ChainSecurity

Security Audit of HelloGold Smart Contracts

This report is public.

ChainSecurity Ltd.

August 19, 2017

HelloGold™ Foundation

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 HelloGold platform

HelloGold calls for a crowdsale of ether through the HelloGold Foundation (HGF) in order to allocate HGT (HelloGold Tokens). The crowdsale aims to raise 8M USD, by selling 300,000,000 HGT, the corresponding ether price will be determined prior to the public sale. The raised ether is retained in the crowdsale contract until the minimum cap of USD 1,000,000 is met. After meeting the minimum cap, all ether is sent to the multisignature contract. Allocated HGT are frozen until the sale is complete. If the crowdsale does not achieve the minimum of USD 1,000,000, the contributors may withdraw their funds.

HelloGold performs an inhouse know your customer (KYC) verification, such that the contributions of each customer are limited to an ether equivalent of USD 50,000.

The price of HGT is divided into 5 tiers, each worth 60M HGT. The time of the crowdsale is initially set to one week. However, each time that a tier is complete (i.e. 60M HGT are allocated), a new HGT price is defined and the crowdsale is extended by one week. This process repeats for each tier, or ends when the total cap is reached.

The crowdsale offers 30% of the total amount of HGT token that are created. The total of 1,000,000,000 HGT will be launched in the HGF_Reserve account.

1.1.1 GoldBacked Token

HGT are directly linked to GoldBacked Tokens (GBT), such that any transfer of HGT triggers a call to the GBT contract.

GBT can be created through two means:

- The HGT holders receive GBT pro-rata to their HGT holdings
- The HGF is allowed to mint GBT to back new gold allocation from HelloGold

GBT can be destroyed through two means:

- GBT accounts are charged 2% fees on their GBT holdings per annum calculated daily.
- GBT can be converted to Gold in a HelloGold account.

1.2 Scope of the Audit

The audit was based on the Ethereum Virtual Machine (EVM) after `EIP-150` and solidity compiler `0.4.14-develop.2017.7.25+commit.7ad42aee.Darwin.appleclang`.

The scope of the audit is limited to the following source code files. For the files we present the hash at the beginning (B) and at the end (E) of the audit. All hashes are `SHA-256`:

- HelloGoldToken.sol
  - B: `8fc759c63ca8cede88c77f646eb7bc214a213ed2e9d6dd24754e668b364b3dc8`
  - E: `8fc759c63ca8cede88c77f646eb7bc214a213ed2e9d6dd24754e668b364b3dc8`
- HelloGoldSale.sol
  - B: `bceaeb0239d1210a848f24ee62bd11e08f5105842743f9bad76ed138b6eb3a2e`
  - E: `5e82eef61d83c108c032872072dd5eb3be4b6450c4adff479bb8cdbe33f3196b`
- GoldFees.sol
  - B: `921e7bdeef0563fc88be1a0fdd6e9fdb1c182bbbbb60bd7a6d0df27e5c042378`
  - E: `d963022a60415010a1628fde86d84f7e107b2211206d50756a5543056cee7adc`
- GoldBackedToken.sol
  - B: `81e51746aa406ec5cd5ec3b59649ac21a9356ced3c646457624234723f2313ea`
  - E: `458eadb91c697a6efc89736b7d326db9657114e20127d1ae0b61f845a34a0cd1`

Third-party libraries used by HelloGold are:

- ERC20.sol
  - B: `1710681a6c89150f4b81918ce7630cc99832a2e0f943a7f63ee5ba3a5e81ad5a`
  - E: `1710681a6c89150f4b81918ce7630cc99832a2e0f943a7f63ee5ba3a5e81ad5a`
- Ownable.sol
  - B: `25018d8f89dc37a9db30aa4f2f3d6af33732ee574f92b8f23a5254eb1e6e4ea5`
  - E: `25018d8f89dc37a9db30aa4f2f3d6af33732ee574f92b8f23a5254eb1e6e4ea5`
- Pausable.sol
  - B: `60f010f950ce3029e1e8507140b28a563d9ed6d6b1567fce7fd6c3b32ea73391`
  - E: `60f010f950ce3029e1e8507140b28a563d9ed6d6b1567fce7fd6c3b32ea73391`
- SafeMath.sol
  - B: `dd5f568c275dc8ed66f80eb04b86ccd478c759f01c8a95f47652ef4bae93265d`
  - E: `dd5f568c275dc8ed66f80eb04b86ccd478c759f01c8a95f47652ef4bae93265d`

- StandardToken.sol
  - B: `02dd3ebe07a67b930ce88a2d4e8a679fdc37ca7cf79ecd855c92bd645d723e1f`
  - E: `5f3a34bcf5a61cb471804d18431f1335258eb22704d49a321bf72ea2a131b0bb`

1.3 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

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.

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 to determine all locations that need to be fixed. ChainSecurity Ltd. has performed extensive auditing in order to discover as many vulnerabilities as possible.

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

3 Details of the Findings

3.1 Unauthorized Users can Potentially Remove Allocations Critical ✓ Fixed

The function `addAllocationPartTwo()` defined in `GoldBackedToken.sol` has no modifier, such as `onlyOwner`. Therefore, any Ethereum user can call the function. When the function is called with argument `numSteps` = 0, line 236 gets executed. Therefore, currentAllocations are overwritten with a potentially empty value of partAllocations. As a result, users lose their GBT.

Listing 1: addAllocationPartTwo in GoldBackedToken.sol

```
219 function addAllocationPartTwo(uint numSteps){
220         for (uint i = 0; i < numSteps; i++ ){
221                 partPos--;
222                 (partAllocations[partPos].amount,partFees) = calcFees(
                        partAllocations[partPos].date,now,partAllocations[partPos].
                        amount);
223 
224                 partAllocations[partPos].amount += partAllocations[partL - 1].
                        amount;
225                 partAllocations[partPos].date    = now;
226                 if (partPos == 0) {
227                         break;
228                 }
229         }
230         if (partPos != 0) {
231                 StillToGo(partPos);
232                 return; // not done yet
233         }
234         PartComplete();
235         FeeOnAllocation(partFees,now);
236         currentAllocations = partAllocations;
237 }
```

Possible Scenario:

1. The token sale finishes.
2. The owner of GBT calls `addAllocation()`. `currentAllocations` contains the new allocation, while `partAllocations` is empty.
3. The token owners use BalanceOf() to see that they have received GBT.
4. Anyone calls `addAllocationPartTwo(0)`. `currentAllocations` gets overwritten with the empty `partAllocations`.
5. If the token owners call BalanceOf() again, they find their balance to be 0 again.

Likelihood High

Impact High

3.2 Allocations might be lost High ✓ Fixed

The current allocations for the GBT are stored inside the `currentAllocations` variable. This variable can be updated in two ways: either by using `addAllocation()` or by using `addAllocationPartOne()` together with `addAllocationPartTwo()`. The current state of the code allows undesirable sequences for these function call, which can lead to the loss of allocations. The most important scenario is as follows:

1. `addAllocation()` is used once or multiple times. Hence, `currentAllocations` contains the allocations while `partAllocations` is empty.
2. Due to rising gas costs of calling `addAllocation()`, `addAllocationPartOne()` and potentially `addAllocationPartTwo()` are called. The new allocation is added to `partAllocations`.
3. Whenever this newly added allocation is complete `currentAllocations` is overwritten with `partAllocations`. One of the two possible locations is given here:

Listing 2: addAllocationPartTwo in GoldBackedToken.sol

```
239         currentAllocations = partAllocations;
```

4. After this operation, all previous allocations in `currentAllocations` are lost and only the least recently added allocation is left.

