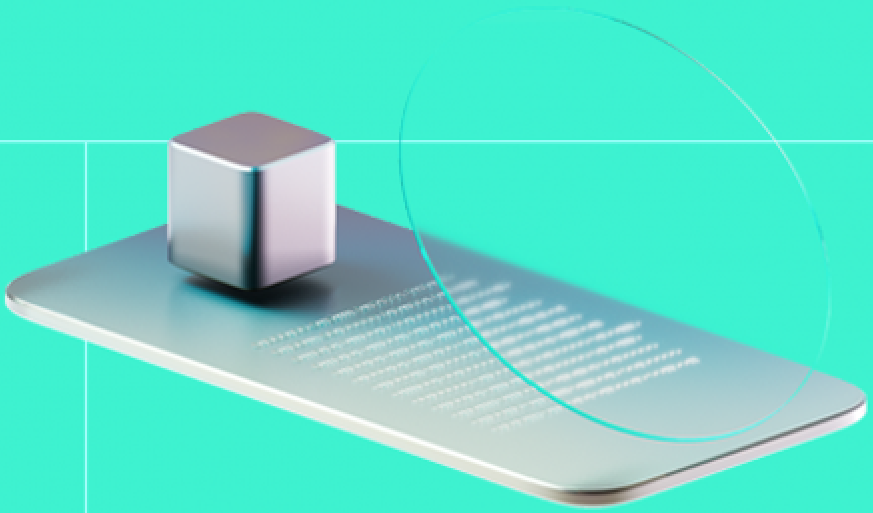
Smart Contract Code Review And Security Analysis Report



Customer: Wormfare

Date: 06/03/2024

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

Wormfare is a next-level gaming experience that goes beyond just play-to-earn. Wormfare offers a unique blend of engaging gameplay, crypto incentives, and magical community quests.

Platform: EVM

Language: Solidity

Tags: ERC20 Presale

Timeline: 22/02/2024 - 06/03/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/wormfare/contracts/commits/main/
Commit	a2b19943777bc505f22e84acdac50ca13b54d6e0
Remediation Commit	39c93e85367d911a7146ea27c70fc0b860df6d2a

Audit Summary

10/10	10/10	98%	10/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 10/10

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

4	4	0	0
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	0
Low	3

Vulnerability	Status
F-2024-1027 - Lack of Parameter Validation in initialize, setApiSigner, and setTokenPriceUsdt Functions	Fixed
F-2024-1029 - Backend API takeover can lead to unlimited token purchase for free	Fixed
F-2024-1030 - DEFAULT_ADMIN_ROLE can frontrun buy() transaction	Fixed
F-2024-1031 - Checks-Effects-Interactions pattern violation	Fixed

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for Wormfare
Audited By	Niccolò Pozzolini, Kornel Świątłowski
Approved By	Przemyslaw Swiatowiec
Website	https://wormfare.com/
Changelog	26/02/2024 - Preliminary Report; 06/03/2024 - Second Review

Table of Contents

System Overview

TokenSale is a smart contract that lets its owners presale an ERC20 token before the actual token is issued. The contract allows users to purchase tokens with USDT, incorporating discount and referral reward mechanisms. The contract works with a backend API that produces and signs the parameters for a user when they want to make a purchase. All USDT received except the referral reward part is transferred to a “treasury” wallet specified during the contract deployment. After purchase, users do not receive actual tokens, the contract only tracks user token balances.

Privileged roles

TokenSale contract uses the AccessControlUpgradeable library from OpenZeppelin to restrict access to important functions. The contract contains DEFAULT_ADMIN_ROLE that can:

- pause and unpause contract,
- update apiSigner address,
- update token price in USDT,
- buy tokens for someone else without generated signature

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are detailed.
- Technical description is robust.

Code quality

The total Code Quality score is **10** out of **10**.

Test coverage

Code coverage of the project is **98.53%** (branch coverage)

- Deployment and basic user interactions are covered with tests.
- Negative cases are covered.

