Zellic

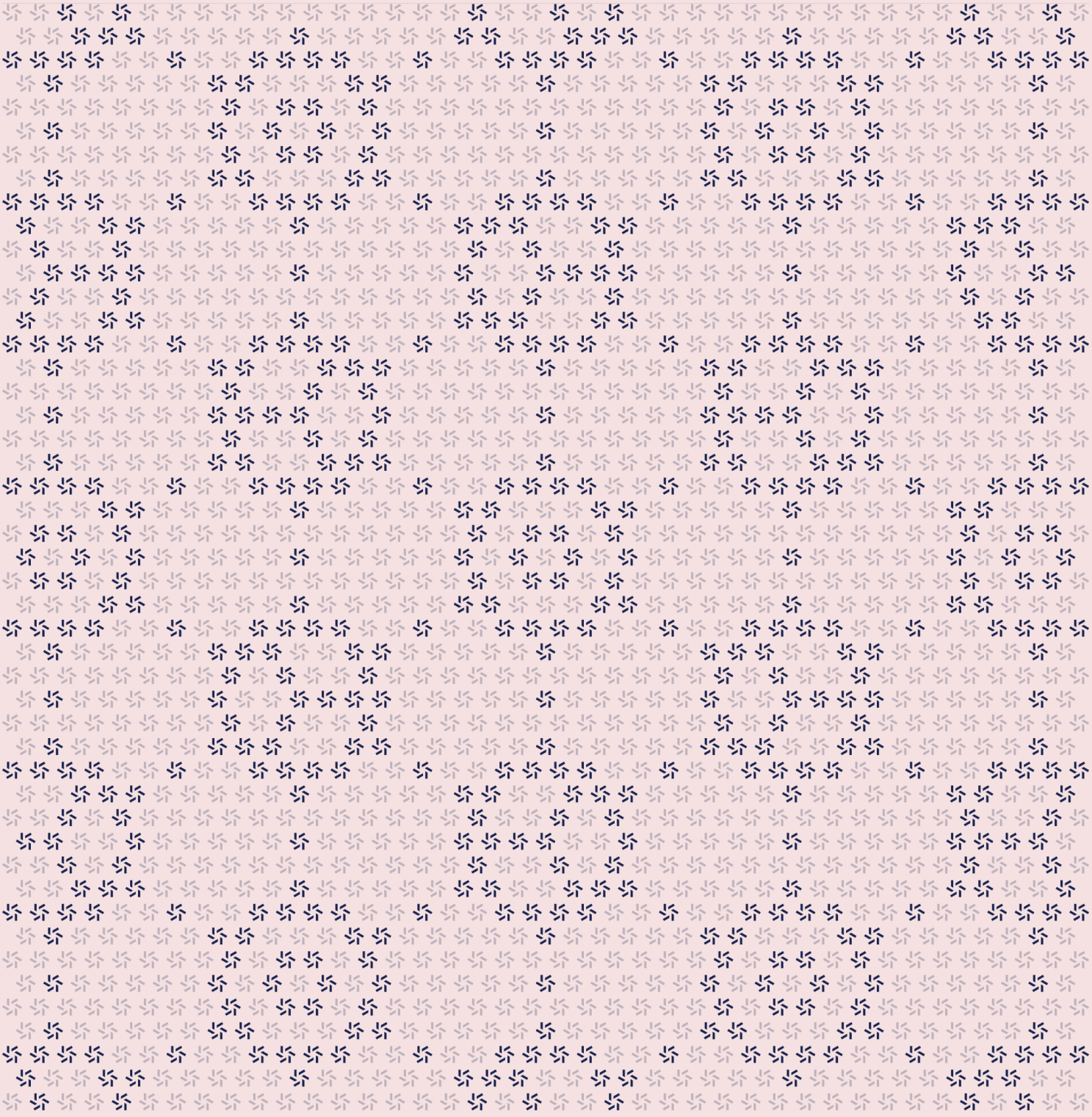
Prepared for
Nic
Voltage Technologies Ltd.

Prepared by
Nipun Gupta
Daniel Lu
Zellic

February 24, 2025

Circuit DAO

Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Voltage Technologies Ltd. from January 6th to February 13th, 2025. During this engagement, Zellic reviewed Circuit DAO's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Could an attacker steal funds from the treasury or the vaults?
- Would it be possible to create fake tokens with lineage to prove that they are protocol coins and perform access-controlled operations?
- Is it possible for attackers to perform griefing attacks on victims such that they could not receive their rewards/tokens?
- Are the conditions returned by puzzles properly filtered such that those conditions could not be treated like protocol-specific conditions?
- Is the accounting logic working as expected?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

Based on the number of severe findings uncovered during the audit, it is our opinion that the project is not yet ready for production. We strongly advise a comprehensive reassessment before deployment to help identify any potential issues or vulnerabilities introduced by necessary fixes or changes. We also recommend adopting a security-focused development workflow, including (but not limited to) augmenting the repository with comprehensive end-to-end tests that achieve 100% branch coverage using any common, maintainable testing framework, thoroughly documenting all function requirements, and training developers to have a security mindset while writing code.

1.4. Results

During our assessment on the scoped Circuit DAO puzzles, we discovered 16 findings. Three critical issues were found. Seven were of high impact, one was of medium impact, two were of low impact, and the remaining findings were informational in nature.

Additionally, Zellic recorded its notes and observations from the assessment for the benefit of Voltage Technologies Ltd. in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
Critical	3
High	7
Medium	1
Low	2
Informational	3

2. Introduction

2.1. About Circuit DAO

Voltage Technologies Ltd. contributed the following description of Circuit DAO:

> Circuit is a CDP protocol built on Chia. Users can permissionlessly borrow Bytecash (BYC), a stablecoin pegged 1:1 to the USD, or earn yield by depositing BYC in a savings vault.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the puzzles.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood.

We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped puzzles itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

Circuit DAO Puzzles

Type	chialisp
Platform	Chia
Target	Circuit DAO
Repository	https://github.com/circuitdao/puzzles ↗
Version	`c9e4393393bc6da57d69f339b4dcb818baeff378`

Programs	`payout.clsp`
	`statutes.clsp`
	`oracle_outlier_resolver.clsp`
	`treasury.clsp`
	`cat_v2_debug.clsp`
	`byc_tail.clsp`
	`oracle.clsp`
	`atom_announcer.clsp`
	`crt_tail.clsp`
	`programs/*.clsp`
	`savings_vault.clsp`
	`surplus_auction.clsp`
	`governance.clsp`
	`recharge_auction.clsp`
	`collateral_vault.clsp`
	`announcer_registry.clsp`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of 10 person-weeks. The assessment was conducted by two consultants over the course of six calendar weeks.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Nipun Gupta
Engineer
nipun@zellic.io

Daniel Lu
Engineer
daniel@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

January 6, 2025	Kick-off call
January 6, 2025	Start of primary review period
February 13, 2025	End of primary review period

3. Detailed Findings

3.1. Broken lineage check due to governance-coin unwrap logic

Target	governance		
Category	Business Logic	**Severity**	Critical
Likelihood	High	**Impact**	Critical

Description

Governance coins are CRT CAT singletons with the governance.clsp puzzle as their inner puzzle. These coins are created from standard CRT coins on an ad hoc basis. The coin allows the coins inside it to be exited to the `INNER_PUZZLE_HASH` argument curried in the puzzle.

If the puzzle hash of the `INNER_PUZZLE_HASH` is same as that of the governance, the coin could be spent like a governance coin and its lineage would be verified as the previous coin that created it was the governance itself. Therefore, by setting `INNER_PUZZLE_HASH` to the puzzle hash of a governance puzzle with the `BILL` argument already curried inside it and the `INNER_PUZZLE_HASH` of the inner argument to be `0`, an attacker could exit the original governance to a fake governance that has the `BILL` already curried inside it.

