FINTEH.ORG | AUDIT

DIFFERENCE AUDIT REPORT 2204041

CWD.GLOBAL BLOCKCHAIN CODE

CUSTOM bb63cb902eda3065200e4ca07da2da679c729719

BASE 5bad558ee1b9a5d3e401e9149a1482f400709b82

Abstract

This audit of CWD.global changes is with respect to the open source fork based on the original BitShares 2.0 code.

Audit conducted by experts from the Fintech experts pool of the non-governmental organization FinTechAssociation (finteh.org) and partners of the NGO FintechAssociation, who are actively working in the Graphene space.

Disclaimer

This audit is the official audit report of the non-governmental organization FinTechAssociation (finteh.org) and has an overall liability waiver.

Due to the peculiarities of creating a fork of BitShares 2.0, the “git diff" method will not work, which significantly complicates the audit process. The commit history may be incorrectly copied and overwritten, including submodules.

The report makes no warranties regarding the security of the code. One audit is not enough. Several independent audits and a public bug bounty program to ensure the code base is secure, recommended for all public projects.

Also, an audit report is not investment advice, totally.

Project overview

Two publicly available GitHub repositories were provided for the audit:

Custom source

`github.com/crowdwiz-biz/crowdwiz-core/commit/bb63cb902eda3065200e4ca07da2da679c729719`

Base Source

`github.com/bitshares/bitshares-core/commit/5bad558ee1b9a5d3e401e9149a1482f400709b82`

In additional, technical specifications and business documentation

github.com/finteh/crowdwiz-core/blob/master/docs/cwd_tech_review_ENG.pdf

github.com/finteh/crowdwiz-core/blob/master/docs/cwd_advantages_review_ENG.pdf

Procedure

The `2.0.20190319` tag release of bitshares-core was compared and reviewed code changes against.

Tasks:

1. Audit changes:
    a. description on what blockchain operations something different happens.
    b. robustness and safety (potential bugs, attack angles, etc.), and anything unusual or against the best practices of blockchain backend code.
2. Audit of compliance with statements in public documents – technical specifications and business documentation, include business logic and gamezone related stuff: flipcoin, lottery, matrix.

Steps:

1. Setting up a local repository
2. Building the software (requires some older dependencies)
3. Preliminary review of code base
4. Review graphene protocol

5. Analysis
6. Starting to sync with the existing chain
7. Cleaning up preliminary findings and add recommendations

The next components was analyzed the basic structure of the changes in blockchain code and review the following components:

1. The on-chain protocol: `libraries/chain/protocol/` and `libraries/chain/include/graphene/chain/protocol/`
2. The blockchain database and business logic in `libraries/chain/` (without `protocol`)
3. Modifications to `graphene:app` (in `libraries/app`)
4. Modifications to the wallet in `libraries/wallet`
5. Modifications to plugins in `libraries/plugins`
6. Modifications to the programs in `programs/`
7. Additional unit tests in `tests/`

Building the software

Introduction

The codebase requires:

- gcc-10
- boost 1.66
- openssl-1.0

cmake:

```
CC=gcc-10 CXX=g++-10 cmake -DBOOST_ROOT=/home/dev/opt/boost_1_66_0
-DCMAKE_BUILD_TYPE=Release -DBoost_NO_BOOST_CMAKE=ON
-DOPENSSL_ROOT_DIR=/usr/lib/openssl-1.0/
-DOPENSSL_CRYPTO_LIBRARY=/usr/lib/openssl-1.0/libcrypto.so
-DOPENSSL_INCLUDE_DIR=/usr/include/openssl-1.0 ../crowdwiz-core
```

Conclusions

The software builds and starts sync starts synchronizing with an existing blockchain through the hard coded seed node synchronizing with an existing blockchain through the hard coded seed nodes:

- `157.230.169.44:1776`
- `185.193.125.52:1776`
- `185.193.125.52:1776`
- `185.193.125.52:1776`
- `185.193.125.52:1776`
- `68.183.242.188:1776`

The random number generators used as well performance due to nested modifiers and multiple levels or referral program can have a concern

finteh.org recommended

SSL version hardcoded by version 1.1.0 was deprecated on 10 September 2019. Very strong recommendation to update it to actual version 1.1.1 or 3.0 accessible from 7 September 2021.