Furthermore, for the integrity of allocations HGF must ensure that whenever there is an incomplete allocation, `addAllocationPartTwo()` is called until the allocation is complete before any other call to `addAllocation()` or `addAllocationPartOne()` is made. These call sequences can also have negative effects.

Likelihood Medium

Impact High

3.3 Minimum Contribution and Rounding Effects Medium ✓ Fixed

It might be desirable to specify a minimum contribution and check it in the code. With the current parameters, any contributions below 10^{10} Wei leads to 0 HGT and no refund. This is due to the integer division in Solidity in the following lines of `HelloGoldSale.sol`.

Listing 3: Token Calculation in HelloGoldSale.sol

```
price = prices[tierNo];                    // price must include the 10^8
uint tokens = safeMul(val, price);         // (val in eth * 10^18) * #tokens per eth
tokens = safeDiv(tokens, 1 ether);         // val is in ether, msg.value is in wei
```

The token calculation can have negative rounding effects for the token buyer. However, these effects are marginal for the current parameter settings.

Contrary, the following lines, used at the end of a tier, can lead to negative rounding effects for HGF:

Listing 4: Token Calculation in HelloGoldSale.sol

```
uint weiCoinsLeftInThisTier = safeMul(coins2buy,1 ether);
uint costOfTheseCoins = safeDiv(weiCoinsLeftInThisTier, price);  // how much did
    that cost?
```

A rather extreme case would be the following with the price of $142,900,000$ taken from the code:

$$\texttt{coins2buy} = 1 \Rightarrow \texttt{weiCoinsLeftInThisTier} = 10^{18}$$
$$\texttt{price} = 142900000 \Rightarrow \texttt{costOfTheseCoins} = 6997900629$$

Therefore, a user receives 1 HGT for $6,997,900,629$ Wei, whereas the normal price would have been $6,997,900,630$ Wei for 1 HGT. The resulting price difference is $< 1.5 \cdot 10^{-8}\%$. The rounding effects are therefore limited for the current parameters.

Likelihood High

Impact Low

3.4 GBT Fee Avoidance by Users Low ✓ Fixed

GBT holders pay a fee of 2% per annum. However, they might try to avoid the fee. The fee is calculated based on the number of days since the last fee calculation:

Listing 5: Fee Calculation in GoldFees.sol

```
        uint256 numberOfDays = (end - start) / (1 days);
        if (numberOfDays == 0) {
            amount = startAmount;
            return;
        }
        amount = (rateForDays(numberOfDays) * startAmount) / (1 ether);
```

Due to the integer division a time span of up to 24 hours, results in no fees. Therefore, an adversarial token holder might transfer his GBT tokens every 23 hours and 59 minutes between two accounts he controls. The token holder has to pay no GBT fees, but has to pay the gas for the additional transactions.

In our measurements, the GBT `transfer` transaction resulted in costs of: 132,042 gas. If the adversarial token holder does this for a year, his gas costs are $365 \cdot 132{,}042$ gas $=$ 48,195,330 gas. At the current gas price of roughly $2 \cdot 10^{-8}$ ETH and with 1 ETH = 300 USD, the token holder would pay 48,195,330 gas = 0.48 ETH = 144.59 USD per year to avoid paying GBT fees. Therefore, anyone with GBT tokens worth at least 7229.3 USD would have an incentive to adopt this strategy, as they would otherwise pay more than 144.59 USD in GBT fees.

We anticipate that this is rather unlikely to happen as those with many GBT tokens might not want to risk the tokens by letting a script constantly send them around. Additionally, were this to happen, HGF has a way to change fee calculation and can therefore make fee calculation more precise.

Side Note: Even though we don't anticipate this, HGF has to ensure not to make too frequent allocations to GBT. In particular, given the current fee calculation and the GBT structure, if HGF would make allocation every 20 hours, no GBT fees would ever be charged. However, according to our understanding of HGF's intentions, we do not perceive this as a likely issue.