If the `veto_conditions` passed to the inner governance is `CREATE_COIN` with the puzzle hash as `0` and a `REMARK` condition, which has the args required for a veto-condition operation, the `final_bill` would be set as `0` as the veto condition resets the bill. The `filter-conditions` in the inner governance will thus return a `CREATE_COIN` condition with the puzzle hash as `0` and a `REMARK` condition as shown below:

```
REMARK PROTOCOL_PREFIX bill_operation () ()
```

As the inner governance act is just like an inner puzzle for the outer governance, the conditions returned from it, `REMARK` and `CREATE_COIN`, will be used to find the bill condition — `REMARK` will be used to get the values of `statutes_inner_puzzle_hash`, `bill_operation`, and `args`, and `CREATE_COIN` will be used to get the `new_coin_amount`.

As per the `REMARK` condition returned from inner governance, the `bill_operation` will be set to `()`. Therefore, no operations will be run, and `CREATE_COIN` will be used to exit the governance to the inner puzzle (which is the inner governance) because the `CREATE_COIN`'s puzzle hash is `0`.

Therefore, the exited governance puzzle will be the inner governance puzzle with the `BILL` already curried in it, and lineage could be proved.

Impact

If a governance puzzle is launched with the `BILL` already curried inside, it will be possible to spend the governance coin along with statutes to change the value of any statute (the global variables) to the desired value immediately.

Recommendations

Instead of allowing exiting to `INNER_PUZZLE_HASH`, the coins could be sent with the help of a settlement puzzle.

Remediation

This issue has been acknowledged by Voltage Technologies Ltd., and a fix was implemented in commit `ccd7ef02` ↗.

3.2. Missing MOD_HASH validation

Target	Protocol Wide		
Category	Business Logic	**Severity**	Critical
Likelihood	High	**Impact**	Critical

Description

The communication between two puzzles take place via the SEND_MESSAGE and RECEIVE_MESSAGE conditions. The third parameter of the condition SEND_MESSAGE is the destination, and for the RECEIVE_MESSAGE, it is the source from which the message originated. The source of the messages in the puzzles are calculated via calculating the tree hash of the mod of the source puzzle along with the parameters. While the source mod hash is usually either curried in the destination mod itself or verified via the statutes announcements, the parameter (args) hash is provided as the solution of the destination puzzle. The args of the destination could therefore be set to any value as long as the source of the message is correctly asserted. In the source puzzle, if we curry incorrect values, it should be prevented via the lineage as it makes sure that the coin to be spent has the parent coin of the same type.

Taking an example of governance and the statutes-mutation interaction, the RECEIVE_MESSAGE in statutes mutation verifies that the message was sent via the following puzzle hash:

```
governance_puzzle_hash (tree_hash_of_apply GOVERNANCE_MOD_HASH
    governance_curried_args_hash)
```

Although the value of governance_curried_args_hash could be controlled by the attacker, the only problem is that these values cannot be arbitrarily set, as the governance mod verifies the lineage via the following code:

```
(list ASSERT_MY_PARENT_ID
  (calculate-coin-id
    parent_parent_id
    (curry_hashes CAT_MOD_HASH
      (sha256 ONE CAT_MOD_HASH)
      (sha256 ONE CRT_TAIL_HASH)
      (curry_hashes MOD_HASH
        (sha256 ONE MOD_HASH)
        (sha256 ONE CAT_MOD_HASH)
        (sha256 ONE CRT_TAIL_HASH)
        (sha256tree STATUTES_STRUCT)
        (sha256 ONE INNER_PUZZLE_HASH)
```

```
        prev_bill_hash
      )
    )
    amount
  )
)
```

This ensures that the parent of the current coin is the governance coin — only if the `MOD_HASH` is set to the `governance_mod_hash`. The issue arises if this coin is created via a CRT CAT coin with the mod hash controlled by the attacker. This way, they could launch a new coin with the puzzle hash of the `governance_mod_hash`, but the `MOD_HASH` value should be that of the previous coin along with the `BILL` already curried inside the coin. This makes it possible to prove the lineage along with the `BILL` arguments to be set to any values, and thus the attacker could enact the bill in the next transaction as they control the `governance_curried_args_hash` passed to the `statutes_mutation puzzle`.

Impact

The vulnerability allows to incorrectly prove lineage of certain coins and send messages to other puzzles, which could be received via passing the incorrect arg hashes. Sending malicious messages could lead to various issues, including changing global variables without governance voting, potentially winning recharge auctions without bidding, and so forth.

Recommendations

While discussing the issue with the Voltage Technologies Ltd. team, we concluded that asserting the puzzle hash of the puzzle during spending should be able to resolve the issue as it makes sure that the mod hash of the puzzle and `MOD_HASH` curried inside are correct.

Remediation

This issue has been acknowledged by Voltage Technologies Ltd., and fixes were implemented in the following commits:

- f877e77e ↗
- ca8ca0c5 ↗

3.3. Improper condition filters in savings vault

Target	savings_vault		
Category	Business Logic	**Severity**	Critical
Likelihood	High	**Impact**	Critical

Description

The conditions returned by the inner puzzle of the savings-vault puzzle are filtered using the `filter-conditions` function. This function allows more than one `CREATE_COIN` condition to be passed. If two coins are created from this savings vault with the same puzzle mod but one contains different curried arguments, the lineage could still be proved.

Impact

New fake coins with nonzero value of `DISCOUNTED_DEPOSIT` could be used to steal from the treasury coin while still maintaining the correct lineage.

Recommendations

We recommend only allowing one `CREATE_COIN` condition to be returned from the inner puzzle of the savings vault.

Remediation

This issue has been acknowledged by Voltage Technologies Ltd., and a fix was implemented in commit 30165496 ↗.

3.4. Missing condition filters in outlier resolver

Target	oracle_outlier_resolver		
Category	Coding Mistakes	**Severity**	Critical
Likelihood	Low	**Impact**	High

Description

The Circuit protocol has an oracle that grabs data from approved announcers. During mutations, if an outlier is detected, price updates are halted while governance is given a chance to resolve the issue. In particular, users can vote on a decision (whether to accept the price) and indicate whether some announcers are to be temporarily banned.

These votes are done with the oracle outlier resolver CRT coin, which when spent announces the decision and ban list, its balance, and the time since the coin's creation.

The puzzle will also include in its conditions the output of the owner's solution to the inner puzzle. These conditions are not filtered.

Impact

As a result, the owner of the coin can insert arbitrary messages into the spend conditions. These include `SEND_MESSAGE` conditions that announce an incorrect balance. Thus, the owner can vote as if they had more shares.

```
(c
  ; ensure this coin was created before the cooldown interval
  (list ASSERT_SECONDS_RELATIVE cooldown_interval)
  (c
    ; ensure the amount is correct
    (list ASSERT_MY_AMOUNT amount)
    (if oracle_coin_id
      ; confirm this coin was spent with correct params
      (c
        (list SEND_MESSAGE
          0x3f
          (concat "C" (sha256tree new_temp_ban_list)
    decision cooldown_interval amount)
          oracle_coin_id
        )
        (a INNER_PUZZLE inner_solution)
```

```
      )
      ; just run the inner solution if no oracle coin id
      ; to allow the owner to exit this puzzle or change the inner puzzle
      (a INNER_PUZZLE inner_solution)
    )
  )
)
```

This can be accomplished either by pointing `oracle_coin_id` to a coin controlled by the owner (and can hence accept the message) or by setting it to `0`, `nil`, and so on.

Recommendations

We recommend filtering the conditions of the inner puzzle to prevent the owner from inserting arbitrary messages. Note Finding 3.14. ↗.

Remediation

This issue has been acknowledged by Voltage Technologies Ltd., and a fix was implemented in commit `8f4f1233` ↗.

3.5. Incorrect `STATUTES_MAX_IDX` for statutes mutation

Target	statutes		
Category	Coding Mistakes	**Severity**	High
Likelihood	Medium	**Impact**	High

Description

The governance-updated statutes has an associated `STATUTES_MAX_IDX` that represents the largest index of the statutes. This value is not correct in the current implementation, since more statutes were added.