On-chain protocol (graphene/chain/protocol)

New operations

Operations 49 to 125 are new to the codebase:

```
/* 49  */   account_status_upgrade_operation,
/* 50  */   flipcoin_bet_operation,  //GAMEZONE
/* 51  */   flipcoin_call_operation,  //GAMEZONE
/* 52  */   flipcoin_win_operation,  //VOP
/* 53  */   flipcoin_cancel_operation,  //VOP
/* 54  */   flipcoin_loose_operation,  //VOP
/* 55  */   lottery_goods_create_lot_operation,  // GAMEZONE
/* 56  */   lottery_goods_buy_ticket_operation,  // GAMEZONE
/* 57  */   lottery_goods_send_contacts_operation,  // GAMEZONE
/* 58  */   lottery_goods_confirm_delivery_operation,  // GAMEZONE
/* 59  */   lottery_goods_win_operation,  // GAMEZONE, VIRTUAL
/* 60  */   lottery_goods_loose_operation,  // GAMEZONE, VIRTUAL
/* 61  */   send_message_operation,
/* 62  */   matrix_open_room_operation,
/* 63  */   matrix_room_closed_operation, // GAMEZONE, VIRTUAL
/* 64  */   matrix_cell_filled_operation, // GAMEZONE, VIRTUAL
/* 65  */   create_p2p_adv_operation, // EXCHANGE
/* 66  */   edit_p2p_adv_operation, // EXCHANGE
/* 67  */   clear_p2p_adv_black_list_operation, // EXCHANGE
/* 68  */   remove_from_p2p_adv_black_list_operation, // EXCHANGE
/* 69  */   create_p2p_order_operation, // EXCHANGE
/* 70  */   cancel_p2p_order_operation, // EXCHANGE
/* 71  */   autocancel_p2p_order_operation, // EXCHANGE, VIRTUAL
/* 72  */   autorefund_p2p_order_operation, // EXCHANGE, VIRTUAL
/* 73  */   call_p2p_order_operation, // EXCHANGE
/* 74  */   payment_p2p_order_operation, // EXCHANGE
/* 75  */   release_p2p_order_operation, // EXCHANGE
/* 76  */   open_p2p_dispute_operation, // EXCHANGE
/* 77  */   reply_p2p_dispute_operation, // EXCHANGE
/* 78  */   resolve_p2p_dispute_operation, // EXCHANGE
/* 79  */   lottery_goods_refund_operation,  // GAMEZONE, VIRTUAL
/* 80  */   credit_system_get_operation, //FINANCIAL
/* 81  */   credit_total_repay_operation, //FINANCIAL, VIRTUAL
/* 82  */   credit_repay_operation, //FINANCIAL
/* 83  */   credit_offer_create_operation, //FINANCIAL
/* 84  */   credit_offer_cancel_operation, //FINANCIAL
/* 85  */   credit_offer_fill_operation, //FINANCIAL
/* 86  */   pledge_offer_give_create_operation, //FINANCIAL
/* 87  */   pledge_offer_take_create_operation, //FINANCIAL
/* 88  */   pledge_offer_cancel_operation, //FINANCIAL
/* 89  */   pledge_offer_fill_operation, //FINANCIAL
/* 90  */   pledge_offer_repay_operation, //FINANCIAL
/* 91  */   pledge_offer_auto_repay_operation, //FINANCIAL, VIRTUAL
/* 92  */   committee_member_update_gamezone_parameters_operation,
/* 93  */   committee_member_update_staking_parameters_operation,
/* 94  */   poc_vote_operation, //PoC
/* 95  */   poc_stak_operation, //PoC
/* 96  */   poc_staking_referal_operation, //PoC, VIRTUAL
/* 97  */   exchange_silver_operation,//PoC
/* 98  */   buy_gcwd_operation,
/* 99  */   approved_transfer_create_operation,
/* 100 */   approved_transfer_approve_operation,
/* 101 */   approved_transfer_cancel_operation,
/* 102 */   approved_transfer_open_dispute_operation,
/* 103 */   approved_transfer_resolve_dispute_operation,
/* 104 */   mass_payment_operation,
/* 105 */   mass_payment_pay_operation,
/* 106 */   change_referrer_operation,
/* 107 */   gr_team_create_operation,
/* 108 */   gr_team_delete_operation,
/* 109 */   gr_invite_send_operation,
/* 110 */   gr_invite_accept_operation,
/* 111 */   gr_player_remove_operation,
/* 112 */   gr_team_leave_operation,
/* 113 */   gr_vote_operation,
/* 114 */   gr_assign_rank_operation,
/* 115 */   gr_pay_rank_reward_operation,
/* 116 */   gr_pay_top_reward_operation,
/* 117 */   gr_apostolos_operation,
/* 118 */   gr_range_bet_operation,
/* 119 */   gr_team_bet_operation,
/* 120 */   gr_range_bet_win_operation,
/* 121 */   gr_range_bet_loose_operation,
/* 122 */   gr_team_bet_win_operation,
/* 123 */   gr_team_bet_loose_operation,
/* 124 */   gr_range_bet_cancel_operation,
/* 125 */   gr_team_bet_cancel_operation
```

