Security Audit of HydroMiner Smart Contract

This report is public.

ChainSecurity Ltd.

October 17, 2017

Contents

1 Introduction

We first and foremost thank you for giving us the opportunity to audit your smart contract code. This documents outlines our methodology, limitations and results for your security audit.

1.1 Overview of the HydroMiner platform

HydroMiner aims to provide eco-friendly, cost effective and profitable mining for cryptocurrencies by using hydro power. HydroMiner is based in Austria and uses the geographic advantage of the alps to provide ample hydro power.

1.2 ICO Description

Through the ICO customers can purchase HydroCoins which generate mining-based rewards. The ICO is split into two phases: presale and crowdsale. The presale and the crowdsale make a total of 25,000,000 of the 100,000,000 HydroCoins available for purchase. The presale is limited to investments of 1500 ether and requires a minimum token purchase of 50 ether. The subsequent crowdsale makes the remaining HydroCoins available and has no minimum token purchase, but requires a registration, which is checked in the contract through the `registered` variable.

The token price depends on the time when the token is bought. During the presale, customers receive 125 HydroCoins per ether. During the crowdsale this rate starts at 120 HydroCoins and drops weekly to 115, 110 and 105.

Overall, the ICO has no minimum funding level. As there is no minimum funding level, there is also no refund option.

1.3 Scope of the Audit

The audit was based on the Ethereum Virtual Machine (EVM) after `EIP-150` and solidity compiler `0.4.16+commit.d7661dd9.Linux.g++`.

The scope of the audit is limited to the following source code files, these files were last retrieved at: September 17th, 2017.

- HydroCoinCrowdsale.sol
  - Final SHA-256: `662b50e32a8a8511f16ef37e3231bb996d3832f3db0d1e5d683d9dacc651ce19`
- HydroCoinPresale.sol
  - Final SHA-256: `969472c0cabf9af414fdce2784bd78991f74df8b8326667ad518c3ab326c9a13`

- HydroCoin.sol
  - Final SHA-256: `dbcebc0e009433644f4ad03ddeb9b9158fc5cfa06ff8d717479b79238fbd10b7`

1.4 Depth of Audit

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

- Scan the presale contract (HydroCoinPresale.sol) and the token contract (HydroCoin.sol) for generic security issues using automated systems and manually inspect the results.
- Audit the crowdsale contract (HydroCoinCrowdsale.sol) for security issues.

Due the tight timelines, ChainSecurity Ltd. was unable to perform an expert audit on the presale and the token contracts.

1.5 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.

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

2 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.

3 Details of the Findings

3.1 No Reentrancies ✔ No Issue

The HydroMiner contract does not contain any vulnerabilities that would allow reentrancy attacks. This is because no untrusted code is ever invoked.

3.2 No Callstack Bugs ✔ No Issue

The HydroMiner contract does not contain any vulnerabilities that would allow attacks based on a callstack overflow. This is because all exception are properly handled and propagated.

3.3 Ether Transfers ✔ No Issue

Ether transfer can lead to variety of issues in Ethereum. These issues include callstack bugs, reentrancies and denial-of-service attacks. For HydroMiner, Ether transfers only occur at the end of the `buyTokens` functions. Here, HydroMiner uses the `transfer` function and performs the transfer to a trusted wallet. Thereby, HydroMiner eliminates callstack bugs during ether transfers.

3.4 Safe Math ✔ No Issue

HydroMiner also uses the popular `SafeMath` library for critical operations to avoid arithmetic over- or underflows and safeguard against unwanted behaviour.

In particular, the critical variables `coinsToSell`, `restrictedTokens`, `weiRaised`, `tokensSold` and `deposits` are only updated using `SafeMath` operations or constants.

3.5 HydroCoin List does not reset last pointer Medium ✔ Fixed

Inside the `remove` function of `HydroCoin.sol` the `last` pointer is not reset if the last element of the list is removed. Therefore, the list might get corrupted over time.

Fix: HydroMiner introduced a code fix to properly reset the `last` pointer.

Listing 1: remove() in HydroCoin.sol

```
82          address next = theList[whom].next;
83          address prev = theList[whom].prev;
84          if (prev != 0x0) {
85              theList[prev].next = next;
86          }
87          if (next != 0x0) {
88              theList[next].prev = prev;
89          }
90          if (last == whom) {
91              last = prev;
92          }
```

Likelihood Medium

Impact Low

3.6 No Event is emitted if the first element of the list is removed Low ✔ Fixed

If members are removed from the list inside HydroCoin a `remove` event is emitted. However, such an event is not emitted if the removed element is the first list element.

Fix: HydroMiner introduced a code fix to properly emit the event:

Listing 2: remove() in HydroCoin.sol

```
76          if (first == whom) {
77              first = theList[whom].next;
78              theList[whom] = XRec(false,0x0,0x0,0);
79              Ev("remove",whom,0);
80              return;
81          }
```

Likelihood Medium

Impact Low

3.7 Incorrect check for finalization of CrowdSale Medium ✔ Fixed

In the `hasEnded` function of `HydroCoinCrowdsale` an incorrect check of the following form happened:

Listing 3: hasEnded() in HydroCoinCrowdsale.sol

```
110   if (tokensSold.add(tokensSoldInPresale) >= coinsToSell)
```

This is incorrect as `tokensSold` already contains `tokensSoldInPresale`.

Fix: HydroMiner introduced a code fix to properly check the condition:

Listing 4: hasEnded() in HydroCoinCrowdsale.sol

```
110   if (tokensSold >= coinsToSell)
```

Likelihood High

Impact Low

3.8 Other Recommendations or Comments ✔ Fixed

- It is likely that the available HydroCoins cannot be sold out completely. This is due to the `rate`, the `minContribution` and the following check:

Listing 5: validPurchase in HydroCoinCrowdsale.sol

```
99      require(msg.value >= minContribution);
100     rate = rates[(now - startTimestamp) / (1 weeks)];
101     uint256 thisGuysTokens = rate.mul(msg.value);
102     require(tokensSold.add(thisGuysTokens) <= coinsToSell);
```

Therefore, it might be that the last coins cannot be sold as their remaining value is smaller than `minContribution`. In the worst case this can be roughly

$$\text{rate} \cdot \text{amount} = 105 \cdot 0.99 \text{ finney} = 0.10395 \text{ HydroCoin}$$

Given the total sale of 25,000,000 HydroCoins, this amount is negligible.

Fix: HydroMiner introduced a code change to allow closing the crowdsale with a small remainder of tokens:

Listing 6: hasEnded in HydroCoinCrowdsale.sol

```
107   function hasEnded() public constant returns (bool) {
108       if (now > endTimestamp)
109           return true;
110       if (tokensSold >= coinsToSell - minContribution.mul(120))
111           return true;
112       return false;
113   }
```

Due to this code change the potential deadlock can be averted.

4 Conclusion

The HydroMiner smart contracts have been analyzed under different aspects, with different open-source tools as well as our fully fledge proprietary inhouse tool. Overall, we found that HydroMiner employ good coding practices and has clean, documented code. We have no remaining security concerns about the HydroMiner smart contracts, as all detected issues were either fixed or addressed.

ChainSecurity

5 Disclaimer