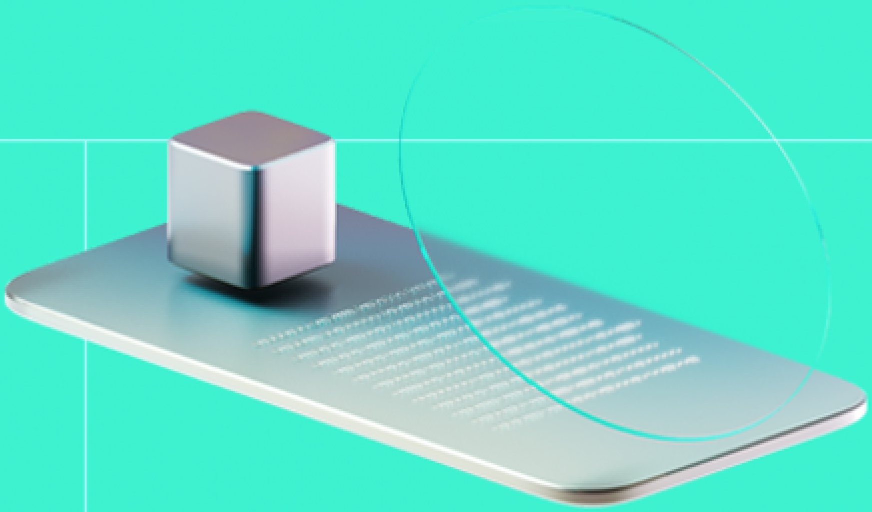
Smart Contract Code Review And Security Analysis Report



Customer: GraFun

Date: 25/09/2024

We express our gratitude to the GraFun team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

The GraFun provides Token Sales protocol that is an Initial Coin Offering (ICO) solution, allowing users to buy and sell tokens, and eventually it deploys PancakeSwap pair for token.

Document

Name	Smart Contract Code Review and Security Analysis Report for GraFun
Audited By	Grzegorz Trawinski
Approved By	Ataberk Yavuzer
Website	https://gra.fun/
Changelog	20/09/2024 - Preliminary Report
	25/09/2024 - Final Report
Platform	BNB Chain
Language	Solidity
Tags	Initial Coin Offering, ICO, Token Sales, ERC20, PancakeSwap
Methodology	https://hackenio.cc/sc_methodology

Review Scope

Repository	Hidden
Commit	504724739624914f0664f6a55e9b466cc629c4c9, dev branch
Remediation commit	1fab268195ea8fb94b2d921246fa450aa8f7addc

Audit Summary

The system users should acknowledge all the risks summed up in the risks section of the report

6	3	3	0
Total Findings	Resolved	Accepted	Mitigated

Findings by Severity

Severity	Count
Critical	0
High	1
Medium	2
Low	3

Vulnerability	Severity
F-2024-6210 - Tokens transfer causes fund loss if token sale is not finished	High
F-2024-6183 - Excessive protocol fee accounting possible	Medium
F-2024-6185 - Arbitrary referralLink can be provided to collect fee	Medium
F-2024-6187 - The finishingTime parameter lacks input validation	Low
F-2024-6191 - Lack of two-step ownership transfer pattern	Low
F-2024-6209 - Lack of emergency stop mechanism	Low

Documentation quality

- The protocol documentation was not provided.

Code quality

- The code represents clear architecture of the solution.
- No NatSpecs were provided.

Test coverage

- Code coverage of the project is **73.53%** (branch coverage).

Table of Contents

System Overview

The Token Sales is an Initial Coin Offering (ICO) solution with the following contracts:

- **Token** - a contract representing the ERC20 token that is a subject of the Initial Coin Offering. The contract instance is created by means of the TokenSaleFactory contract.
- **TokenSaleFactory** - a contract that allows users to start new Initial Coin Offering, buy and sell tokens, redeem purchases after token sale's deadline. It also deploy the PancakeSwap instances.
- **TokenSaleHelpers** - a library with a set of functions used by the aforementioned contracts.

Privileged roles

- The owner of the TokenSaleFactory contract can upgrade the contract.