The actual business logic behind those operations is implemented in `graphene::chain` and will be reviewed at a later point in time.

Changes to chain parameters

The chain parameters define many parameters in a way that they can be changed by the `committee-account` later on. These parameters include `block_time` and fees.

Two changes stuck out:

- `committee_proposal_review_period`, and
- `maximum_proposal_lifetime`

where fixed values have been added instead of using the recommended `GRAPHENE_DEFAULT_*` macros.

finteh.org recommended

Add back `GRAPHENE_DEFAULT_*` macros in `libraries/chain/include/graphene/chain/config.hpp` instead

Additional parameters have been added:

```
uint16_t   ref_01_percent_of_fee           = (20*GRAPHENE_1_PERCENT);
///< percent of transaction fees paid to ref_01
uint16_t   ref_02_percent_of_fee           = (10*GRAPHENE_1_PERCENT);
///< percent of transaction fees paid to ref_02
uint16_t   ref_03_percent_of_fee           = (6*GRAPHENE_1_PERCENT);
///< percent of transaction fees paid to r ef_03
uint16_t   ref_04_percent_of_fee           = (4*GRAPHENE_1_PERCENT);
///< percent of transaction fees paid to r ef_04
uint16_t   ref_05_percent_of_fee           = (3*GRAPHENE_1_PERCENT);
///< percent of transaction fees paid to r ef_05
uint16_t   ref_06_percent_of_fee           = (2*GRAPHENE_1_PERCENT);
///< percent of transaction fees paid to r ef_06
uint16_t   ref_07_percent_of_fee           = (2*GRAPHENE_1_PERCENT);
///< percent of transaction fees paid to r ef_07
uint16_t   ref_08_percent_of_fee           = (3*GRAPHENE_1_PERCENT);
///< percent of transaction fees paid to r ef_08
uint8_t    status_levels_00      = 0;
uint8_t    status_levels_01      = 3;
uint8_t    status_levels_02      = 5;
uint8_t    status_levels_03      = 7;
uint8_t    status_levels_04      = 8;
bool       denominator           = true;
uint16_t   status_denominator_00           = (0*GRAPHENE_1_PERCENT);
uint16_t   status_denominator_01           = (70*GRAPHENE_1_PERCENT);
uint16_t   status_denominator_02           = (80*GRAPHENE_1_PERCENT);
uint16_t   status_denominator_03           = (90*GRAPHENE_1_PERCENT);
uint16_t   status_denominator_04           = (100*GRAPHENE_1_PERCENT);
uint8_t    denominator_bonus_level         = 4;
uint8_t    nv_levels  = 30;
uint8_t    min_nv_status         = 3;
uint8_t    ref_levels            = 8;
uint8_t    compression_levels    = 30;
bool       compression           = true; /// Referral reward program
have compression
bool       cashback   = true; /// Cashback - first level reward paid to
self
bool       allow_non_partner_register      = true; ///
uint8_t    min_not_compressed    = 3;
share_type            compression_limit     =
(GRAPHENE_BLOCKCHAIN_PRECISION * int64_t(0));
uint64_t   referral_statistic_seconds      = 90*24*60*60;
account_id_type       root_account         = account_id_type(27);
share_type            nv_level_threshold_01      =
(GRAPHENE_BLOCKCHAIN_PRECISION * int64_t(15000));
share_type            nv_level_threshold_02      =
(GRAPHENE_BLOCKCHAIN_PRECISION * int64_t(35000));
share_type            nv_level_threshold_03      =
(GRAPHENE_BLOCKCHAIN_PRECISION * int64_t(100000));
share_type            nv_level_threshold_04      =
(GRAPHENE_BLOCKCHAIN_PRECISION * int64_t(250000));
share_type            nv_level_threshold_05      =
(GRAPHENE_BLOCKCHAIN_PRECISION * int64_t(450000));
share_type            nv_level_threshold_06      =
(GRAPHENE_BLOCKCHAIN_PRECISION * int64_t(700000));
share_type            nv_level_threshold_07      =
(GRAPHENE_BLOCKCHAIN_PRECISION * int64_t(1500000));
share_type            nv_level_threshold_08      =
(GRAPHENE_BLOCKCHAIN_PRECISION * int64_t(3000000));
share_type            status_threshold_01   =
(GRAPHENE_BLOCKCHAIN_PRECISION * int64_t(3000));
share_type            status_threshold_02   =
(GRAPHENE_BLOCKCHAIN_PRECISION * int64_t(2000));
share_type            status_threshold_03   =
(GRAPHENE_BLOCKCHAIN_PRECISION * int64_t(1000));
share_type            status_threshold_04   =
(GRAPHENE_BLOCKCHAIN_PRECISION * int64_t(0));
uint16_t   nv_level_reward_01    = (1*GRAPHENE_1_PERCENT);
uint16_t   nv_level_reward_02    = (2*GRAPHENE_1_PERCENT);
uint16_t   nv_level_reward_03    = (3*GRAPHENE_1_PERCENT);
uint16_t   nv_level_reward_04    = (4*GRAPHENE_1_PERCENT);
uint16_t   nv_level_reward_05    = (5*GRAPHENE_1_PERCENT);
uint16_t   nv_level_reward_06    = (6*GRAPHENE_1_PERCENT);
uint16_t   nv_level_reward_07    = (7*GRAPHENE_1_PERCENT);
uint16_t   nv_level_reward_08    = (8*GRAPHENE_1_PERCENT);
```

