



**GVISP 1**  
your space

# GVISP1 LTD.

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

2018.03.12.

**COIN PAYMENT PROCESSOR.ORG**

**#OPEN CONSORTIUM cPRO**

[development@coinpaymentprocessor.org](mailto:development@coinpaymentprocessor.org)

**GVISP1 LTD**

**#CP PROCESSOR GOLD PARTNER**

[office@gvisp.com](mailto:office@gvisp.com)

**PERSONAL ETH CONVERSION**

**ETHEREUM SMART CONTRACT AUDIT**



## 1. Audit description

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

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

**PersonalEthConversion.sol** is a contract that converts sent ETH to stable tokens by storing ETH as personal conversion convert.

Later, if conditions are satisfied, one can exchange stable tokens for supplements: DoES, UpES, DiXX and InXX.

Also, stored ETH can later be repurchased for stable tokens by superAdmin.

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

**PersonalEthConversion.sol** is a contract that converts between values: ETH to stable tokens, stable tokens to Supplements and back.

- ✓ Constructor parameters for **PersonalEthConversion.sol** are: Main.sol contract address, Supplements.sol contract address and specific stable token contract address.
- ✓ All functions and state variables are well commented which is good in order to understand quickly how everything is supposed to work.
- ✓ This contract must be deployed after Main.sol, Supplements.sol and specific StableToken.sol as it takes their addresses as parameters.

## 3. A brief review of contract's functionalities

Contract contains many functions of which only `withdrawEthFee` and `withdrawStablesFee` need (standard) authorization.

Other functions can be called by anyone if requirements are satisfied. Those two functions are used to withdraw fee (both ETH and stable tokens) from the contract.

There is one "constant" function that returns personal supplements for each address. All other functions are used to convert between values (ETH to stable tokens, stable tokens to DoES and/or UpES tokens, stable tokens for ETH and supplements for stable tokens).

- ✓ `buy` - converts sent ETH to stable tokens at current price. Sent ETHs are stored as convert and stable tokens are minted to msg.sender. Can be called if processes are not paused (which is defined inside Main.sol contract) and if specific stable token is "activated" for conversion (which is also defined inside Main.sol);
- ✓ `fallback` function calls `buy` function;
- ✓ `buybackEth` - converts stable tokens to ETH at current price, but only the sum of sent ETH (from msg.sender's address) can be bought back. Can be called if processes are not paused, stable token is "activated" and msg.sender has enough cPRO tokens on his account (required amount is defined inside Main.sol contract)
- ✓ `personalSupplements` - (constant) function that returns the supplements specific address has;
- ✓ `buyDoES` - converts stable tokens to DoES tokens. Can only be called if processes are not paused, stable token is "activated" for conversion, DoES capitalization is at

least 5% lower than capitalization of UpES, and there is some personal or global supplements (in case of personal supplements, only excess stable tokens can be put as convert for DoES tokens)

- ✓ `buyUpES` - converts stable tokens to UpES tokens. Analogue rules apply for UpES conversion as for DoES conversion.
- ✓ `buybackStablesForDoES` - converts DoES tokens to stable tokens at current price. Msg.sender can repurchase only stable tokens he stored as a convert. Function call requirements are same as `buybackEth` call requirements.
- ✓ `buybackStablesForUpES` - converts UpES tokens to stable tokens at current price. Msg.sender can repurchase only stable tokens he stored as a convert. Function call requirements are same as `buybackEth` call requirements.
- ✓ `withdrawEthFee` - takes fee in ETH from the contract and transfers it to oracle and buyback contract address. Function can only be called by authorized addresses.
- ✓ `withdrawStablesFee` - takes fee in stables and transfers it to msg.sender.

#### **4. Functionalities test**

- fallback: ✓
- buy: ✓
- buybackEth: ✓
- personalSupplements: ✓
- buyDoES: ✓
- buyUpES: ✓
- buybackStablesForDoES: ✓
- buybackStablesForUpS: ✓
- withdrawEthFee: ✓
- withdrawStablesFee: ✓

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

After deployment, only **Main.sol**, **Supplements.sol** and specific **StableToken.sol** contract addresses are set, which means all of these contracts must be deployed before **PersonalEthConversion.sol**.

PersonalEthConversion.sol contract “reads authorizations” from Main.sol contract in order to secure some function are called only by authorized addresses.

After deployment, “mint authorization” must be set for its address inside Main.sol so it could actually call `mint` function within StableToken.sol and change state variables inside Supplements.sol contract.