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **0** medium, and **3** low severity issues, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **10**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

No additional risks were identified.

Findings

Vulnerability Details

F-2024-1027 - Lack of Parameter Validation in initialize, setApiSigner, and setTokenPriceUsdt Functions - Low

Description:

The `initialize`, `setApiSigner`, and `setTokenPriceUsdt` functions in the `TokenSale.sol` contract does not perform any validation on its parameters. This lack of validation can lead to potential misconfigurations due to human errors.

Specifically, parameters such as `_usdtContract`, `_treasuryWallet`, and `_totalTokensForSale` in the `initialize` function, if set incorrectly, cannot be changed without redeploying the contract. This can lead to significant inconvenience and potential loss of funds.

Assets:

- contracts/TokenSale.sol [https://github.com/wormfare/contracts/commit/a2b19943777bc505f22e84acdac50ca13b54d6e0]

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 2
Impact [1-5]: 2
Exploitability [0-2]: 0
Complexity [0-2]: 0
Final Score: 2.0 (Low)
Hacken Calculator Version: 0.6

Recommendations

Recommendation:

To mitigate this issue, it is recommended to:

1. Implement checks in these functions to validate that the parameters are not zero values.
2. Check address parameters against the zero address.

Remediation(Commit: 39c93e85): Validation of input parameters has been added to `initialize()`, `setApiSigner()`, and `setTokenPriceUsdt()` functions.

F-2024-1029 - Backend API takeover can lead to unlimited token purchase for free - Low

Description:

The Backend API facilitates the creation of signatures enabling users to purchase tokens during the presale. These signatures are generated using specific data fields, including:

- recipient address,
- token amount in USDT,
- discount percentage,
- referral wallet address,
- referral reward percentage,
- sender address.

In a situation when the Backend API is compromised, an attacker can generate a valid signature with a referral reward percentage set to `100%`. With this signature, the attacker can purchase tokens and subsequently withdraw all `Tether` paid during the `buy()` transaction. This loophole allows the attacker to repeat the process indefinitely, acquiring an unlimited number of tokens during the presale. Notably, the current signature mechanism permits infinite reuse without any implemented deadline for verification.

Assets:

- contracts/TokenSale.sol [https://github.com/wormfare/contracts/commit/a2b19943777bc505f22e84acdac50ca13b54d6e0]

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 1
Impact [1-5]: 5
Exploitability [0-2]: 2
Complexity [0-2]: 1
Final Score: 1.7 (Low)
Hacken Calculator Version: 0.6

Recommendations

Recommendation:

It is recommended to incorporate validation within the `buyWithReferral()` function to ensure that the `_referralRewardPercent` argument value remains below the introduced cap.

Remediation(Commit: 39c93e85): Validation of `_referralRewardPercent` argument value has been added to the `internalBuy()` function.

F-2024-1031 - Checks-Effects-Interactions pattern violation - Low

Description:

State variables are updated after the external calls to the token contract.

As explained in Solidity Security Considerations, it is best practice to follow the checks-effects-interactions pattern when interacting with external contracts to avoid reentrancy-related issues.

```
uint _treasuryWalletAmountUsdt = _amountUsdt;
if (_referralWallet != address(0)) {
//@audit external in buyWithReferral() -> usdtContract.safeTransferFrom(
)
_treasuryWalletAmountUsdt = buyWithReferral(
_to,
_amountUsdt,
_referralWallet,
_referralRewardPercent
);
}
//@audit state variables are updated here
tokenBalances[_to] += _tokenAmount;
totalSoldTokens += _tokenAmount;
```

Assets:

- contracts/TokenSale.sol [https://github.com/wormfare/contracts/commit/a2b19943777bc505f22e84acdac50ca13b54d6e0]

Status: Fixed

Classification

Severity: Low

Impact:

Likelihood [1-5]: 2
Impact [1-5]: 2
Exploitability [0-2]: 0
Complexity [0-2]: 1
Final Score: 1.8 (Low)
Hacken Calculator Version: 0.6

Recommendations

Recommendation:

It is suggested to follow the checks-effects-interactions pattern when interacting with external contracts. Specifically, the storage variables `tokenBalances` and `totalSoldTokens` should be updated before calling the function `buyWithReferral` which contains an external call.