The meaning of those variables only becomes clear when reviewing the business logic in `graphene::chain` at a later point in time.

NOTE

In order to ensure that all parameters of new operations are properly reflected, it makes sense to use the script `programs/build_helpers/check_reflect.py`, it will ensure all attributes are properly reflected! In order for this script to work, the codebase needs to be compiled (in particular, it requires `cmake doxygen` to be run).

Interestingly, some definitions of operation structures come with predefined values for attributes, such as:

```
  struct create_p2p_adv_operation : public base_operation
{
      struct fee_parameters_type {
           uint64_t fee          = 5 * GRAPHENE_BLOCKCHAIN_PRECISION;
           uint32_t price_per_kbyte = 1 *
GRAPHENE_BLOCKCHAIN_PRECISION;
      };
      asset            fee;
      account_id_type            p2p_gateway;
      bool             adv_type = true;
      string           adv_description;
      share_type       max_cwd;
      share_type       min_cwd;
      share_type       price;
      string           currency;
      uint32_t         min_p2p_complete_deals = 0;
      uint8_t          min_account_status = 0;
      uint32_t         timelimit_for_reply = 3600;
      uint32_t              timelimit_for_approve = 3600;
      string           geo;
      account_id_type            fee_payer() const { return
p2p_gateway; }
      void             validate()const;
      share_type       calculate_fee(const fee_parameters_type& k)const;
};
```

It is worth noting that these values (e.g. `true`, `0`, `3600`) are not taken into consideration anywhere. The reason is that the variables are not `optional<T>` and thus when using this (and other operations), a value for those attributes must be provided by the user which replaces the values here!

