DRIVEN security

TECHNICAL AUDIT
HNW

April 20, 2022

Table of Contents

Project Details

Project Name
HNW - HfuelToken.sol

Project Type
ERC20 Token

Contract Address
Not Yet Deployed

Blockchain
Not Yet Deployed

Project Website
https://hnwdefi.com/

This Audit Was Created On
April 20, 2022

Technical Audit Overview: HfuelToken.sol

GENERAL ISSUES

- Security Issues: **PASSED**
- Gas & Fees Issues: **PASSED**
- ERC Errors: **PASSED**
- Compilation Errors: **PASSED**
- Design Logic: **PASSED**
- Timestamp Dependence: **PASSED**
- Buy/Sell/Transfer: ***The owner can exclude addresses from paying fees and can set a custom fee for each address***

SECURITY AGAINST CYBER ATTACKS

- Private user's data: **PASSED**
- Reentrancy: **PASSED**
- Cross-Function Reentrancy: **PASSED**
- Front Running: **PASSED**
- Taxonomy Attacks: **PASSED**
- Integer Overflow and Underflow: **PASSED**
- DoS (Denial of Service) with Unexpected Revert: **PASSED**
- DoS (Denial of Service) with Block Gas Limit: **PASSED**
- Insufficient Gas Griefing: **PASSED**
- Forcibly Sending ETH to a Contract: **PASSED**

Technical Audit Details: HfuelToken.sol

INHERITED CONTRACTS

- abstractMintableToken

LIBRARIES

- SafeMath for uint256
- INHEREITED CONTRACTS
- abstractMintableToken
- -- vars --
- {static}[[string]] name
- {static}[[string]] symbol
- {static}[[uint8]] decimals
- {static}[[uint256]] targetSupply
- [[uint256]] totalTxs
- [[uint256]] players
- [[uint256]] mintedSupply_
- [[mapping address=>Stats]] stats
- [[address]] vaultAddress
- {static}[[uint8]] taxDefault
- [[mapping address=>uint8]] _customTaxRate
- [[mapping address=>bool]] _hasCustomTax
- [[mapping address=>bool]] _isExcluded
- [[address]] _excluded

FUNCTIONS

- setVaultAddress()
- mint()
- finishMinting()
- calculateTransactionTax()
- transferFrom()
- transfer()
- GETTERS
- calculateTransferTaxes()

TECHNICAL AUDIT DETAILS (CONTINUED)

