GVISP 1
your space

2018.03.12.

COIN PAYMENT PROCESSOR.ORG

#OPEN CONSORTIUM cPRO

development@coinpaymentprocessor.org

GVISP1 LTD

#CP PROCESSOR GOLD PARTNER

office@gvisp.com

PERSONAL ETH COLLATERAL

ETHEREUM SMART CONTRACT AUDIT

GVISP1 LTD.

GVISP1 Ltd. **** Your space**** Bul.Vojvode Misica 17, 11000 Belgrade, Serbia, EU

1. Audit description

One contract was checked: **PersonalEthCollateral.sol.**

The purpose of this audit is to check all functionalities of **PersonalEthCollateral.sol** contract, and to determine level of security and probability of adverse outcomes.

PersonalEthCollateral.sol is a contract that stores received ETH, mapps them to msg.sender's address and mints stable tokens by calling \`mint\` function from StableToken contract. Stored ETH may be repurchased any time for stable tokens at the same price they were bought for (FIFO methodology).

In the following lines, the contract is being referenced by the name of the file where it was written in. The file contains exactly one contract, so there is no room for confusion.

2. Quick review

- ✓ All functions and state variables are well commented which is good in order to understand quickly how everything is supposed to work. Contract contains while loop in one of its functions which is generally not recommended but corresponds to requirements.
- ✓ The contract was written in accordance with solidity's aesthetic standards (names of state variables and functions start with lowercase letter, names of events start with capital letter, etc)
- ✓ During deployment, stable token, oracle, buyback contract and cPRO token contract address must be provided as parameters.

3. A brief review of contract's functionalities

Contract's functions are:

- ✓ \`buy\` function (implemented in fallback function) stores received ETH and mints stable tokens to msg.sender's address.
- ✓ \`takeFromPersonalCollateral\` - is called when someone wants to withdraw ETH from personal collateral by sending the equivalent amount of stable tokens.
- ✓ \`authorize\` and \`unauthorize\` functions regulate privileges for calling the above functions, and only superAdmin can call them.

Functions only authorized addresses can call are:

- ✓ \`changeOracleWalletAddress\`-changes the address of oracle wallet,
- ✓ \`changeOraclePercent\` - changes the portion of fee that goes to oracle,
- ✓ \`withdrawFee\` - sends fee to oracle and buyback contract,
- ✓ \`setMinCProAmount\` sets the minimum amount of cPRO tokens one must have in order to repurchase ETH from personal ETH collateral. To convert ETH in stable tokens user do not need to have minimum amount of cPRO tokens.

Functions for informational purposes are:

- ✓ \`getSpecificCollateralStables\` - shows the amount of stable tokens minted for a specific collateral
- ✓ \`getSpecificCollateralEth\` - shows the amount of ETH stored inside a specific collateral
- ✓ \`getSpecificCollateralTimestamp\` - shows the timestamp when a specific collateral was created

4. Functionalities test

- Total fee: ✓
- Last time fee taken: ✓
- Collateral profiles: ✓
- Fallback: ✓
- buy: ✓
- changeOracleWalletAddress: ✓
- changeOraclePercent: ✓
- setMinCProAmount: ✓
- takeFromPersonalCollateral: ✓
- withdrawFee: ✓
- authorize: ✓
- unauthorize: ✓
- getSpecificCollateralStables: ✓
- getSpecificCollateralEth: ✓
- getSpecificCollateralTimestamp: ✓

5. Detailed code check (line-by-line)

State variables of the contract:

- uint256 **totalFee** - total amount of fee (in ETH) stored inside the contract
- uint256 **lastTimeFeeTaken** - timestamp of a moment when the commision was last withdrawn from the contract
- uint256 **oraclePercent** - fee percentage that goes to oracle wallet
- uint256 **minCProAmount** - minimum amount of cPRO tokens one must have in order to call `takeFromPersonalCollateral` function
- address **stableTokenAddress** - address of stable token (unchangeable)
- address **oracleWalletAddress** - address of oracle wallet
- address **buybackContractAddress** - address of buyback contract (unchangeable)
- address **cProContractAddress**- address of cPRO token (unchangeable)