Virtual operations (such as `aurorefund_p2p_order_operation` and others) should define the `validate()` method and throw an exception. Usually we have these in the header files for virtual operations:

```
void            validate()const { FC_ASSERT( !"virtual operation" ); }
```

This ensures that virtual operations do not end up on the blockchain. Everything else equal, this is the only technical difference to a regular operation!

We highly suspect that brackets are missing in `libraries/chain/protocol/exchange.cpp`:

```
void payment_p2p_order_operation::validate()const
{
FC_ASSERT( fee.amount >= 0 );
if (file_hash)
      FC_ASSERT( fc::raw::pack_size(file_hash)>=10);
```

```
     FC_ASSERT( fc::raw::pack_size(file_hash)<1000 );
}
```

Business logic (graphene/chain)

The modifications that took place in `libraries/chain/*.cpp` files and thus implemented the "business logic" of each new operation have been reviewed.

Configuration

In `config.hpp` you predefine an account and link it with id `x.y.65735`. This requires that at least `65735+1` accounts exist on the blockchain at launch. Otherwise you risk someone else registering this account by chance. Further, if your code makes use of this account by `get_account` instead of `find_account` you risk exceptions that may cause the blockchain to get stuck!

```
/// Represents Apostolos Account
#define GRAPHENE_APOSTOLOS_ACCOUNT
(graphene::chain::account_id_type(65735))
```

Furthermore, we still found these in the code: account_id_type(65735). Should be replaced with the macro above.

You may further want to get rid of the STEALTH asset (which only exists on the original BitShares blockchain).

```
#define GRAPHENE_FBA_STEALTH_DESIGNATED_ASSET (asset_id_type(743))
```

The maintenance interval has been reduced to every hour. Considering the amount of extra efforts that need to take place in the maintenance interval, it becomes crucial to estimate the time the maintenance block takes in order to not miss blocks.

```
#define GRAPHENE_DEFAULT_MAINTENANCE_INTERVAL  (60*60) // seconds, aka:
1 hour
```

Use of `enums`

In `libraries/chain/include/graphene/chain/gamezone_object.hpp` (and others), we ran into this:

```
uint8_t status = 0; //0 = new, 1 = sold out, 2 = completed, 3 = owner
paid, 4 = canceled
```

in the `lottery_goods_object`. We encourage you to use an `enum` instead (see for example: `vesting_balance_object.hpp`).

Modifications to graphene:app (in libraries/app)

In `graphene::app`, the only changes were due to additional API calls within the `database_api`. While we encourage building custom plugins for reach functional aspect of a graphene blockchain (e.g. one for the `lottery/coinflipping`, one of p2p adv, and another for the great race) adding APIs to the quite general `database_api` is just as fine if the additional compile complexity is acceptable.

The list of new API calls in `database_api`:

```
vector<flipcoin_object> get_active_flipcoin() const;
vector<scoop_lots> lottery_goods_get_active() const;
vector<lottery_goods_object> lottery_goods_get_by_owner(const
std::string owner, uint8_t status) const;
vector<lottery_goods_object> lottery_goods_get_by_winner(const
std::string winner, uint8_t status) const;
vector<lottery_goods_object> lottery_goods_need_contacts(const
std::string winner) const;
vector<lottery_goods_object> lottery_goods_get_by_status_limit(uint16_t
limit, uint8_t status) const;
vector<matrix_object> get_active_matrix() const;
vector<matrix_rooms_object> get_rooms_by_player(const std::string
player) const;
vector<p2p_adv> get_p2p_adv(const std::string account_id_or_name, bool
adv_type) const;
vector<p2p_adv_object> get_my_p2p_adv(const std::string
account_id_or_name, uint8_t status) const;
vector<p2p_ord> get_p2p_orders(const std::string account_id_or_name)
const;
vector<p2p_ord> get_gateway_p2p_orders(const std::string
account_id_or_name, uint8_t status) const;
size_t get_gateway_total_orders(const std::string account_id_or_name)
const;
vector<credit_offer_object> credit_get_offers() const;
vector<credit_offer_object> credit_get_offers_by_account(const
std::string account_id_or_name) const;
vector<account_statistics_object> credit_get_debitors(const std::string
account_id_or_name) const;
vector<pledge_offer_object> pledge_get_offers() const;
vector<pledge_offer_object> pledge_get_offers_by_account(const
std::string account_id_or_name) const;
```

```
vector<gr_rating_obj> gr_get_rating(const std::string rating_type)
const;
vector<gr_invite_obj> gr_get_invites(const std::string
account_id_or_name) const;
vector<gr_range_bet_api_obj> gr_get_range_bets() const;
vector<gr_team_bet_api_obj> gr_get_team_bets() const;
```

We have not investigated the actual implementations of each API call but the first impression was that they have mostly been implemented according to reasonable standards.

