CryptoTestament

Smart Contract Audit

coinspect

CryptoTestament

Smart Contract Audit

V220314

Prepared for IOV • March 2022

1. Executive Summary

In **February 2022, liberdapps** engaged Coinspect to perform a source code review of **CryptoTestament**. The objective of the project was to evaluate the security of the smart contracts.

The following issues were identified during the assessment:

High Risk	Medium Risk	Low Risk
1	0	0
Fixed	Fixed	Fixed
1	0	0

The high-risk vulnerability TES-1 is caused by lack of restrictions on the callers to the `receive` function in `CryptoTestament` contract, allowing any address to perform a griefing attack against the beneficiary and prevent the beneficiary from getting the funds after the testator has died.

Update: As of commit `460835894f93c9f18c74fe498750641fe626f08d` of **March 9, 2022** the TES-1 issue has been fixed.

2. Assessment

The audit started on **February 8, 2022** and was conducted on the repository at https://github.com/liberdapps/CryptoTestament as of commit `0b7ce4cc37bfe11047581a46492b6aaa3f13894f` of **February 2, 2020** tagged as `v1.0.0`.

The scope of the assessment was limited to the contracts `CryptoTestament` and `CryptoTestamentService` in file `contracts/CryptoTestament.sol` with sha256sum `3cf297c74268be24e2dacc0538357245a84dd9b13288291e7fa27cee796bfd96`.

The `CryptoTestamentService` contract (prior to Coinspect's audit) was deployed in the RSK network at 0x9f386392833fa09b9064cc49f0acbb20d4d1937b and the dapp is available at https://cryptotestament.io.

The system allows users (*testators*) to deposit RBTC in a `CryptoTestament` contract with a designated *beneficiary*. The testator can deposit or withdraw funds at any time. The testator must show proof of life periodically by calling a function of the testament contract. If a specified amount of time passes since the testator last gave proof of life, the testator is assumed dead and the testament contract can be executed, resulting in the funds being transferred to the beneficiary.

The code is very clear and well written. The contracts are specified to be compiled with Solidity compiler >= 0.8.7. The two contracts are self-contained and don't depend on any third-party code.

The repository doesn't include any tests. In general it is advisable to develop tests together with the contracts and ensure tests have full coverage of the contracts code.

The `CryptoTestament` contract has a number of storage variables that are set in the constructor and never changed, and it is recommended to make those variables *`immutable`*. These variables include: `creationTimestamp`, `testatorAddress`, `serviceAddress` and `serviceFeeRate`. Also in the `CryptoTestamentService` contract the storage variable `serviceOwner` can be marked *`immutable`*.

Both `CryptoTestament` and `CryptoTestamentService` contracts contain many `require` statements without a reason string. In general it is recommended to always put a reason string in `require` statements, to make it easier to test and debug the contracts or any services interacting with them.

The `CryptoTestamanent` contract implements the `receive` function. The testator can call this function to deposit funds in the testament contract. Receiving funds is considered "proof of life", i.e. when the `receive` function is called the `lastProofOfLifeTimestamp` variable is set to `block.timestamp`:

```
    receive() external payable {
        // Only allow deposits if testament is still locked.
        require (status == TestamentStatus.LOCKED);
        require (block.timestamp - lastProofOfLifeTimestamp <= proofOfLifeThreshold);

        // Calculate and pay service fees.
        uint256 serviceFee = (msg.value * serviceFeeRate) / 10000;
        if (serviceFee > 0) {
            (bool sent, ) = serviceAddress.call{value: serviceFee}("");
            require (sent, "Send failed.");
        }

        // Update proof of life.
        lastProofOfLifeTimestamp = block.timestamp;
    }
```

However, notice that the `receive` function can be called by any address, not just by the testator address. This means that anyone can call the function to force an update of the `lastProofOfLifeTimestamp` variable and prevent the beneficiary from getting the funds after the testator has died. It is recommended to allow only the testator to call the `receive` function (see TES-1).

3. Update

Fixes were verified as of commit `460835894f93c9f18c74fe498750641fe626f08d` of **March 9, 2022.** The final revision reviewed by Coinspect to verify fixes after the audit contains the following Solidity source files with their respective sha256sum hashes:

```
81f3c877b52be60c057e38eb52d105427dec923490d4532e309121f23efa4e0b  CryptoTestament.sol
4fd6092bdfa8b42f19d535c5ac69c4323b0b894717c699e58d5552eeabd04cd4  Migrations.sol
```

The high-risk issue TES-1 was fixed by allowing only the testator address to deposit funds by calling the function `receive`. The update also includes other improvements following Coinspect's suggestions, such as the use of the *`immutable`* keyword for storage variables that don't change during the contract's lifetime, and the inclusion of message strings in all revert statements.

4. Summary of Findings

Id	Title	Total Risk	Fixed
TES-1	Griefing attack against testament execution	High	✔

5. Detailed Findings

TES-1 Griefing attack against testament execution

Total Risk	Impact	Location
High	High	`CryptoTestament.sol`
Fixed	Likelihood	
✔	High	

Description

The `CryptoTestamanent` contract implements the `receive` function. The testator can call this function to deposit funds in the testament contract. Receiving funds is considered "proof of life", i.e. when the `receive` function is called the `lastProofOfLifeTimestamp` variable is set to `block.timestamp`:

```
receive() external payable {
    // Only allow deposits if testament is still locked.
    require (status == TestamentStatus.LOCKED);
    require (block.timestamp - lastProofOfLifeTimestamp <= proofOfLifeThreshold);

    // Calculate and pay service fees.
    uint256 serviceFee = (msg.value * serviceFeeRate) / 10000;
    if (serviceFee > 0) {
        (bool sent, ) = serviceAddress.call{value: serviceFee}("");
        require (sent, "Send failed.");
    }

    // Update proof of life.
    lastProofOfLifeTimestamp = block.timestamp;
}
```

However, notice that the `receive` function can be called by any address, not just by the testator address. This means that anyone can call the function to force an update of the `lastProofOfLifeTimestamp` variable and prevent the testament to be executed and the beneficiary from getting the funds after the testator has died.

Recommendation

It is recommended to only allow the testator address to call the `receive` function in `CryptoTestamanet` contract.

6. Disclaimer

The information presented in this document is provided "as is" and without warranty. The present security audit does not cover any off-chain systems or frontends that communicate with the contracts, nor the general operational security of the organization that developed the code.