Potential Risks

Potentially Impossible Redemption: The protocol is designed to allow buy and sell tokens until the token sale period ends. The sale period ends either when PancakeSwap instance is deployed or deadline is reached. Whenever users transfer tokens between accounts, they lose possibility to redeem tokens. In the event of token sale period end without PancakeSwap instance deployment, users who did not sell tokens previously will likely encounter financial loss, as tokens held may have no economic value.

Single Points of Failure and Control: The project is fully or partially centralized, introducing single points of failure and control. This centralization can lead to vulnerabilities in decision-making and operational processes, making the system more susceptible to targeted attacks or manipulation.

Administrative Key Control Risks: The digital contract architecture relies on administrative keys for critical operations. Centralized control over these keys presents a significant security risk, as compromise or misuse can lead to unauthorized actions or loss of funds.

Single Entity Upgrade Authority: The token ecosystem grants a single entity the authority to implement upgrades or changes. This centralization of power risks unilateral decisions that may not align with the community or stakeholders' interests, undermining trust and security.

Flexibility and Risk in Contract Upgrades: The project's contracts are upgradable, allowing the administrator to update the contract logic at any time. While this provides flexibility in addressing issues and evolving the project, it also introduces risks if upgrade processes are not properly managed or secured, potentially allowing for unauthorized changes that could compromise the project's integrity and security.

Absence of Upgrade Window Constraints: The contract suite allows for immediate upgrades without a mandatory review or waiting period, increasing the risk of rapid deployment of malicious or flawed code, potentially compromising the system's integrity and user assets.

Findings

Vulnerability Details

F-2024-6210 - Tokens transfer causes fund loss if token sale is not finished - High

Description:

The `TokenSaleFactory` contract allows to configure and deploy a token sale. Within the token sale period users can buy tokens and sell them within the `TokenSaleFactory` contract. When token sale is finished, a PancakeSwap instance is deployed with WETH and token. If token sale is not finished, users can redeem their purchases after specified time. Also, the `Token` instance can be transferred between users during the token sale. However, whenever bought tokens are transferred to other user, the purchaser's redeem record is decreased by the amount transferred. Also, new record is not created for the tokens receiver.

```
    function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual override {
        if (!_tokenSaleFactory.isFinished(address(this))) {
            require(!_restrictedAddresses[from] && !_restrictedAddresses[to], "TSF: tokensale didn't finished");

            // Check vulnerability if user transfers token to token sale factory contract
            if (from != address(_tokenSaleFactory) && to != address(_tokenSaleFactory)) {
                _tokenSaleFactory.onTransfer(from, amount);
            }
        }
    }
```

```
    function onTransfer(address from, uint256 amount) external {
        address token = msg.sender;
        TokenSaleInfo storage tokenSale = _tokenSaleInfo[token];
        require(tokenSale.sqrtPriceX96 != 0, "TSF: call not from token");

        RedeemBalance storage redeemBalance = _redeemBalances[token][from];
        uint256 tokenBought = redeemBalance.tokenBough;
        if (tokenBought == 0) return;
        amount = _min(amount, tokenBought);

        uint256 wethSpend = redeemBalance.wethSpend;
        uint256 amountToSubtract = wethSpend *  amount / tokenBought;
        redeemBalance.wethSpend = wethSpend - amountToSubtract;
        redeemBalance.tokenBough = tokenBought - amount;
        tokenSale.totalWethRedeem -= amountToSubtract;
    }
```

This implementation has two outcomes:

- The token's holder cannot redeem them anymore by means of the `redeem` function, when tokens sale time is finished and PancakeSwap instance is not created.
- The token's holder cannot sell them anymore by means of the `sell` function, when tokens sale time is in progress.

The `redeem` for such holder is not possible as `redeemBalance.tokenBough` and `redeemBalance.wethSpend` are equal to 0.

```
    function redeem(address token, uint256 amount) external {
        ...
        RedeemBalance storage redeemBalance = _redeemBalances[token][msg.send
er];

        require(
            amount <= redeemBalance.tokenBough &&
            redeemBalance.wethSpend > 0 &&
            tokenSale.totalWethRedeem > 0, "TSF: overredeem"
        );
        ...
```