Structures:

EthCollateral – contains:

- uint256 **amountOfCollateralStables** (stable tokens minted within this specific collateral),
- uint256 **amountOfCollateralEth** (amount of ETH stored as this specific collateral),
- uint256 **collateralTimestamp** (timestamp of a moment when transaction occured)

EthCollateralProfile – contains:

- uint256 **numberOfEthCollaterals** (total number of collaterals address owns),
- uint256 **totalCollateralStables** (total amount of stable tokens bought "with collateral"),
- uint256 **totalCollateralEth** (total amount of ETH exchanged for tokens "with collateral"),
- uint256 **lastCollateralAdded** (serial number of the last added collateral),
- uint256 **lastCollateralTaken** (serial number of the last taken collateral)

Mappings:

- ➢ (address => bool) authorizedAddresses - remembers whether an address is authorized or not
- ➢ (address => EthCollateralProfile) ethCollateralProfiles - mapps addresses to their collateral profiles

Modifiers:

- ➢ onlySuperAdmin - checks if msg.sender is superAdmin
- ➢ onlyAuthorized - checks if msg.sender is authorized to call `changeOraclePercent`, `changeOracleAddress` and `withdrawFee` function
- ➢ isNotPaused - checks if "pause" is on
- ➢ cProRequirement - checks if person has the required amount of cPRO tokens

During deployment, creator must specify addresses of Stable Token, oracle, cPRO token and buyback contract. Stable Token address is needed so that **PersonalEthCollateral.sol** could mint stable tokens, while fee goes to oracle and buyback address. Buyback address is unchangeable which maybe should be changed. cPRO contract address is needed so contract could check the balance of msg.sender.

- ✓ authorize - function that gives privileges to other addresses (only superAdmin can call this function)
- ✓ unauthorize - function that denies privileges (only superAdmin can call this function)
- ✓ changeOracleWalletAddress - changes the oracle address and only authorized addresses can call it
- ✓ changeOraclePercent - changes the portion of fee that goes to oracle and only authorized addresses can call it
- ✓ withdrawFee - sends fee from the contract to oracle and buyback contract (only authorized addresses can call it)
- ✓ setMinCProAmount - sets the minimum amount of cPRO tokens one must have in order to call `takeFromPersonalCollateral` function
- ✓ buy - stores sent ETH inside personal collateral and mints stable tokens to msg.sender's address
- ✓ takeFromPersonalCollateral - burns received stable tokens and sends ETH to msg.sender
- ✓ getSpecificCollateralStables - (for informational purposes), shows the amount of stable tokens minted within a specific collateral
- ✓ getSpecificCollateralEth - (for informational purposes), shows the amount of ETH stored inside a specific collateral
- ✓ getSpecificCollateralTimestamp - (for informational purposes), shows the timestamp when specific collateral was created
- ✓ `takeFrompersonalCollateral` function iterates through collaterals starting from the oldest one in order to return ETH by FIFO method. Daily fee equals 0.01% (of initial collateral ETH) for each collateral. ETH transfer happens at the bottom of the function which is a good practice.

6. Static analysis test, vulnerabilities and outcomes

- ✓ Static analysis of the code was conducted and no security flows were found.

    https://oyente.melon.fund
    browser/stable.sol:PersonalEthCollateral
    EVM Code Coverage :
    Callstack Depth Attack Vulnerability : False
    Re-Entrancy Vulnerability : False
    Assertion Failure: False
    Parity Multisig Bug 2 : False
    Transaction-Ordering Dependence (TOD) : False

- ✓ Overflows are only possible in `buy` function, and will occur if one specific address owns 2**256 collaterals. But that is not actually possible because there is not enough ETH circulating to pay the gas for 10**77 transactions.

7. Final comments

Function that changes the address of buyback contract should be added. It would make more sense if fee (in `takeFromPersonalCollateral` function) was calculated as a compound interest rate, but that would lead to overflow after ~100 days because of solidity's inability to handle decimal numbers.