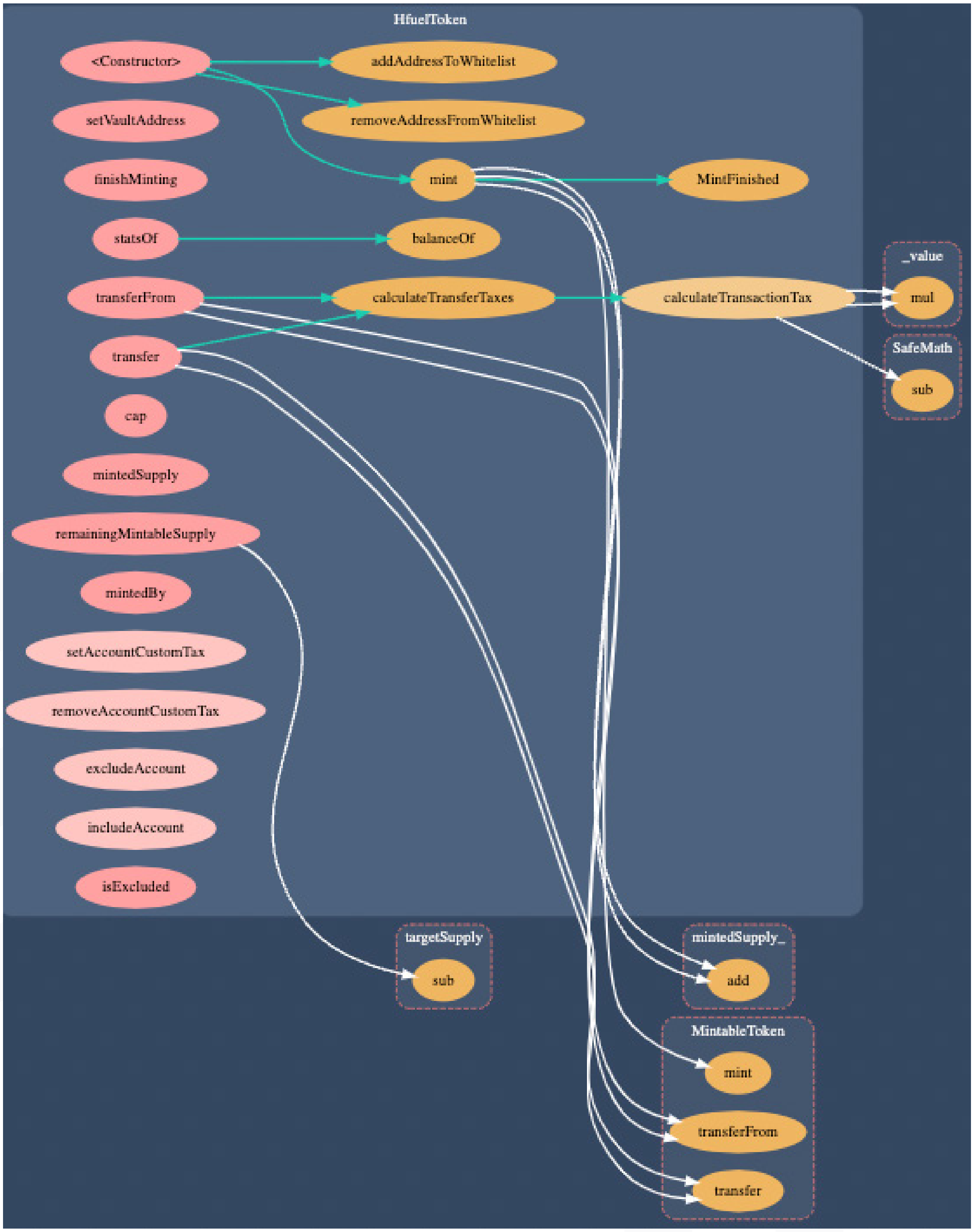
GETTERS

- remainingMintableSupply()
- cap()
- mintedSupply()
- statsOf()
- mintedBy()
- isExcluded()

SETTERS

- setAccountCustomTax()
- removeAccountCustomTax()
- excludeAccount()
- includeAccount()

Contract Architecture: HfuelToken.sol



Penetration Testing: HfuelToken.sol

RE-ENTRANCY

What is Re-Entrancy?
A re-entrancy attack can arise when you write a function that calls another untrusted contract before resolving any consequences. If the attacker has authority over the untrusted contract, he can initial a recursive call back to the original function, repeating interactions that would otherwise not have occurred after the effects were resolved.

Attackers can take over the smart contract's control flow and make modifications to the data that the calling function was not anticipating.

To avoid this, make sure that you do not call an external function until the contract has completed all of the internal work.

TEST: PASSED

CROSS-FUNCTION RE-ENTRANCY

What is Cross-Function Re-Entrancy?
When a vulnerable function shares the state with another function that has a beneficial effect on the attacker, this cross-function re-entrancy attack is achievable. This re-entrancy issue that is the employment of intermediate functions to trigger the fallback function and a re-entrancy attack is not unusual.

Attackers can gain control of a smart contract by calling public functions that use the same state/variables as "private" or "onlyOwner" functions.

To avoid this, make sure there are no public functions that use private variables, and avoid calling routines that call external functions or use mutex (mutual exclusion).

TEST: PASSED

FRONT RUNNING

What is Front Running?
Front running indicates that someone can obtain prior information of transactions from other beneficial owners by technology or market advantage, allowing them to influence the price ahead of time and result in economic benefit, which usually results in loss or expense to others

Since all transactions are visible in the block explorer for a short period of time before they are executed, network observers can see and react to an action before it is included in a block.

Attackers can front run transactions because every transaction is visible to the blockchain, even if it is in the "processing" or "indexing" state. This is a very low security vulnerability because it is based on the blockchain rather than the contract.

