CERTIK

Security Assessment

WingRiders

Mar 15th, 2022

Table of Contents

Summary

This report has been prepared for WingRiders to discover issues and vulnerabilities in the source code of the WingRiders project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Formal Verification 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 several findings, but only at an "informational" level of severity. We recommend addressing these findings to ensure a high level of security standards and industry practices.

In addition to the manual audit, CertiK has formally verified several of the core functions. The verification proves that the core business logic functions satisfy important properties and invariants, which builds confidence that the program logic is correct.

Overview

Project Summary

Project Name	WingRiders
Platform	Cardano
Language	Plutus
Codebase	https://github.com/WingRiders/core-contracts/tree/audit
Commit	ef208a1ff2dd4731369ee341ccc8a3e1bbb6013c

Audit Summary

Delivery Date	Mar 15, 2022 UTC
Audit Methodology	Manual Review, Formal Verification

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Mitigated	Partially Resolved	Resolved
Critical	0	0	0	0	0	0	0
Major	0	0	0	0	0	0	0
Medium	0	0	0	0	0	0	0
Minor	0	0	0	0	0	0	0
Informational	5	0	0	1	0	0	4
Discussion	0	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
PSC	src/DEX/OnChain/Core/PoolState.hs	cb1282a48f278e285f017d8dca48b3fe3b3cd72e9f7bdf7f4d16bb971af743e3
PPC	src/DEX/OnChain/Common/ProtocolParams.hs	41109958c8a349921cce7738059a10adaa791d3df8ddffa48475fac779cc32e5
TOC	src/DEX/OnChain/Treasury	
UUO	src/DEX/OnChain/Util/Unsafe.hs	24ac46707c0d3d23a4b5ce02c0b058db093765dd02d6765cf8050f3bde492dc1
COC	src/DEX/OnChain/Common	
AMM	src/DEX/OnChain/Core/AMM.hs	d1bd3e6a47c20ef0af800f501465034400247fd9e7ef3ab8cfa16a1cf5baee5f
UOC	src/DEX/OnChain/Util	
FAC	src/DEX/OnChain/Core/FlatAssets.hs	d6d31c3bebf20bebc4e32a565f843ca8f16913b3f97e02b7c09aac86e19d10b0
PSC	src/DEX/OnChain/Core/PoolState.hs	ad599a25a4a8a2d2f6c29e85fe7fd04d9b7b0e41159dbb0505fda7cb45447a9a
AMM	src/DEX/OnChain/Core/AMM.hs	d1bd3e6a47c20ef0af800f501465034400247fd9e7ef3ab8cfa16a1cf5baee5f
FAC	src/DEX/OnChain/Core/FlatAssets.hs	88fc25d808654d49b6c0e269aa2dea2dd2efff4c63182607c362679a8f776b79
RCO	src/DEX/OnChain/Core/Request.hs	50c9d92c886dca5b50b4d4186a409ecbc58f269a0a878512200d4ff1ab3e326d
RDU	src/DEX/OnChain/Util/RawData.hs	8766bc99c5b51a8c8116b5d9185b5a3a7b7ea5b865a8d9de23fd55bae871c798
ROC	src/DEX/OnChain/Request.hs	0d352469cf99d8d7bd8e18116f4fe803b2fc0547279c8ae217edc869599d0ef4
PPC	src/DEX/OnChain/Common/ProtocolParams.hs	41109958c8a349921cce7738059a10adaa791d3df8ddffa48475fac779cc32e5
MOC	src/DEX/OnChain/Mint.hs	ea661df0fa509772f5174f1ec7136fbd8e0912b2960c384ad651250df3c7be57

ID	File	SHA256 Checksum
RDU	src/DEX/OnChain/Util/RawData.hs	2e56d9a02dab559644271474fd9ffc94128f8ef0c212fbd796db0cbc96d06947
COC	src/DEX/OnChain/Common	
COD	src/DEX/OnChain/Core	
TCC	src/DEX/OnChain/Common/Types.hs	53b6d61c53732f1e217da74e950f740d3ac68dd9755d9ef1019a4789dde5d519
THT	src/DEX/OnChain/Treasury/TreasuryHolder.hs	06bc81a3df30f183e8ef2a825a5b73d476b9d7b6815ad81873dbae3526228269
PTC	src/DEX/OnChain/Common/Types/Pool.hs	28e82c4e1f42729a45fc1a5f1a852f749c8ff50aea796569364e658068a47866
TCC	src/DEX/OnChain/Common/Types.hs	53b6d61c53732f1e217da74e950f740d3ac68dd9755d9ef1019a4789dde5d519
FOC	src/DEX/OnChain/Factory.hs	f22936e135156dadbe287aa52de5048daa8212fe514376564caf5210e9bd3a1f
UUO	src/DEX/OnChain/Util/Unsafe.hs	24ac46707c0d3d23a4b5ce02c0b058db093765dd02d6765cf8050f3bde492dc1
CCO	src/DEX/OnChain/Common/Constants.hs	56b233207f62cd3a662f24bfd81c9103ac773ec92d4b52b73cad3422ab7a24a1
MTC	src/DEX/OnChain/Common/Types/Misc.hs	ac76f8fc2a0239c7d6bf4df8d48e644f000c9d8a10018f343a1b27ab96ec2ac0
RCO	src/DEX/OnChain/Core/Request.hs	50c9d92c886dca5b50b4d4186a409ecbc58f269a0a878512200d4ff1ab3e326d
FSP	src/Helper/FixedSupplyPolicy.hs	affe27789566b64ee3b1e0ae161fd0c21d53379eeeca8fe26d55b78d1d83befe
PSC	src/DEX/OnChain/Core/PoolState.hs	ad599a25a4a8a2d2f6c29e85fe7fd04d9b7b0e41159dbb0505fda7cb45447a9a
DUO	src/DEX/OnChain/Util/Debug.hs	e484ca05e01605dcc64fe3c0b94b0e4401c582e7398daec9a85c66fba6fc1da9
TOC	src/DEX/OnChain/Treasury.hs	3a591f9e256e8d53de3a53e52ebd3fc6fd362dd132be222265b143c1fa8b61d9
BTC	src/DEX/OnChain/Common/Types/Base.hs	63023c059023cfa7f4cc38863ac5ee71bee14eda6300de884e495fef0fabf16b

