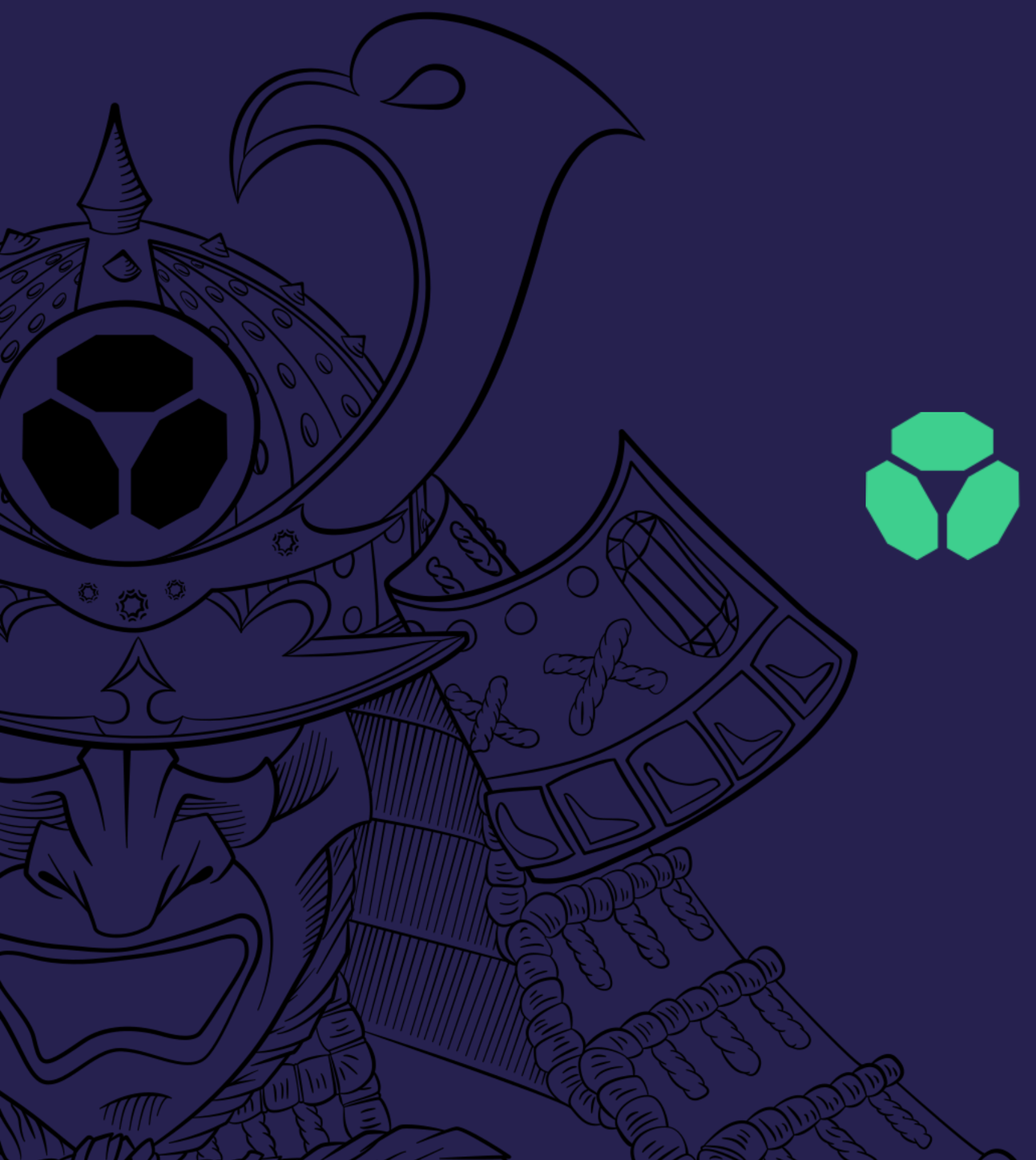

smart state

> Smart
Contract
Audit #

XP.NETWORK

Oct 26
2021

TABLE OF CONTENTS

METHODOLOGY

MAIN TESTS LIST:

- Best code practices
- ERC20/BEP20 compliance (if applicable)
- Logical bugs
- General Denial Of Service(DOS)
- Locked ether
- Private data leaks
- Using components with known vulns
- Weak PRNG
- Unsed vars
- Uncheked call return method
- Code with no effects
- Pool Asset Security (backdoors in the underlying ERC-20)
- Function visibility
- Use of deprecated functions
- Authorization issues
- Re-entrancy
- Arithmetic Over/Under Flows
- Hidden Malicious Code
- External Contract Referencing
- Short Address/ Parameter Attack
- Race Conditions / Front Running
- Uninitialized Storage Pointers
- Floating Points and Precision
- Signatures Replay

STRUCTURE OF CONTRACT

LIB.RS

CONTRACT METHODS ANALYSIS:

- event_cnt_inc(&self)
  Vulnerabilities not detected

- init(
  &self,
  token: TokenIdentifier,
  nft_token: TokenIdentifier,
  wegld_token: TokenIdentifier,
  min_valid: usize,
  #[var_args] validators_args:
  ManagedVarArgs<ManagedAddress>,
)
  We suggest you have a logical mistake here and validators.
  len() should be compared to min_valid. Fixed in commit
  090cd59f604a83b98a2d0ea662949efde493c76d

- require_unpaused(&self)
  Vulnerabilities not detected

- require_paused(&self)
  Vulnerabilities not detected

- get_esdt_and_token_fees(&self)
  Vulnerabilities not detected

- freeze_send(
  &self,
  #[payment] value: BigUint,
  chain_nonce: u64,
  to: String,
  real_value: BigUint,
)
  User can send any amount of fees he would like to, due to this his tokens can potentially get stuck on the bridge. We recommend to set some minimal amount for commision.