The only possible attack is seeing transactions made by bots. Using transaction fees, you can avoid bots.

TEST: PASSED

PENETRATION TESTING (CONTINUED)

TAXONOMY ATTACKS

These taxation attacks can be made in 3 ways:

1. Displacement
Performed by increasing the gasPrice higher than network average, often by a multiplier of 10.

2. Insertion
Outbidding transaction in the gas price auction

3. Suppression (Block Stuffing)
The attacker sent multiple transactions with a high gasPrice and gasLimit to custom smart contracts that assert to consume all the gas and fill up the block's gasLimit.

This type of attack occurs mainly for exchanges, so this smart contract is secured

TEST: PASSED

INTEGER OVERFLOW AND UNDERFLOW

Overflow
An overflow occurs when a number gets incremented above its maximum value.

In the audited contract: uint8 private _decimals = 9;

(_decimals can't reach a value bigger than it's limit)

TEST: PASSED

Underflow

An overflow occurs when a number gets decremented below its maximum value.

(There are no decrementation functions for parameters and users can't call functions that are using uint values);

TEST: PASSED

PENETRATION TESTING (CONTINUED)

DOS (DENIAL OF SERVICE) WITH UNEXPECTED REVERT

DoS (Denial of Service) attacks can occur in functions when you attempt to transmit funds to a user and the functionality is dependent on the successful transfer of funds.

This can be troublesome if the funds are given to a bad actor's smart contract (when they call functions like "Redeem" or "Claim"), since they can simply write a fallback function that reverts all payments.

TEST: PASSED

DOS (DENIAL OF SERVICE) WITH BLOCK GAS LIMIT

Each block has an upper bound on the amount of gas that can be spent, and thus the amount of computation that can be done. This is the Block Gas Limit. If the gas spent exceeds this limit, the transaction will fail. This leads to a couple possible Denial of Service vectors.

TEST: PASSED

INSUFFICIENT GAS GRIEFING

This attack can be carried out against contracts that accept data and use it in a sub-call on another contract.

This approach is frequently employed in multisignature wallets and transaction relayers. If the sub-call fails, either the entire transaction is rolled back or execution is resumed.

TEST: PASSED.

FORCIBLY SENDING ETH TO THE SMART CONTRACT

TEST: PASSED

Project Details

Project Name
HNW - Refinery V1

Project Type
Staking Smart Contract

Contract Address
Not Yet Deployed

Blockchain
Not Yet Deployed

This Audit Was Created On
April 20, 2022

Technical Audit Overview: Refinery V1

GENERAL ISSUES

- Security Issues: **PASSED**
- Gas & Fees Issues: **PASSED**
- ERC Errors: **PASSED**
- Compilation Errors: **PASSED**
- Design Logic: **FAILED**
- Timestamp Dependence: **PASSED**
- Transfer: ***The owner can exclude addresses from paying fees and can set a custom fee for each address;***

SECURITY AGAINST CYBER ATTACKS

- Private User's Data: **PASSED**
- Reentrancy: **PASSED**
- Cross-function Reentrancy: **PASSED**
- Front Running: **PASSED**
- Taxonomy attacks: **PASSED**
- Integer Overflow and Underflow: **PASSED**
- DoS (Denial of Service) with Unexpected Revert: **PASSED**
- DoS (Denial of Service) with Block Gas Limit: **PASSED**
- Insufficient gas griefing: **PASSED**
- Forcibly Sending ETH to a Contract: **PASSED**

In-Depth Analysis: Refinery V1

DESIGN LOGIC

FAILED

Unused variables and the smart contracts is using a fallback function but not a receive function

Technical Audit Details: Refinery V1

INHERITED CONTRACTS

- {abstract}OwnableUpgradeable

LIBRARIES

- {abstract}SafeMath for [[uint256]]