ID	File	SHA256 Checksum
VTO	src/DEX/OnChain/Treasury/Voting.hs	03399b23b6d8ec24ccd39db753b81e0870094e0beb938aa97314aefed19cc507
AMM	src/DEX/OnChain/Core/AMM.hs	d1bd3e6a47c20ef0af800f501465034400247fd9e7ef3ab8cfa16a1cf5baee5f
RDU	src/DEX/OnChain/Util/RawData.hs	2e56d9a02dab559644271474fd9ffc94128f8ef0c212fbd796db0cbc96d06947
CCO	src/DEX/OnChain/Common/Constants.hs	c110da77f5190c21a0cec4f1b08b242568c64134904a398e8824bb540beb0124
OCD	src/DEX/OnChain	
UOC	src/DEX/OnChain/Util	
COE	src/DEX/OnChain/Core.hs	269f41e604512dc30609e4f341a753131739df53226100d08a39761aa5ce2345
PTC	src/DEX/OnChain/Common/Types/Pool.hs	28e82c4e1f42729a45fc1a5f1a852f749c8ff50aea796569364e658068a47866
MOC	src/DEX/OnChain/Mint.hs	181a766bfe9d370a14ddc1b49b1436e5901e157ab2d720a5dabf011cb44abe36
TOC	src/DEX/OnChain/Treasury	
PCO	src/DEX/OnChain/Core/Pool.hs	dac3605dd5b7e7123d01711af6597be2ae43b972363e1e171f86ea4bb79b1345
PCO	src/DEX/OnChain/Core/Pool.hs	9491b4a92feaee21bdcdc3cee76e6c79a0bc20b94d66951c8e542f581b3541c4
OCD	src/DEX/OnChain	
TCO	src/DEX/OnChain/Common/Types	
ROC	src/DEX/OnChain/Request.hs	0d352469cf99d8d7bd8e18116f4fe803b2fc0547279c8ae217edc869599d0ef4
DUO	src/DEX/OnChain/Util/Debug.hs	11527b7b7bf6e32704e4e4d6d7ce52b0144b6128521ca17979a033c9f06b8107
RMS	src/DEX/OnChain/Staking/RewardMint.hs	8820e52af964bcad360b4210f565c23b5ef9f7da7ba1f32ca5908d7b1ab6d5e4
OCD	src/DEX/OnChain	

ID	File	SHA256 Checksum
PPC	src/DEX/OnChain/Common/ProtocolParams.hs	91c4942ed5fd734e3a59bc4dfbaa5146b486a829fbf36bcc2133a1a518c1a6fa
PCO	src/DEX/OnChain/Core/Pool.hs	9491b4a92feaee21bdcdc3cee76e6c79a0bc20b94d66951c8e542f581b3541c4
COD	src/DEX/OnChain/Core	
MUO	src/DEX/OnChain/Util/Misc.hs	af4039a6366810b09778a113844b60cce745e2ec358d5d32be1dd2544ccbb0aa
TCO	src/DEX/OnChain/Common/Types	
MOC	src/DEX/OnChain/Mint.hs	181a766bfe9d370a14ddc1b49b1436e5901e157ab2d720a5dabf011cb44abe36
MUO	src/DEX/OnChain/Util/Misc.hs	68e5985d00a75eae41b7b9b9c3346c74ad5c97628000b87aee1a8d0b9c61804a
RTC	src/DEX/OnChain/Common/Types/Request.hs	a2a6a9925474e27add28b0f16e3e96d8979fd3ba3c6c531268468147080 1f3bd
FTC	src/DEX/OnChain/Common/Types/Factory.hs	d35898d8b72580d93fe2e3203102dd4d219c89edd378fbabc7e1585d601 48dea
TCO	src/DEX/OnChain/Common/Types.hs	3a0c4be9aff14911dd43acb7d2d28bdb551b6f6ac8c4ec01b07e196b44daecac
FAC	src/DEX/OnChain/Core/FlatAssets.hs	88fc25d808654d49b6c0e269aa2dea2dd2efff4c63182607c362679a8f776b79
RMS	src/DEX/OnChain/Staking/RewardMint.hs	8820e52af964bcad360b4210f565c23b5ef9f7da7ba1f32ca5908d7b1ab6d5e4
MUO	src/DEX/OnChain/Util/Misc.hs	a6060c9794535ba6ec4d7d4f68e2908c149d245191597250bd343be556b208a9
COD	src/DEX/OnChain/Core	
POC	src/DEX/OnChain/Pool.hs	ca7a6e0a57197a961f43d64c62164c0ec6de9d4f741728b5a7327386ff3c9fa2
MTC	src/DEX/OnChain/Common/Types/Misc.hs	ac76f8fc2a0239c7d6bf4df8d48e644f000c9d8a10018f343a1b27ab96ec2ac0
FOC	src/DEX/OnChain/Factory.hs	5a1375dfbe7f564393a5e32f1327f1b9c23d7cae7fa4dca4faeaacab0d4e3b9e