PersonalEthConversion.sol also reads Supplements.sol contract in order to get the information about global supplements.

##### State variables of the contract:

- address **mainContractAddress** - address of Main.sol contract
- address **supplementsContractAddress** - address of Supplements.sol contract
- address **stableTokenAddress** - address of specific StableToken.sol contract
- uint256 **totalEthFee** - total amount of fee (in ETH) stored inside the contract
- uint256 **totalStableTokenFee** - total amount of fee (in specific stable token) stored inside the contract

- uint256 **lastTimeEthFeeTaken** - timestamp of the moment ETH fee was last withdrawn
- uint256 **lastTimeStablesFeeTaken** - timestamp of the moment specific stable token fee was last withdrawn

#### Mappings:

- (address => EthConvert) ethConverts - maps addresses to their ETH converts
- (address => StableConvert) stableConverts - maps addresses to their stables convert

#### Modifiers:

- onlyAuthorized - checks if msg.sender is authorized
- isNotPaused - checks if processes are not paused
- cProRequirement - checks if msg.sender has enough cPRO tokens to call certain functions
- activatedStable - checks if stable token is activated for conversion
- checkForDoES - compares capitalizations of DoES and UpES tokens in order to tell if DoES token is available for purchase (based just on this criteria)
- checkForUpES - compares capitalizations of DoES and UpES in order to tell if UpES token is available for purchase (based just on this criteria)

#### Structures:

- EthConvert - state of ETH convert for specific address; contains information about the amount of stable tokens minted, and the amount of ETH stored inside itself
- StableConvert - state of stable tokens convert for specific address; contains information about the amount of stable tokens locked inside the convert, and the amount of both DoES and UpES tokens generated by those stables
- Both structures could be replaced with mappings if unnecessary information about the amount of minted stables and DoES and UpES tokens was deleted, but then personalSupplements could not be calculated.

#### Functions:

- `fallback` function - implements `buy` function
- `buy` - converts sent ETH to stable tokens by storing ETH to msg.sender's convert; 'isNotPaused' and 'activatedStable' modifiers must be on in order for function to work. Price is read from **StableToken.sol** contract itself (and is provided by centralized oracle service). 0.2% of sent ETH is stored inside the contract as a fee, and the rest is expressed in stable tokens and minted to msg.sender's address
- `buybackEth` - converts stable tokens to ETH by burning them. Msg.sender can buyback only ETH he initially sent. Function can only be called if msg.sender has the required amount of cPRO tokens on his balance, stable token is activated (inside **Main.sol** contract) and processes are not paused. ETH is bought back at current price of ETH in specific stable token.
- `personalSupplements` - (constant) function calculates the amount of non-converIALIZED stable tokens for specific address
- `buyDoES` - converts stable tokens to DoES tokens; stables sent must be non-converIALIZED (either as a consequence of personal or global supplements). Fee equals 0.4% and is stored inside the contract itself. Can be called if processes are not paused, stable token is activated and "checkForDoES" modifier allows it.
- `buyUpES` - works the same as `buyDoES`, but with "checkForUpES" modifier instead of "checkForDoES"

- `buybackStablesForDoES` - converts DoES tokens for stable tokens. Msg.sender can buy only the amount of stables he sent this way, and stables are bought at current price. Processes must not be paused, msg.sender must have enough cPRO tokens and stable token must be activated in order for function to work.
- `buybackStablesForUpES` - works the same as `buybackStablesForDoES` but with UpES tokens.
- `withdrawEthFee` - takes fee (in ETH) from the contract and transfers it to oracle account and cPRO buyback contract. Only authorized addresses can call this function.
- `withdrawStablesFee` - takes fee (in stable tokens) and sends it to msg.senders' address. Only authorized addresses can call this function.

## **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:PersonalEthConversion*

*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*

- ✓ Over and underflows are not possible in this contract.

## **7. Final comments**

Modifier 'activatedStable' is already checked when calling 'buy' function (or sending ETH via fallback), so there is no need for require (Main(mainContractAdress).activatedCovertibles(stableTokenAddress))" in line 99.

Generally, the code is well commented.Contract imports four interfaces: **StableToken.sol**, **StandardToken.sol**, **Main.sol** and **Supplements.sol**.

Import of both StableToken.sol and StandardToken.sol is unnecessary because both of them have all ERC20 standard functions and 'mint' function, so just one of them would do the job for both stable token and DoES/UpES token, but the interface shouldn't be removed for simplicity (it wouldn't be obvious which token is minted in code).