Impact

Higher-index statutes cannot be updated by governance as intended. This action would fail in the statute-mutation operation.

```
(assert
  (size_b32 governance_curried_args_hash)
  (> mutation_index -2) ; -1 for custom conditions, >= 0 for statutes
  (> STATUTES_MAX_IDX mutation_index)
  (if (= mutation_index CUSTOM_CONDITIONS_MUTATION_INDEX)
    ; vote for announcements only, so this should just be a list
    (l mutation_value)
    ; should be  a list with 5 elements if mutating statutes
    (not (r (r (r (r (r mutation_value))))))
  )
```

Recommendations

Update the `STATUTES_MAX_IDX` to the correct value. It may be desirable to find a systematic way to keep this value up-to-date.

Remediation

This issue has been acknowledged by Voltage Technologies Ltd., and a fix was implemented in commit `7b23a5fa` ↗.

3.6. Mistakes in auction-collateral calculations

Target	vault_keeper_bid		
Category	Coding Mistakes	**Severity**	High
Likelihood	High	**Impact**	High

Description

The puzzle vault_keeper_bid.clsp allows keepers to place bids on the collateral in a vault that is being liquidated. The keepers provide the bid amount in BYC, and this amount is used to calculate the collateral amount returned to the bidder based on the current auction price. The auction price is calculated based on the time that has passed since the start of the auction and the price-decrease factor.

The amount of collateral returned should thus be equal to the `byc_bid_amount` divided by the current price after reduction, and the current price after reduction should be calculated as price reduced per interval * total intervals, which when expanded, results in the following formula:

$$\frac{\text{start\_price} \times \text{step\_price\_decrease\_factor}}{\text{PERCENT\_PRECISION}} \times \left(\frac{\text{current\_timestamp} - \text{auction\_start\_time}}{\text{step\_time\_interval}} \right)$$

So finally, the formula to calculate collateral comes out to be

$$\frac{\text{byc\_bid\_amount} \times \text{PRECISION} \times \text{MOJOS}}{10 \times \left(\text{start\_price} - \frac{\text{start\_price} \times \text{step\_price\_decrease\_factor}}{\text{PERCENT\_PRECISION}} \times \frac{\text{current\_timestamp} - \text{auction\_start\_time}}{\text{step\_time\_interval}} \right) \times \text{PRECISION}}$$

where 10/MOJOS is for unit conversion. To compare, the calculation of `bid_xch_collateral_amount_pre` in the puzzle is as follows:

```
bid_xch_collateral_amount_pre (/
  (*
    (/
      (* byc_bid_amount PRECISION)
      (*
        (- start_price
            (* ; current price factor
              (/
                (*
                  start_price
```

```
                  step_price_decrease_factor
                )
                PERCENT_PRECISION
              )
              ; current price
              (/ (- current_timestamp auction_start_time) step_time_interval)
            )
          )
          10
        )
      )
      MOJOS)
    PRECISION
)
```

That is, it multiplies the numeration by an extra `start_price` and divides by an extra `PERCENT_PRECISION` before subtracting it from `start_price`, which should not be there.

Impact

The calculation of collateral to be returned to the user is incorrect, which will lead to incorrect accounting and potentially loss of funds.

Recommendations

We recommend changing the code of the puzzle by removing the extra `start_price` and `PERCENT_PRECISION` from the calculation.

Remediation

This issue has been acknowledged by Voltage Technologies Ltd., and a fix was implemented in commit `1102b793` ↗.

3.7. Outlier resolution–logic bugs

Target	oracle		
Category	Coding Mistakes	**Severity**	High
Likelihood	High	**Impact**	High

> Note: This issue was independently raised by the Voltage Technologies Ltd. team during the review period.

Description

The following issues were discovered in the oracle resolution puzzle:

1. As we explain in section 5.6. ↗, the oracle coin stores price data with the oldest price at the head. However, the resolution operation appends the newest price to the head.

```
prices (if price_to_apply
  (c price_to_apply PRICES)
  PRICES
)
```

2. Additionally, the outcome of the resolution vote is improperly accounted for when determining the price. If, after maturity, the `current_decision_bool` is false, the `price_to_apply` should be empty, but the puzzle is using the price from the `OUTLIER_INFO` instead.

Impact

This means that the oracle coin's accounting can be broken if an outlier resolution vote occurs.

Recommendations

The issues discussed above are already fixed by the Circuit DAO team as follows:

1. They added the new `price_to_apply` at the end of the `PRICES`.
2. They added an if condition to correctly set the `price_to_apply` value.

Remediation

This issue has been acknowledged by Voltage Technologies Ltd., and a fix was implemented in commit `cef2f579`.

3.8. Unverified solution parameters in `announcer_registry` could lead to loss of rewards

Target	announcer_registry		
Category	Business Logic	**Severity**	High
Likelihood	Medium	**Impact**	High

Description

Announcers can periodically claim rewards from the announcer registry. These rewards are in the form of CRT tokens that are minted by the registry. When a reward claim is made, the rewards are automatically distributed to all the registered announcers. The `MINT` operation, which is used to distribute rewards, expects the caller to include the following solution parameters (along with lineage and the opcode of the operation):

```
statutes_inner_puzzle_hash
statutes_price_updates
crt_credits_per_interval
claim_interval
issuance_coin_info
cat_mod_hash
crt_tail_hash
change_receiver_hash
offer_mod_hash
```

The operation calculates the rewards to be distributed per announcer and then calculates the issuance coin ID to which it sends the message to issue CRT tokens. Then it finally asserts announcement from the CAT with `offer_mod_hash` as the inner puzzle to verify that the rewards are correctly distributed as follows:

```
issuance_coin_id (calculate-coin-id
  (f issuance_coin_info)
  (curry_hashes cat_mod_hash
    (sha256 ONE cat_mod_hash)
    (sha256 ONE crt_tail_hash)
    (f (r issuance_coin_info))
  )
  (f (r (r issuance_coin_info)))
)

; ...
```

```
(list SEND_MESSAGE 0x3f
    (concat
        PROTOCOL_PREFIX
            (sha256tree (c STATUTES_STRUCT (c offer_mod_hash
    crt_credits_per_interval)))
    )
        issuance_coin_id
    )

; ...

(list ASSERT_PUZZLE_ANNOUNCEMENT
        (sha256
          (curry_hashes cat_mod_hash
              (sha256 ONE cat_mod_hash)
              (sha256 ONE crt_tail_hash)
              offer_mod_hash
          )
          (sha256tree
            (c
              0 ; nonce
              (generate-offer-assert
                ANNOUNCER_REGISTRY
                (/ crt_credits_per_interval announcers_count)
                (if (> change_amount 0)
                  ; we reward the mint spender with remainder if any
                  (list
                    (list
                      change_receiver_hash
                      change_amount
                      (list change_receiver_hash)
                    )
                  )
                  ()
                )
              )
            )
          )
        )
      )
```

The parameters `cat_mod_hash`, `crt_tail_hash`, and `issuance_coin_info` are provided via the solution parameters and could be different from the values expected. A malicious user could therefore grief the legitimate announcers by providing incorrect values of `cat_mod_hash`, `crt_tail_hash`, and `issuance_coin_info` and receiving/announcing the messages from fake

puzzles such that the coin spend does not fail.

Impact

Legitimate announcers might lose their rewards due to the malicious solution parameters.

Recommendations

We recommend to verify the validity of `cat_mod_hash` and `crt_tail_hash` from statutes announcements.

Remediation

This issue has been acknowledged by Voltage Technologies Ltd., and a fix was implemented in commit `3818d5ba` ↗.

