SECURITY AUDIT OF

GLUTEU

Public Report

Feb 10, 2025

Verichains Lab

info@verichains.io

https://www.verichains.io

Driving Technology > Forward

ABBREVIATIONS

Name	Description
Ethereum	An open source platform based on blockchain technology to create and distribute smart contracts and decentralized applications.
Ether (ETH)	A cryptocurrency whose blockchain is generated by the Ethereum platform. Ether is used for payment of transactions and computing services in the Ethereum network.
Smart contract	A computer protocol intended to digitally facilitate, verify or enforce the negotiation or performance of a contract.
Solidity	A contract-oriented, high-level language for implementing smart contracts for the Ethereum platform.
Solc	A compiler for Solidity.
ERC20	ERC20 (BEP20 in Binance Smart Chain or *x*RP20 in other chains) tokens are blockchain-based assets that have value and can be sent and received. The primary difference with the primary coin is that instead of running on their own blockchain, ERC20 tokens are issued on a network that supports smart contracts such as Ethereum or Binance Smart Chain.
ERC721	The ERC-721 introduces a standard for NFT, in other words, this type of Token is unique and can have different value than another Token from the same Smart Contract, maybe due to its age, rarity or even something else like its visual.

EXECUTIVE SUMMARY

This Security Audit Report was prepared by Verichains Lab on Feb 10, 2025. We would like to thank the Gluteus Maximus team for trusting Verichains Lab in auditing smart contracts. Delivering high-quality audits is always our top priority.

This audit focused on identifying security flaws in code and the design of the GLUTEU. The scope of the audit is limited to the source code files provided to Verichains. Verichains Lab completed the assessment using manual, static, and dynamic analysis techniques.

During the audit process, the audit team had identified some vulnerable issues in the smart contracts code.

TABLE OF CONTENTS

1. MANAGEMENT SUMMARY

1.1. About GLUTEU

Gluteus Maximus is a sentient AI agent preparing for the Crypto Wars, a social crypto game where humans and AI agents can compete against each other. **This litepaper** outlines the staking, governance, and rewards mechanisms within the **Gluteus Maximus** ecosystem, with detailed gameplay to be revealed in a forthcoming whitepaper. The governance token, **$GLUTEU**, is currently live on three chains: **Base Network** (most liquid) at `0x06A63c498eF95AD1fA4FfF841955e512b4B2198a`, **Solana (Wormhole)** at `mMykZc4tun2kFNgxbd7WiApAoGEaX8bxFuHjxzjnkrV`, and **Ethereum Mainnet (Wormhole)** at `0x7A78c790250FEf60ce7E8Ef85557d67Cc4216A52`. Over **$1 million** in liquidity is fully burned or locked on DEX platforms such as **Uniswap** (Base, Ethereum Mainnet) and Raydium (Solana), and **$GLUTEU** is also listed on centralized exchanges like **MEXC.com, BITMART.com, TOOBIT.com**, and **WEEX.com**. The utility token **$LEGION**, which will represent the primary staking rewards and be essential for participating in the Crypto Wars, will be airdropped to Senator NFT holders. While **$LEGION** is not yet live, updates on its release can be found on **Gluteus Maximus**'s official social channels, including **X**, **Telegram**, and **Linktree**.

1.2. Audit Scope

This audit focused on identifying security flaws in code and the design of the GLUTEU. It was conducted on commit `13eda4a2e6ec76d5533b48c852d86857c7fc0d42` from git repository: https://github.com/theconsulgm/gluteus-maximus

The latest version of the following files were made available in the course of the review:

SHA256 Sum	File
dc70afc35ce7ab3236531d87aa14006ca444a6b892716ac661e52604f1a062fa	./SenatorNFTCollection.sol
5bb92f1737adb4a58887b83abf4d1fbceb333a40ac41e668d3880fc4a48018ae	./StakingGLUTEU.sol
6c69c210b5d053da7b42319f437bbc9bd8a115167ac70f7485694bb115464ded	./GluteusMaximusToken.sol

1.3. Audit Methodology

Our security audit process for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using public and RK87, our in-house smart contract security analysis tool.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that were considered during the audit of the smart contract:

- Integer Overflow and Underflow
- Timestamp Dependence
- Race Conditions
- Transaction-Ordering Dependence
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Usage, Gas Limit and Loops
- Redundant fallback function
- Unsafe type Inference
- Reentrancy
- Explicit visibility of functions state variables (external, internal, private and public)
- Logic Flaws

For vulnerabilities, we categorize the findings into categories as listed in table below, depending on their severity level:

SEVERITY LEVEL	DESCRIPTION
CRITICAL	A vulnerability that can disrupt the contract functioning; creates a critical risk to the contract; required to be fixed immediately.
HIGH	A vulnerability that could affect the desired outcome of executing the contract with high impact; needs to be fixed with high priority.
MEDIUM	A vulnerability that could affect the desired outcome of executing the contract with medium impact in a specific scenario; needs to be fixed.
LOW	An issue that does not have a significant impact, can be considered as less important.

Table 1. Severity levels

1.4. Disclaimer

Gluteus Maximus team acknowledges that the security services provided by Verichains, are conducted to the best of their professional abilities but cannot guarantee 100% coverage of all security vulnerabilities. Gluteus Maximus team understands and accepts that despite rigorous auditing, certain vulnerabilities may remain undetected. Therefore, Gluteus Maximus team agrees that Verichains shall not be held responsible or liable, and shall not be charged for any hacking incidents that occur due to security vulnerabilities not identified during the audit process.