ID	File	SHA256 Checksum
UOC	src/DEX/OnChain/Util	
SOC	src/DEX/OnChain/Staking	
POC	src/DEX/OnChain/Pool.hs	ca7a6e0a57197a961f43d64c62164c0ec6de9d4f741728b5a7327386ff3c9fa2
FOC	src/DEX/OnChain/Factory.hs	5a1375dfbe7f564393a5e32f1327f1b9c23d7cae7fa4dca4faeaacab0d4e3b9e
RCO	src/DEX/OnChain/Core/Request.hs	2cb6952ca39b5170499d9d95e7177b25ee64afe569f0d09def5065158e0beb98
FTC	src/DEX/OnChain/Common/Types/Factory.hs	d35898d8b72580d93fe2e3203102dd4d219c89edd378fbabc7e1585d60148dea
RTC	src/DEX/OnChain/Common/Types/Request.hs	a2a6a9925474e27add28b0f16e3e96d8979fd3ba3c6c5312684681470801f3bd
CCO	src/DEX/OnChain/Common/Constants.hs	56b233207f62cd3a662f24bfd81c9103ac773ec92d4b52b73cad3422ab7a24a1
VTO	src/DEX/OnChain/Treasury/Voting.hs	03399b23b6d8ec24ccd39db753b81e0870094e0beb938aa97314aefed19cc507
DUO	src/DEX/OnChain/Util/Debug.hs	11527b7b7bf6e32704e4e4d6d7ce52b0144b6128521ca17979a033c9f06b8107
BTC	src/DEX/OnChain/Common/Types/Base.hs	63023c059023cfa7f4cc38863ac5ee71bee14eda6300de884e495fef0fabf16b
SOC	src/DEX/OnChain/Staking	
COC	src/DEX/OnChain/Common	
THT	src/DEX/OnChain/Treasury/TreasuryHolder.hs	06bc81a3df30f183e8ef2a825a5b73d476b9d7b6815ad81873dbae3526228269

Review Notes

Introduction

DEXs (decentralized exchanges) facilitate exchange of financial assets without the need for a third party to oversee such transactions. AMMs (automated market makers) are one of the most popular DEXs used on blockchains. AMMs achieve decentralization by maintaining liquidity pools for assets in smart contracts, and incentivizing various users (liquidity providers) to add liquidity to these reserves by interacting with these smart contracts. Users who want to exchange an asset X for another asset Y, deposit a certain amount of X, and get Y in return. The price for this exchange is calculated using a mathematical formula based on the current reserves of the assets in the smart contract. WingRiders uses the “constant product” formula, which sets the price so that the product of the two reserves remains (approximately) constant.

The WingRiders AMM

WingRiders adapts the AMM concept to make it more suitable for the Cardano chain.

In order to adapt the principle of the AMM to the eUTxO model, liquidity pools are represented by UTxOs, where there is a unique UTxO corresponding to pool reserves for each token pair. Any transaction involving a particular pool must contain this UTxO as input. Since there is only one UTxO for each pool, the WingRiders DEX protocol introduces several new components to enable processing multiple exchange transactions or liquidity transactions within one block:

- **Agents:** some wallets can be allowed by a specific Agent Token to process requests (see below) that constitute a batch transaction(see below). Agents are incentivized by the fees that the protocol charges for creating a request to the users.
- **Requests:** The protocol supports 6 types of requests, most of which are first validated by an agent, before being included in a batch transaction. Users who submit requests for interaction with the pool can reclaim those requests if they wish to do so even before the requests have expired. The requests currently supported are:
    1. **Swap Requests:** Requests created by users who want to exchange one asset for another. The input for such a request should be the amount of tokens to be swapped in for and ADA, and the output would be a request token and any change the user may receive. The user receives the other asset once this request is processed by a batch transaction.
    2. **Add Liquidity Requests:** These are created by users who want to add liquidity to a given pool. In this case the input should contain both kinds of tokens that constitute the pool and ADA, and the output would again be a request token and any change the liquidity provider

might receive. The liquidity provider receives their share tokens once this request is processed by a batch transaction.

3. **Remove Liquidity Requests:** These are created by users who want to withdraw liquidity from a given pool. In this case the input should contain the share tokens for the pool that the user has and ADA. The output will be the request token and any change the user may receive. Once again, the user receives the appropriate amount of liquidity pool tokens in exchange for their share tokens once this request is applied by a batch transaction.
4. **Extract Treasury Requests:** This request drains the treasury funds from the liquidity pool and adds them to the interim treasury script address. The input for such a request should be ADA to cover transaction fees and the output would be a request token and any change the user may receive. The treasury funds are directed to the provided interim treasury address once the request is processed by a batch transaction.
5. **Add Staking Rewards Requests:** Unlike the other requests, these requests can only be submitted by a Staking Agent. Hence the input should contain a reward token specific to the liquidity pool, the ADA they want to donate to this pool and the output would be a request token and any change they might receive. The agent receives the reward token back once this request is processed by a batch transaction. (A Staking Agent is a wallet which contains a token identifying it as a staking agent. Staking Agents are not the same as the agents that process requests for batch transactions. They are governed by a DAO.)

In addition to the Agent-mediated requests, any user can do an **Emergency Withdrawal**. These deal with a situation when the Agents are not processing requests, e.g. because they were stopped to deal with a security problem. Users can then directly withdraw their liquidity, and the transaction is accepted if the pool has been idle for a certain amount of time. The inputs in this case are the pool UTxO and the liquidity provider's UTxO containing the amount of share tokens worth of liquidity they want to withdraw. The output would be the pool UTxO with updated balances, and the withdrawn liquidity.

Bulk/Batch Transactions: To process multiple transactions with a single liquidity pool efficiently and bypass the concurrency issue, any exchange with a given liquidity pool (except emergency withdrawals) is first submitted as a request transaction to the chain. An agent collects the requests corresponding to the same liquidity pool, which are then submitted as inputs of a batch transaction in the FIFO order.

