SmartDec

Tigereum Smart Contracts Security Audit

Abstract

In this report we consider the security of the Tigereum project. Our task is to find and describe security issues in the smart contracts of the platform.

Procedure

In our analysis we consider Tigereum whitepaper (sha1sum 0a21f688e1b7eddceaf9285bdbb49fdd0607af65 *970dc6_2c6e379920be4d38afb324a8aa076df4.pdf) and smart contracts code (sha1sum 02805d71084468f6c16db7a1c7115a6319cc9361 *tgICOv1.zip).
We perform our audit according to the following procedure:

- automated analysis
  - we scan project's smart contracts with our own Solidity static code analyzer SmartCheck
  - we scan project's smart contracts with several publicly available automated Solidity analysis tools such as Remix, Oyente, Securify (beta version since full version was unavailable at the moment this report was made), and Solhint.
  - we manually verify (reject or confirm) all the issues found by tools
- manual audit
  - we manually analyze smart contracts for security vulnerabilities
  - we check smart contracts logic and compare it with the one described in the whitepaper
  - we run tests
- report
  - we report all the issues found to the developer during the audit process
  - we reflect all the gathered information in the report

Disclaimer

The audit does not give any warranties on the security of the code. One audit can not be considered enough. We always recommend proceeding to several independent audits and a public bug bounty program to ensure the security of the smart contracts.

Checked vulnerabilities

We have scanned Tigereum smart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered (the full list includes them but is not limited to them):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Use of tx.origin
- Exception disorder
- Gasless send
- Balance equality
- Byte array
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Redundant fallback function
- Send instead of transfer
- Style guide violation
- Unchecked external call
- Unchecked math
- Unsafe type inference
- Implicit visibility level

About The Project

Project Architecture

Smart contracts are provided as a Truffle project with js-tests. The project consists of:

- mintable token
  - Tigereum.sol inherited from OpenZeppelin MintableToken.sol
- crowdsale
  - TigereumCrowdsale.sol inherited from OpenZeppelin Crowdsale.sol

Code Logic

Tigereum is ERC20 mintable token: symbol — TIG, decimals — 18, name — Tigereum. Since the token is mintable, its TotalSupply depends on how many tokens were sold at presale.
The crowdsale is capped crowdsale with the following parameters:

- 33,500,000 are to be sold during the crowdsale
- after the end of the crowdsale 16,500,000 are to be send to the specified addresses (advisors, presale, team, bounty, founders — the distribution is described in the code)
- the payed ETH go to:
  - 1% — to the specified advisor address
  - 99% — to the contract owner
- funds (ETH) are forwarded to owner and advisor immediately after the payment is sent
- tokens are minted immediately after the payment is sent
- the crowdsale start is 8.12.2017 9:00
- the crowdsale end is 18.12.2017 23:59
- the price is 1 ETH = 1000 TIG
- during the first 12 hours the is a 33% discount (1 ETH = 1330 TIG)
- the founders' tokens are locked for 90 days after crowdsale end

However, there are bugs in the implementation due which the crowdsale will not end when the cap is reached (besides, it is impossible to exceed the cap).

Automated Analysis

We used several publicly available automated Solidity analysis tools. Here are the combined results of their analysis. All the issues found by tools were manually checked (rejected or confirmed).

Tool	Rule	false positives	true positives
SmartCheck	Constant functions		1
	ERC20 approve		1
	ERC20 transfer return false		2
	Style guide violation		1
	Using throw		1
	Var	2	
	Address hardcoded	1	
	DOS with revert	5	
	DOS with throw	1	
	Functions returns type	25	
	Malicious libraries	1	
	Private modifier	17	
	Reentrancy external call	18	
	Should be pure but is not	8	
	Should be view but is not	12	
	Timestamp dependence	7	
	Unchecked math	18	
Total SmartCheck		115	6
Securify*	Transactions May Affect Ether Receiver	1	4
	Transactions May Affects Ether Amount	5	
Total Securify*		6	4
Remix	Use of "this" for local functions		1
	Gas requirement of function	23	
	Is constant but potentially should not be	1	
	Potential Violation of Checks-Effects-Interaction pattern	4	
	use of "block.timestamp"	2	
	use of "now"	3	