VARIABLES

- [[address]] refineryVaultAddress
- [[ITokenMint]] tokenMint
- [[IToken]] xHNWToken
- [[IToken]] hfuelToken
- [[RatesController]] ratesController
- [[IRefineryVault]] refineryVault
- [[mapping address=>User]] users
- [[mapping address=>Airdrop]] airdrops
- [[mapping address=>Custody]] custody
- [[uint256]] CompoundTax
- [[uint256]] ExitTax
- [[uint256]] payoutRate
- [[uint256]] ref_depth
- [[uint256]] ref_bonus
- [[uint256]] minimumInitial
- [[uint256]] minimumAmount
- [[uint256]] deposit_bracket_size
- [[uint256]] max_payout_cap
- [[uint256]] deposit_bracket_max
- [[uint256]] ref_balances
- [[uint256]] total_airdrops
- [[uint256]] total_users
- [[uint256]] total_deposited
- [[uint256]] total_withdraw
- [[uint256]] total_bnb
- [[uint256]] total_txs
- {static}[[uint256]] MAX_UINT

TECHNICAL AUDIT DETAILS (CONTINUED)

FUNCTIONS

Only-Owner Functions

- updatePayoutRate()
- updateRefDepth()
- updateRefBonus()
- updateInitialDeposit()
- updateMinimumAmount()
- updateCompoundTax()
- updateExitTax()
- updateDepositBracketSize()
- updateMaxPayoutCap()
- updateHoldRequirements()

Functions for Users

- checkin()
- deposit()
- claim()
- roll()

Internal Functions

- _setUpline()
- _deposit()
- _refPayout()
- _heart()
- _roll()
- _claim_out()
- _claim()

TECHNICAL AUDIT DETAILS (CONTINUED)