Factory: The Factory contract is responsible for minting unique UTxOs corresponding to each liquidity pool. Any user can interact with the factory to create a pool. The factory assigns a hash to each liquidity pool, which is constructed by a nested application of the SHA256 hash function and concatenation of the hashes for the two tokens that form a part of the liquidity pool. In order to ensure that duplicate liquidity pools are not created, the factory tracks the hash intervals which are unused. These are open intervals that are a part of the factory UTxO. An interval (a, b) being a part of a factory UTxO indicates that the entire range in the interval constitutes unused hashes. Every time a user wants to create a new liquidity pool, a

hash for this pool is computed (based on the tokens involved), and there is a check that it belongs to one of the available intervals. If it passes this check, the factory UTxO is split further into two intervals at the hash corresponding to the new pool. Then a pool is created by the factory with different parameters governing the pool. A set of Share Tokens which represent the shares to be allocated to liquidity providers are also minted at the time of pool creation. This represents the maximum shares that can be allocated for each liquidity pool in the protocol.

Treasury: A small portion of the trading fees is sent to a DAO representing the project. To avoid the overhead of many small payments this is not transferred for each bulk transaction. Instead, a portion of the funds in the contract are marked as part of the treasury, which is conceptually distinct from the token reserves, and for efficiency agents should submit at most a single request to withdraw treasury funds in each batch. Collection of these funds is through the extract treasury request which is processed by a batching agent as part of the requests that constitute the batch transaction.

Audit Scope and limitations

We audit the source files that are used to build the on-chain contract code. Specifically that means the following files:

- src/Helper/FixedSupplyPolicy.hs
- src/DEX/OnChain/Common/Constants.hs
- src/DEX/OnChain/Common/ProtocolParams.hs
- src/DEX/OnChain/Common/Types.hs
- src/DEX/OnChain/Common/Types/Request.hs
- src/DEX/OnChain/Common/Types/Factory.hs
- src/DEX/OnChain/Common/Types/Misc.hs
- src/DEX/OnChain/Common/Types/Base.hs
- src/DEX/OnChain/Common/Types/Pool.hs
- src/DEX/OnChain/Core/AMM.hs
- src/DEX/OnChain/Core/FlatAssets.hs
- src/DEX/OnChain/Core/Request.hs
- src/DEX/OnChain/Core/PoolState.hs
- src/DEX/OnChain/Core/Pool.hs
- src/DEX/OnChain/Util/Unsafe.hs
- src/DEX/OnChain/Util/RawData.hs
- src/DEX/OnChain/Util/Debug.hs
- src/DEX/OnChain/Util/Misc.hs
- src/DEX/OnChain/Staking/RewardMint.hs
- src/DEX/OnChain/Mint.hs

- src/DEX/OnChain/Request.hs
- src/DEX/OnChain/Factory.hs
- src/DEX/OnChain/Pool.hs

The off-chain parts of the repository are not part of the audit scope. In particular, we have not done an in-depth review of the deployment scripts, so we assume that the various access-control tokens are minted and distributed correctly.

Also, the off-chain code relies on a WingRiders-specific fork of the `plutus-apps` repository, in order to submit transactions that deal with staking addresses for scripts. (The forked repo can be viewed publically at `https://github.com/WingRiders/plutus-apps/commit/6c46c470b6c06ea3ab099f7ad6cf03891c730a17`.) This should not affect the on-chain code, but in any case our audit was done on the source-code level, not the compiled and deployed scripts.

Since some properties are expensive or impossible to check on-chain, the WingRiders design will use a reputation system for the Agents. Since they are not checked for in the on-chain code, they are not considered as attacks for the purposes of this audit.

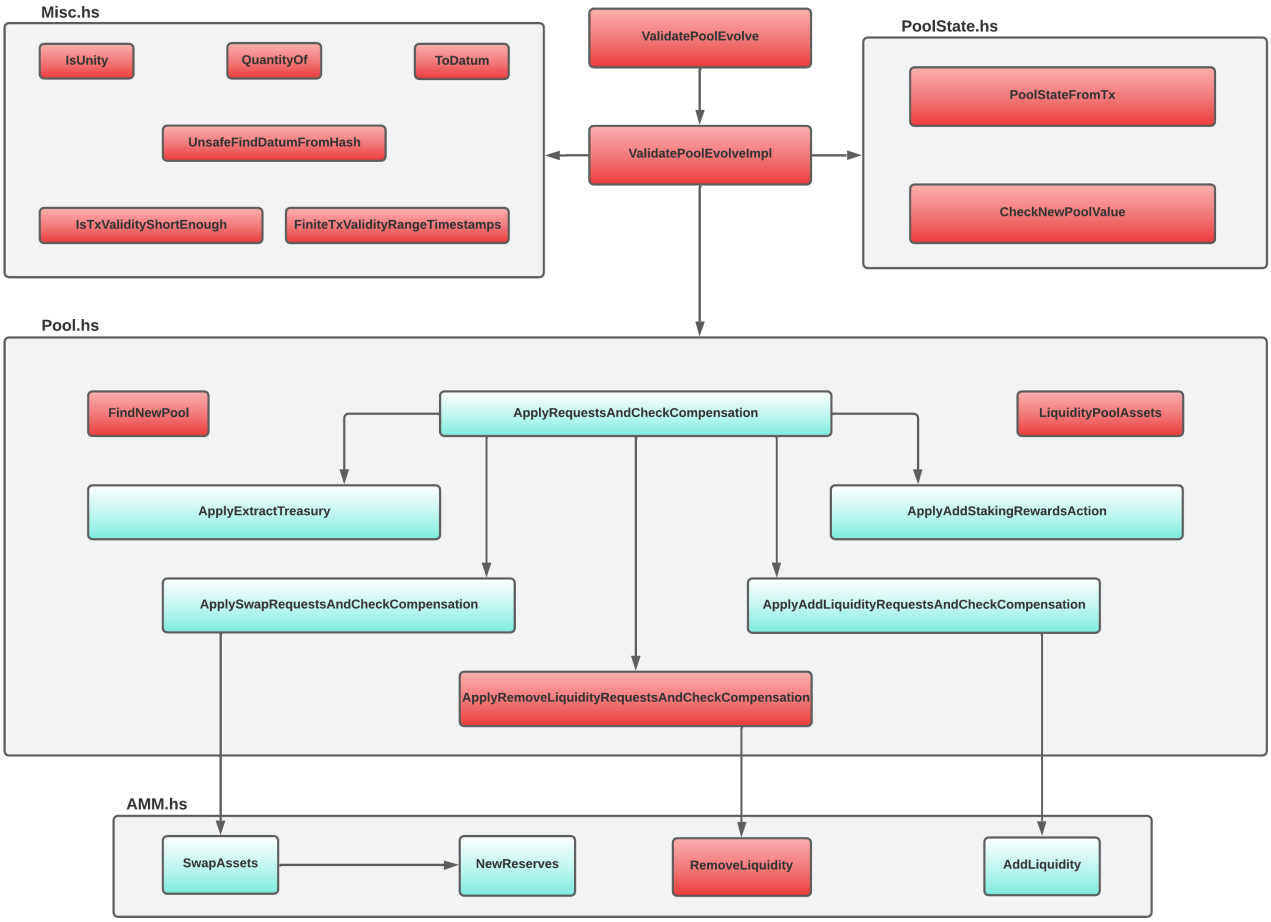
- **Frontrunning:** The agents should submit requests in the order as the where added to the blockchain, and changing this order will result in users paying unfair prices. Agents are incentivized to submit the requests in the correct order to maintain their reputation.
- **Stoppage:** The DEX cannot function except for supporting emergency withdrawals if all agents stop processing requests.
- **Denial of service attack:** Agent doesn’t process requests but executes dummy requests to keep the pool alive so you can’t use the emergency withdraw action. This way, a malicious agent could carry out a denial of service attack. However this can be expensive to the agent since it costs the agent the protocol fees.