3.9. Unverified solution parameters in `recharge_win` could lead to loss of tokens

Target	recharge_win		
Category	Business Logic	**Severity**	High
Likelihood	Medium	**Impact**	High

Description

A recharge auction can be launched whenever the BYC balance of the treasury drops below the treasury minimum. In a recharge auction, the protocol acquires BYC to refill the treasury by auctioning off CRT. Once the recharge auction has concluded, the winner can mint an amount of CRT as specified in the bid by asserting the corresponding announcement from the auction coin, and the BYC amount bid gets paid to the treasury. The recharge-win operation allows the caller to provide the following solution parameters:

```
ttl
current_timestamp
treasury_coins
treasury_mod_hash
funding_coin_id
treasury_minimum
target_puzzle_hash
```

The `funding_coin_id` is the ID of the issuance coin, which is used to issue CRT coins to the winner of the bid. As the operation could be called by anyone, the callers could provide an incorrect value for `funding_coin_id`, which is used to send a message to the coin to issue the CRY coin, as shown below:

```
(list SEND_MESSAGE 0x3f
  (concat
    PROTOCOL_PREFIX
    (sha256tree
      (c STATUTES_STRUCT
        (c target_puzzle_hash winning_crt_bid_amount)
      )
    )
  )
  funding_coin_id
)
```

The malicious user could thus receive this message in a fake funding coin to spend the

recharge-auction coin without reverting the transaction.

Impact

The winner of the bid would not receive the CRT amount they won if a malicious user receives the message in a fake funding coin.

Recommendations

It is important to make sure that the funding coin ID is the expected one, and hence either the funding coin ID could be calculated in the puzzle itself or the recharge-win operation could be allowed to be run only by the winner of the bid.

While calculating the funding coin ID in the puzzle, it is important to make sure that the `cat_mod_hash`, `crt_tail_hash`, and the `inner_puzzle` of that CAT are all validated as the message could be received to invalid coins, if these are invalid.

Remediation

This issue has been acknowledged by Voltage Technologies Ltd., and fixes were implemented in the following commits:

- `badd1d21` ↗
- `b80b2663` ↗

3.10. Missing timestamp validation in collateral vault

Target	collateral_vault		
Category	Coding Mistakes	**Severity**	High
Likelihood	Medium	**Impact**	High

Description

The withdraw operation of the collateral vault allows providing the following parameters in the solution:

```
(@ args
  (
    withdraw_amount
    price_info
    liquidation_ratio
    current_timestamp
    statutes_cumulative_stability_df
    current_stability_df
  )
)
```

While the other parameters are verified via the statutes announcements, the `current_timestamp` is not verified to be the valid value via `ASSERT_BEFORE_SECONDS_ABSOLUTE` and `ASSERT_SECONDS_ABSOLUTE` conditions.

Impact

An attacker could use an old timestamp, which would result in an incorrect (old) value of `cumulative_stability_df`. This cumulative stability discount factor is used to calculate the undiscounted principal and hence the minimum collateral required in the vault such that it is above the liquidation threshold. While the old cumulative stability discount factor could only be used up until the timestamp of the `price_info`, manipulation of the value could lead to users withdrawing some amount from the vault, even if the vault may be liquidatable.

Recommendations

We recommend verifying the validity of `current_timestamp` via `ASSERT_BEFORE_SECONDS_ABSOLUTE` and `ASSERT_SECONDS_ABSOLUTE` conditions.

Remediation

This issue has been acknowledged by Voltage Technologies Ltd., and a fix was implemented in commit 4d420834 ↗.

3.11. Treasury withdrawals can create unauthorized treasury coins

Target	treasury		
Category	Business Logic	**Severity**	Medium
Likelihood	Low	**Impact**	Medium

Description

The treasury coins support any number of user-provided `CREATE_COIN` conditions, simply prepending one of its own recreate itself. Although it prevents users from splitting the treasury coin with the exact same puzzle hash, users may still create new, different treasury coins.

This is an issue because treasury-coin validity is lineage based. Specifically, there are states like `LAUNCHER_ID` and `RING_PREV_LAUNCHER_ID` that enforce important protocol invariants, such as in puzzles that require knowledge of the entire treasury's balance.

Impact

Users can create small pockets of treasury coins that look like the entire thing. This could result in auctions being started despite the real treasury balance being too high.

Recommendations

We recommend implementing stronger condition filtering on the coin.

Remediation

This issue has been acknowledged by Voltage Technologies Ltd., and a fix was implemented in commit `d98ddd52` ↗.

3.12. Unverified `MIN_DEPOSIT` value

Target	atom_announcer		
Category	Business Logic	**Severity**	High
Likelihood	Low	**Impact**	Low

Description

The atom announcer verifies the following parameters during the launch:

```
(assert
  (= DEPOSIT 0)
  (= DELAY 0)
  (= COOLDOWN_START 0)
  (= APPROVED 0)
  (= ATOM_VALUE 0)
  (= TIMESTAMP_EXPIRES 0)
  (= LAST_PENALTY_INTERVAL 0)
  (list ASSERT_MY_PARENT_ID LAUNCHER_ID)
)
```

However, it does not verify the `MIN_DEPOSIT` parameter, which should not be zero or a low value. As the atom announcer can only be approved via the governance, and hence the parameter can only be set via the governance, it is important to verify the validity of the parameter.

Impact

If the `MIN_DEPOSIT` value is low, the announcers could set their deposit to a very low value, and thus their penalty amount would be quite less. These announcers would be able to misbehave without penalty. This is not necessarily an issue because announcer approval passes through governance — which can verify their correct configuration. But this creates additional burden off chain, when the necessary checks could easily be done in the puzzles.

Recommendations

We recommend verifying the `MIN_DEPOSIT` amount in the atom_announcer puzzle.

Remediation

This issue has been acknowledged by Voltage Technologies Ltd., and fixes were implemented in the following commits:

- b7e18f7c ↗
- 33f0bfe8 ↗

3.13. Veto from proposal coin has limited effect

Target	governance		
Category	Business Logic	**Severity**	Low
Likelihood	High	**Impact**	Low

Description

One consideration by the authors of the protocol is that CRT votes towards proposals may be tied up or not immediately available to veto a bill with lower support. So governance coins are designed such that proposals can still veto other bills. But this mitigation might not quite interact well with how the protocol is envisioned to work.

The governance system is not designed with a voting mechanism. Rather, all proposals are a single governance coin and must be vetoed by a single governance coin. In order for CRT holders to actually collectively vote or collaborate on proposals or vetos, they must themselves design an inner puzzle for the governance coin that aggregates their tokens.

So, while funds in proposals can be used to veto other proposals, they cannot be aggregated with other funds to veto a proposal. Similarly, there is not an obvious way for funds in an aggregated proposal to be split towards vetoing other proposals, in the case where voters disagree.

Impact

Unwelcome proposals can cause disruption to in-progress proposals, if they need to be ended in order to free funds.

Recommendations

Voltage Technologies Ltd. may consider other designs (like an explicit voting system) to prevent this issue.

Remediation

Voltage Technologies Ltd. acknowledged this issue and added a reference voting implementation in commit `ca8ca0c5` ↗.

3.14. Outlier resolver messages are not protocol messages

Target	oracle_outlier_resolver		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The oracle outlier resolver puzzle communicates with the oracle by generating a condition as follows.

```
(list SEND_MESSAGE
  0x3f
  (concat "C" (sha256tree new_temp_ban_list)
    decision cooldown_interval amount)
  oracle_coin_id
)
```

We note that this message is generally not considered a protocol message, despite beginning with the prefix `C`. This is because only messages of length 33 are considered privileged.

```
(defun-inline is-valid-msg-cond (condition_body)
  (not
    (and
      (l condition_body)
      (l (r condition_body))
      (not (l (f (r condition_body))))
      (= (strlen (f (r condition_body))) 33)
      (= (substr (f (r condition_body)) 0 1) PROTOCOL_PREFIX)
    )
  )
)
```

Impact

This may be risky when maintaining the protocol, especially if condition filtering is added, as suggested in Finding 3.4. ↗. Namely, if the standard utilities for approving messages are used, the spender may be able to forge a protocol message.

Recommendations

We recommend including all components of the outlier resolution message inside the hash.

Remediation

This issue has been acknowledged by Voltage Technologies Ltd., and a fix was implemented in commit 8f4f1233 ↗.