Remediation(Commit: 39c93e85): `tokenBalances` and `totalSoldTokens` storage variables are now updated before external calls. internalBuy() function follows Checks-Effects-Interactions pattern.

F-2024-1030 - DEFAULT_ADMIN_ROLE can frontrun buy() transaction - Info

Description:

Users are able to purchase new tokens with USDT with buy() function. The quantity of tokens allocated to a user is determined by the current USDT price, which is stored in the tokenPriceUsdt variable. However, entities possessing the DEFAULT_ADMIN_ROLE role have the ability to manipulate this price using the setTokenPriceUsdt() function. Exploiting this vulnerability allows such entities to front-run user transactions, altering the price to be less favorable for the user, resulting in them receiving fewer tokens despite paying the same value in USDT.

```
uint _tokenAmount = ((_amountUsdt * 1 ether) / _tokenPriceUsdt);
```

Assets:

- contracts/TokenSale.sol [https://github.com/wormfare/contracts/commit/a2b19943777bc505f22e84acdac50ca13b54d6e0]

Status: Fixed

Classification

Severity: Info

Impact:

Likelihood [1-5]: 1
Impact [1-5]: 4
Exploitability [0-2]: 2
Complexity [0-2]: 1
Final Score: 1.5 (Informational)
Hacken Calculator Version: 0.6

Recommendations

Recommendation:

It is recommended add the current USDT price to the signature and validating whether the passed price matches the value stored within the tokenPriceUsdt variable.

Remediation(Commit: 39c93e85): tokenPriceUsdt has been added to signature.

Observation Details

F-2024-1021 - Gas inefficiency due to missing usage of Solidity custom errors - Info

Description:

Starting from Solidity version 0.8.4, the language introduced a feature known as "custom errors". These custom errors provide a way for developers to define more descriptive and semantically meaningful error conditions without relying on string messages. Prior to this version, developers often used the `require` statement with string error messages to handle specific conditions or validations. However, every unique string used as a revert reason consumes gas, making transactions more expensive.

Custom errors, on the other hand, are identified by their name and the types of their parameters only, and they do not have the overhead of string storage. This means that, when using custom errors instead of `require` statements with string messages, the gas consumption can be significantly reduced, leading to more gas-efficient contracts.

Assets:

- contracts/TokenSale.sol [https://github.com/wormfare/contracts/commit/a2b19943777bc505f22e84acdac50ca13b54d6e0]

Status: Fixed

Recommendations

Recommendation:

It is recommended to use custom errors instead of reverting strings to reduce increased Gas usage, especially during contract deployment. Custom errors can be defined using the `error` keyword and can include additional information.

Remediation(Commit: 39c93e85): Usage of custom errors has been added.

F-2024-1022 - Floating pragma - Info

Description:

The project uses floating pragmas ^0.8.23.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Assets:

- contracts/TokenSale.sol [https://github.com/wormfare/contracts/commit/a2b19943777bc505f22e84acdac50ca13b54d6e0]

Status:

Fixed

Recommendations

Recommendation:

Consider locking the pragma version.

Remediation(Commit: 39c93e85): Pragma version has been locked to `0.8.23`.

F-2024-1023 - Public functions that should be external - Info

Description:

Functions that are meant to be exclusively invoked from external sources should be designated as `external` rather than `public`. This is essential to enhance both the gas efficiency and the overall security of the contract.

External visibility can be added to `initialize()` function.