Some API calls come with their own returning object structure.
Our only remark would be to limit the outcome of each API request by `100` items and implement pagination where necessary.
The default value of `enable_subscribe_to_all` has been set to `true` which results in the behavior that all objects that are queries by the user are also subscribed to. This not only leads to an increased per-connection state on the server, but also comes with potentially significant more traffic between the server and the subscribing user. Given that subscriptions are client-side, please consider disabling this by default.

Modifications to the wallet in `libraries/wallet`

Only a few new calls have actually been implemented in the `cli_wallet`:

```
signed_transaction register_test_account(string name, string
referrer_account);
signed_transaction upgrade_status(string name, uint8_t status);
signed_transaction open_room(matrix_id_type matrix_id,
           string player,
           uint8_t matrix_level,
           bool broadcast = false);
```

There seem to be implementations for a few other calls, but those were commented out.
Further, the `review_period_seconds` attribute for `proposal_create_operation` has been modified on two occupations which would break in case you re-enable the evaluation of said parameter on the blockchain.

Modifications to plugins in `libraries/plugins`

The only changes in the plugin relate to the replacing of `BitShares` into `CrowdWiz`. No functional changes were made.

Modifications to the programs in `programs/`

For some reason, the `witness_node` and the `delayed_node` no longer accept `plugins` as configuration parameters. No other functional changes have been made.

Additional unit tests in `tests/`

The entire `tests/` folder from bitshares has been removed for unknown reasons. We encourage you to implement unit tests for every single functionality of your blockchain to prevent regressions on upgrades and misbehavior.

gr_interval

There is a global pseudo counter `current_gr_interval` that increments every maintenance interval and restarts after `14`. Depending on the interval, various aspects of the blockchain take place.

Operations

For the ordering of operations into categories, we used the operations names and comments as per `operations.hpp`. The categories seem to be independent of each other, when discussing proof of stake.

Proof of Crowd Staking

> **finteh.org recommended**
>
> ***The "proof-of-crowd staking" consensus must be validated by tokenomics experts to conform to the principles of game theory and economics.***

Here we investigate the operations which are defined in `poc_evaluator.cpp`:

- `poc_vote_operation`: This operations takes 3 assets `poc[3|6|12]_vote` and deduct them from the account balance. Asset must be `CWD` and the amount must be `>=0`. It appears that the assets are moved to accumulated_fees and are thus a fee for the user who gets his account updated for the corresponding amount in the corresponding staking period.

- `poc_stak_operation`: This operation seems to allow to stake an amount that is higher than a minimum that is derived from some committee parameters depending on the duration (`staking_type`). The equations could be implemented slightly easier to review but seem to be o.k. A percentage of the `stak_amount` in the operation is moved into newly created vesting balances for the user that is vesting for a certain amount in time (3, 6, 12 months).
    - In the case of 3 months, the first vesting balance contains the newly created staking reward that is unlocked and can be claimed linearly over 3 months. The second vesting balance contains the `stak_amount` that is deducted from the account and is locked for the entire duration but vests fully right after it.
    - In the case of 6 months, there is only one vesting balance containing both, the initial `stak_amount` plus reward and vests after 6 months.
    - In the case of 12 months, there is only one vesting balance containing both amount and reward. The balance starts vesting immediately over a duration of 12 months.

    Afterwards, it appears that a 8 level deep referral program takes place where each level gets a fraction of the `stak_amount` (newly issued token). The fraction depends on the depth in the referral program, the `referral_status` of the referrer as well as global staking parameters.

    In the case the operation is called while `gr_interval` in `[2,4,6,9,11,13]`, an additional change in the `gr_teams` volumes takes place.