- freeze_send_nft(&self, chain_nonce: u64, to: String)
  User can send any amount of fees he would like to, due to this his tokens can potentially get stuck on the bridge. We recommend to set some minimal amount for commision.

- withdraw_nft(&self, to: String)
  Vulnerabilities not detected

- withdraw(&self, to: String)
  Vulnerabilities not detected

- deposit(&self)
  Vulnerabilities not detected

- user_role(&self, user: ManagedAddress)
  Vulnerabilities not detected

- validate_action(
  &self,
  id: BigUint,
  ac: Action<Self::Api>,
)
  Validators can reject an action only in case of sending an incorrect action type with the provded is, so the read_cnt is incremented and validator is not included in signers. We would recommend to include a boolean variable which will be true or false in case validator wants to accept action or reject it. In our vision this logic would be better for understanding and easier for tracking. Also it it recommended to emit an event in case validator accepts/rejects action.

- validate_add_validator(
  &self,
  uuid: BigUint,
  board_member_address: ManagedAddress,
)

  Vulnerabilities not detected

- validate_remove_validator(
  &self,
  uuid: BigUint,
  user_address: ManagedAddress,
)

  Vulnerabilities not detected

- validate_set_validator(
  &self,
  uuid: BigUint,
  new_quorum: usize,
)

  Vulnerabilities not detected

- validate_unfreeze(
  &self,
  uuid: BigUint,
  to: ManagedAddress,
  amount: BigUint,
)

  Vulnerabilities not detected

- validate_unfreeze_nft(
  &self,
  uuid: BigUint,
  to: ManagedAddress,
  token: TokenIdentifier,
  nonce: u64,
)

  Vulnerabilities not detected

- validate_send_wrapped(
  &self,
  uuid: BigUint,
  chain_nonce: u64,
  to: ManagedAddress,
  amount: BigUint,
)

  Vulnerabilities not detected

- validate_send_nft(
  &self,
  uuid: BigUint,
  chain_nonce: u64,
  to: ManagedAddress,
  id: ManagedBuffer,
)

  Vulnerabilities not detected

- validate_pause(&self, uuid: BigUint)

  Vulnerabilities not detected

- validate_unpause(&self, uuid: BigUint)

  Vulnerabilities not detected

- validate_withdraw_fees(&self, uuid: BigUint)

  Vulnerabilities not detected

- change_user_role(&self, user_address: ManagedAddress, new_role: UserRole)
  Vulnerabilities not detected

- perform_action(
  &self,
  action: Action<Self::Api>,
)
  Vulnerabilities not detected

STRUCTURE OF CONTRACT

ACTIONS.RS

CONTRACT METHODS ANALYSIS:

- eq(&self, other: &Self)
  Vulnerabilities not detected

STRUCTURE OF CONTRACT

EVENTS.RS

CONTRACT METHODS ANALYSIS:

- new(event: Event<BigUint>)
  Vulnerabilities not detected

STRUCTURE OF CONTRACT

USER_ROLE.RS

CONTRACT METHODS ANALYSIS:

- can_sign(&self)
  Vulnerabilities not detected

STRUCTURE OF CONTRACT

SWAP.RS

CHECK SUMMARY:

The model of this contract is to mint some wrapped tokens and provide them to the caller for his EGLD. But when someone returns his wrapped tokens, they are not burned, but marked as free. Which means if everyone will return their tokens there will be wrapped tokens on this contract that are not supported by any EGLD tokens. The whole point of wrapped tokens is that they are minted for an original token and burned when someone wants to return their original toknes. This is what makes them completely equal to original tokens. In your case tokens are not burned, but stored, so in order to statisfy wrapped tokens financial patters we would recommned to burn them instead of storing.

CONTRACT METHODS ANALYSIS:

- init(&self)
  Vulnerabilities not detected

- issue_wrapped_egld(
  &self,
  token_display_name: ManagedBuffer,
  token_ticker: ManagedBuffer,
  initial_supply: BigUint,
  #[payment] issue_cost: BigUint,
)

  Vulnerabilities not detected

- esdt_issue_callback(
  &self,
  caller: &ManagedAddress,
  #[payment_token] token_identifier: TokenIdentifier,
  #[payment] returned_tokens: BigUint,
  #[call_result] result: ManagedAsyncCallResult<()>,
)

  Vulnerabilities not detected

- esdt_mint_callback(
  &self,
  caller: &ManagedAddress,
  amount: &BigUint,
  #[call_result] result: ManagedAsyncCallResult<()>,
)

  Vulnerabilities not detected

- wrap_egld(&self, #[payment] payment: BigUint)

  Vulnerabilities not detected

- unwrap_egld(
  &self,
  #[payment] wrapped_egld_payment: BigUint,
  #[payment_token] token_identifier: TokenIdentifier,
)

  Vulnerabilities not detected

- get_locked_egld_balance(&self)

  Vulnerabilities not detected

VERIFICATION CHECK SUMS

Contract Name	Bytecode hash (SHA 256)
Action.rs	45f8ad2a286ad1ef828118f6bb8f1d4555e21596ec41686f70c3e05c7c16afdc
Events.rs	2082912c3968b997f7cc0ca440f44357e4de5d51f2a6f86e8497b23c4eef75c1
Lib.rs	203e338a97756a78f2ad932b89e53085015136358fef1dfe032d0393fe44eb28
User_role.rs	5648b16d02bd9ca6b0341d39b9c939b5ef48f062fbed2a57aa9982ca1bf3dcc3
Swap.rs	ec5df3d87fd443c0136191717cb4316cba378865e011f7fd3faf6379b826cb08

Get In Touch

info@smartstate.tech

smartstate.tech