The `sell` for such holder is not possible as `_setSellRedeemValues` function reverts with `panic: division or modulo by zero (0x12)` error message. This happens because the `oldTokenRedeemValue` variable can be equal to 0 when user had no purchased tokens before.

```
    function _setSellRedeemValues(
        address token,
        address user,
        uint256 tokenAmount,
        uint256 wethAmount
    ) internal {
            TokenSaleInfo storage tokenSale = _tokenSaleInfo[token];
            RedeemBalance storage redeemBalance = _redeemBalances[token][user
];

            uint256 oldTokenRedeemValue = redeemBalance.tokenBough;
            uint256 newTokenRedeemValue = _min(tokenAmount, oldTokenRedeemVal
ue);
            uint256 oldWethRedeemValue = redeemBalance.wethSpend;
```

```
          uint256 newWethRatio = oldWethRedeemValue * newTokenRedeemValue /
oldTokenRedeemValue;
          ...
```

As a result, in the event of the unsuccessful token sale, user encounter funds loss as he/she holds tokens which are not redeemable and swapable back to WETH.

Assets:

- TokenSaleFactory.sol [https://github.com/dexe-network/grafun-contrakts]
- Token.sol [https://github.com/dexe-network/grafun-contrakts]

Status: Fixed

Classification

Impact: 5/5

Likelihood: 3/5

Exploitability: Independent

Complexity: Simple

Severity: High

Recommendations

Remediation: It is recommended to review the protocol's design and defined business rules and decide whether:

- Transfer of tokens should not be possible before the tokens sale is finished.
- The redeem records accounting should be updated to take into account scenario when tokens are transferred between token sales participants.

Resolution: The `sell` function is now updated, so it allows to sell tokens for users, who did not buy them previously. (commit Id: 5b391c80379274e6dca890b486f1450016057080).

Additional comment from the Client's team:

> Buy and sell work only until `tokenSale` is finished, this is the expected behavior. If `tokenSale` come to deadline not finished, user could redeem everything he bought but did not transferred or sold. If Alice buys 1 token and transfers it to

Bob, Bob cannot redeem it but he could transfer it freely before or after deadline or token sale end.

Evidences

Proof of Concept

Reproduce:

1. As a protocol owner, deploy and configure the solution.
2. As a creator, create new token sale instance.
3. As a user1, buy an amount of tokens.
4. As a user1, transfer bought tokens to the user2.
5. As a user2, attempt to sell tokens within token sale instance. Observe the transaction reverts with `TSF: can't redeem` error.
6. Forward blockchain time so the token sale is finished unsuccessfully.
7. As a user2, attempt to redeem tokens within token sale instance. Observe the transaction reverts with `panic: division or modulo by zero (0x12)` error.

F-2024-6183 - Excessive protocol fee accounting possible - Medium

Description:

Within the `_buy` function the protocol `fee` is accounted from the `wethAmount` provided to the function. However, the protocol firstly accounts the fee, then it calculates the `wethAmountToReturn` amount, which should be returned to the user whenever excessive amount was provided. The protocol accepts maximum number of tokens equal to `TOTAL_BNB_TO_COLLECT` which is **31 ether**.

This implementation leads to possibility of excessive `fee` accounted from the tokens, which should be returned back to the user. Such situation may happen whenever multiple users call the `buy` function without proper amount set in slippage, represented by the `minTokenAmount` input parameter. Whenever the `TOTAL_BNB_TO_COLLECT` threshold is nearly reached, the victim user will pay overestimated fee from provided native token and acquire low number of residual tokens.

```
function _buy(
        address token,
        uint256 minTokenAmount,
        address referralLink,
        uint256 wethAmount
    ) internal {
        TokenSaleInfo storage tokenSale = _tokenSaleInfo[token];
        require(!isFinished(token), "TSF: tokensale finished");

        require(wethAmount > 0, "TSF: zero buy");

        uint256 fee = _calculateSwapFee(wethAmount);

        require(wethAmount > fee, "TSF: Insufficient fee");
...
        uint256 wethAmountToReturn;
        if (wethAmount + wethBalance > TOTAL_BNB_TO_COLLECT) {
            wethAmountToReturn = wethAmount - (TOTAL_BNB_TO_COLLECT - wethBal
ance);
            wethAmount = TOTAL_BNB_TO_COLLECT - wethBalance;
        }
...
    }
```

Assets:	• TokenSaleFactory.sol [https://github.com/dexe-network/grafun-contrakts]
Status:	Fixed

Classification

Impact:	4/5
Likelihood:	2/5
Exploitability:	Independent
Complexity:	Simple
Severity:	Medium

Recommendations

Remediation:	It is recommended to calculate the protocol `fee` based on the amount of tokens which does not include amount to return in every case.
Resolution:	The fee is now adjusted and calculated correctly, based on the WETH amount that must be returned to the buyer (commit Id: 1fab268195ea8fb94b2d921246fa450aa8f7addc).