    Further changes w.r.t. the credit system takes place and modifies the `allowed_to_repay` attribute of the referrer as well as the monthly income value in account statistics.

- `poc_staking_referal_operation`: This is a virtual operation that is created in case the referrer account pays back some amounts in a credit.

finteh.org recommended

To effectively review these mechanics, formal specifications are required! A major concern could be the nested modifications of database objects which take a considerable amount of time. It might be necessary (depending on growth and usage) to optimize here in order not to slow down block production unnecessarily.

Gamezone

In contrast to proof of crowd staking, gamezone comes with a gamezone objects file which usually contains contract specific objects (think, accounts or assets). In this case there is a `gamezone_object.cpp` (all implementation details are in the .hpp file). The operations are managed in `gamezone_evaluator.cpp` as usual.

Gamezone comes with its own set of global parameters that define `matrix_lasts_block`, `matrix_idle_blocks` as well as `matrix_level_*_cells` and `matrix_level_*_price` for the matrix game. There is `flipcoin_min_bet_amount` for the flip coin game and `lottery_goods_total_participants`, `lottery_goods_expiration` for lottery. All variables are predefined with values we could validate.

Flipcoin

Flipcoin appears to be a game to deal with simply flipping a coin for heads or tails. There is a `flipcoin_object` define that stores previous bets together with a `status` as well as an index

The following operations have been implemented:

- `flipcoin_bet_operation`: This operation creates a new `flipcoin_object` independent of whether there already is an object that it could match to. The matching appears to take place elsewhere. Each bet comes with a user provided nonce and a user-provided expiration time. Initial status is `0`. This operation deducts the betting amount from the user's balance.
- `flipcoin_call_operation`: The purpose of this operation is to match an existing `flipbet_object` as created with the previous operation. Given that this is

user-controlled, the UX would be that users are provided with existing pending games to pick one to complete with this operation. Apparently, a single bet can be matched with one or more call operations.

- `flipcoin_cancel_operation`: This operation is not used anywhere, e.g. it is not implemented. The existence of this operation indicates that a bet may be canceled before being matched, however, it has no meaning right now.
- `flipcoin_win_operation`: Virtual operation pushed when processing the bets
- `flipcoin_loose_operation`: Virtual operation pushed when processing the bets

The bets themselves are resolved in `db_update.cpp` every block for all flipcoin objects with expiration in the past. The random number generator that is used to throw heads or tails is based on the head block id and takes the `nonce`-th position in the hexadecimal representation of the block id (starting at position 8 and capped to 39 for unclear and undocumented reasons).

Old flipcoin objects are removed from the database.

finteh.org recommended

A random number generator, implemented by the CWD.global team, must be reviewed very carefully in order to avoid exploitation there.

Lottery

Lottery allows the creation of individual lottery games where people can buy tickets. An object called `lottery_goods_object` stores the lotteries.

- `lottery_goods_create_lot_operation`: This operation is used to create a new lottery. As arguments it accepts the owner, the number of participants (note: should probably be called tickets instead), the price of a single ticket, an expiration time, some latency that appears to extend the expiration time when the last ticket is bought, and image url as well as a description. The corresponding object is created with this operation. Apparently, a certain referral status is required in order to be allowed to use this operation. The implementation mentions an admin account with id `27` (should probably be a constant macro instead of a number right away).

- `lottery_goods_buy_ticket_operation`: This operation is used to buy a ticket from a given lottery `lot_id`. The operation comes with an amount that must match the lottery's price. Hence, only a single ticket can be bought at a time.
- `lottery_goods_send_contacts_operation`: After the lottery winner has been picked, this operation can be used by the winner to put a `winner_contacts` information into the lottery object.
- `lottery_goods_confirm_delivery_operation`: This operation can only be called by the winner in order to claim the profit.
- `lottery_goods_refund_operation`: Virtual operation to indicate that a lottery was not fully sold out, thus refunded
- `lottery_goods_win_operation`: Virtual operation pushed when processing the bets
- `lottery_goods_loose_operation`: Virtual operation pushed when processing the bets

Expired lotteries are processed every block in `proceed_lottery_goods()`. The random number generator is different compared to flipcoin. In this case, the number of participants to select a substring from the current head block. This substring is converted into a long unsigned integer that is used to select a winner from the list of participants. The lottery object is modified accordingly.