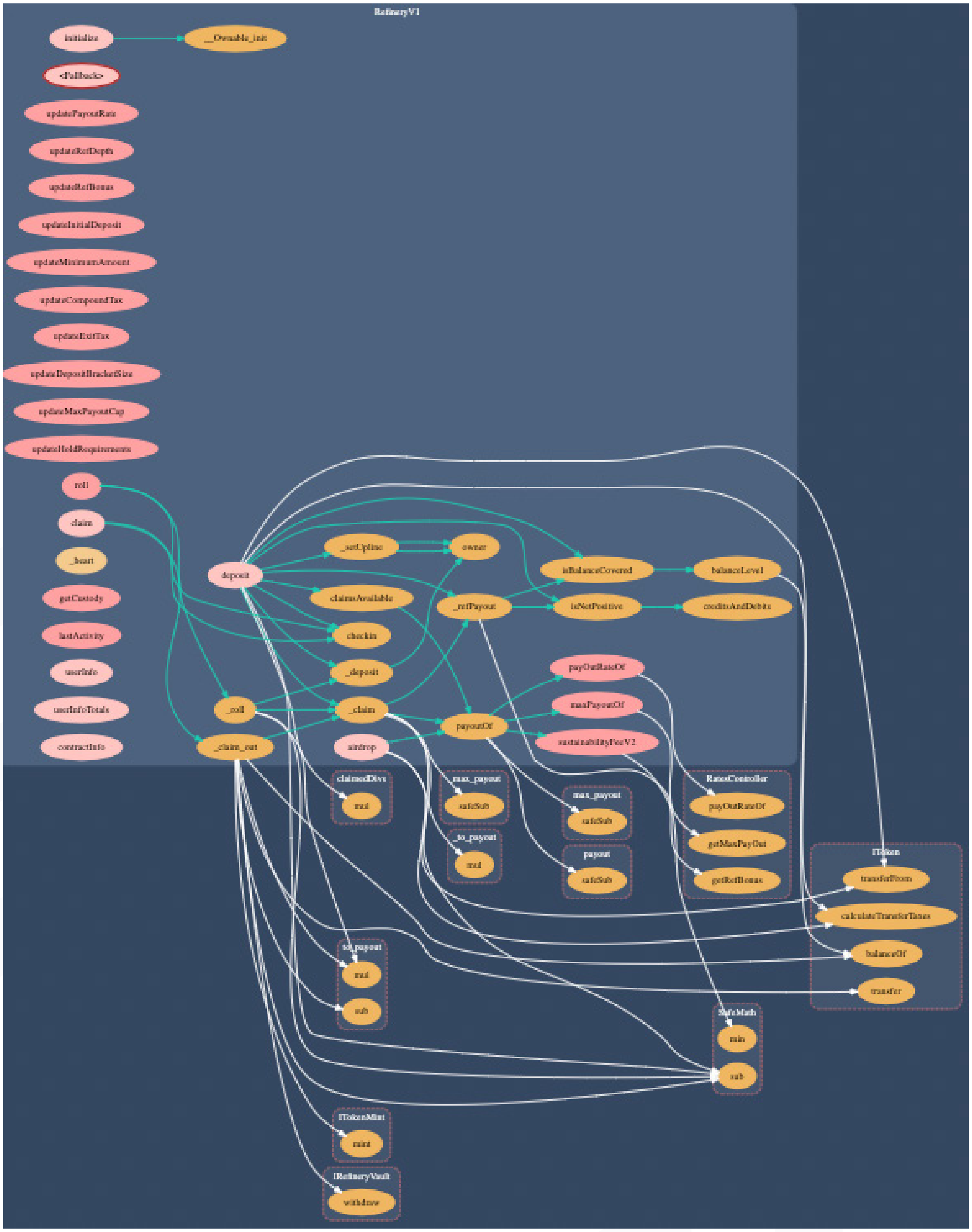
GETTERS

- isNetPositive()
- creditsAndDebits()
- isBalanceCovered()
- balanceLevel()
- getCustody()
- lastActivity()
- claimsAvailable()
- maxPayoutOf()
- sustainabilityFeeV2()
- payOutRateOf()
- payoutOf() userInfo()
- userInfoTotals()
- contractInfo()

PUBLIC EXTERNAL

- airdrop()

Contract Architecture: Refinery V1



Project Details

Project Name
HNW - TokenMint.sol

Project Type
Token Minter

Contract Address
Not Yet Deployed

Blockchain
Not Yet Deployed

Description
Considering the functions, this smart contract was designed so people can buy tokens using a smart contract not a DEX (like PancakeSwap). Such methods are used in order to avoid the high price impact on big transactions (buy / sell)

This Audit Was Created On
April 20, 2022

Technical Audit Overview: TokenMint.sol

GENERAL ISSUES

- Security Issues: **PASSED**
- Gas & Fees Issues: **PASSED**
- ERC Errors: **PASSED**
- Compilation Errors: **PASSED**
- Design Logic: **PASSED**
- Timestamp Dependence: **PASSED**

SECURITY AGAINST CYBER ATTACKS

- Private User's Data: **PASSED**
- Reentrancy: **PASSED**
- Cross-function Reentrancy: **PASSED**
- Front Running: **PASSED**
- Taxonomy attacks: **PASSED**
- Integer Overflow and Underflow: **PASSED**
- DoS (Denial of Service) with Unexpected Revert: **PASSED**
- DoS (Denial of Service) with Block Gas Limit: **PASSED**
- Insufficient gas griefing: **PASSED**
- Forcibly Sending ETH to a Contract: **PASSED**

Technical Audit Details: TokenMint.sol

INHERITANCE

- {abstract}Whitelist

VARIABLES

- address tokenAddress
- Token token
- address exchangeableToken

FUNCTIONS

ONLY WHITELISTED FUNCTION

- mint()

INTERNAL FUNCTIONS

- intForExchange()

ONLY OWNER FUNCITONS

- setExchangeableToken()