3.15. Miscellaneous coding inaccuracies

Target	Protocol Wide		
Category	Optimization	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

> Note: Some of these issues were raised independently by the Voltage Technologies Ltd. team during the review period.

Description

Several coding inaccuracies were found during the review; they are detailed below.

The `start_only` parameter could be provided even after `AUCTION_STATE` is set

The vault_keeper_start_auction puzzle could be used to start the liquidation auction for a collateral vault. A vault keeps track of the state of a liquidation auction in the `AUCTION_STATE` curried arg. When the initiator starts the auction, they provide some parameters needed to start the auction in the `start_only` solution parameter. If a time-out occurs without all debt having been recovered but with collateral left in the vault, then a liquidation auction can be restarted by calling the start operation again. The `start_only` parameter is only needed for starting the auction the first time. But the `start_only` values could be provided even if the `AUCTION_STATE` is not nil, that is, even if the auction has already been started.

Redundant timestamp check during oracle update

User-provided timestamps are checked in conditions to confirm their accuracy. During price updates, an `ASSERT_SECONDS_ABSOLUTE` condition is produced in both the oracle and statues puzzles to check the same timestamp. It is only necessary to check the timestamp in either the oracle or statues puzzle.

Extraneous `temp_ban_list` logic in oracle mutation

The oracle mutation operation unconditionally assigns `nil` to `temp_ban_list_to_apply` and later sets `temp_ban_list` by checking if `temp_ban_list_to_apply` is nil or not. As the `temp_ban_list_to_apply` is already nil, the if condition would always go to the else statement, hence the if condition could be removed.

```
(price_to_apply temp_ban_list_to_apply outlier_info) (if OUTLIER_INFO
  ; outlier is set, block all price mutations, only price resolver can be used
  (x)
  (if (or (not cut_price_infos) (above-threshold cut_price_infos mean_price
    auto_approve_price_threshold_bps))
    ; we have a price outlier, set the outlier info
    (list 0 () (c () (c mean_price current_timestamp)))
    ; price is within threshold, no change, no outlier info
    (list (c mean_price current_timestamp) () ())
  )
)

; ...

temp_ban_list (if temp_ban_list_to_apply
      ; outlier is resolved, set temp ban list to what outlier resolution has
      (sha256tree temp_ban_list_to_apply)
      (if TEMP_BAN_LIST
        (if expired_ban_list
          ; ban has expired, reset it
          ()
          ; ban is still active
          TEMP_BAN_LIST
        )
        ()
      )
    )
```

Unnecessary `my_coin_id` parameter in savings vault

The savings vault expects a user to provide the `my_coin_id` solution parameter, but it is not used anywhere in the puzzle and thus could be removed along with the condition that verifies its validity.

Impact

The issues discussed above do not pose any security risks and are solely recommendations for enhancing the code.

Recommendations

We recommend resolving the aforementioned inaccuracies.

Remediation

This issue has been acknowledged by Voltage Technologies Ltd., and fixes were implemented in the following commits:

- 139aa3a6 ↗
- 9c685d58 ↗
- b3d095c7 ↗
- 30165496 ↗

3.16. Incorrect access control for announcer registry

Target	vault		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The announcer registry is not allowed to withdraw or deposit funds to the treasury. The treasury verifies the correctness of the message source by calculating the hash of the approver where `approver_mod_hash` is provided via the solution. The treasury verifies that `approver_mod_hash` is in `approval_mod_hashes` via the following,

```
(contains approval_mod_hashes approver_mod_hash)
```

as the approval mod hashes contain the hash of the following five mods, which also contain the announcer registry.

```
collateral_vault.clsp
surplus_auction.clsp
recharge_auction.clsp
savings_vault.clsp
announcer_registry.clsp
```

Impact

The announcer registry is given an additional access to deposit/withdraw funds from the treasury, but it should not be given that access as it may allow the announcer registry to withdraw funds from the treasury. As the conditions returned from the announcer registry are verified and do not contain any such condition to the treasury, any future changes may lead to potential issues.

Recommendations

We recommend to verify that the `approver_mod_hash` is from the first four mods of the `approval_mod_hashes`.

Remediation

Voltage Technologies Ltd. commented that future changes to the announcer registry and treasury are not expected. As such, because this is not currently a vulnerability, no remediation is necessary.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Vote aggregation

As explained in Finding 3.13. ↗, the governance system does not have an explicit voting mechanism and instead relies on the community to design a puzzle for aggregating votes into proposal and veto coins.

We note that until this system is in place, the protocol has some centralization risk. Individual users with large balances may push proposals without community consent, if there is no time to organize a veto.

4.2. Treasury-ring correctness relies on governance

Each coin in the treasury keeps track of where it is in a logical ring. This enables operations to be performed on the treasury as a whole. We note that the correctness of this ring is enforced by the governance system, not at all on chain. For the safety of the protocol, we recommend that off-chain tools are developed to make proposal validity clear to voters.

4.3. Confusion of atom-announcer identity

Possibly as a consequence of code changes, the atom announcer code has some misleading variable names. While announcers identify themselves by inner puzzle hash, other puzzles refer to this data as `launcher_id`. We recommend making it clear how announcers are to be identified for the purpose of bans.

4.4. Oracle-resolution voting

Like the governance system, oracle resolution does not have an explicit voting system. Depending on parameters, it can in fact be more risky. If some votes are used towards a resolution decision that is later maliciously overturned, these votes may not be available to fix the decision (depending on how the resolution delays are set).

4.5. Test code coverage

Overall, the coverage of unit tests is good, but integration tests could be improved. There are a few components of the protocol that are not well-tested, and adding more unit/integration tests could be helpful. Specifically,

1. testing the oracle mutation puzzle;
2. testing the end-to-end flows of BYC/CRT with collateral vaults, recharge auctions, and surplus auctions; and
3. testing end-to-end flows for statutes along with other coin spends.

Preferably, the test coverage for unit tests and integration tests should be evaluated independently. This helps not only testing individual coin-spend calls but also the complex integration that returns conditions that interact with other puzzles.

5. System Design

This provides a description of the high-level components of the system and how they interact, including details like a function's externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be upheld.

Not all components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

5.1. Design overview

Most coins that are part of the Circuit protocol are structured similarly. Generally, their authorization is proved through lineage — sometimes with Chia's standard singleton puzzle, sometimes using custom logic. A protocol coin usually has a collection of operations, each one governing a way the coin can be spent.

Operations

The majority of protocol coin types have operations stored by module hash in curried parameters. Precisely how these operations are formatted appears ad hoc, based on factors like access control. Here are a few examples.

- Collateral vaults have separate `OWNER_OPERATIONS` and `KEEPER_OPERATIONS` lists, and the collateral-vault puzzle manages access control for which operations can be invoked.
- Atom announcers have all operations stored in an `OPERATIONS` list, despite having both keeper and owner operations. The atom announcer passes whether the spender is the owner or keeper as a parameter to the operation, which can choose to fail if the spender is not authorized.
- The oracle coin only has two operations, taking their module hashes explicitly as parameters (`MUTATION_PROGRAM_HASH` and `OUTLIER_RESOLUTION_PROGRAM_HASH`) instead of in a list.

Regardless of how these operations are stored, the general pattern for how coins are spent is the same. The spender reveals an operation to execute and provides necessary data. The coin confirms the operation is permitted and executes the module. Then, the return value is used to determine the final spend conditions.

Coins obtain the target operation and parameters in two main ways.

1. For the keeper operations, this data is provided directly in the solution. See the collateral vault's keeper operations or all statutes operations for examples.
2. Coins that have owners (like governance CRTs) obtain the operation and parameters from `REMARK` conditions when the inner puzzle is solved.

The precise format of operation results also varies across coins. For example, the governance operations return both a bill and a list of conditions. The oracle operations just return the exact list

of conditions for the spend. The statutes operations output a list of seven items, including the new statutes and additional spend conditions.