Likelihood Low

Impact Medium

3.5 GBT Fees can be increased by Other Users Low ✓ Addressed

In Section 3.3 we have already discussed rounding effects for HGT, similarly they can have effects on GBT. In the GBT, during fee calculation multiple such roundings take place. An adversary can trigger them to lower the overall balance of another GBT user. We present an example scenario:

There is a victim V and an adversary A. V starts with 1.43 GBT ($\texttt{balance} = 1.43 \cdot 10^{18}$) and wants to keep its GBT for six days before transferring them. Therefore, after six days V would have:

$$\texttt{GoldFees.calcFees}(0, 6 \cdot 86400, 1430000000000000000)$$
$$\rightarrow (\texttt{amount} = 1429529927411320519, \texttt{fees} = 470072588679481)$$

However, if A sends 1 GBT cent (10^-18 GBT) to V after three days, V's balance after three days is calculated as:

$$\begin{aligned}&\texttt{GoldFees.calcFees}(0, 3 \cdot 86400, 1430000000000000000)\\&\rightarrow (\texttt{amount} = 1429764944387079432, \texttt{fees} = 235055612920568)\end{aligned}$$

Then 1 is added due to the transfer so that V has an amount of 1429764944387079433 after three days. After the full six days V's balance is re-calculated as:

$$\begin{aligned}&\texttt{GoldFees.calcFees}(3 \cdot 86400, 6 \cdot 86400, 1429764944387079433)\\&\rightarrow (\texttt{amount} = 1429529927411320487, \texttt{fees} = 235016975758913)\end{aligned}$$

Therefore, V's final balance is 1429529927411320487 instead of 1429529927411320519 a difference of 32 GBT cents.

There are certainly more extreme examples, but we find that the effect should not be major. Furthermore, adversary A has no incentive to perform this attack as it pays in GBT and in gas. Finally, as stated above, HGF has the possibility to change the fee calculation to address this issue.

Likelihood Low

Impact Medium

3.6 Blocked Reentrancy Low ✔ Addressed

There is a potential reentrancy in the code. However, according to current EVM specification, the reentrancy is not exploitable. The `transfer()` call allows the recipient of the refund to regain the control flow. This happens before `coinsLeftInTier` is adjusted and a potentially higher tier has been selected. Therefore, the recipient could send more money and keep purchasing at the lower tier.

However, this is currently not possible as the `transfer()` call only provides the recipient with a 2300 gas to perform his second call which is not sufficient as a call with non-zero value alone requires 9000 gas.

Listing 6: Blocked Reentrancy in HelloGoldSale.sol

```
if (refund > 0) {
    ethRaised = safeSub(ethRaised, refund);
    recipient.transfer(refund);
}
if (coinsRemaining <= (MaxCoinsR1 - minimumCap)){ // has passed success criteria
    if (!multiSig.send(this.balance)) {          // send funds to HGF
        log0("cannot forward funds to owner");
    }
}
coinsLeftInTier = safeSub(coinsLeftInTier,actualTokens);
```

Likelihood Low

Impact Medium

3.7 Non Updated Libraries Medium ✓ Fixed

The StandardToken.sol library used, does not check whether the addresses allowance is reduced to zero before changing the approve amount. This has been pointed out here on the most recent StandardToken.sol:
`https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/token/StandardToken.sol#L50`.

3.8 Minor vulnerabilities

3.8.1 Minimum Fee not enforced Low ✓ Fixed

The comments says that GoldFees has a minimum fee of 1, and `fee` is set to 1 in line 87, but `fee` is immediately overwritten afterwards. Therefore, line 87 has no effect.