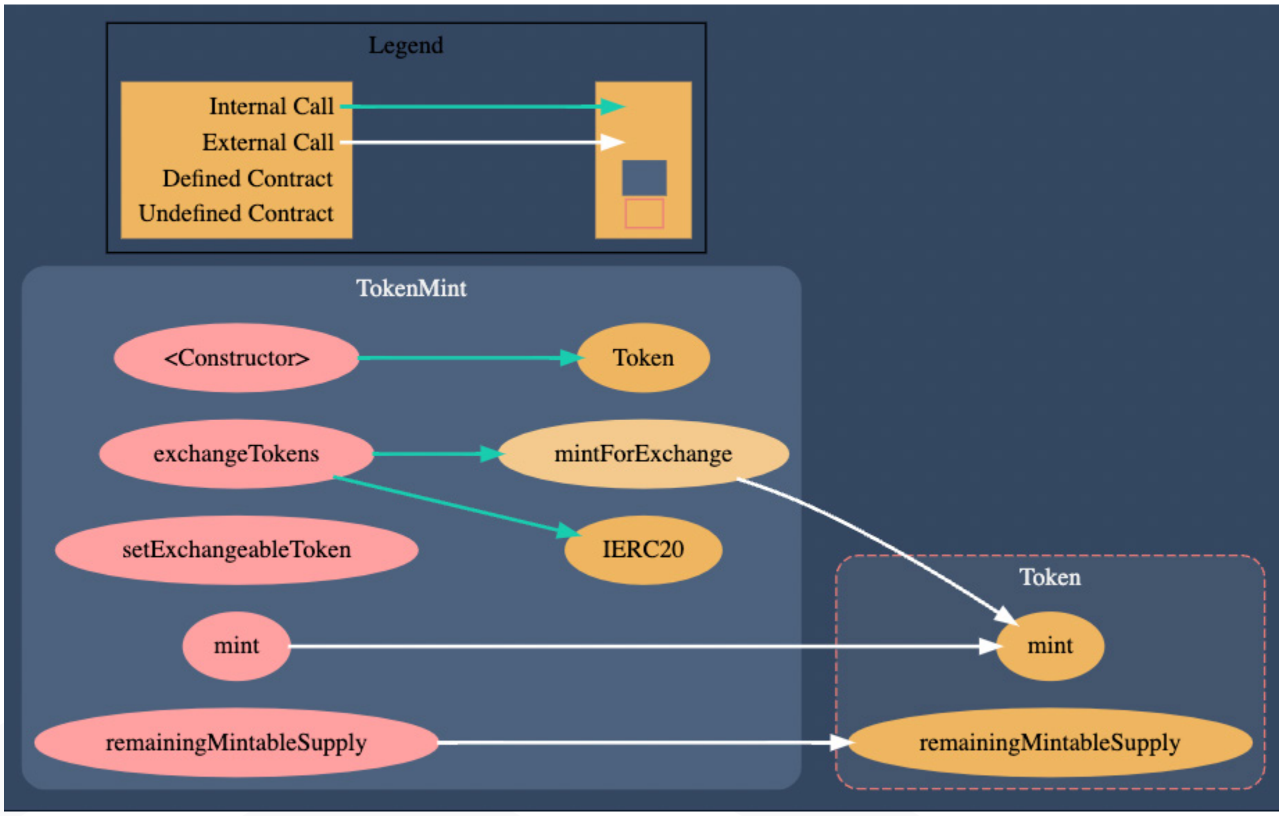
PUBLIC FUNCTIONS

- exchangeTokens()

GETTERS

- remainingMintableSupply()

Contract Architecture: TokenMint.sol



Project Details

Project Name
HNW - RefineryVault.sol

Project Type
Custom Smart Contract

Contract Address
Not Yet Deployed

Blockchain
Not Yet Deployed

Description
In this contract the fees are sent from trades made with Hfuel Token (HfuelToken.sol). Only whitelisted addresses are able to withdraw them.

This Audit Was Created On
April 20, 2022

Technical Audit Overview: RefineryVault.sol

GENERAL ISSUES

- Security Issues: **PASSED**
- Gas & Fees Issues: **PASSED**
- ERC Errors: **PASSED**
- Compilation Errors: **PASSED**
- Design Logic: **PASSED**
- Timestamp Dependence: **PASSED**

SECURITY AGAINST CYBER ATTACKS

- Private User's Data: **PASSED**
- Reentrancy: **PASSED**
- Cross-function Reentrancy: **PASSED**
- Front Running: **PASSED**
- Taxonomy attacks: **PASSED**
- Integer Overflow and Underflow: **PASSED**
- DoS (Denial of Service) with Unexpected Revert: **PASSED**
- DoS (Denial of Service) with Block Gas Limit: **PASSED**
- Insufficient gas griefing: **PASSED**
- Forcibly Sending ETH to a Contract: **PASSED**

