Assure DeFi®

THE VERIFICATION GOLD STANDARD

Security Assessment

Venko

Date: 02/08/2024

Audit Status: PASS

Audit Edition: Solana

ASSURE DEFI®

THE VERIFICATION GOLD STANDARD

Risk Analysis

Vulnerability summary

Classification	Description
🔴 High	High-level vulnerabilities can result in the loss of assets or manipulation of data.
🟠 Medium	Medium-level vulnerabilities can be challenging to exploit, but they still have a considerable impact on smart contract execution, such as allowing public access to critical functions.
🟡 Low	Low-level vulnerabilities are primarily associated with outdated or unused code snippets that generally do not significantly impact execution, sometimes they can be ignored.
🟢 Informational	Informational vulnerabilities, code style violations, and informational statements do not affect smart contract execution and can typically be disregarded.

Executive Summary

According to the Assure assessment, the Customer's smart contract is **Well Secured.**

Insecure	Poorly Secured	Secured	Well Secured

Scope

Target Code And Revision

For this audit, we performed research, investigation, and review of the Venko verifying the functional and superficial part of the contract since it is generated in SPL token creator and we do not have access to the base code.

Target Code And Revision

Project	Assure
Language	Rust
Codebase	File [SHA256] - Venkonw.zip 0b532fde706958c81789970b8b84d2de45ee32 ad6dce90c0790cec6dec2f908f
Audit Methodology	Static, Manual

AUDIT OVERVIEW

HIGH

No high severity issues were found.

MEDIUM

No medium severity issues were found.

LOW

No low severity issues were found.

INFORMATIONAL

1. Updating URL Generator to Point to Mainnet in VenkoToken/Index.js

File: VenkoToken/Index.js

Function: generateExplorerTxUrl

Issue: The URL generator currently points to the devnet, which is intended for development and testing purposes, the URL should be updated to point to the mainnet instead of the devnet.

Solution: Update the generateExplorerTxUrl function in the Token/Index.js file to ensure that the URL generator points to the mainnet. This involves modifying the base URL within the function to use the mainnet endpoint rather than the devnet endpoint.

Annexes

Testing code:

Token test.js:

```
const {
  Connection,
  Keypair,
  SystemProgram,
  Transaction,
  PublicKey,
  clusterApiUrl,
  LAMPORTS_PER_SOL,
} = require('@solana/web3.js');
const {
  ExtensionType,
  createInitializeMintInstruction,
  mintTo,
  getMintLen,
  getTransferFeeAmount,
  unpackAccount,
  TOKEN_2022_PROGRAM_ID,
  createInitializeTransferFeeConfigInstruction,
  transferCheckedWithFee,
  withdrawWithheldTokensFromAccounts,
  createAssociatedTokenAccountIdempotent,
} = require('@solana/spl-token');
const { expect } = require('chai');

describe('Solana Token Tests', function() {
  this.timeout(30000); // Extend timeout for Solana transactions

  const opts = {
    preflightCommitment: 'confirmed',
  };

  const network = clusterApiUrl('devnet');
  const connection = new Connection(network, opts.preflightCommitment);

  let payer;
  let mintAuthority;
  let newAuthority;
  let mintKeypair;
  let mint;
  let transferFeeConfigAuthority;
  let withdrawWithheldAuthority;
  let decimals;
  let feeBasisPoints;
  let maxFee;
  let mintAmount;

  before(async () => {
    payer = Keypair.generate();
    await connection.requestAirdrop(payer.publicKey, 2 * LAMPORTS_PER_SOL);

    mintAuthority = payer;
    newAuthority = Keypair.generate();
    mintKeypair = Keypair.generate();
    mint = mintKeypair.publicKey;
    transferFeeConfigAuthority = Keypair.generate().publicKey;
    withdrawWithheldAuthority = Keypair.generate().publicKey;

    decimals = 6;
    feeBasisPoints = 200; // 2%
    maxFee = BigInt(100000 * Math.pow(10, decimals)); // 100000 tokens
    mintAmount = BigInt(1_000_000 * Math.pow(10, decimals)); // Mint 1,000,000 tokens
  });
  it('should create a new token', async () => {
    const extensions = [ExtensionType.TransferFeeConfig];
    const mintLen = getMintLen(extensions);
    const mintLamports = await
connection.getMinimumBalanceForRentExemption(mintLen);

    const mintTransaction = new Transaction().add(
      SystemProgram.createAccount({
        fromPubkey: payer.publicKey,
        newAccountPubkey: mint,
        space: mintLen,
        lamports: mintLamports,
        programId: TOKEN_2022_PROGRAM_ID,
      }),
      createInitializeTransferFeeConfigInstruction(
        mint,
        transferFeeConfigAuthority,
        withdrawWithheldAuthority,
        feeBasisPoints,
        maxFee,
        TOKEN_2022_PROGRAM_ID
      ),
      createInitializeMintInstruction(
        mint,
        decimals,
        mintAuthority.publicKey,
        null,
        TOKEN_2022_PROGRAM_ID
      )
    );

    const newTokenTx = await sendAndConfirmTransaction(
      connection,
      mintTransaction,
      [payer, mintKeypair]
    );

    expect(newTokenTx).to.be.a('string');
  });

  it('should mint tokens to the associated account', async () => {
    const owner = payer;
    const sourceAccount = await createAssociatedTokenAccountIdempotent(
      connection,
      payer,
      mint,
      owner.publicKey,
      {},
      TOKEN_2022_PROGRAM_ID
    );

    const mintSig = await mintTo(
      connection,
      payer,
      mint,
      sourceAccount,
      mintAuthority,
      mintAmount,
      [],
      undefined,
      TOKEN_2022_PROGRAM_ID
    );

    expect(mintSig).to.be.a('string');

    const accountInfo = await connection.getParsedAccountInfo(sourceAccount);
    const tokenAmount =
accountInfo.value.data['parsed']['info']['tokenAmount']['uiAmount'];
    expect(tokenAmount).to.equal(Number(mintAmount) / Math.pow(10, decimals));
  });
```

```
it('should transfer tokens with fee', async () => {
  const recipient = Keypair.generate();
  const sourceAccount = await createAssociatedTokenAccountIdempotent(
    connection,
    payer,
    mint,
    payer.publicKey,
    {},
    TOKEN_2022_PROGRAM_ID
  );

  const destinationAccount = await createAssociatedTokenAccountIdempotent(
    connection,
    payer,
    mint,
    recipient.publicKey,
    {},
    TOKEN_2022_PROGRAM_ID
  );

  const transferSig = await transferCheckedWithFee(
    connection,
    payer,
    sourceAccount,
    mint,
    destinationAccount,
    payer.publicKey,
    mintAmount / BigInt(2),
    decimals,
    BigInt(2000000000),
    []
  );

  expect(transferSig).to.be.a('string');
```

```
    const sourceAccountInfo = await connection.getParsedAccountInfo(sourceAccount);
    const destinationAccountInfo = await
connection.getParsedAccountInfo(destinationAccount);


expect(sourceAccountInfo.value.data['parsed']['info']['tokenAmount']['uiAmount']).to
.equal(Number(mintAmount) / Math.pow(10, decimals) / 2);

expect(destinationAccountInfo.value.data['parsed']['info']['tokenAmount']['uiAmount'
]).to.equal(Number(mintAmount) / Math.pow(10, decimals) / 2);
  });

  it('should fetch and withdraw withheld tokens from accounts', async () => {
    const mintPublicKey = mint;

    const allAccounts = await connection.getProgramAccounts(
      TOKEN_2022_PROGRAM_ID,
      {
        commitment: 'confirmed',
        filters: [
          {
            memcmp: {
              offset: 0,
              bytes: mintPublicKey.toString(),
            },
          },
        ],
      }
    );

    const accountsToWithdrawFrom = [];

    for (const accountInfo of allAccounts) {
      const account = unpackAccount(
        accountInfo.pubkey,
        accountInfo.account,
        TOKEN_2022_PROGRAM_ID
      );
      const transferFeeAmount = getTransferFeeAmount(account);
      if (transferFeeAmount !== null && transferFeeAmount.withheldAmount > BigInt(0)) {
        accountsToWithdrawFrom.push(accountInfo.pubkey);
      }
    }

    expect(accountsToWithdrawFrom.length).to.be.a('number');

    const sourceAccount = await createAssociatedTokenAccountIdempotent(
      connection,
      payer,
      mint,
      newAuthority.publicKey,
      {},
      TOKEN_2022_PROGRAM_ID
    );

    const withdrawSig = await withdrawWithheldTokensFromAccounts(
      connection,
      newAuthority,
      mint,
      sourceAccount,
      withdrawWithheldAuthority,
      [],
      accountsToWithdrawFrom
    );

    expect(withdrawSig).to.be.a('string');
  });
});
```

Assessment Results

Score Results

Review	Score
Global Score	**90/100**
Assure KYC	Not completed
Audit Score	90/100

The Following Score System Has been Added to this page to help understand the value of the audit, the maximum score is 100, however to attain that value the project must pass and provide all the data needed for the assessment. Our Passing Score has been changed to 84 Points for a higher standard, if a project does not attain 85% is an automatic failure. Read our notes and final assessment below. The Global Score is a combination of the evaluations obtained between having or not having KYC and the type of contract audited together with its manual audit.

Audit PASS

This audit report presents an in-depth analysis of the sol contract functions, emphasizing its immutability, renounced ownership, and financial controls.

The project has successfully passed the audit process.

Disclaimer

Assure Defi has conducted an independent security assessment to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the reviewed code for the scope of this assessment. This report does not constitute agreement, acceptance, or advocating for the Project, and users relying on this report should not consider this as having any merit for financial advice in any shape, form, or nature. The contracts audited do not account for any economic developments that the Project in question may pursue, and the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude, and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are entirely free of exploits, bugs, vulnerabilities or deprecation of technologies.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any person reading this report should invest their funds without sufficient individual due diligence, regardless of the findings presented. Information is provided ‘as is, and Assure Defi is under no covenant to audit completeness, accuracy, or solidity of the contracts. In no event will Assure Defi or its partners, employees, agents, or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions or actions with regards to the information provided in this audit report.

The assessment services provided by Assure Defi are subject to dependencies and are under continuing development. You agree that your access or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies with high levels of technical risk and uncertainty. The assessment reports could include false positives, negatives, and unpredictable results. The services may access, and depend upon, multiple layers of third parties.