Authorization

One of the key challenges in the protocol is distinguishing between trusted coins, carrying meaningful state, and coins that are not relevant to the protocol (or are malicious). Chia provides a few primitives for maintaining logical state on chain, like singleton and CAT. But these tools are not always easily applicable. For instance, the governance coins need to be stateful while holding CAT balance at the same time. This is addressed by implementing custom singleton-like behavior.

Since this pattern is so common across protocol coins, we will quickly outline how it works. It would be convenient if we could verify a coin's authorization simply by verifying that it comes from a correct puzzle hash. But this is not always possible, because coins can have stateful curried parameters. What stops an attacker from creating a new coin with the same puzzle hash and choosing these maliciously?

A solution is to carefully limit which coins can be spent. For instance, if we want to enforce requirements on a coin's creation (such as initial state, minimum balance, approval by governance, etc.), we require that either

1. the coin is created by a coin of the same type, or
2. the coin was created in accordance with these requirements

in order to spend it. Then, we carefully enforce what coins our trusted ones can create. For example, a collateral coin can only be spent if a) it was created by a collateral coin (which can set a nonzero collateral), or b) it has zero collateral. Thus, every collateral coin spent originates from a collateral coin with zero collateral.

Condition filtering

The protocol coins communicate through announcements, but there are situations where the spender needs to add their own. For example, when withdrawing collateral, the spender needs to tie the receipt of XCH to the collateral coin spent. Otherwise, the spend bundle might be split by an attacker to take the XCH for themselves. So protocol coins have to allow spenders to add announcements while preventing them from forging protocol communication.

This is accomplished by protecting specific condition formats. In particular, announcements and messages that begin with a specific byte and are 33 bytes long cannot be provided by spenders. Remarks that begin with that same byte are also protected.

5.2. The statues coin

The statutes coin is a standard singleton puzzle that is spent during nearly all Circuit actions. It is mainly responsible for 1) allowing governance to update protocol parameters and 2) announcing

these parameters to be enforced by other puzzles.

The puzzle's mutable state primarily consists of

- a list of statutes,
- price data, and
- fee and interest data.

Statutes are announced by index and value. For example, the protocol's price-delay parameter is communicated by an announcement of index 8, along with its value.

The protocol distinguishes between immutable statutes and statutes controlled by governance. Immutable statutes are announced with negative indexes baked into the contract, while governance statutes are indexed by their position in the statutes list.

In addition to the value of a statute, each entry in the statutes list additionally includes governance parameters for its next update. For instance, every statue has a proposal threshold that must be met before it can be changed. This data is set along with the value in governance proposals.

Operations

The statues coin can also be spent merely to announce the statute and price data. Additionally, it has mutation and update-price operations. The return value of these operations is formatted to include 1) the new state of the statues coin and 2) additional spend conditions.

Then, 1) is used to inform the `CREATE_COIN` condition that signals state changes to the parent singleton puzzle, and 2) is included in the spend conditions.

The mutation operation allows the spender to update the statutes list (or emit custom conditions) in the presence of approval by a governance coin. The update-price operation allows the spender to update the price according to the oracle's announcement. It also computes cumulative fees and interest.

Correctness and test coverage

The statutes coin needs to accurately announce protocol parameters and price data. It must only allow mutations when approved by governance and price updates from the oracle. Attempts to update statutes at out-of-bounds indexes should fail. And the statutes coin should correctly enforce its ordering rules that, among other things, prevent different values of the same statute from being used simultaneously.

Most of these properties are covered by the existing test suite. But we recommend better testing of how coins interact in the conditions they return.

5.3. Governance

A CRT holder's balance represents their power in protocol governance. Coins storing CRT balance are exercised by transferring them to a governance puzzle, with an appropriate owner. The owner can then use the coin to propose statutes changes or veto others' proposals.

The primary mutable state of the governance coin is its bill. In general, coins without a bill are considered veto coins while coins with a bill are proposals. All governance coins are granted the full ability to authorize statute mutations, so it is the responsibility of the governance puzzle to ensure that proposals pass through the correct voting process before being enacted.

Operations

A governance coin can be spent in a few ways, namely to

- propose a statute update,
- cancel a proposal,
- veto a different proposal,
- get vetoed by another proposal, or
- enact a mature proposal.

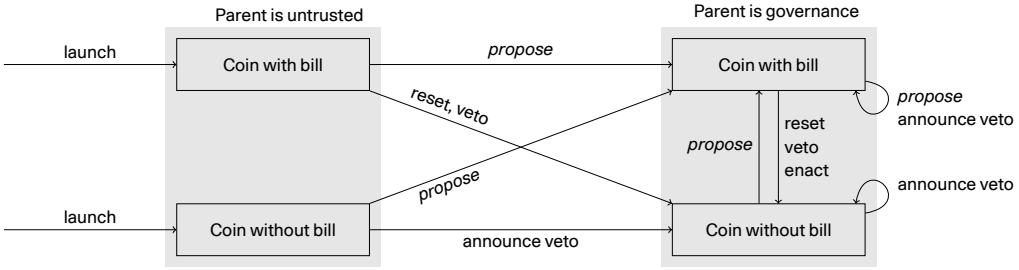
In implementation, the bill operation and coin amounts are extracted from `REMARK` and `CREATE_COIN` conditions in the inner puzzle's solution. Each operation returns a bill and a list of conditions. When the coin is spent, it creates a new governance coin with the bill. It also filters those conditions and includes them in the output.

Additionally, a governance coin can be unwrapped back to the owner into a regular CRT.

Authorization

Governance coins do not strictly enforce the state of their first spend. But it is critical that governance coins pass through the proposal process. This is because proposing a bill enforces requirements against statutes (such as the enactment delay and proposal threshold) that could be skipped if the bill were set in an untrusted launch.

This invariant is enforced by carefully managing state changes during spending. Then, the governance-coin bill data is only trusted if it was once spent from another governance coin. We can illustrate this reasoning.



Italicized is the propose operation, which is the only way to obtain a coin with a bill with a governance parent (which can be enacted). Notice that announce veto cannot be performed by a coin with a bill if its parent was not a governance coin. Otherwise, it would allow the proposal process to be skipped.

The last type of spend (unwrapping a governance coin) is not shown in the diagram but should be considered because the coin created may be a governance coin with a bill. We discuss this issue in Finding 3.1. ↗.

Correctness and test coverage

A governance coin must correctly enforce statutes parameters during proposals and correctly enact statutes mutations. Users should not be able to enact governance coins with bills they directly control by currying. The veto process should allow proposals to be vetoed by governance coins with greater balance during the veto period.

As with other coins, most of these properties are covered by the existing test cases. But we recommend including end-to-end tests that verify the behavior of output conditions.

5.4. Atom announcer

The atom announcers are responsible for providing the price of XCH. Announcer prices are used to update the Oracle price. Each such update requires a minimum of M-of-N announcer price announcements from distinct Announcers that have been approved by governance.

These announcers are custom singletons and are owned by the data providers. Any new announcer is approved by the governance after which they could register themselves and announce the atom values.

Operations

The following operations could be performed on the announcer.

- **Governance operations.** The announcer-govern operation can only be performed when it receives a message from the statutes to toggle the activation and updates the `APPROVAL` parameter.

- **Owner operations.** The announcer-register operation is used to register the puzzle and send a message to the announcer registry to register this puzzle. It is important to verify that the `APPROVED` value is set and the `inner_puzzle_hash` is provided to make sure that the operation can only be performed by the owner. The announcer-mutate operation can be used by the owner to set the atom value, and the delay is updated to the current timestamp + `DELAY`. Lastly, the announcer-configure operation allows to change the inner puzzle hash, deposit, delay, activation status, cooldown start, and the atom value.
- **Keeper operations.** The announcer-announce operation creates an announcement with the launched ID and the atom value. The announcer-penalize operation penalizes an announcer by reducing the amount of tokens from the puzzle.