F-2024-6185 - Arbitrary referralLink can be provided to collect fee - Medium

Description: The `TokenSaleFactory` contract allows to set `referralLink` for both `buy` and `sell` functions. Whenever an address is set, this particular wallet receives the `referrerAmount` of native tokens, which decreases the protocol fee. The referrer fee is set to **10%**. However, there is no list of allowed referrers implemented. Thus, user can set arbitrary account, e.g. under his/her control and receive **10%** of the fee back.

```
    function _checkReferal(address referal) internal returns (address) {
        address oldReferal = _referrees[msg.sender];
        if (oldReferal == address(0)) {
            _referrees[msg.sender] = referal;
            _referrers[referal].push(msg.sender);
            return referal;
        } else {
            return oldReferal;
        }
    }
```

```
...
        if (referralLink != msg.sender && referralLink != address(0)) {
            uint256 referrerAmount = fee * PERCENT_TO_REFERRER / ONE_HUNDREED
_PERCENTS;
            _transferETH(referralLink, referrerAmount);
            _transferETH(treasury, fee - referrerAmount);
        } else {
            _transferETH(treasury, fee);
        }
...
```

Assets:

- TokenSaleFactory.sol [https://github.com/dexe-network/grafun-contrakts]

Status: Accepted

Classification

Impact: 2/5

Likelihood: 5/5

Exploitability:	Independent
Complexity:	Simple
Severity:	Medium

Recommendations

Remediation:	It is recommended to implement a whitelist of approved referrers. This would prevent users from setting an arbitrary account under their control and receiving a portion of the fee back.
Resolution:	The Client's team claimed this finding as an expected behavior. Therefore, the finding is accepted.

F-2024-6187 - The finishingTime parameter lacks input validation - Low

Description: The `finishingTime` parameter within the `_createTokenSale` function can be used to set the deadline when token sale has ended and redemption of the sale process is possible. However, it lacks input validation. Thus, this parameter can be set into the past value or to too short time in the future. Eventually, such token sale can finish immediately.

```
    function _createTokenSale(
        address token,
        string calldata metadata_,
        address referralLink,
        bool withDao,
        uint256 finishingTime,
        uint256 maxWalletAmount
    ) internal returns (address) {
        IToken sellToken = IToken(token);
...
        if (withDao) {
            tokenSale.finishingTime = finishingTime == 0 ? type(uint256).max
: finishingTime;
        } else {
            tokenSale.finishingTime = type(uint256).max;
        }
```

Assets:

- TokenSaleFactory.sol [https://github.com/dexe-network/grafun-contrakts]

Status: Fixed

Classification

Impact:	3/5
Likelihood:	2/5
Exploitability:	Independent
Complexity:	Simple
Severity:	Low

Recommendations

Remediation: It is recommended to validate the `finishingTime` parameter to ensure that it is not set to a past value or to a very short time in the future. This can be implemented by adding an assertion like this:

```
require(finishingTime > block.timestamp + minimumTime, "finishingTime is too
soon");
```

Where `minimumTime` is a predefined constant that represents the minimum acceptable duration for the token sale. This will prevent the token sale from finishing immediately after it is created.

Resolution: The input validation is now improved. The `finishingTime` parameter can be set to 0 or must be set above current `block.timestamp` (commit Id: 5b391c80379274e6dca890b486f1450016057080).

[F-2024-6191](F-2024-6191) - Lack of two-step ownership transfer pattern - Low

Description: The `TokenSaleFactory` implements single step of ownership transfer. In the event of a transfer to invalid address, the transfer is immediate and the authority is lost. Thus, access to all functionalities protected by the `restricted` modifier will be permanently lost.

```
contract TokenSaleFactory is OwnableUpgradeable, UUPSUpgradeable {
...
```

Assets:

- TokenSaleFactory.sol [https://github.com/dexe-network/grafun-contrakts]

Status: Accepted

Classification

Impact: 5/5

Likelihood: 1/5

Exploitability: Dependent

Complexity: Simple

Severity: Low

Recommendations

Remediation: It is recommended to implement two-step ownership transfer. In the first step a new proposal address should be provided. In the second step the proposal address should confirm the transfer. This can be achieved by importing OpenZepplin's `Ownable2StepUpgradeable`.

