Security Audit of FuzeX Smart Contract

This report is public.

ChainSecurity Ltd.

January 11, 2018

Contents

Token Name	FUZEX TOKEN (FXT)
Decimals	18
Smallest Unit (Atom)	10^{-18} FXT
Token Price	1 ETH = 6000 FXTs
Maximum tokens sold	480,000,000
Maximum tokens issued (including rewards and tokens issued to the FuzeX team)	980,000,000
Percentage of tokens for sale to the public	60%
Private-sale cap	20,000 ETH
Private-sale minimum contribution	500 ETH
Pre-sale cap	26,667 ETH
Pre-sale minimum contribution	20 ETH
Main-sale cap	33,333 ETH
Main-sale minimum contribution	0.1 ETH
Total cap	80,000 ETH
Minimum cap	10,000 ETH
Rewards	Yes (see Section 1.2)
Refund	Yes (see Section 1.4)

Table 1: Facts about the FXT token and the token sale.

We first and foremost thank FUZEX for giving us the opportunity to audit their smart contracts. This documents outlines our methodology, limitations, and results.

1 System Overview

The FuzeX team aims to launch a secure and easy-to-use e-card for cryptocurrency, credit, debit, and reward payments. The FuzeX card can store up to 15 cryptocurrency accounts, including 10 credit/debit cards and 5 rewards cards, and would feature real-time exchange rates that would enable users to spend their cryptocurrency and rewards anytime and anywhere. For security purposes, the FuzeX card would feature two-factor authentication (a PIN for the FuzeX card as well as a password to secure the mobile application) for private key management and a built-in loss prevention features. An E-Paper display would make the FuzeX card easy to use, allowing users to select their desired method of payment and see their balance.

In the following we describe the FUZEX TOKEN (FXT) and its corresponding token sale. Table 1 gives the general overview.

1.1 TGE Overview

FXTs will be sold at the price of 6000 FXT for 1 ETH. The total number of tokens that will be available for sale is 480,000,000. The tokens will be sold in three phases:

- Private sale: contribution cap is set to 20,000 ETH with 120,000,000 FXTs maximum sold,
- Pre-sale: contribution cap is set to 26,667 ETH with ≈ 160,000,000 FXTs maximum sold, and
- Main-sale: contribution cap is set to 33,333 ETH with 200,000,000 FXTs maximum sold.

Unsold tokens in the private sale are transferred to the pre-sale. Unsold tokens in the pre-sale are not available for sale.

1.2 Token Rewards

Contributors will be given rewards as follows:

- Contributors in the private sale are rewarded with 40% extra FXTs. For example, if a contributor buys $3,000,000$ FXTs for 500 ETH during the private sale, the contributor is rewarded with $1,200,000$ FXTs, receiving a total amount of $4,200,000$ FXTs.
- Contributors in the pre-sale are rewarded with 20% extra FXTs.
- Contributors that buy tokens during the first three days of the main-sale are rewarded 10% extra FXTs. After the first three days, contributors that buy tokens during the following seven days of the main-sale are rewarded 5% extra FXTs.

1.3 Token Distribution

The tokens issued to contributors (including tokens issued as rewards to contributors) will constitute 60% of the total number of issued FXTs. The additionally issued tokens (40% of the total number of tokens) will be allocated as follows:

- 5% of the tokens will be allocated to advisors and partners.
- 5% of the tokens will be allocated for bounty and blockchain industry donations.
- 15% of the tokens will be allocated for technology acquisition for FuzeX ecosystem.
- 15% of the tokens will be allocated to the FuzeX team. Part of these tokens will be given to compensate FuzeX employees.

1.4 Extra Features

Pausable FuzeX has the power to pause the tokens. During this time, no token transfers and approvals can be made.

Refunds If the minimum cap of 10,000 is not reached, then contributors can request a refund. Refunds are not issued automatically, and contributors must explicitly call the `refund` function defined in the `FXTVault` contract.