Formal verification

Formal guarantees about code being executed on blockchains can come in the form of properties about the algorithm which makes its logic secure against hacks, as well as in the form of invariants that should always hold for the code to be deemed safe. As part of the audit we apply formal verification in to prove properties and invariants about some of the functions in the contract. We prove them as theorems in the Coq proof assistant.

In the case of WingRiders, the code is structured into a set of business-logic functions that compute the new state of the contract after applying an action, and a set of validation functions that use the business logic to check if a proposed transaction is correct. Because there currently is no production-ready formal specification of the Cardano blockchain we do not formally verify the validation functions, but we do formally prove properties about the business logic. The security case for the full contract then relies on manual review of the validation functions, as well as manual review of the business logic code for additional assurance, but during that review the auditor can rely on the verified properties.

The following figure shows the functions in WingRiders that are subject to formal verification. The colors indicate which functions have been formally verified (green) and which have only been manually audited (red).



The business logic functions operate on a datatype `PoolState` which is simply five numbers:

- the reserves of token A and B,
- the treasury amounts of A and B,
- and the amount of available shares.

To validate a batch transaction, the validator parses the pool UTxO datum to find the current values, and then calls `ApplyRequestsAndCheckCompensation` to compute the new values after carrying out all the actions. `ApplyRequestsAndCheckCompensation` in turn calls functions for each individual requestes.

We manually check that inside `validatePoolEvolveImpl` certain properties about the state of the liquidity pool (modelled as a `PoolState`) hold. Then we formally state those properties and use them as assumptions for the theorem, that the business logic function `applyRequestsAndCheckCompensation` satisfies certain invariants.

A concrete example of the above methodology is establishing a formal proof that at no point can the DEX allocate more liquidity shares than the initially preminted liquidity tokens. Since a UTxO cannot carry a negative Value, the `checkNewPoolValue` function ensures that allocated shares cannot exceed the

maximum initial value. This in turn helps in establishing this invariant for the `applyAddLiquidityRequestAndCheckCompensation` function.

Verification results

The machine-checked formal statements are quoted below. Here we give an informal overview of them.

First, we have formally proven that the *increasing k property* holds after each swap operation for a pool. Here `k` represents the product of the two token reserves. The property states that if `r_1new` and `r_2new` are the liquidity reserves in a pool after a swap operation, and if `r_1old`, `r_2old` were the reserves in the pool before the swap operation, then `r_1new * r_2new >= r_1old * r_2old`.

This property is characteristic of the DEXs based on the constant-product market maker formula. If trading fees were zero *k* would be exactly constant, while including the fees it will increase. Although most papers describing AMMs treat the pricing formula as a simple equation, in a real contract like WingRiders it is more complicated: the numbers are integers rather than real numbers and need to be rounded correctly, and the *treasury* should not be considered part of the reserves and must be subtracted appropriately. Our proof checks that the pricing is done correctly.

The *increasing k property* is important because it gives confidence that the contract behaves as intended. On the one hand, it shows that trading operations cannot “drain” both reserves at the same time: if one of them decreases the other one must increase. Historically, there have been attacks on DEXes where the swap operation was implemented incorrectly (e.g. due to bad rounding or bad unit conversions) so an attacker could steal the locked funds in the contract, but this is ruled out by our proof. We formally prove this corollary in Coq (theorem `non_depletion_property`). On the other hand, it helps show that the economic theory of the contract is as expected, because it suggests that it is not profitable for a user to split a large trade into two consecutive smaller trades with the DEX, doing so would just incur more trading fees. (We have not formalized a proof of the no-splitting-trades property for WingRiders, but informally it follows from increasing k.)