Resolution: The Client's Team accepted the finding.

F-2024-6209 - Lack of emergency stop mechanism - Low

Description: The `TokenSaleFactory` contract does not implement any emergency stop mechanism. This means that whenever the protocol will be under adversary attack the temporary halt of the processing will be not possible, until the protocol is upgraded. Thus, key functions such as `buy` and `sell` could be continuously exploited.

```
contract TokenSaleFactory is OwnableUpgradeable, UUPSUpgradeable {
```

Assets:

- TokenSaleFactory.sol [https://github.com/dexe-network/grafun-contrakts]

Status: Accepted

Classification

Impact:	2/5
Likelihood:	2/5
Exploitability:	Independent
Complexity:	Simple
Severity:	Low

Recommendations

Remediation: It is recommended to implement emergency stop mechanism. This can be achieved by importing OpenZepplin's `PausableUpgradable` contract and making use of `whenNotPaused` modifier.

Resolution: The Client's Team accepted the finding.

Observation Details

F-2024-6179 - The _tokenSaleFactory can be immutable - Info

Description: The `_tokenSaleFactory` is set only once in the constructor, thus it can be set as `immutable`.

The `immutable` variables can only be assigned during contract creation (inside the constructor). Once assigned, their value cannot be changed. This is a good practice for variables that should not change after the contract is deployed.

Assets:

- Token.sol [https://github.com/dexe-network/grafun-contrakts]

Status: Fixed

Recommendations

Remediation: It is recommended to mark the `ITokenSaleFactoryMin` as immutable:

```
ITokenSaleFactoryMin internal immutable ITokenSaleFactoryMin;
```

Resolution: The `ITokenSaleFactoryMin` state variable is now set to immutable (commit Id: 5b391c80379274e6dca890b486f1450016057080).

F-2024-6181 - The Ownable inheritance is redundant - Info

Description: The `Token` contract inherits from the `Ownable` library. This is done only to protect `addRestricted` function call used within the `restrictAddresses` function. However, the `Token` contract already saves the value of the `TokenSaleFactory` instance in the `_tokenSaleFactory` parameter, which can be used to protect the `addRestricted` function against unauthorised access.

Assets:

- Token.sol [https://github.com/dexe-network/grafun-contrakts]

Status: Accepted

Recommendations

Remediation: It is recommended to remove the `Ownable` library usage to save some Gas and use the `_tokenSaleFactory` state variable for the purpose of the authorisation.

Resolution: The Client's Team accepted the finding.

F-2024-6182 - TokenSaleFactory implementation is not disabled - Info

Description: The `TokenSaleFactory` contract is upgradable, however, it does not disable the implementation. Thus, the implementation can be hijacked by the threat actor and leveraged for the further attacks, such as phishing. The finding was reported as a deviation from leading security practices.

```
 contract TokenSaleFactory is OwnableUpgradeable, UUPSUpgradeable {
...
    constructor(
        address v3Factory,
        address weth
    ) {
        PANCAKE_V3_FACTORY = v3Factory;
        WETH = weth;
    }

    function __TokenSaleFactory_init(address _treasury, address _poolFactory)
public initializer {
        __Ownable_init();

        treasury = _treasury;
        poolFactory = _poolFactory;
    }
```

Assets:

- TokenSaleFactory.sol [https://github.com/dexe-network/grafun-contrakts]

Status: Accepted

Recommendations

Remediation: It is recommended to disable the implementation by means of the `_disableInitializers()` function.

```
    constructor(
        address v3Factory,
        address weth
    ) {
        PANCAKE_V3_FACTORY = v3Factory;
        WETH = weth;
        _disableInitializers()
    }
```

Resolution: The Client's Team accepted the finding.