	Variables have very similar names	71	
Total Remix		104	1
Oyente	Assertion Failure	2	
Total Oyente		2	
Solhint	Event name must be in CamelCase		1
	Function order is incorrect, public function can not go after internal function		5
	throw is deprecated, avoid to use it		1
	Avoid to make time-based decisions in your business logic	5	
Total Solhint		5	7
Overall Total		232	18

Securify* — beta version, full version is unavailable.
Securify, Oyente — these tools do not support 0.4.18 compiler version; the specified version was changed in the code to 0.4.16 and 0.4.17 respectively for these tools.
Cases when these issues lead to actual bugs or vulnerabilities are described in the next section.

Manual Analysis

Contracts were completely manually analyzed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of automated analysis were manually verified. All confirmed issues are described below.

Critical issues

Critical issues seriously endanger smart contracts security. We highly recommend fixing them.

Incorrect tokensale finalization

TigereumCrowdsale.sol contains the following problems:

- TigereumCrowdsale.sol, line 184
  `if(state == State.NormalSale && msg.value*rate > cap) {...}`

First, the comparison must be non-strict (otherwise users can not buy all tokens). Second, the comparison should be not with `cap`, but with the remaining number of tokens (`tokensLeft`). Third, here the `rate` is not updated, though it can become smaller if the state changes.

- TigereumCrowdsale.sol, line 191
  `tokensLeft = SafeMath.sub(tokensLeft, numTokens);`

An exception will be thrown when trying to subtract if `tokensLeft < numTokens`. Thus, if the user wants to buy more tokens than he/she has left, he/she will not be able to buy anything. We recommend checking that `tokensLeft >= numTokens` before subtracting.

- TigereumCrowdsale.sol, line 213
  `uint256 tokensToRefundFor = cap.sub(tokensTotalWithFullBuy);`

An exception will be thrown when trying to subtract if `cap < tokensTotalWithFullBuy`. In this case, as in the previous one, if the user wants to buy more tokens than there is left, his/her transaction will not succeed and he/she will not buy anything. This is a serious bug since this part of code is supposed to handle this particular situation. We highly recommend fixing the bug and implementing the logic of selling all the remaining tokens and sending extra ETH (change) to the investor.

- TigereumCrowdsale.sol, line 215:
  `uint256 weiAmountToRefund = tokensToRefundFor.div(rate);`

Since `tokensToRefundFor` may not be divided exactly the `rate`, in this case less wei will be returned to the user than it should be. (However, this last part of bug is not critical, but medium)

Medium severity issues

Medium issues can influence smart contracts operation in current implementation. We highly recommend addressing them.

Code coverage

TigereumCrowdsale.sol test coverage is not sufficient:

- 47.25% of statements
- 36.54% of branches
- 63.16% of functions

- 48.54% of lines

We highly recommend improving tests for:

- `canMint` modifier
- `forwardFundsAmount` function
- `refundAmount` function
- `fixAddress` function
- `buyTokensUpdateState` function (lines 183 and 184)
- `buyTokens` function (lines 193-196)
- `lastTokens` function
- `withdrawLockupTokens` function
- `finalizeUpdateState` function (line 240)
- `endMintingBreak` function

Tests failed

The tests provided with the code have failed. We highly recommend fixing the code so that the tests will run successfully.