1.5. Acceptance Minute

This final report served by Verichains to the Gluteus Maximus team will be considered an Acceptance Minute. Within 7 days, if no any further responses or reports is received from the Gluteus Maximus team, the final report will be considered fully accepted by the Gluteus Maximus team without the signature.

2. AUDIT RESULT

2.1. Overview

The GLUTEU was written in `Solidity` language, and built upon the **OpenZeppelin** library.

`GluteusMaximusToken` is an **ERC20 token** with a fixed supply of 1 billion tokens. This contract has a required version of `Solidity` to be `0.8.19`.

PROPERTY	VALUE
Name	Gluteus Maximus by Virtuals
Symbol	GLUTEU
Decimals	18
Supply	1,000,000,000 (x10^{18}) Note: the number of decimals is 18, so the total representation token will be 1,000,000,000 or 1 billion.

Table 2. Gluteus Maximus Token's properties

`SenatorNFTCollection` and `StakingGLUTEU` were written with a required version to be `0.8.17`. SenatorNFTCollection is an **ERC721-based NFT** collection with a maximum supply of `500` unique tokens. Key features of the contract include controlling Minting and Burning mechanisms with these functions: `mintWithId` and `burnWithId`. Only staking contracts can call those functions. The owner can set staking contracts.

`StakingGLUTEU`, enables users to stake **1,000,000 GLUTEU (1 million GLUTEU)** tokens to receive a randomly assigned Senator NFT from a limited pool of **500 unique IDs** (the maximum supply mentioned above). It employs **Chainlink VRF v2.5** for randomness and includes safeguards for edge cases like **VRF** failure or pool exhaustion.

- `stakeAndGetNFT` - **STAKE & REQUEST VRF** mechanism: The user stakes **1 million GLUTEU** and requests **VRF** for 1 random word. A new `StakeRequest` is created and stored in the stakes mapping under the `requestId`. The `requestId` is also recorded in the userRequests mapping, allowing the user to track their VRF requests.
- `fulfillRandomWords`: The **Chainlink VRF** callback provides a random word, which is stored in the corresponding `StakeRequest`. The `randomReady` is set to true, indicating that the randomness is ready for use. This mechanism is only for the internal actions.

- `claimNFT`: If there are available IDs in the pool, the user mints an NFT corresponding to the assigned random ID. If the pool is empty (no IDs left), the contract automatically refunds the user's **GLUTEU** and marks the stake as inactive.
- If the `randomReady` flag is false (indicating the randomness callback never occurred), the user can reclaim their **GLUTEU** tokens without receiving an NFT through the `unstakeNoNFT` mechanism.
- Burn NFT and Unstake (`unstakeAndBurnNFT`): After the `waitPeriod` has elapsed, the current owner of a minted NFT can burn it to reclaim the **GLUTEU tokens** associated with that NFT. The burned `tokenId` is returned to the pool of available IDs (availableIds) though `availableIds.push(tokenId);`.
- Only the owner can call the `setWaitPeriod` and `updateVRFSettings` mechanisms.

2.2. Findings

#	Title	Severity	Status
1	Users can unstake without locking their staking tokens.	MEDIUM	Fixed
2	Unsafe Transfer Token	INFORMATIVE	Fixed

2.2.1. MEDIUM - Users can unstake without locking their staking tokens.

Positions:

- `StakingGLUTEU.sol`#unstakeNoNFT()

Description:

According to the **Chainlink documentation**, re-requesting or canceling randomness in **VRF v2.5** is incorrect and poses security risks. However, the `unstakeNoNFT` function allows users to cancel their request after staking. Users can track the random `requestID` in the `fulfillRandomWords` index while it is pending on **Chainlink**. If the `requestID` does not match their own `ID`, they can use this function to cancel the randomness request and reclaim their staked amount before `fulfillRandomWords` state.

```
function unstakeNoNFT(uint256 requestId) external nonReentrant {
        StakeRequest storage sr = stakes[requestId];
        require(sr.staker == msg.sender, "Not your stake");
        require(sr.stakeActive, "Inactive or used");
        require(!sr.claimed, "Already claimed");
        require(!sr.randomReady, "Random assigned; can't forfeit now");

        sr.stakeActive = false;
        gluteuToken.safeTransfer(msg.sender, STAKE_AMOUNT);
        emit UnstakeNoNFT(msg.sender, requestId);
    }
```

RECOMMENDATION

User staking amounts should be locked for a period to ensure `fulfillRandomWords` completes successfully or fails.

UPDATES

- *Feb 10, 2025*, The Gluteus Maximus team has implemented required statement that locks the amount for 1 hour.

2.2.2. INFORMATIVE - Unsafe Transfer Token

Positions:

- `StakingGLUTEU.sol`#L196
- `StakingGLUTEU.sol`#L277
- `StakingGLUTEU.sol`#L321
- `StakingGLUTEU.sol`#L344

Description:

The contract uses the `transfer` and `transferfrom` functions to transfer in/out tokens. Although there are required statements, `SafeERC20` is recommended to be used to handle more cases, as convenience is already ensured.

RECOMMENDATION

Use the `SafeERC20` library to handle token transfers.

UPDATES

- *Feb 6, 2025*, The Gluteus Maximus team has implemented our recommendations.

3. VERSION HISTORY

Version	Date	Status/Change	Created by
1.0	*Jan 24, 2025*	Public Report	Verichains Lab
1.1	*Feb 6, 2025*	Public Report	Verichains Lab
1.2	*Feb 7, 2025*	Public Report	Verichains Lab
1.3	*Feb 10, 2025*	Public Report	Verichains Lab

Table 3. Report versions history