Condition filtering and solution-parameters verification

The `REMARK`, `ANNOUNCEMENTS`, and `MESSAGE` conditions returned by the inner puzzles are filtered to make sure these are not protocol-specific announcements. In the case of the keeper operations, the conditions are directly passed as the solution. The conditions should also contain only one `CREATE_COIN` condition, which is used to get the operation to be performed, along with the arguments, the new puzzle hash, and the deposit amount.

It is also important for the singleton to prove its lineage when it is spent or verify that the parameters are 0 when it is initialized; this is to make sure that the curried parameters are not preset to nonzero values.

Test coverage

Intended branches

- ☑ All the operations return the expected conditions for the respective operations.
- ☑ The conditions returned must contain a `CREATE_COIN` condition to create a new coin.
- ☑ The governance operation can only be performed if it receives a message from the statute puzzle.
- ☑ The register operation should send a message to the announcer registry to register the announcer.
- ☑ Deactivation cooldown can be started using the configure operation.
- ☑ Penalize operation should decrease the deposit amount from the vault.

Negative behavior

- ☑ Attempting to melt an approved announcer should fail.
- ☑ Penalizing twice in the same interval should fail.

5.5. Announcer registry

The announcer registry keeps track of the registered announcers and allows the announcers to claim their rewards. It is a custom singleton that recreates itself after every spend. There are two opcodes allowed, which perform the following actions:

1. `MINT_OPCODE` — The mint operation is used to claim rewards. The operation can be performed by any registered announcer and automatically sends a rewards payment to each registered announcer. The puzzle calculates the CRT credits per announcer and sends a message to the issuance coin, which issues these coins. These coins are then distributed to the announcers.
2. `REGISTER_OPCODE` — The register operation could be used via announcers to register themselves in the announcer registry. The registry receives a message from the announcer and adds the announcer to the `ANNOUNCER_REGISTRY` variable.

Condition filtering and solution-parameters verification

There are no inner puzzles, and hence no condition filtering is required. The solution parameters are verified via announcements from the statutes.

The solution must also provide the lineage proof, which verifies the coin's validity.

Test coverage

Intended branches

- ☑ The mint opcode returns the expected conditions for asserting the announcements and sending messages to the issuance coin.
- ☑ The register opcode returns the expected conditions for receiving the message from atom announcer.
- ☑ The `ASSERT_MY_PARENT_ID` condition is returned, which checks the coin's lineage.
- ☑ The `CREATE_COIN` condition is returned.

Negative behavior

- ☑ The mint operation without any registered announcers should fail.
- ☐ The spend should revert if the lineage is incorrect.

5.6. Oracle

This price data broadcasted by approved announcers is used to feed the oracle coin. The coin puzzle is a standard singleton that stores

- a list of price data,

- a list of temporarily banned announcers, and
- data for outlier resolution.

The list contains pairs of prices and timestamps, with the oldest data at its head. A price is considered matured to the protocol when it has outlived the `STATUTE_PRICE_DELAY` protocol parameter. When new prices are added, all immature prices are kept; but only the newest mature price is retained.

Given this price data, the oracle's other primary responsibility is to announce them to the protocol (and other protocols). When an oracle is spent for an announcement, the spender can decide what prices are regarded as mature by providing a price delay. For example, the most recent price can be obtained by announcing with a delay of 0.

When oracle data is consumed by the protocol, it is enforced by other puzzles that the provided price delay matches the `STATUTE_PRICE_DELAY` parameter.

Mutation

In the normal course of operation, any user may add a new price once the latest price is sufficiently old (past the `PRICE_UPDATE_DELAY`). This is done by selecting enough unique approved announcers and providing their prices. The oracle checks via announcement that these announcers indeed announce the given prices.

The current timestamp and the provided prices are added to the list. Additionally, all matured prices except the most recent are removed.

If a substantial price deviation is detected, the oracle will enter an outlier resolution state. The price data is not updated and the oracle will not announce prices.

Outlier resolution

The coin is in the outlier resolution mode when `OUTLIER_INFO` is set. This parameter contains both voting state and the proposed price data that triggered the resolution.

The outlier voting mechanics are similar to the governance proposal process. Each proposed decision is associated with an amount of previously locked CRT. A resolution decision from a coin with more locked CRT replaces the previous decision.

Correctness and test coverage

While most functionality is covered by the existing test suite, we notice that the outlier resolution process does not have sufficient test coverage. As with the other components, negative test cases on the effects of spend conditions would additionally ensure correctness.

5.7. Treasury

The treasury is a collection of BYC coins that hold fees and absorb bad debt. It consists of multiple coins rather than just one in order to support simultaneous withdrawals. This poses a challenge for operations involving the entire treasury (such as knowing the total balance), which is handled by putting a ring structure on the coin set.

Authorization

The parent of a treasury coin may or may not be a treasury coin. The coin is trusted if the spender provides proof that its parent is a treasury coin. Otherwise, the coin can only be spent if it and its curried parameters are approved by governance.

Some actions require knowledge of the entire treasury's balance. The purpose of a recharge auction is to sell CRT for BYC to replenish the treasury and cover bad debt. The protocol only allows recharge auctions to be started if the treasury balance is below a threshold.

In order to enforce that all treasury coins are spent in these cases, the coins are placed in a ring. Each coin knows its launcher (original nontreasury creator) as well as the launcher of the next coin. Then, a protocol operation can require the spender to provide the coins in an order that matches the ring, which serves as proof that the entire treasury is being spent.

Under section 4.2. ↗, we briefly describe how the ring structure is enforced.

Spends

A treasury coin can be spent in two main ways. With the approval of governance, a treasury coin can complete its launch or have its ring structure updated. With the approval of some protocol coins, a treasury coin can have its balance changed.

Correctness and test coverage

The treasury coin must only update structural data when approved by governance. Changes to balance should be authorized by puzzles approved in statutes. Correct lineage proofs should be accepted, and incorrect ones rejected—this is particularly important for operations like recharge auctions that need knowledge of the entire treasury balance.

Voltage Technologies Ltd. includes positive and negative test cases for many of these facts, like verifying that withdrawal requests from unapproved puzzles fail. In general, we recommend improving test coverage of end-to-end flows like fee deposits.

5.8. Recharge auction

A recharge auction can be launched whenever the BYC balance of the treasury drops below the treasury minimum. The treasury auctions off CRT for BYC, and the auction can only be launched by

the governance approval.

Operations

The following operations could be performed on the recharge auction.

- **Governance operations.** The launch operation launches a recharge auction coin. This is only possible if it receives a custom message from the statutes puzzle to start the auction. This operation creates a new coin with the `LAUNCHER_ID` set.
- **Keeper operations.** The start-auction operation starts the auction by setting the `start_time`. The operation also verifies that the treasury balance is below treasury minimum by spending the treasury ring. The bid operation allows users to place a bid for CRT. If the bid is outbid by another bid, the BYC held in the auction coin is refunded to the target puzzle hash for the previous bidder. For the win operation, once the auction has concluded, the winner can mint an amount of CRT as specified in the bid by asserting the corresponding announcement from the auction coin.

Condition filtering and solution-parameters verification

The `REMARK`, `ANNOUNCEMENTS`, `MESSAGE`, and `CREATE_COIN` conditions should not be allowed to be directly passed via solution and returned without filtering. The function `filter-conditions` is responsible for filtering out these conditions. It does not allow the `CREATE_COIN` condition, which could lead to incorrect lineage.

The solution parameters are verified via the announcement assertions from the statutes coin.

Test coverage

Intended branches

- ☑ All the operations return the expected conditions for the respective operations.
- ☑ Bids must be accepted if the CRT price is higher.
- ☑ Wins can only be called if the auction time has ended.

Negative behavior

- ☑ Bids must fail if the price is lower than the previous bid.
- ☑ Bids must fail if the price increase is too low.

5.9. Surplus auction

A surplus auction can be triggered whenever the BYC balance of the treasury exceeds the treasury maximum plus the surplus auction lot amount, and it will auction a BYC amount equal to the lot balance. The amount of CRT offered in the winning bid gets melted, and the BYC is returned to the

bidder who won the bid.

