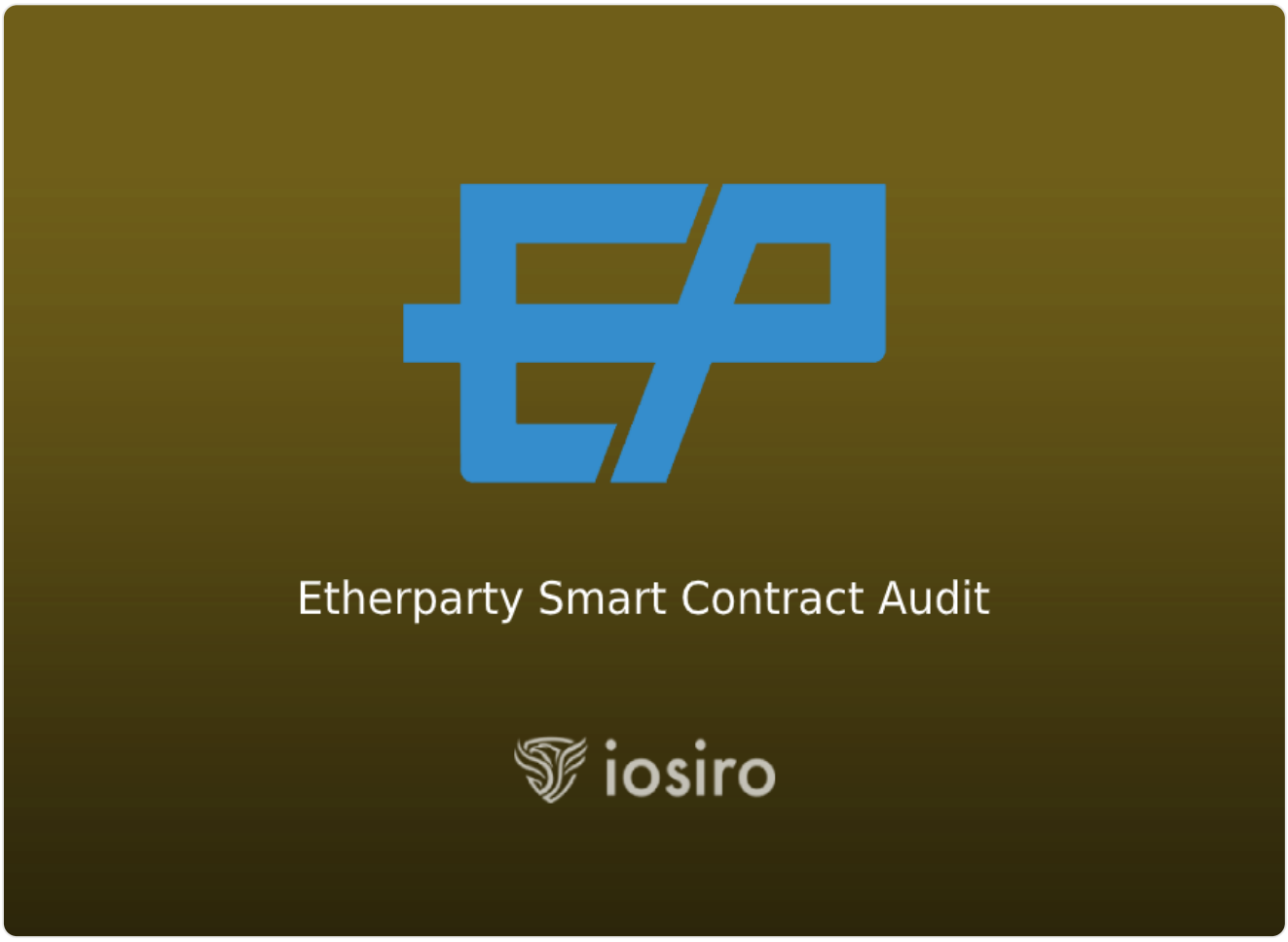
Etherparty Smart Contract Audit



1. Introduction

iosiro was commissioned by Etherparty to conduct an audit on their token and crowdsale platform smart contracts. The audit was performed between 6 September 2018 and 7 September 2018. An audit review was performed on 11 September 2018 to verify that changes in the original audited had been correctly addressed.