Old lotteries are removed from the database.

Matrix

Given that we have no specifications for this game, we are trying to identify its logic from code. Some bullet points:

- There is a matrix_object that contains side information about an ongoing matrix game including, start and finish block numbers, amounts and prices and prizes per level.
- An 8 level game
- There can technically be multiple active and open matrix games. It is unclear whether that actually happens.

- A player can open multiple rooms for a specific matrix as long as the user's `referral_status` allows.
- For unclear reasons, each room appears to have its own level, called `matrix_level`.
- A room can technically hold many people/accounts.
- Each level comes with a fixed price defined when opening a new matrix (automated and autonomous in `proceed_matrix()`)
- Parameters, such as price levels and prizes are defined in a committee controlled parameter called `gamezone_parameters`
- When a matrix is finished (e.g. finish block reached and some average value over number of idle blocks is not reached - else finishing delayed by number of idle blocks)
- It is unclear what values `matrix_object.status` can take and what they represent. A room can have 3 states, open cells closed, open cells opened and closed. Apparently, opening a room for a specific level may lead to another room being `filled` if room status is `open cells opened` and the room is full (room specific size of cells). The person that had opened the room would receive the `reward/prize` (depends on matrix level)
- A referral program with 1 level takes place
- Some custom logic executes if the matrix is owned by the users referrer
- The first too rooms are treated differently (probably due to adjusted mechanics in hardforks)
- `matrix_open_room_operation`: This opens a room within a matrix. To open a room, a level-specific fee must be paid that goes into the matrix.
- `matrix_room_closed_operation`: Virtual operation
- `matrix_cell_filled_operation`: Virtual operation

finteh.org recommended

Some hard fork took place to fine-tune the mechanics. We would recommend defining macros in the `hardfork.h` file and using those instead of putting (block) numbers into the source code. This would make running an independent testnet easier.

Conclusion of the expert council

Some parts of the documentation should be clarified more.

The witnesses' vote number, for which the community has updated their witness nodes to match the new source code, should be `2/3+1` of the total votes.

Decentralization is achieved thanks to the consensus mechanism that enables multiple parties to append new blockchain onto the existing blockchain and thus update the state of the database. The consensus scheme requires at least 11 of these block producers that take turns. Thus, a single-point of failure is avoided and the consensus scheme becomes decentralized.

“Cryptocurrency is a finite resource” - ETH, HIVE/STEEM, EOS and many more have no hard cap.

“40% is distributed among Gold Crowd holders, 2% of undistributed profit from the affiliate program is consumed to replenish the reserve fund” - community should know, block producers and workers are paid from which fund.

“the implementation of NFT functionality is also possible” - it’s already possible. BitShares implements NFTs using regular assets with max_supply: `1`.

FinTechAssociation recommended for safety user’s funds

1. The Crowd Collateral algorithm needs a special audit for the.

2. The Crowd Credit needs careful consideration in order not to be exploited.

What if the user gets a loan and his referred users don't pay any transaction fees anymore? Maybe they are Sybille's even.

3. The "proof-of-crowd staking" consensus must be validated by tokenomics experts to conform to the principles of game theory and economics.

4. The random number generator has to be reviewed very thoroughly as CWD.global wants to avoid this being exploited.

It's worth noting that the Peerplays project also has a random number generator and some "games" implemented on chain - even though they never managed to get traction it's potentially worth a review for CWD.global developers. Check this https://github.com/peerplays-network/peerplays

“The winner is determined when generating the next block based on the hash of that block” means that the block producer that gets to create the block can define the outcome of the random number generator! CWD.global developers should review peerplays algorithms. It is based on a commit+reveal scheme of all block producers.

And when the Helper bot is open source, the CWD.global community can offer other interesting ideas for it!

CWD.global project looks like a very interesting fork of the BitShares blockchain with several innovations in economy mechanics.

Most of the business logic functions declared in the documentation correspond to the description.

The application from the repository was built from the source code correctly

Non-governmental organization
FINTEH
FINTECH ASSOCIATION
Warsaw