Technical Audit Details: RefineryVault.sol

INHERITANCE

- {abstract}Whitelist

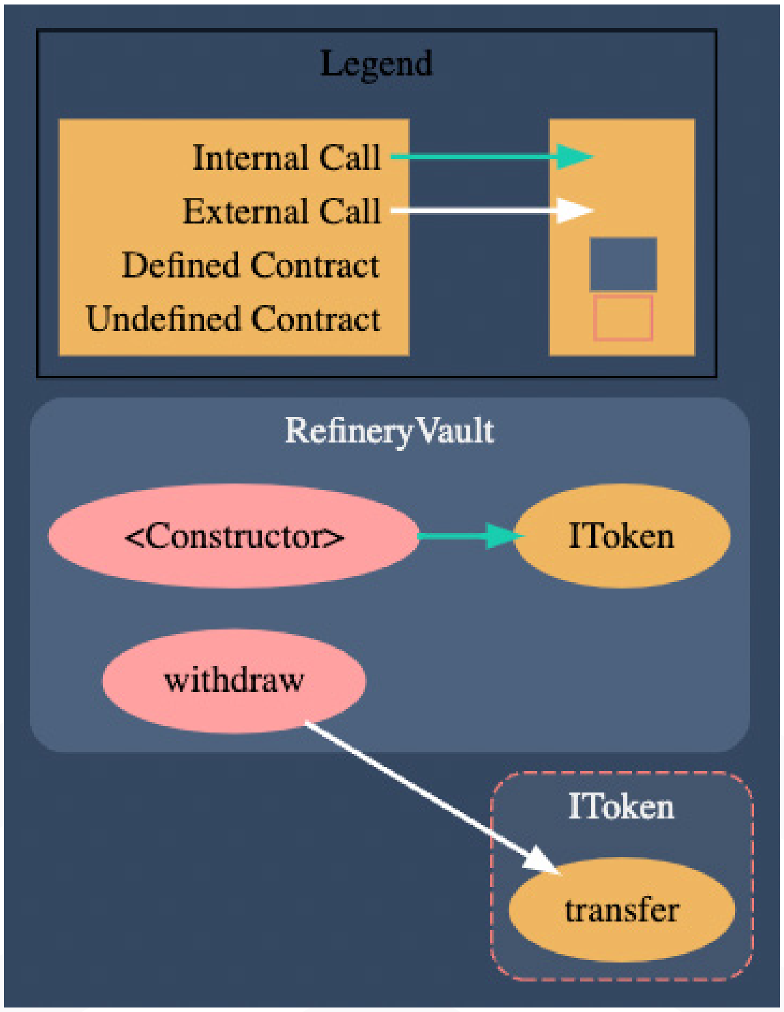
VARIABLES

- IToken token

FUNCTIONS FOR WHITELISTED

- withdraw()

Contract Architecture: RefineryVault.sol



Project Details

Project Name
HNW - Rates Controller

Project Type
Custom Smart Contract

Contract Address
Not Yet Deployed

Blockchain
Not Yet Deployed

Description
This contract only computes and returns values

This Audit Was Created On
April 20, 2022

Technical Audit Overview: Rates Controller

GENERAL ISSUES

- Security Issues: **PASSED**
- Gas & Fees Issues: **PASSED**
- ERC Errors: **PASSED**
- Compilation Errors: **PASSED**
- Design Logic: **PASSED**
- Timestamp Dependence: **PASSED**

SECURITY AGAINST CYBER ATTACKS

- Private User's Data: **PASSED**
- Reentrancy: **PASSED**
- Cross-function Reentrancy: **PASSED**
- Front Running: **PASSED**
- Taxonomy attacks: **PASSED**
- Integer Overflow and Underflow: **PASSED**
- DoS (Denial of Service) with Unexpected Revert: **PASSED**
- DoS (Denial of Service) with Block Gas Limit: **PASSED**
- Insufficient gas griefing: **PASSED**
- Forcibly Sending ETH to a Contract: **PASSED**

Technical Audit Details: Rates Controller

INHERITANCE

- {abstract}Ownable

LIBRARIES

- {abstract}SafeMath for [[uint256]]