This report is organized into the following sections.

- **Section 2 - Executive Summary:** A high-level description of the findings of the audit.
- **Section 3 - Audit Details:** A description of the scope and methodology of the audit.
- **Section 4 - Design Specification:** An outline of the intended functionality of the smart contracts.

- **Section 5 - Detailed Findings:** Detailed descriptions of the findings of the audit.

The information in this report should be used to understand the risk exposure of the smart contracts, and as a guide to improve the security posture of the smart contracts by remediating the issues that were identified. The results of this audit are only a reflection of the source code reviewed at the time of the audit and of the source code that was determined to be in-scope.

---

2. Executive Summary

This report presents the findings of an audit performed by iosiro on the Etherparty platform token and crowdsale smart contracts. The purpose of the audit was to achieve the following.

- Ensure that the smart contracts functioned as intended.
- Identify potential security flaws.

Due to the unregulated nature and ease of transfer of cryptocurrencies, operations that store or interact with these assets are considered very high risk with regards to cyber attacks. As such, the highest level of security should be observed when interacting with these assets. This requires a forward-thinking approach, which takes into account the new and experimental nature of blockchain technologies. There are a number of techniques that can help to achieve this, some of which are described below.

- Security should be integrated into the development lifecycle.
- Defensive programming should be employed to account for unforeseen circumstances.
- Current best practices should be followed wherever possible.

At the conclusion of the audit, only informational findings and design recommendations remained open. These included minor deviations from best-practice, unnecessary variable declaration, and instances where code could be simplified.

Despite the findings, the code was of a high standard. This was evident in the use of separated token and crowdsale logic, as well as making use of commonly used libraries, where reasonable. A comprehensive test suite was also provided by the development team.

The risk posed by the smart contracts can be further mitigated by using the following controls prior to releasing the contracts to a production environment.

- Use a public bug bounty program to identify security vulnerabilities.
- Perform additional audits using different teams.

3. Audit Details

3.1 Scope

The source code considered in-scope for the assessment is described below. Code from any other files are considered to be out-of-scope.

3.1.1 Etherparty Smart Contracts

Project Name: eth-smart-contracts
Commit: 39ab46c

3.2 Methodology

A variety of techniques were used to perform the audit, these are outlined below.

3.2.1 Dynamic Analysis

The contracts were compiled, deployed, and tested using both Truffle tests and manually on a local test network. A number of pre-existing tests were included in the project.

3.2.2 Automated Analysis

Tools were used to automatically detect the presence of potential vulnerabilities, such as reentrancy, timestamp dependency bugs, transaction-ordering dependency bugs, and so on. Static analysis was conducted using Mythril and Oyente. Additional tools, such as the Remix IDE, compilation output and linters were used to identify potential security flaws.

3.2.3 Code Review

Source code was manually reviewed to identify potential security flaws. This type of analysis is useful for detecting business logic flaws and edge-cases that may not be detected through dynamic or static analysis.

3.3 Risk Ratings

Each Issue identified during the audit is assigned a risk rating. The rating is dependent on the criteria outlined below..

- **High Risk** - The issue could result in a loss of funds for the contract owner or users.
- **Medium Risk** - The issue results in the code specification operating incorrectly.
- **Low Risk** - A best practice or design issue that could affect the security standard of the contract.
- **Informational** - The issue addresses a lapse in best practice or a suboptimal design pattern that has a minimal risk of affecting the security of the contract.
- **Closed** - The issue was identified during the audit and has since been addressed to a satisfactory level to remove the risk that it posed.

---

4. Design Specification

The following section outlines the intended functionality of the smart contracts.

4.1 Etherparty Platform Token Template

The Etherparty platform token template is described below.

ERC20 Token

The token should implement the ERC20 standard.

Field	Value
Symbol	SYMBOL
Name	NAME
Decimals	18
Total Supply	Set during initialisation

Lock

Token transfers should be locked until the end date of the crowdsale or the crowdsale fund allocation is depleted.

Time Lock

Pre-sale token allocations may be specified with a time lock period. Tokens should only be withdrawn after their associated time lock period has expired. All token transfers should remain locked until the end of the crowdsale.

4.2 Etherparty Platform Crowdsale Template

The Etherparty platform crowdsale template is described below.

