e

CD SECURITY

AUDIT REPORT

Shogun

November 2024

Prepared by
ladboy
Pelz
0xluk3

Introduction

A time-boxed security review of the **Shogun** protocol was done by **CD Security**, with a focus on the security aspects of the application's implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs, and on-chain monitoring are strongly recommended.

About **Shogun**

SHOGUN is a DeFi token ecosystem that features buy-and-burn mechanisms, tax-based liquidity funding, and decentralized perpetual auctions. Every day over the first 28 days, 100M SHOGUN tokens are given away at auction. Perpetual auctions distribute 8% of the war chest every day after launch. The proceeds of transactions are split between feeding the war chest, burning, and liquidity after an 8% tax. To guarantee fair trading, the ecosystem makes use of MEV-resistant techniques and Uniswap V2 liquidity pools. During the first stage, Proof of Burn offers incentives to builders. SHOGUN is an Ethereum-based company that prioritizes community-driven growth and sustainable tokenomics.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic, and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - 9aca6c8e888c3c923d3c7dbac6b7181ab7d15e05**

***review commit hash* - 01ef756d374387ea3bd25ce2fdc0f8cbe62046cd**

Scope

The following smart contracts were in scope of the audit:

- `contracts/*`

The following number of issues were found, categorized by their severity:

- Critical & High: 0 issues
- Medium: 0 issues
- Low & Info: 6 issues

Findings Summary

ID	Title	Severity	Status
[L-01]	Missing Balance Check in `createInitialLiquidity` Function	Low	Fixed
[L-02]	Missing Sanity Check for Zero `amount` in `enterAuctionX28Liquid`	Low	Fixed
[L-03]	The use of msg.sender == tx.origin invariant may be broken in future	Low	Acknowledged
[L-04]	Lack of safeMath usage with solidity version below 0.8	Low	Acknowledged
[I-01]	Potential Risk with `renounceOwnership` Function	Informational	Acknowledged
[I-02]	Multiple BuyAndBurn contracts remain registered after contract address update	Informational	Acknowledged

[L-01] Missing Balance Check in `createInitialLiquidity` Function

Severity

Impact: Low
Likelihood: Low

Description

The `createInitialLiquidity` function in `BuyAndBurnShogun.sol` allows anyone to initialize liquidity. However, the function does not validate whether the contract has sufficient `X28` tokens to meet the liquidity requirements for proper error handling. If the balance is insufficient, the transaction will fail when calling `_mintPosition`.

Vulnerable Code

```
function createInitialLiquidity() public {
    require(s_poolAddress != address(0), "NoPoolExists");
    require(!s_initialLiquidityCreated, "LPHasCreated");

    s_initialLiquidityCreated = true;
    TransferHelper.safeApprove(s_shogunAddress, address(this),
type(uint256).max);
    TransferHelper.safeApprove(X28, address(this), type(uint256).max);

    IShogun(s_shogunAddress).mintLPTokens();
    _mintPosition();
}
```

Recommendations

Add a check to ensure that the contract's balance of `X28` tokens is at least the required amount (`INITIAL_LP_X28`) before proceeding:

```
require(IERC20(X28).balanceOf(address(this)) >= INITIAL_LP_X28,
"Insufficient X28 balance");
```

Suggested Fix

```
function createInitialLiquidity() public {
    require(s_poolAddress != address(0), "NoPoolExists");
    require(!s_initialLiquidityCreated, "LPHasCreated");
    require(IERC20(X28).balanceOf(address(this)) >= INITIAL_LP_X28,
"Insufficient X28 balance");

    s_initialLiquidityCreated = true;

    IShogun(s_shogunAddress).mintLPTokens();
    _mintPosition();
}
```

[L-02] Missing Sanity Check for Zero `amount` in `enterAuctionX28Liquid`

Description

The `enterAuctionX28Liquid` function currently lacks a sanity check to validate that the `amount` parameter passed by the caller is greater than zero.