Discrepancies with the **whitepaper**

There are several discrepancies between the whitepaper and the contracts code:

1. According to the whitepaper, p.23 and p.34, the amount of tokens for investors is 80%, for founders 17%, for team members and advisors 2% and 1% for bounty. But according to the contracts code, there are 79% for participants, for founders 14,36%, for team members and advisors 4,64% (3,64% and 1% respectively) and 2% for bounty.
2. According to the whitepaper, p.34, the tokensale ends at 8 Dec 2017, GMT 23:00. But according to the contracts code, it ends at 8 Decr 2017, GMT 23:59.

Incorrect state transition

- If no purchases are made during the bonus period (i.e., within the first 12 hours after the beginning of the crowdsale), the first purchase after the end of the bonus period will still occur at the bonus price, since the state transition in the `buyTokens` function (TigereumCroudsale.sol, line 187) occurs only after purchase (TigereumCroudsale.sol, lines 188 and 189).
- If the time of the beginning of the crowdsale exactly matches with the timestamp of the current block, it will be possible to purchase tokens without switching the state from `BeforeSale` to `Bonus`. This happens because there is a strict check in the `buyTokensUpdateState` function in line 181, and in the `validPurchase` function (Crowdsale.sol), which is called in the `buyTokens` and `lastTokens` functions, the check is non-strict. Therefore, if this transaction tries to buy more tokens than `cap` then the contract will not change state to `ShouldFinalize` and an attempt to make a regular purchase of tokens (via the `buyTokens` function) can be made. Thus, it will be possible to buy more tokens than `cap`.
- There is a similar issue in case if there is an attempt to buy more tokens than `cap` at the Bonus stage. This bug is even more severe, because there is no need for the specific time match (as in previous case).

We highly recommend considering all possible combinations of conditions in the `buyTokensUpdateState` function.

Sending tokens and ethers to unusual investors

A contract is exposed to this vulnerability if a miner (who executes and validates transactions) can reorder the transactions within a block in a way that affects the receiver of ether. In TigereumCroudsale.sol, this applies to all transfers of ethers or tokens to addresses that are stored in the crowdsale fields (lines 26-48). The owner can change these fields using the `fixAddress` (line 144) function.
In addition, the owner should be warned that the transfer of ethers/tokens to these addresses depends on the order of the mining. Let us say, the crowdsale is completed, and the owner wants to call the `finalize` function, but before that he/she wants to change the addresses and calls `fixAddress` and then calls `finalize`. Depending on the order of the mining of these transactions, the tokens will be sent to different addresses.

Function Calling Order

The `withdrawLockupTokens` function can be called before the `finalize` function (because at TigereumCroudsale.sol, line 196 state changes to `Lockup`). This requires that no one calls `finalize` during the `Lockup` period (which is possible). As a result, tokens will not be given to any of users with addresses that are stored in the crowdsale fields (lines 26-48), except founders.

User can not withdraw all his/her funds

In functions `forwardFunds` (TigereumCroudsale.sol, line 128) and `forwardFundsAmount` (line 134), 99% and 1% of received wei are calculated. In this case, since the result of division can be only integer, the sum of received funds will not be equal to the transferred amount.
For example, if we pass 199 wei then

```
onePercent=1 wei
```

, so

```
onePercent.mul(99)=99 wei
```

Thus, 99 wei will be lost. That is, weis will be lost on every token purchase transaction. Also, the problem is that some amounts may not converge.

Production deploy

The provided code does not contain production deploy scripts, only test deploy with test addresses. Bugs and vulnerabilities in deploy scripts often appear and severely endanger contracts' security. We highly recommend not only developing deploy scripts very carefully but also performing audit of them.

ERC20 approve issue

There is ERC20 approve issue (StandardToken.sol, line 51). We recommend explicitly warning investors and users not to use `approve` directly and to use `increaseApproval`/`decreaseApproval` functions (or to change the approved amount to 0 and then to the desired value) instead - link.

Low severity issues

Low severity issues can influence smart contracts operation in future versions of code. We recommend to take them into account.