Pricing Rounds

The crowdsale may be initialised with an array specifying multiple pricing rounds, setting the token purchase price depending on the current time.

Token Allocations

Pre-sale tokens are allocated dynamically to addresses on contract initialisation via the crowdsale constructor with optional time lock functionality. Additionally, a crowdfund address may also be provided to allocate tokens to be used in the crowdsale. After this initial allocation, the total supply is fixed.

Token Forwarding / Burnable Functionality

Upon closing of the crowdsale, remaining unsold tokens within the crowdfund may be optionally forwarded to a specified address or burned upon closing of the crowdsale.

Rescheduling

The start time of the crowdsale may be rescheduled by the crowdsale contract owner. However, it may only be rescheduled a minimum of four hours before the previously scheduled start time.

Whitelist

The crowdsale may be initialised with whitelist functionality. If enabled, participants need to be whitelisted before being able to contribute funds to the crowdsale.

---

5. Detailed Findings

The following section includes in depth descriptions of the findings of the audit.

5.1 High Risk

No high risk issues were present at the conclusion of the audit.

5.2 Medium Risk

No medium risk issues were present at the conclusion of the audit.

5.3 Low Risk

5.4 Informational

5.4.1 Design Comments

General

The following describes possible actions to improve the functionality and readability of the codebase.

Unnecessary declaration of the return variable name `balance`

BasicToken.sol: line 47

It was found that the `balanceOf(...)` function explicitly defined the return variable name `balance`. Although not strictly a security vulnerability or functional issue, the definition of the return variable name was unnecessary due to the fact that it was never used in the function code body. Instead, the `balances[_owner]` result was returned directly thus rendering the return variable name irrelevant.

Use of `assert` instead of `require` for transfer validation

SafeERC20.sol: lines 15, 19, 23

It was found that the `assert` operator was used to validate transfers, it is recommended that the `require` operator be used instead. Typically, `assert` should only be used for invariant checking and not conditional checking since it burns all gas upon failure.

Use of implicit integer sizes in StandardToken implementation

StandardToken.sol: lines 73, 89, 90

It was found that the `increaseApproval(...)` and `decreaseApproval(...)` functions contained implicit `uint` type declarations. It is suggested that these `uint` type declarations be replaced by the `uint256` type declaration instead.

Redundant `if` statements can be replaced with `require`

Crowdfund.sol: lines 157, 176, 212, 227, 232, 253

It was found that in several instances, the design pattern using an `if` statement for validation followed by a `revert()` if validation failed was used. It is suggested that this design pattern be replaced by the more appropriate `require` pattern. An example (from line 157) is shown below.

```
if (!token.changeCrowdfundStartTime(startsAt)) {
    revert();
}
```

Can simply be expressed as:

```
require(token.changeCrowdfundStartTime(startsAt));
```

Please note that the conditional statement in the `require` is an inversion of the conditional statement of the equivalent `if` statement.

Timestamp

The `now` keyword (an alias for `block.timestamp`) is used to determine the current time. This value can be marginally affected by miners (by up to 900 seconds), so a common best practice is to rather use `block.number` to determine the time. However, the risk posed in this circumstance is inconsequential, and it is simply listed for completeness. No action is necessary.

5.5 Closed

5.5.1 Uncessary Self-Destruct Functionality (Low Risk)

Crowdfund.sol: Line 266

Description

The crowdsale contract was found to have `selfdestruct` functionality. This results in a single point of failure and unnecessarily extends the attack surface of the contract. The risk would be that if the crowdsale is self-destructed during the crowdsale, unsuspecting users could still send ether to the crowdsale contract resulting in lost funds.

Since the fallback function contains all the necessary fault tolerance, it would be unlikely that a large sum of ether would accidentally be locked in the contract, requiring self-destruct functionality to withdraw funds. Additionally, if it was deemed necessary to stop token transfers or cancel the crowdsale, the owner would simply withhold calling the `closeCrowdfund()` function.

As a result, the risk presented by the functionality outweighs any potential benefit that it may offer.

Remedial Action

It is recommended that the `kill()` function is removed entirely. Alternatively, it would be possible to rework the function to halt the crowdsale, but also withdraw ether from the contract in a similar fashion to self-destruct.

Implemented Action

The `kill()` function was removed in f8153a4 as per the recommendation.