Second, we also prove that the treasury value will increase monotonically by each action, except for the treasury extraction action itself (which sets it to zero). This also serves as a sanity check that there are no bugs in the trading code.

Finally, since the wingriders DEX protocol premints liquidity pool share tokens, it is important to ensure that at no point can the DEX allocate more shares than what is available. We formally prove this as an invariant. We also ensure that even if liquidity providers withdraw liquidity, the amount of liquidity pool tokens cannot exceed the maximum value set by the number of preminted tokens. These are two invariants which we prove hold for the liquidity pool shares.

Verification details

We show the theorem statements for the results we verified. The Definitions block contains definitions used to state the results, and then there are several statements which prove a property or invariant for an individual function. They lead up to a theorem for the top level business-logic function `applyRequestsAndCheckCompensations`.

Definitions

```
Definition extract_Request (l : list (Types.ParsedRequest * PlutusTypes.Value))
    : (list Types.ParsedRequest) :=
  List.map fst l.

Definition extract_value (l : list (Types.ParsedRequest * PlutusTypes.Value))
    : (list PlutusTypes.Value) :=
  List.map snd l.

Definition extract_Request (l : list (Types.ParsedRequest * PlutusTypes.Value)) : (list
Types.ParsedRequest) :=
List.map fst l.

Definition tx_instance (v : PlutusTypes.Value) : Prop :=
  exists (tx : Tx),
       validValuesTx tx = true
    /\ List.elem v (List.map PlutusTypes.txOutValue (txOutputs tx)) = true.

Fixpoint tx_instance_list (l : list PlutusTypes.Value) : Prop :=
match l with
 | x :: xs => (tx_instance x) /\ (tx_instance_list xs)
 | nil => True
end.

Definition flatAssets_request (x : Types.ParsedRequest) (oilAda : GHC.Num.Int): Prop :=
let
 '(Types.MkParsedRequest rAct rAssets cAssets) := x
in
 exists (val : PlutusTypes.Value) (assetA assetB shares : PlutusTypes.AssetClass),
 tx_instance val /\
FlatAssets.flatAssetsFromValue (FlatAssets.MkLiquidityPoolAssets assetA assetB shares)
(val, oilAda) = rAssets.

Fixpoint flatAssets_request_list (l : list Types.ParsedRequest) (oilAda : GHC.Num.Int) :
Prop :=
match l with
 | x :: xs => (flatAssets_request x oilAda) /\ (flatAssets_request_list xs oilAda)
 | nil => True
end.

Fixpoint checkNewPool_true_request_list
```

```
      (l : list (Types.ParsedRequest*PlutusTypes.Value))
      (p : Types.PoolState)
      (lpVT : Types.LPValidityAssetClass)
      (lpAssets : FlatAssets.LiquidityPoolAssets)
     : bool :=
  match l with
   | (x1 , x2) :: l' =>
     let '(p1, Pool.MkIntentionallyUnused)
       := applyRequestsAndCheckCompensations (p, Pool.MkIntentionallyUnused)
                                             (x1 :: nil)
     in
       (Pool.checkNewPoolValue lpVT lpAssets x2 p1)
       && checkNewPool_true_request_list l' p1 lpVT lpAssets
   | nil => true
  end.

Fixpoint checkRequestListforRemoveLiquidity (l : list Types.ParsedRequest ) : bool :=
match l with
 | x :: xs => match x with
              | (Types.MkParsedRequest (Types.WithdrawLiquidityAction _ _) _ _) => true
              | _ => checkRequestListforRemoveLiquidity xs
             end
 | nil => false
end.

Fixpoint checkRequestListforExtractTreasury (l : list Types.ParsedRequest) : bool :=
match l with
 | x :: xs => match x with
              | (Types.MkParsedRequest Types.ExtractTreasuryAction _ _) => true
              |  _ => checkRequestListforExtractTreasury xs
             end
 | nil => false
end.
```

Verification #1

Code

About the function ***newReserves*** in the file `/DEX/OnChain/Core/AMM.hs`.

```
newReserves :: (Integer, Integer) -> Integer -> ((Integer, Integer), (Integer, Integer))
```

Specification

```
Lemma increasing_k :
forall (r_1old r_2old d_amt r_1new r_2new r_amt p_fee : GHC.Num.Int),
  AMM.newReserves (r_1old, r_2old) d_amt = ((r_1new, r_2new), (r_amt, p_fee)) ->
```

```
  r_1new GHC.Num.* r_2new >= r_1old GHC.Num.* r_2old.
```

✅ The code meets the specification.

Verification #2

Code

About the function ***swapAssets*** in the file `/DEX/OnChain/Core/AMM.hs`.

```
swapAssets :: PoolState -> (SwapDirection, Integer) -> (PoolState, Integer)
```

Specification

```
Lemma swapAssets_increasing_k :
  forall (oldA newA : GHC.Num.Int)
         (oldB newB : GHC.Num.Int)
         (oldTreasuryA newTreasuryA : GHC.Num.Int)
         (oldTreasuryB newTreasuryB : GHC.Num.Int)
         (d_amt r_amt : GHC.Num.Int)
         (shares_old shares_new : GHC.Num.Int)
         (swap_dir : SwapDirection),
    AMM.swapAssets (MkPoolState oldA oldB shares_old oldTreasuryA oldTreasuryB)
                   (swap_dir , d_amt)
      = ((MkPoolState newA newB shares_new newTreasuryA newTreasuryB) , r_amt) ->

    newA GHC.Num.* newB >= oldA GHC.Num.* oldB.
```

✅ The code meets the specification.