Code duplication

Functions `forwardFundsAmount` (TigereumCroudsale.sol, line 134) and `forwardFunds` (line 128) are almost exactly the same. Instead of the function `forwardFunds` one can simply call `forwardFundsAmount(msg.value)`.

Discrepancy with the ERC20 standard

In Tigereum.sol, line 14, according to the ERC20 standard, the variable `decimals` should be declared as `uint8`.

Check `ICOadvisor1` and `admin` addresses

The ether is redirected to addresses `ICOadvisor1` and `admin`. So, one needs to check in advance that these addresses are not addresses of contracts. In if case if, for example, address of ICOadvisor1 is a contract and its fallback function is implemented as follows:

```
function {
      throw;
}
```

no purchase transaction will work.

Suboptimal code

- At each attempt to purchase tokens, the `buyTokensUpdateState` (TigereumCroudsale.sol, line 180) function is called. It would be enough to call only the status check and `calculateCurrentRate`
- it is possible to `calculateCurrentRate` only once
- `hasEnded` duplicates the state check in `finalize`

Unused code

Event `numIs` (TigereumCroudsale.sol, line 71) and modifier `canMint` (line 73) are not used.

Use of "this" for local functions

In TigereumCrowdsale.sol in line 246 the internal function `hasEnded` is called as an external with the use of `this`. We recommend not using `this` to call functions in the same contract, it consumes more gas than normal local calls.

Code style

Codestyle issues influence code conciseness and readability and in some cases may lead to bugs in future. We recommend to take them into account.

Constants readability

Constants `lockupPeriod` (TigereumCroudsale.sol, line 62), `bonusPeriod` (line 63) are defined not clearly for the reader. We recommend defining them as a product of values (number of days, day length, etc., e.g. `90 * 1 days` or `12*1` hours), so that it will be obvious that they are calculated correctly.

Variables that are constants, but not marked `constant`

In the `TigereumCrowdsale` contract, the following variables are constants but not marked so: `cap` (TigereumCroudsale.sol, line 18), `startRate` (line 19), `toDec` (line 16), `hundredKInvestorSum` (line 33), `additionalPresaleInvestorsSum` (line 36), `preSaleBotReserveSum` (line 39), `ICOadvisor2Sum` (line 42), `teamSum` (line 45), `bountySum` (line 48), `foundersSum` (line 53), `startTimeNumber` (line 59), `endTimeNumber` (line 60), `lockupPeriod` (line 62), `bonusPeriod` (line 63), `bonusEndTime` (line 65).

Misleading comment

In TigereumCrowdsale.sol, line 33, the comment

```
//2 - 2.8% - 2,500,000
```

was written. But according to the whitepaper the comment should be:

```
//2 - 5% - 2,500,000
```

Using throw

In TigereumCrowdsale.sol, line 183 `throw` is used. This function is deprecated, we recommend to avoid using it.

Conclusion

In this report we have considered the security of Tigereum smart contracts. We performed our audit according to the procedure described above.
The audit showed several issues of different severity level. We highly recommend addressing them.

This analysis was performed by SmartDec

COO Sergey Pavlin 

December 1, 2017

Appendix

Code coverage

```
-----------------------|----------|----------|----------|----------|----------------|
File | % Stmts | % Branch | % Funcs | % Lines |Uncovered Lines |
-----------------------|----------|----------|----------|----------|----------------|
contracts/ | 47.25 | 36.54 | 63.16 | 48.54 | |
Tigereum.sol | 100 | 100 | 100 | 100 | |
TigereumCrowdsale.sol | 47.25 | 36.54 | 63.16 | 48.54 |... 234,235,275 |
-----------------------|----------|----------|----------|----------|----------------|
All files | 47.25 | 36.54 | 63.16 | 48.54 | |
-----------------------|----------|----------|----------|----------|----------------|
```