2 Audit Overview

2.1 Scope of the Audit

The scope of the audit is limited to the following source code files. All of these source code file were received on January 4th, 2018:

- FXTMainsale.sol
  - Final SHA-256: `971812be8c57bf4fe1b8c54665ed452006d39645b8e4cd69f2b8df42b63dd283`
- receiver/PaymentFallbackReceiver.sol
  - Final SHA-256: `093eaa6e3802e8eebcb29ef7799e1a4868926376d1cadc99c2bc71d0591070b6`
- receiver/PresaleFallbackReceiver.sol
  - Final SHA-256: `4160351cbfd38cb15f005d869b2fb4c5045157ad1e7b07fd1eefce43ab2bb8c4`
- FXTPresale.sol
  - Final SHA-256: `51aa6016e7bf8284e2d3b7bbc105cb582a3ac0c0dde2338f0b51981ffbc962e1`
- SampleMainsale.sol
  - Final SHA-256: `bf80cc2edaff17814279f5e62873ae4a0ac61b9a0d70dd62e727671b73e214ee`
- FXT.sol
  - Final SHA-256: `918e58073c71409d77e2a4e8283d727e99cfa91162376b172b1de0bc5d463bae`
- minime/TokenController.sol
  - Final SHA-256: `30830501232f6e8272332d9b6b8a5f50cbcf6d1df0bf4343a9610487854b3b16`
- PresaleVault.sol
  - Final SHA-256: `df20659e03601eb538cc0d1a1845e63fe209fe956f7d9d2429aae45abda3a468`
- MainsaleVault.sol
  - Final SHA-256: `4f505d91bef9a6be6d8828748eebdf66240dd71c34c7cab1c1bf2a656a4169d2`
- BTCPayment.sol

  - Final SHA-256: `28ccc5a5f505226ea24933b0e1a43a7d19687112e74b14d5e33e97859f63352a`
- TeamTimeLock.sol
  - Final SHA-256: `041682464086c653234332a9ddfcd556b12f57585f6b19bfc8cc9d37b93c6139`

2.2 Depth of Audit

The scope of the security audit conducted by ChainSecurity Ltd. was restricted to:

- Scan the contracts listed above for generic security issues using automated systems and manually inspect the results.
- Manual audit of the contracts listed above for security issues.

2.3 Terminology

For the purpose of this audit, we adopt the following terminology. For security vulnerabilities, we specify the *likelihood*, *impact* and *severity* (inspired by the OWASP risk rating methodology[1]).

Likelihood represents the likelihood of a security vulnerability to be encountered or exploited in the wild.

Impact specifies the technical and business related consequences of an exploit.

Severity is derived based on the likelihood and the impact calculated previously.

We categorize the findings into 3 distinct categories, depending on their criticality:

- Low - can be considered as less important
- Medium - needs to be considered to be fixed
- High - should be fixed very soon
- Critical - needs to be fixed immediately

During the audit concerns might arise or tools might flag certain security issues. If our careful inspection reveals no security impact, we label it as ✓ No Issue. Finally, if during the course of the audit process, an issue has been addressed technically, we label it as ✓ Fixed, while if it has been addressed otherwise we label it as ✓ Addressed.

Findings that are labelled as either ✓ Fixed or ✓ Addressed are resolved and therefore pose no security threat. Their severity is still listed, but just to give the reader a quick overview what kind of issues were found during the audit.

[1] `https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology`

3 Limitations

Security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a secure smart contract. However, auditing allows to discover vulnerabilities that were overlooked during development and areas where additional security measures are necessary.

In most cases, applications are either fully protected against a certain type of attack, or they lack protection against it completely. Some of the issues may affect the entire smart contract application, while some lack protection only in certain areas. We therefore carry out a source code review trying to determine all locations that need to be fixed. Within the customer-determined timeframe, ChainSecurity Ltd. has performed auditing in order to discover as many vulnerabilities as possible.

4 Details of the Findings

4.1 Anyone can finalize the presale High ✓ Fixed

The `presaleFinished` boolean variable indicates whether the presale has finished. If set to true, then the mainsale has started. The `presaleFinished` variable is set to true by the following function defined in the `BTCPayment` contract:

```
function presaleFallBack(uint256 _presaleWeiRaised) public returns (bool) {
    if (presaleFinalized)
        return false;
    presaleFinalized = true;
    return true;
}
```

Since `presaleFinalized` is initially false and this method is public, any user can call it and set `presaleFinalized` to true.

Likelihood: High

Impact: Medium

Post-audit fix: The FuzeX team fixed the issue by requiring that only the `presale` contract can finalize the presale:

```
function (uint256) public returns (bool) {
    require(msg.sender == address(presale));
    ...
}
```

4.2 Mainsale does not start automatically if the presale is over Low ✓ Fixed

If the presale sells out, FuzeX has no opportunity to advance to the main sale. Instead, FuzeX has to wait until the time limit is reached, which might take up to one month. Alternatively, FuzeX could allow the execution of `finalizePresale` once `maxReached()` returns `true`.

