Kamari Token Audit

June 2021

by Adam Zigdon

Table of Contents

Introduction

This is an audit for the token protocol created by Kamari.
In this document, I will provide a summary of my findings together with proposed solutions.

Summary

The contracts audited are from the Kamari repository.
The audit is based on the commit `ca63df94ed0f99123259a9d24989a9d88e5c704c` from June 1st, 2021.

Audited Contracts

- `token/Kampay.sol` - represents the Kampay token.

Analyses Performed

Here are some of the vulnerabilities I checked during the process:

- Balance equality.
- Code and contract interaction complexity.
- Complex code and contract interactions.
- Failure to use a withdrawal pattern.
- Fraudulent or erroneous code.
- Functions with excessive gas cost.
- Incorrect handling of cryptographic signatures.
- Insufficient validation of the input parameters
- Integer rounding errors, overflow, underflow and related usage of SafeMath functions.
- Malicious libraries.
- Missing or misused function qualifiers.
- Misuse of block timestamps.
- Misuse of the different call methods: call.value(), send() and transfer().
- Outdated Solidity compiler.
- Race conditions such as reentrancy attacks or front running.
- Softlock DoS attacks.
- Wrong or missing error handling.

Severity Classification

- **Critical**: Issues that put the system at serious risk. Must be fixed **immediately**.
- **Medium:** These are issues that might put the system at risk if misused. Should be fixed **as soon as possible**.
- **Minor:** Issues that represent smaller issues that shouldn’t stop deployment. Should be taken into account and be fixed **when possible**.
- **Enhancements/Fixes:** Any findings that don’t represent a current or future security risk. Can be either changed or ignored.

Manual Analysis - Issues by Severity

Critical

No critical issues were found.

Medium

No medium issues were found.

Minor

No minor issues were found.

Enhancements/Fixes

1. Tokens allocation

Affected contracts: `Kampay.sol`

All tokens are being sent to a single address (the deployer) and the distribution plan is being documented in a comment just above the constructor:

```
// public sale 2% - 20M
// private sale 20% - 200M
// Advisors 10% - 100M
// Team 20% - 200M
// Marketing and production partners 15% - 150M
// Ecosystem 20% - 200M
// treasury 13% - 130M

constructor() ERC20("Kampay","KAMPAY") {
    _mint(msg.sender, 1000000000 * 10 ** 18);
}
```

It requires the distribution to be done later in the process, and can be subject to human errors which are not covered by tests.
If such an error occurs, it will require redeployment of the token causing Kamari to spend redundant BNB on gas fees and have two or more different tokens with the same name which can confuse token holders and traders.
In addition, when the distribution is done manually, it's harder to trace what role each address has and it can lead to trust issues of the community.

Proposed resolution:

- Assign addresses prior to deploying the tokens and distribute tokens at the constructor.
  OR
- Make the distribution process safer, either by automating it or by doing it very carefully (and supervised), while making the information about the different addresses publicly available elsewhere (e.g. website).

Automated Tools

Mythril

Mythril didn't find any issues in the contracts.

Slither

Severity

Slither have three different severity levels:

- High
- Medium
- Low

Issues

`Kampay.sol`

all issues found by this tool were of low severity and weren't originated in the current contract.

Conclusion

The Kamari token protocol contract is pretty straight forward.
It uses the well documented and audited OpenZeppelin's ERC20 contract and doesn't introduce any new dangerous code.
I found an issue which might compromise the trust of the community when misused.
I recommend either fixing it prior to deployment or handling it very carefully if fixing it is not applicable at this time (e.g. missing addresses).

Disclaimer

This audit report is not a security warranty, investment advice, or an approval of the Kamari/Kampay project.
It also doesn't guarantee the smart contract code to be faultless.