Smithii Audit

May 2024

By CoinFabrik

v202311

Executive Summary

CoinFabrik was asked to audit the contracts for the Smitii project.

During this audit we found one critical issue and two high severity issues.

Scope

The audited files are from the git repository located at https://github.com/SmithiiDev/smithii_evm_token_creator_contract. The audit is based on the commit 4c75ee3412e5def669c2f13ae74106d71fc5a8eb. Fixes were checked on commit 2d23dcc343e8c563a4c6f0cc7f56a18153dc2669. All issues were resolved.

The scope for this audit includes and is limited to the following files:

- `ERC20TokenFactory.sol`: Contracts deployers.
- `tokens/ERC20/ERC20Template.sol`: Enhanced ERC20 contract template.
- `ERC20PoolFactory.sol`: Manages liquidity pool as ERC721 token.
- `ERC20AntiWhale.sol`: Protection against whale-like actors.
- `utils/Shallowed.sol`: AntiWhale modifier.
- `ERC20AntiBot.sol`: Protection against bot-like actors.
- `utils/Secured.sol`: AntiBot modifier.
- `ERC20TokenMultiSender.sol`: ASM-optimized multisender.
- `marketplace/Payments.sol`: Register payments for the service.
- `marketplace/Indexer.sol`: Indexer of projects and contracts.
- `utils/Payable.sol`: Payments glue logic.
- `utils/Indexable.sol`: Indexer glue logic.

No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

Methodology

CoinFabrik was provided with the source code, including automated tests that define the expected behavior, and general documentation about the project. Our auditors spent two weeks auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function, understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. Without being limited to them, the audit process included the following analyses.

- Arithmetic errors
- Outdated version of Solidity compiler
- Race conditions
- Reentrancy attacks
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

ID	Title	Severity	Status
CR-01	RegisterBlock() Frontrunning Blocks All Updates	Critical	Resolved
HI-01	Bot Protection Blocks Exempted Contracts	High	Resolved
HI-02	AntiWhale Protection Time Limit Block	High	Resolved

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. Blocking bugs are also included in this category. They must be fixed **immediately**.

- **High:** These refer to a vulnerability that, if exploited, could have a substantial impact, but requires a more extensive setup or effort compared to critical issues. These pose a significant risk and **demand immediate attention**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of, but might be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit has one of the following statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision. The reported risk is accepted by the development team.
- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.

Critical Severity Issues

CR-01 RegisterBlock() Frontrunning Blocks All Updates

Location:

- `contracts/ERC20AntiBot.sol:41`
- `contracts/ERC20AntiWhale.sol:48`

Classification:

- CWE-284: Improper Access Control[1]

The AntiBot mechanism works by registering the block number using `ERC20AntiBot.registerBlock()` so there can be only one call to *update() for each block*. *We can see how the function* `ERC20Template._update()` do the `noBots()` check and then register the block:

[1]https://cwe.mitre.org/data/definitions/284

```
function _update(
        address sender,
        address recipient,
        uint256 amount
    ) internal virtual override whenNotPaused noBots(sender) noWhales(recipient, amount) {
        registerBlock(recipient);
        registerBlockTimeStamp(sender);
…
```

However, as the `registerBlock()` function is public, anybody can call this function with any token and any address, and prevent any update as every sender would be detected as a bot. A malicious attacker can front-end all `_update()` calls and prepend it with a `registerBlock()` call that blocks it.

Recommendation

Make the `ERC20AntiBot.registerBlock()` function private.

Status

Resolved. The `registerBlock()` function is now external, but it associates the block with the msg.sender address, so a malicious attacker can no longer register blocks for any addresses.

High Severity Issues

HI-01 Bot Protection Blocks Exempted Contracts

Location:

- `contracts/ERC20AntiBot.sol:32`

Classification:

- CWE-862: Missing Authorization[2]

While the AntiBot contract has a mechanism to register exemptions (`setExempt()` and `isExempt()`) the exempt mapping is never used in the `isBotDetected()` function, meaning that an exempted contract will still be treated like a regular contract. Note that this bug is not present at the AntiWhale protection as it correctly applies exemptions.

Recommendation

Add the exemption checking in the `isBotDetected()` function.

[2]https://cwe.mitre.org/data/definitions/862.html

Status

Resolved. Added exemption checking at `isBotDetected()` function.

HI-02 AntiWhale Protection Time Limit Block

Location:

- `contracts/ERC20AntiWhale.sol:48`

Classification:

- CWE-284: Improper Access Control[3]

The AntiWhale mechanism has several checks that detect big movements that manipulate the asset values. The last of those checks is time-based, as we can see in the `ERC20AntiWhale.isWhaleDetected()` function:

```
   function isWhaleDetected(address _token, address _to, uint256 _amount) public view
returns(bool) {
…
      if (isActive(_token)) {
           if(_amount > canUseAntiWhale[_token].maxAmountPerTrade) isWhale = true;
           if(_amount + IERC20(_token).balanceOf(_to) >
canUseAntiWhale[_token].maxAmountTotal) isWhale = true;
           if(block.timestamp < buyBlock[_to] + canUseAntiWhale[_token].timeLimitPerTrade)
isWhale = true;
       }
       return isWhale;
```

And in a similar way as CR-01, the `ERC20AntiWhale.registerBlockTimeStamp()` function is public, so anybody can modify the `buyBlock[_to]` value. A malicious attacker can front-end a call to `isWhaleDetected()` and update the `buyBlock[]` mapping so the time check always returns true.

Recommendation

Make the ERC20AntiWhale.registerBlockTimeStamp() function private.

Status

Resolved. While the `registerBlockTimeStamp()` is still external, it now can only add a timestamp to the `msg.sender` address mapping, so a malicious attacker can no longer register timestamps for all addresses.

Medium Severity Issues

No issues found.

[3]https://cwe.mitre.org/data/definitions/284

Minor Severity Issues

No issues found.

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest to other stakeholders of the project, including users of the audited contracts, token holders or project investors.

Centralization

The indexer, Anti-bot and anti-whale protections need an authorized operator.

The owner of the payments contract can withdraw all funds.

The contract uses Uniswap V3 to implement the liquidity pools.

Upgrades

No upgrade mechanism was implemented.

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

ERC20AntiBot.sol

Operator

The Operator can set if a token can use AntiBot protection without paying the fee.

Owner

The Owner role can set if a token can use AntiBot protection, but it must pay the fee.
Also this role can set exemptions on the contracts anti-bot checks.

ERC20AntiWhale.sol

Operator

The Operator can set if a token can use AntiWhale protection without paying the fee.

Owner
The Owner role can set if a token can use AntiWhale protection, but it must pay the fee. Also this role can set exemptions on the contracts, and set trading limits and time limits per trade.

ERC20TokenFactory.sol

Operator
This role can deploy contracts

ERC20TokenMultiSender.sol

Operator
This role can airdrop tokens.

marketplace/Payments.sol

Operator
This role can add a service and set a service active.

Admin
This role can withdraw funds from a project as payments for a service

Owner
This role can withdraw funds from all projects and all services

marketplace/Indexer.sol

Operator
This role can register a project.

tokens/ERC20/ERC20Template.sol

Owner
This role can add addresses to blacklist, set an address as non-taxable, set the tax deposit address and stop airdrop mode.

Changelog

- 2024-5-31 – Initial report based on commit 4c75ee3412e5def669c2f13ae74106d71fc5a8eb
- 2024-08-14 - Final report based on commit 2d23dcc343e8c563a4c6f0cc7f56a18153dc2669

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the Smithii project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.