VARIABLES

- [[IERC20]] xSK_Token
- [[IERC20]] xHNW_Token
- [[uint256]] xSKBalances
- [[uint256]] xHNWBalances
- [[uint256]] rates
- [[uint16]] maxPayOutRates
- [[uint256]] refBonuses

FUNCTIONS

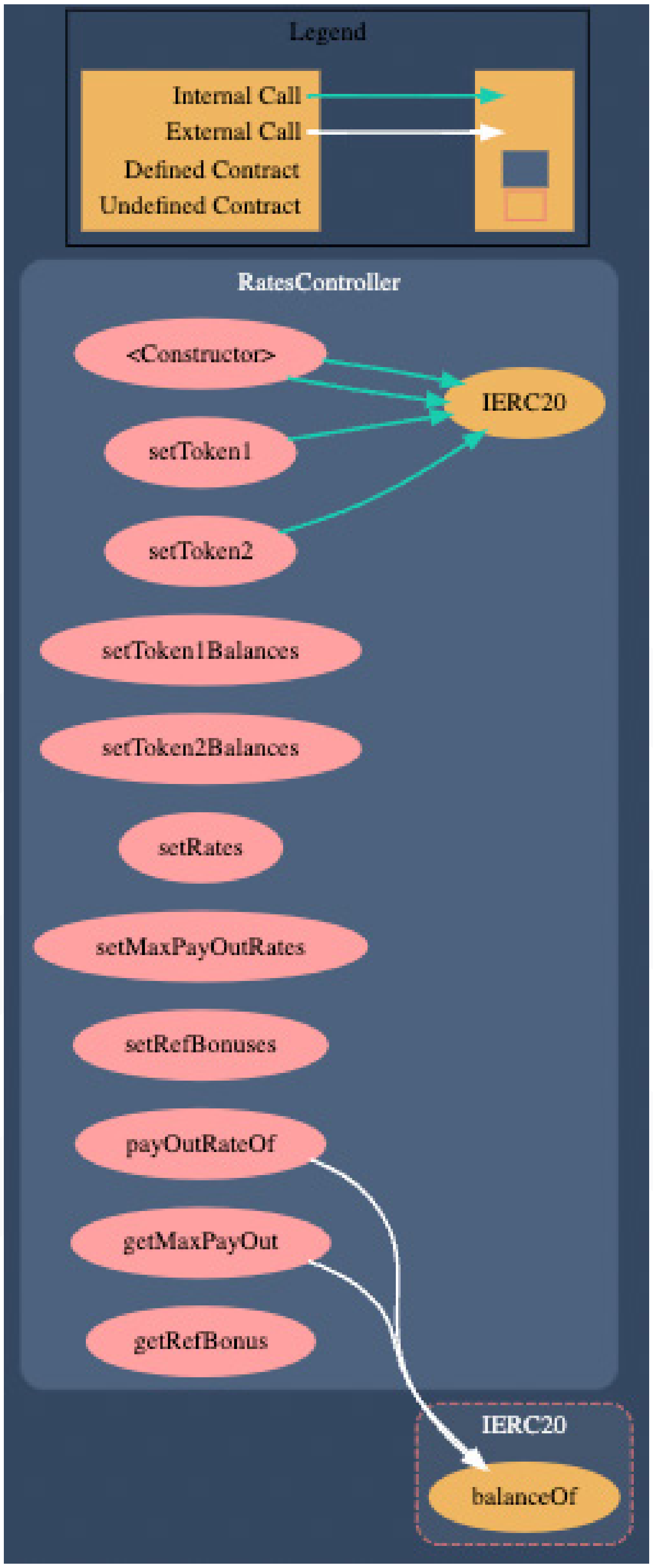
Only Owner Functions

- setToken1()
- setToken2()
- setToken1Balances()
- setToken2Balances()
- setRates()
- setMaxPayOutRates()
- setRefBonuses()

GETTERS

- payOutRateOf()
- getMaxPayOut()
- getRefBonus()w

Contract Architecture: Rates Controller



This audit was created by

DRIVENsecurity