Listing 7: GoldFees.sol

```
77      // minimum fee is 1
78      function calcFees(uint256 start, uint256 end, uint256 startAmount) constant
            returns (uint256 amount, uint256 fee) {
79          if (startAmount == 0) return;
80          uint256 numberOfDays = (end - start) / (1 days);
81          if (numberOfDays == 0) {
82              amount = startAmount;
83              return;
84          }
85          amount = (rateForDays(numberOfDays) * startAmount) / (1 ether);
86          if ((fee == 0) && (amount != 0)) fee = 1;
87          fee = safeSub(startAmount,amount);
88      }
```

At the current parameters, we except fee to be at least 1 if the end of the function is reached. Therefore, the this issue would only have an impact if the parameters would be changed.

Likelihood High

Impact Low

3.9 Recommendations per file ✓ Fixed

3.9.1 readme

In the readme, HGF lists the steps for the "Post Launch setup". This is an excellent idea. We noticed, that the freezing of HelloGoldToken using its `pause()` function can not be found on this list. This should be performed before the start of the crowdsale, but after the approval and the preallocation of tokens. Note, that further items might be missing on this list.

The readme says "Each time a tranche is complete the new price comes into effect and the sale is extended by a further week until either last tranche is complete or total cap reached". It is not entirely clear if a new tier should begin if the previous tier was not sold out, but the time of the previous tier has expired. Currently, there appears to be no functionality to extend the crowdsale if the previous tier was not sold out.

3.9.2 HelloGoldSale.sol

- Why was `multiSig` not used in `complete()` instead of `msg.sender`?
- The name of the variable "price" is misleading, as it is not a price but a number of tokens per ether.
- A minor comment on readability:

  Listing 8: HelloGoldSale.sol

  ```
  237 uint256 coins2buy = (coinsLeftInTier > coinsRemaining) ? coinsRemaining :
          coinsLeftInTier;
  ```

  Using the already inherited `min256()` might be more readable.

3.9.3 GoldFees.sol

We plot in Figure 1 how the `rateForDays` function evolves over time. This appears to be the desired behavior as fees of roughly 2% per annum would be deducted. For this plot, we took the default values for `rateN = 9999452054794520548` and `rateD = 19`, while maxDays is set through the constructor in `calcMax()`.

- It appears that the variable `maxDiv` is never used within the contract. However, it may be intended for outside use.

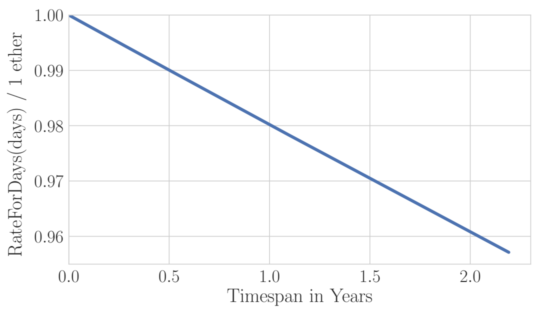

Figure 1: Output of function rateForDays for up to 800 days.

3.9.4 GoldBackedToken.sol

- It appears that GoldBackedToken is not an ERC20 token. Is this intentional? The function name is "BalanceOf" instead of "balanceOf".

4 Conclusion

The HGF smart contracts have been analyzed under different aspects. We have been able to identify several security issues. HGF's response has been fast and professional, they were able to fix most of the issues quickly through code changes.

Some of the issues, which are harder to address on a technical level, will be addressed through other means. As HGF has the option to change GBT fee calculation in the future, they will be monitoring for the possible issue described in Section 3.5 and adapt the calculation should the need arise. Furthermore, as HelloGoldSale customers have to go through a KYC process, HGF can address the issue described in Section 3.6.

Overall ChainSecurity Ltd. is not aware of any remaining issues in the HGF contracts.

ChainSecurity

5 Disclaimer

UPON REQUEST BY HELLOGOLD FOUNDATION, CHAINSECURITY AGREES MAKING THIS AUDIT REPORT PUBLIC. THE CONTENT OF THIS AUDIT REPORT IS PROVIDED "AS IS", WITHOUT REPRESENTATIONS AND WARRANTIES