F-2024-6186 - The RedeemBalance struct has typo - Info

Description: The `RedeemBalance` has typo: the parameter name should be `tokenBought` instead of `tokenBought`

```
    struct RedeemBalance {
        uint256 tokenBough;
        uint256 wethSpend;
    }
```

Assets:

- TokenSaleFactory.sol [https://github.com/dexe-network/grafun-contrakts]

Status: Accepted

Recommendations

Remediation: The typo in the `RedeemBalance` struct should be corrected to ensure that the code is accurate and functions as expected. The parameter name `tokenBough` should be changed to `tokenBought`. This change should be propagated throughout the codebase to ensure that all references to this struct are accurate.

Resolution: The Client's Team accepted the finding.

F-2024-6189 - Lack of Monitoring on Key Functions - Info

Description: The protocol currently lacks monitoring for its key functions, specifically `buy` , `sell`, `redeem` and `transferOwnership`. Without proper monitoring, it is challenging to detect and respond to unauthorized activities, irregular behaviors, or potential hacking attempts in real time. This absence of surveillance increases the risk of undetected exploits or issues that could compromise the contract's security and integrity.

Assets:

- TokenSaleFactory.sol [https://github.com/dexe-network/grafun-contrakts]
- TokenSaleHelpers.sol [https://github.com/dexe-network/grafun-contrakts]

Status: Accepted

Recommendations

Remediation: It is recommended to implement a comprehensive monitoring mechanism for the key functions `buy` , `sell`, `redeem` and `transferOwnership`. within the protocol. Continuous monitoring of these functions is essential to ensure that the contract operates as intended and to detect any anomalies or unauthorized activities in real time. By closely observing these critical areas, it is possible to identify and respond promptly to any signs of hacking or unusual behavior. This proactive approach not only helps in maintaining the integrity and security of the smart contract but also provides an additional layer of assurance for all stakeholders involved.

Resolution: The Client's Team accepted the finding.

F-2024-6193 - Lack of IERC20 transfer/transferFrom return value check - Info

Description: Within the `TokenSaleFactory` multiple instances of `IERC20`'s `transfer()` and `transferFrom()` methods usage were observed to handle token transfers. However, no return value is checked upon finishing the transfer. As the protocol aims to be used with `WBNB` token and custom EIP-20 compliant token, this finding is reported as a deviation from leading security practices.

```
 ...
        tokenContract.transfer(treasury, TOTAL_SUPPLY * INCENTIVE_FEE / ONE_H
UNDREED_PERCENTS); //@audit no safetransfer
...
        tokenContract.transfer(daoAddress, sentToDao);
...
        if (amount0Owed > 0) {
            IERC20(token0).transfer(msg.sender, amount0Owed);
        }
        if (amount1Owed > 0) {
            IERC20(token1).transfer(msg.sender, amount1Owed);
        }
...
```

Assets:

- TokenSaleFactory.sol [https://github.com/dexe-network/grafun-contrakts]
- TokenSaleHelpers.sol [https://github.com/dexe-network/grafun-contrakts]

Status: Accepted

Recommendations

Remediation: It is recommended to use OpenZeppelin's `SafeERC20` library, which provides `safeTransfer()` and `safeTransferFrom()` functions that handle standard and non-standard cases gracefully. `SafeERC20` wraps the standard ERC20 functions and ensures compatibility with both standard-compliant and non-compliant tokens. Additionally, `SafeERC20` methods automatically perform the necessary checks for allowance and balance, making the code cleaner and more concise.

Resolution: The Client's Team accepted the finding.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Definitions

Severities

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Potential Risks

The "Potential Risks" section identifies issues that are not direct security vulnerabilities but could still affect the project’s performance, reliability, or user trust. These risks arise from design choices, architectural decisions, or operational practices that, while not immediately exploitable, may lead to problems under certain conditions. Additionally, potential risks can impact the quality of the audit itself, as they may involve external factors or components beyond the scope of the audit, leading to incomplete assessments or oversight of key areas. This section aims to provide a broader perspective on factors that could affect the project's long-term security, functionality, and the comprehensiveness of the audit findings.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details	
Repository	Hidden
Commit	5047247
Remediation commit	1fab2681
Whitepaper	N/A
Requirements	N/A
Technical Requirements	N/A

Contracts in Scope
./contracts/Token.sol
./contracts/TokenSaleFactory.sol
./contracts/TokenSaleHelpers.sol
./contracts/libs/SqrtPriceX96.sol