Verification #3

Code

About the function ***applyAddLiquidityRequestAndCheckCompensation*** in the file `/DEX/OnChain/Core/Pool.hs`.

```
applyAddLiquidityRequestAndCheckCompensation :: PoolState -> ParsedRequest -> PoolState
```

Specification

```
Theorem applyAddLiquidityRequestAndCheckCompensation_inv :
forall (oldA newA : GHC.Num.Int)
       (oldB newB : GHC.Num.Int)
       (oldqtyShares newqtyShares : GHC.Num.Int)
       (oldTreasuryA newTreasuryA : GHC.Num.Int)
       (oldTreasuryB newTreasuryB : GHC.Num.Int)
       (pRequest : Types.ParsedRequest)
       (lpValidityToken : Types.LPValidityAssetClass)
       (lpAssets : FlatAssets.LiquidityPoolAssets)
       (newPoolValue : PlutusTypes.Value),
  oldqtyShares >= 0 ->
  tx_instance newPoolValue ->
  applyAddLiquidityRequestAndCheckCompensation (Types.MkPoolState oldA
                                                                  oldB
                                                                  oldqtyShares
                                                                  oldTreasuryA
                                                                  oldTreasuryB)
                                               pRequest
    = (Types.MkPoolState newA newB newqtyShares newTreasuryA newTreasuryB) ->
  Pool.checkNewPoolValue lpValidityToken
                         lpAssets
                         newPoolValue
                         (Types.MkPoolState newA newB newqtyShares newTreasuryA
newTreasuryB) = true ->

  newqtyShares <= oldqtyShares /\ newqtyShares >= 0.
```

✅ The code meets the specification.

Verification #4

Code

About the function ***applySwapRequestAndCheckCompensation*** in the file `/DEX/OnChain/Core/Pool.hs`.

```
applySwapRequestAndCheckCompensation :: PoolState -> ParsedRequest -> PoolState
```

Specification

```
Theorem applySwapRequestAndCheckCompensation_increasing_k
   : forall (oldA newA : GHC.Num.Int)
            (oldB newB : GHC.Num.Int)
            (oldqtyShares newqtyShares : GHC.Num.Int)
            (oldTreasuryA newTreasuryA : GHC.Num.Int)
            (oldTreasuryB newTreasuryB : GHC.Num.Int)
            (pRequest : Types.ParsedRequest),
   oldqtyShares >= 0 ->
```

```
    applySwapRequestAndCheckCompensation (
         Types.MkPoolState oldA oldB oldqtyShares oldTreasuryA oldTreasuryB)
         pRequest
      = (Types.MkPoolState newA newB newqtyShares newTreasuryA newTreasuryB) ->

    newA GHC.Num.* newB >= oldA GHC.Num.* oldB.
```

✅ The code meets the specification.

Verification #5

Code

About the function ***newReserves*** in the file `/DEX/OnChain/Core/AMM.hs`.

```
newReserves :: (Integer, Integer) -> Integer -> ((Integer, Integer), (Integer, Integer))
```

Specification

```
Lemma non_depletion_property
  : forall (initialA initialB : GHC.Num.Int)
           (lockA : GHC.Num.Int)
           (finalA finalB : GHC.Num.Int)
           (outY pFee : GHC.Num.Int),
   AMM.newReserves (initialA , initialB) lockA = (finalA, finalB , (outY, pFee)) ->

   finalA > 0 /\ finalB > 0.
```

✅ The code meets the specification.

Comment: This result ensures that trade requests cannot completely drain the funds of the contract.

Verification #6

Code

About the function ***applyRequestsAndCheckCompensations*** in the file `/DEX/OnChain/Core/Pool.hs`.

```
applyRequestsAndCheckCompensations :: (PoolState, IntentionallyUnused) -> [ParsedRequest]
-> (PoolState, IntentionallyUnused)
```

Specification

```
Theorem applyRequestsAndCheckCompensation_inv :
forall (l1 : list (Types.ParsedRequest * PlutusTypes.Value)),
forall (initialA initialB oilAda : GHC.Num.Int)
       (finalA finalB : GHC.Num.Int)
       (initialTreasuryA finalTreasuryA : GHC.Num.Int)
       (initialTreasuryB finalTreasuryB : GHC.Num.Int)
       (initialShares finalShares : GHC.Num.Int)
       (lpVT : Types.LPValidityAssetClass)
       (lpAssets : FlatAssets.LiquidityPoolAssets)
       (l2 : list Types.ParsedRequest),
  extract_Request l1 = l2 ->
  initialA > 0 ->
  initialB > 0 ->
  initialShares >= 0 ->
  checkRequestListforExtractTreasury l2 = false ->
  checkRequestListforRemoveLiquidity l2 = false ->
  tx_instance_list (extract_value l1) ->
  flatAssets_request_list l2 oilAda ->
  checkNewPool_true_request_list l1
                                 (Types.MkPoolState
                                   initialA
                                   initialB
                                   initialShares
                                   initialTreasuryA
                                  initialTreasuryB)
                                 lpVT l
                                 pAssets = true ->
  applyRequestsAndCheckCompensations ((Types.MkPoolState
                                       initialA
                                       initialB
                                       initialShares
                                       initialTreasuryA
                                       initialTreasuryB),
                                      Pool.MkIntentionallyUnused)
                                     l2
  = ((Types.MkPoolState
       finalA
       finalB
       finalShares
       finalTreasuryA
       finalTreasuryB), Pool.MkIntentionallyUnused) ->

  finalA GHC.Num.* finalB >= initialA GHC.Num.* initialB /\
  finalShares >= 0 /\
  (finalTreasuryA >= initialTreasuryA) /\
  (finalTreasuryB >= initialTreasuryB).
```

✅ The code meets the specification.

Comment: this result shows that the business logic for each batch transaction preserves the invariants we described above. Its proof relies on the previous results for the functions that it calls.