Likelihood: Low

Impact: Low

Post-audit fix: The FuzeX team has modified the required pre-condition in the function `finalizePresale` to

```
function finalizePresale(address _mainsale) public onlyOwner {
    require(!isFinalized);
    require(maxReached() || now > endTime);
    ...
}
```

The new precondition allows the owner of the presale contract to finalize the presale after the ether cap of the presale is reached.

4.3 Reentrancy Analysis ✓ No Issue

We did not discover any reentrancy issues. This is because the FuzeX contracts use the `transfer` function to transfer funds, which only forwards a limited amount of gas to potentially untrusted callers. Whenever unrestricted calls to untrusted code are made, the FuzeX contracts are in a consistent state and therefore not vulnerable to reentrancy attacks.

4.4 No Callstack Bugs ✓ No Issue

We did not discover any callstack issues.

4.5 Ether Transfers ✓ No Issue

We did not discover unusual or dangerous ether transfers in the code.

4.6 Safe Math ✓ No Issue

The FuzeX contracts use the safe math library to avoid over-/under-flows. In particular, critical variables such as `weiRaised` and `beneficiaryFunded` are always handled with calls to the safe math library.

5 Recommendations

- The `BTC` to FXT formula appears to be misunderstandable. On the website and inside the white paper it says:

$$\frac{1\ \texttt{BTC}}{X\ \texttt{ETH}} \cdot \frac{1\ \texttt{ETH}}{6000\ \texttt{FXT}} \qquad X = \frac{\text{BTC Price}}{\text{ETH Price}}$$

  Given that it is the `BTC` to FXT formula, it should probably say:

$$X \cdot \frac{6000\ \texttt{FXT}}{1\ \texttt{ETH}} \qquad X = \frac{\text{BTC Price}}{\text{ETH Price}}$$

  Currently this would mean:

$$X = \frac{17{,}197\ \text{USD}}{815\ \text{USD}} \approx 21.1 \qquad X \cdot \frac{6{,}000\ \texttt{FXT}}{1\ \texttt{ETH}} \approx \frac{126{,}600\ \text{FXT}}{1\ \texttt{BTC}}$$

  Therefore 2 `BTC` would buy:

$$2\ \texttt{BTC} \cdot \frac{126{,}600\ \text{FXT}}{1\ \texttt{BTC}} = 253{,}200\ \text{FXT}$$

- The following documentation of the `onlyRegistered` function defined in `KYC.sol`

```
1 @param _isPresale bool Whether the address is registered to presale or
      mainsale
```

  can be misunderstood. The argument `_isPresale` indicates the contribution phase (presale or mainsale), not whether the address is registered for one of these phases.

- A more appropriate name for the function `setAdmin` in `KYC.sol` is `addAdmin` because this function does not remove the current admin(s) from the list of admins.
- If necessary, consider implementing a `removeAdmin` to `KYC.sol`.
- The function `getRate()` calculates the number of FXTs that can be purchased for 1 ETH. The rate is determined based on the base rate `baseRate`, the bonus coefficient `BONUS_COEFF`, and the reward rate for the presale `PRE_BONUS`. All three variables are not modified, which means that `getRate()` returns a constant value that can be pre-computed to avoid unnecessary computations.

- Typo in comment in `FXTPresale.sol`: “privavte-sale”
- It would be good to clarify, whether or not there is a potential refund. Currently there is this comment in the code:

```
1       // TODO: enable refund if min cap not reached
```

  But no minimum cap or refund functionality has been implemented.

- The argument `_presaleWeiRaised` is not used in the following function:

```
1 function presaleFallBack(uint256 _presaleWeiRaised) public returns (bool)
      {...}
```

  This function is defined in the `BTCPayment` contract.

- The comments in `KYC.sol` still reference the `ASTCrowdsale` contract.
- It is unclear, why the `KYC` would differentiate between registrations for presale and main sale. Removing this differentiation could reduce the complexity of the `KYC` and would also reduce the amount of transactions FuzeX has to perform to add users. Currently, FuzeX needs to perform two transactions to register a user for the presale and the main sale.

6 Disclaimer