Assets:

- contracts/TokenSale.sol [https://github.com/wormfare/contracts/commit/a2b19943777bc505f22e84acdac50ca13b54d6e0]

Status: Fixed

Recommendations

Recommendation:

To optimize gas usage and improve code clarity, declare functions that are not called internally within the contract and are intended for external access as `external` rather than `public`. This ensures that these functions are only callable externally, reducing unnecessary gas consumption and potential security risks.

Remediation(Commit: 39c93e85): `external` visibility has been used in `initialize()` function.

F-2024-1024 - Missing events emitting for critical functions - Info

Description:

Events for critical state changes should be emitted for tracking actions off-chain.

Events are crucial for tracking changes on the blockchain, especially for actions that alter significant contract states or permissions. The absence of events in these functions means that external entities, such as user interfaces or off-chain monitoring systems, cannot effectively track these important changes.

It was observed that events are missing events in the following functions:

- setApiSigner()
- setTokenPriceUsdt()

Assets:

- contracts/TokenSale.sol [https://github.com/wormfare/contracts/commit/a2b19943777bc505f22e84acdac50ca13b54d6e0]

Status: Fixed

Recommendations

Recommendation:

Consider emitting the corresponding events in the critical functions.

Remediation(Commit: 39c93e85): Events has been added to `setApiSigner()` and `setTokenPriceUsdt()` functions.

F-2024-1026 - Misleading Parameter Name and Lack of Detailed Information in NatSpec - Info

Description:

The `_amountUsdt` parameter in the `buy`, `buyFor` and `internalBuy` functions in the `TokenSale.sol` file has a potentially misleading name. The USDT token has 6 decimals, but this parameter is expected to have 18 decimals. This discrepancy can lead to confusion and potential errors.

Furthermore, this behavior is not detailed in the functions' NatSpec comments, which should provide comprehensive information about the function and its parameters for better understanding and usage.

Assets:

- contracts/TokenSale.sol [https://github.com/wormfare/contracts/commit/a2b19943777bc505f22e84acdac50ca13b54d6e0]

Status: Fixed

Recommendations

Recommendation:

To improve clarity and prevent potential misunderstandings, it is recommended to update the NatSpec comments for these functions to clearly indicate the expected format of this parameter.

Remediation(Commit: 39c93e85): NatSpec has been updated with information about expected format of `_amountUsdt` parameter.

F-2024-1028 - Redundant Math in Token Sale Calculation - Info

Description:

In the TokenSale.sol file, there is a block of code that performs unnecessary calculations. Specifically, when checking if the totalSoldTokens plus _tokenAmount exceeds totalTokensForSale, it calculates _redundantTokens and then subtracts this from _tokenAmount.

This operation is redundant because _tokenAmount can be directly set to totalTokensForSale - totalSoldTokens, which represents the remaining tokens available for sale. This simplification can improve code readability and efficiency.

Here is the affected code block:

```
if (totalSoldTokens + _tokenAmount > totalTokensForSale) {
uint _redundantTokens = totalSoldTokens +
_tokenAmount -
totalTokensForSale;
_tokenAmount -= _redundantTokens;
_amountUsdt = getUsdtPrice(_tokenAmount, _tokenPriceUsdt);
}
```

Assets:

- contracts/TokenSale.sol [https://github.com/wormfare/contracts/commit/a2b19943777bc505f22e84acdac50ca13b54d6e0]

Status: Fixed

Recommendations

Recommendation:

Remove the redundant math calculations by setting _tokenAmount to totalTokensForSale - totalSoldTokens.

Remediation(Commit: 39c93e85): Redundant math calculation has been removed, suggested solution has been implemented.

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. Severity Definitions

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.

Appendix 2. Scope

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

Scope Details

Repository	https://github.com/wormfare/contracts/commits/main/
Commit	a2b19943777bc505f22e84acdac50ca13b54d6e0
Whitepaper	https://whitepaper.wormfare.com/
Requirements	https://github.com/wormfare/contracts/tree/main/docs
Technical Requirements	https://github.com/wormfare/contracts/tree/main/docs

Contracts in Scope

./contracts/TokenSale.sol