Findings

Critical	**0** (0.00%)
Major	**0** (0.00%)
Medium	**0** (0.00%)
Minor	**0** (0.00%)
Informational	**5** (100.00%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
AMM-01	Unclear Comment	Coding Style	Informational	Resolved
MUO-01	`divideCeil` Is Defined For Negative And Positive Integers	Logical Issue	Informational	Resolved
OCD-01	`finiteTxValidityRangeTimestamps` And `isTxValidityRangeShortEnough` Do Not Perform Enough Checks	Logical Issue, Mathematical Operations	Informational	Resolved
PCO-01	List Of Parsed Requests Not Checked For Possibly More Than One ExtractTreasury Actions.	Inconsistency	Informational	Acknowledged
RMS-01	Wrong Comment	Coding Style	Informational	Resolved

AMM-01 | Unclear Comment

Category	Severity	Location	Status
Coding Style	Informational	src/DEX/OnChain/Core/AMM.hs (old commit): 22, 25	Resolved

Description

The comments of the function `swapFee` say that the fees are 0.30% of the locked amount, however, the correct percentage is 0.35%.
0.30% is the *swap fees* minus the *protocol fees*.

Recommendation

We advise rewriting the comments so the percentage is consistent with the function logic.

Alleviation

The comment has been fixed in revision `282ea70b6284e6d12aa467c596f73e1f1f8bab0b`.

MUO-01 | `divideCeil` Is Defined For Negative And Positive Integers

Category	Severity	Location	Status
Logical Issue	Informational	src/DEX/OnChain/Util/Misc.hs: 174~177	Resolved

Description

The functions `divideCeil` use the functions `divide` and `modulo`. Therefore this function is defined for negative numbers which contradict the comment of this function.

Recommendation

We advise rewriting the comment of this function for consistency--e.g. change the comment to say "is not intended to be called with" rather than "is not defined" for negative number. The current comment could be taken to mean that it will throw an error if the numbers are negative.

Currently, this function is used by `swapFee` and `newReserves` in the `AMM.hs` contract and the functions calling them seem to prevent that `divideCeil` could be used with negative value.

Alleviation

This comment has been fixed in revision `282ea70b6284e6d12aa467c596f73e1f1f8bab0b`.

OCD-01 | `finiteTxValidityRangeTimestamps` And `isTxValidityRangeShortEnough` Do Not Perform Enough Checks

Category	Severity	Location	Status
Logical Issue, Mathematical Operations	Informational	src/DEX/OnChain/Util/Misc.hs: 244~247, 250~252 src/DEX/OnChain/Factory.hs: 107, 113 src/DEX/OnChain/Core/Pool.hs: 141, 143, 480	Resolved

Description

As the comment L252 says: the function `isTxValidityRangeShortEnough` checks if a given interval is valid so we can be sure it will not be rejected by the Cardano node. However, it is possible to have a `POSIXTimeRange` with a `LowerBound` greater than the `UpperBound`. The Plutus blockchain should reject such a transaction, but rather than relying on that, it may be wise to check the wellformedness here too. When converting a `POSIXTimeRange` to a couple `(POSIXTime, POSIXTime)` using the function `finiteTxValidityRangeTimestamps` no checks are made. The function `isTxValidityRangeShortEnough` will then always return true if `endTimestamp < startTimestamp`.

Recommendation

We advise adding more checks to ensure that the interval is really valid.

Alleviation

This issue has been fixed in revision `282ea70b6284e6d12aa467c596f73e1f1f8bab0b`, by adding an additional check that `startTimestamp < endTimestamp`.

PCO-01 | List Of Parsed Requests Not Checked For Possibly More Than One ExtractTreasury Actions.

Category	Severity	Location	Status
Inconsistency	Informational	src/DEX/OnChain/Core/Pool.hs: 300~313	Acknowledged

Description

According to the design, agents should submit at most one `ExtractTreasuryAction` in a batch. However, there is no check in the function `applyRequestsAndCheckCompensations` that the list of `ParsedRequests` it receives as an input contains at most one such action. There *is* a check that each `ExtractTreasuryAction` widthdraws a nonzero amount, but that still allows an attacker to submit hoax `ExtractTreasuryAction` for half of the request in the batch, preventing other requests from being processed and included in the bulk transaction.

Recommendation

Add a check to ensure that the list of `ParsedRequests` passed as input to the `applyRequestsAndCheckCompensations` functions contains at most one `ExtractTreasuryAction`.

Alleviation

WingRiders have considered this possibility, but determined that the impact of the proposed attack is minor enough that it is not necessary to include any additional on-chain checks.

"It is up to the batching agent to include only 1 extract treasury request in 1 batch. Although it is possible for the agent to carry out an extract treasury request as every other request in the transaction, this would merely slow down the overall processing speed and not stop the processing overall. Agent doing this is subject to a reputational damage and a loss of collateral. What's more, every extract treasury request would still need to produce an interim treasury UTxO that needs to contain at least min-ada ADA value. This means that if the swap request in between extract treasury requests was not big enough, agent doing this would end up donating ADA to the treasury for every extract treasury request."

RMS-01 | Wrong Comment

Category	Severity	Location	Status
Coding Style	Informational	src/DEX/OnChain/Staking/RewardMint.hs: 95	Resolved

Description

The linked line, from the comments of the function `validateBurnRewardTokens`, is not true, indeed `validateBurnRewardTokens` does not check that a single reward token was minted. This comment seems to have been copied and pasted from the comments of the function `validateMintSingleRewardToken` and has not been completely corrected.

Recommendation

We advise rewriting this comment so it is consistent with the logic of the function `validateBurnRewardTokens`.

Alleviation

This comment has been fixed in revision `282ea70b6284e6d12aa467c596f73e1f1f8bab0b`.

Appendix

Finding Categories

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.

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.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

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

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.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS

AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.