Operations

The following operations can be performed on the surplus auction:

- **Keeper operations.** For the start-auction operation, to start the surplus auction, a keeper executes the start operation on a surplus auction coin in eve state. At the same time, a payout coin and one or several treasury coins must be spent to transfer an amount of BYC equal to the surplus auction lot amount from the treasury to the payout coin. For the bid operation, bidding occurs by spending the surplus auction coin using the bid operation. A bid involves the bidder increasing the amount of CRT offered, specifying a target puzzle hash to which they wish to receive the BYC amount and supplying the current timestamp. If this bid is outbid by another user, the previous user is repaid their CRT amount. For the win operation, when the auction concludes, the surplus auction coin is melted, reducing the supply of CRT in existence, and in return, the winner receives the surplus auction lot amount from the payout coin.

Condition filtering and solution-parameters verification

The `REMARK`, `ANNOUNCEMENTS`, `MESSAGE`, and `CREATE_COIN` conditions should not be allowed to be directly passed via solution and returned without filtering. The function `filter-conditions` is responsible for filtering out these conditions. It does not allow the `CREATE_COIN` condition, which could lead to incorrect lineage.

The solution parameters are verified via the announcement assertions from the statutes coin.

Test coverage

Intended branches

- ☑ All the operations return the expected conditions for the respective operations.
- ☑ An auction can only be launched if the sum of the amounts of the treasury coins are more than the max amount needed to launch the auction.

Negative behavior

- ☑ If the bid is too low, it should fail.
- ☑ Any protocol-specific announcements/messages should fail if passed in the solution.

5.10. Collateral vault

The collateral vault puzzle enables XCH to be locked and BYC to be minted based on XCH price. Their vaults are custom singletons and can be created permissionlessly by anyone. The inner puzzle of the collateral vault must be chosen/designed suitably as it could lead to the

operations/funds becoming inaccessible. Loans that are taken out from the collateral vault accrue stability fees. If the collateral locked falls below the liquidation threshold, the vault becomes eligible for liquidation and the owner operations become inaccessible. The vault also keeps track of the following:

- Collateral in the vault
- The principal amount of outstanding loans (total borrowed - total repaid)
- Auction state - the state of the liquidation auction if it has started
- Inner puzzle hash
- Discounted principal

The debt in the vault is not stored in a separate variable but calculated on an ad hoc basis as follows:

```
debt(t) = discounted_principal * CCSFDF_t  (where CCSFDF_t is the Current
    Cumulative Stability Fee Discount Factor at time t)
```

The stability fee could therefore be calculated as

```
SF = debt - principal
```

Operations

The following operations can be performed on the collateral vault:

- **Owner operations.** These operations can only be triggered by the owner.
  - The deposit-collateral operation increases the collateral amount in the vault.
  - The withdraw-collateral operation decreases the collateral amount in the vault while verifying that the collateral remaining is above the liquidation threshold.
  - The borrow-BYC (create debt) operation signals the tail to issue a certain BYC amount while verifying that the requested amount of BYC to mint does not take the vault below liquidation threshold.
  - The repay-BYC (debt + fees) operation repays the provided amount to the vault and transfers the stability fee to a treasury coin.
  - The transfer-the-vault operation allows changing the inner puzzle hash of the vault.
  - The transfer-fees-to-treasury operation transfers the stability fee to the treasury.
- **Keeper operations.** These operations can be triggered by keepers.
  - The start-auction operation sets the `auction_state` parameters. If the collateral in the vault falls below the minimum collateral required, an auction could be started. A small amount from the total fee is stored as the

initiator-incentive balance to incentivize keepers to initiate liquidations.
- The bid operation allows keepers to bid on the collateral. The puzzle calculates the amount of XCH the bidder will receive based on the amount of BYC they bid. (Read more about an issue discovered in the calculation in Finding 3.6. ↗.) The bid amount first goes to the initiator, then to the treasury, and the remaining amount is melted.
- For the recover-bad-debt operation, if all the collateral has been sold off to bidders without all debt being repaid, this puzzle could be used. This amount is withdrawn from the treasury coin and melted.

Condition filtering and solution-parameters verification

The vault operations, args, and the statutes inner puzzle hash is provided via the `REMARK` condition from the inner puzzle if the operation is an owner operation and directly from the solution provided by the keeper if the operation is a keeper operation. It is important for the puzzle to accurately filter out the `REMARK` condition. Furthermore, it is crucial that the inner puzzle does not output any other protocol-specific conditions and does not have any `CREATE_COIN` condition as it might lead to incorrectly proving lineage with curried-in parameters.

The parameters provided in the solution must also be verified to be accurate, and this is done via either asserting their values from the statutes announcements or returning other conditions from the vault that verify their correctness. For instance, the `statutes_cumulative_stability_fee_rate` and `current_timestamp` are verified via the statutes announcement and the `ASSERT_SECONDS_ABSOLUTE/ASSERT_BEFORE_SECONDS_ABSOLUTE` condition, respectively. Failure to verify any such parameter might lead to critical issues as, in that case, these parameters could be set to user-provided values without being verified (read more about this in Finding 3.10. ↗).

Test coverage

Intended branches

- ☑ All the operations return the expected conditions for the respective operations.
- ☑ Deposit should increase the collateral amount in the vault.
- ☑ Withdrawal should decrease the collateral amount and should only be possible if it does not cause the vault to cross the liquidation threshold.
- ☑ Liquidation can only start for an undercollateralized vault.
- ☑ Melting the amount equal to BYC minted + fee should repay the debt.

Negative behavior

- ☑ Minting BYC that causes the vault to cross the liquidation threshold should fail.
- ☑ Minting less than the minimum amount should fail.
- ☑ Bidding too much or a negative amount should fail.
- ☑ Minting a negative amount of BYC should fail.

5.11. Savings vault

Savings vaults are BYC CAT singletons that can be permissionlessly created by anyone to earn yield. The amount deposited in the vault is converted to `DISCOUNTED_DEPOSIT`, which is stored in the vault and used to calculate the accumulated yield by the vault. There are two possible operations on the vault: deposit and withdraw.

While depositing and withdrawing, users can provide the treasury-coin info, which is used to calculate the treasury coin from which interest is paid.

Condition filtering and solution-parameters verification

The puzzle filters out the `REMARK`, `ANNOUNCEMENTS`, and `MESSAGE` conditions but allows more than one `CREATE_COIN` condition. This is a serious concern as it allows users to create a savings vault with parameters already curried in and still prove the lineage (read more about this in Finding 3.3. ↗).

Test coverage

Intended branches

- ☑ Deposits and withdrawals are functioning as expected.

Negative behavior

- ☑ Withdrawal must fail if treasury coin does not have enough balance.

5.12. BYC and CRT

Both the BYC and CRT tokens are standard CATs. Their tail puzzles support creating new coins and burning coin balance.

BYC tail

The BYC tail listens for tagged messages from collateral vault coins, which indicate how much is to be issued or melted. It checks that the sender has a puzzle matching the collateral vault by looking up the hash in statutes.

CRT tail

When melting is attempted, the CRT tail checks that the action and amount are authorized by the surplus coin. For issuing, the tail confirms that it is either protocol genesis, authorized by the recharge auction or authorized by the announcer registry. Like the BYC tail, these puzzle hashes are checked against the list committed in `STATUTES_APPROVAL_MOD_HASHES_HASH`.

Correctness and test coverage

Tests are included for verifying the main operations, which are issuing and melting tokens with the approval of particular puzzles. We recommend including negative tests to confirm that minting is not possible. Additionally, we recommend testing the end-to-end flows with collateral vaults, recharge auctions, and so forth rather than simply the puzzle's execution with those parameters.

6. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Chia mainnet.

During our assessment on the scoped Circuit DAO puzzles, we discovered 16 findings. Three critical issues were found. Seven were of high impact, one was of medium impact, two were of low impact, and the remaining findings were informational in nature.

6.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.