If a user inadvertently or maliciously passes `amount = 0` the `AuctionEntered` event will be emitted with a `0` amount, leading to misleading or excessive data in the frontend. The affected code:

```
function enterAuctionX28Liquid(
    uint256 amount
) external dailyUpdate(s_buyAndBurnAddress) nonReentrant {
    // Transfer burn amount to X28 BNB, call public burnCAX28() to burn
X28
    uint256 burnAmount = (amount * BURN_PERCENT) / PERCENT_BPS;
    IX28(X28).transferFrom(msg.sender, X28_BNB, burnAmount);
    IX28(X28).burnCAX28(X28_BNB);

    // Additional transfers and updates
    ...
    emit AuctionEntered(msg.sender, getCurrentContractDay(), amount);
}
```

Without a check for `amount > 0`, a zero value can result in events being emitted a the fronend over populated with invalid data.

Recommendations

Ensure that the `amount` parameter is greater than zero before proceeding with the function logic:

```
require(amount > 0, "Amount must be greater than 0");
```

[L-03] The use of msg.sender == tx.origin invariant may be broken in future

Description

The `buyX28` and `buynBurn` functions in the **BuyAndBurnShogun.sol** contract enforce a restriction using `msg.sender == tx.origin` to prevent interactions from other smart contracts:

```
require(msg.sender == tx.origin, "InvalidCaller");
```

This check is currently valid because **EOAs** (Externally Owned Accounts) cannot contain contract code. However, with the introduction of **EIP-7702**, EOAs may hold contract code, enabling contracts to mimic EOAs (`msg.sender == tx.origin`).

The proposed change in Ethereum would:

1. Allow contracts to batch transactions on behalf of an EOA.

2. Break the assumption that `msg.sender == tx.origin` guarantees the transaction is initiated by an EOA.

You can read ore about the eip here

Recommendation

Remove the check

[L-04] Lack of safeMath usage with solidity version below 0.8

Description

The BuyAndBurn contract uses Solidity 0.7.6 which doesn't include automatic overflow/underflow protection, but fails to use SafeMath library:

```
pragma solidity ^0.7.6; contract BuyAndBurnShogun is ReentrancyGuard {
```

Without SafeMath, arithmetic operations can silently overflow/underflow.

However it was not proven that present functions could easily overflow, except for view functions if bogus values are passed to them as arguments. This could affect potential external integrations, but not the protocol itself.

Recommendations

Import and use SafeMath for calculations in solidity below 0.8

[I-01] Potential Risk with `renounceOwnership` Function

Description

The `renounceOwnership` function in the `BuyAndBurnShogun.sol` contract allows the current owner to renounce ownership, setting the `s_ownerAddress` to the zero address. While this feature may be intentional, its inclusion poses a potential risk to the functionality of the contract, as the `BuyAndBurnShogun` contract relies on the owner for various configurations and settings. Removing ownership could render certain functions inoperable, potentially disrupting critical operations.

Vulnerable Code

```
function renounceOwnership() public {
    require(msg.sender == s_ownerAddress, "InvalidCaller");
```

```
        s_ownerAddress = address(0);
    }
```

Recommendations

Given the importance of the owner in managing the contract, the ability to renounce ownership should be removed to prevent accidental or malicious use.

[I-02] Multiple BuyAndBurn contracts remain registered after contract address update

Description

The Shogun contract never removes old BuyAndBurn contracts from the registry when setting a new one:

```
function setBuyAndBurnContractAddress(address contractAddress) external
onlyOwner {
   s_buyAndBurnAddress = contractAddress;
   s_buyAndBurnAddressRegistry[contractAddress] = true;
   s_taxExclusionList[contractAddress] = true;
}
```

While this doesn't pose security risks since auction state is managed by the Shogun contract, it creates potential confusion for integrators about which contract is currently active. Multiple registered contracts maintain the ability to call functions like burnCAShogun() and dailyUpdate(), though these operations remain safe due to state management controls.

Recommendations

Revoke old bnb contracts when setting a new one.