulty: - | Category: Best practices]

Description

It is recommended to branch the actions in the contract based on the product type, and not based on the operation type. Not doing so makes for more convoluted contracts.

In the case of the EXA Finance contracts, the first verification performed is based on the “OnCompletion” field of the application call transaction, and after redirecting the action flow to the desired section of the code, the checks on the product type are performed.

The analyzed approval_program does not present any challenges in regards to its readability, but once new features are added, it might become harder to read and understand the contract.

Status

The issue has been acknowledged by the client.

B06: Unused values remain in the stack

[Severity: Low | Difficulty: - | Category: Best practices]

Description

When checking the return of the asset_holding_get opcode, executed in order to verify if previous bidders can receive a refund once their bid has been surpassed, in case the previous bidder opts out from the price asset, a uint64 zero value remains in the stack for all the remaining execution of the contract for that application call. The value does not affect the flow of execution of the contract, but is always recommended to remove unnecessary values from the stack.

A value can be left in the stack on the following operation:

Values left in stack

```
1275   // check if last bidder still has the asset opted-in,
       otherwise skip refund
1276   load 8
1277   load 5
1278   asset_holding_get AssetBalance
1279   bz handle_regular_auction_bid__skip_refund
1280   pop
```

In case a jump happens to the tag “`handle_regular_auction_bid__skip_refund`”, in line 1279, the remaining value in the stack is not discarded.

Status

The issue has been fully addressed.