Compilation output

```
Compiling .\contracts\Migrations.sol...
Compiling .\contracts\Tigereum.sol...
Compiling .\contracts\TigereumCrowdsale.sol...
Compiling .\zeppelin-solidity\contracts\crowdsale\Crowdsale.sol...
Compiling .\zeppelin-solidity\contracts\math\SafeMath.sol...
Compiling .\zeppelin-solidity\contracts\ownership\Ownable.sol...
Compiling .\zeppelin-solidity\contracts\token\BasicToken.sol...
Compiling .\zeppelin-solidity\contracts\token\ERC20.sol...
Compiling .\zeppelin-solidity\contracts\token\ERC20Basic.sol...
Compiling .\zeppelin-solidity\contracts\token\MintableToken.sol...
Compiling .\zeppelin-solidity\contracts\token\StandardToken.sol...


Compilation warnings encountered:


/C/Users/user/work/sc/audit/Tigereum/tgICOv1/contracts/Migrations.sol:11:3: Warning: No
visibility specified. Defaulting to "public".
function Migrations() {
^
Spanning multiple lines.
,/C/Users/user/work/sc/audit/Tigereum/tgICOv1/contracts/Migrations.sol:15:3: Warning: No
visibility specified. Defaulting to "public".
function setCompleted(uint completed) restricted {
^
Spanning multiple lines.
,/C/Users/user/work/sc/audit/Tigereum/tgICOv1/contracts/Migrations.sol:19:3: Warning: No
visibility specified. Defaulting to "public".
function upgrade(address new_address) restricted {
^
Spanning multiple lines.
,/C/Users/user/work/sc/audit/Tigereum/tgICOv1/zeppelin-
solidity/contracts/ownership/Ownable.sol:20:3: Warning: No visibility specified. Defaulting to
"public".
function Ownable() {
^
Spanning multiple lines.
,/C/Users/user/work/sc/audit/Tigereum/tgICOv1/zeppelin-
solidity/contracts/crowdsale/Crowdsale.sol:43:3: Warning: No visibility specified. Defaulting
to "public".
function Crowdsale(uint256 _startTime, uint256 _endTime, uint256 _rate, address _wallet) {
^
Spanning multiple lines.
,/C/Users/user/work/sc/audit/Tigereum/tgICOv1/zeppelin-
solidity/contracts/crowdsale/Crowdsale.sol:64:3: Warning: No visibility specified. Defaulting
to "public".
function () payable {
^
Spanning multiple lines.
,/C/Users/user/work/sc/audit/Tigereum/tgICOv1/contracts/TigereumCrowdsale.sol:183:97: Warning:
"throw" is deprecated in favour of "revert()", "require()" and "assert()".
```

```
if(state == State.ShouldFinalize || state == State.Lockup || state == State.SaleOver) { throw;
}
^---^
,/C/Users/user/work/sc/audit/Tigereum/tgICOv1/zeppelin-
solidity/contracts/math/SafeMath.sol:9:3: Warning: Function state mutability can be restricted
to pure
function mul(uint256 a, uint256 b) internal constant returns (uint256) {
^
Spanning multiple lines.
,/C/Users/user/work/sc/audit/Tigereum/tgICOv1/zeppelin-
solidity/contracts/math/SafeMath.sol:18:3: Warning: Function state mutability can be
restricted to pure
function div(uint256 a, uint256 b) internal constant returns (uint256) {
^
Spanning multiple lines.
,/C/Users/user/work/sc/audit/Tigereum/tgICOv1/zeppelin-
solidity/contracts/math/SafeMath.sol:25:3: Warning: Function state mutability can be
restricted to pure
function sub(uint256 a, uint256 b) internal constant returns (uint256) {
^
Spanning multiple lines.
,/C/Users/user/work/sc/audit/Tigereum/tgICOv1/zeppelin-
solidity/contracts/math/SafeMath.sol:30:3: Warning: Function state mutability can be
restricted to pure
function add(uint256 a, uint256 b) internal constant returns (uint256) {
^
Spanning multiple lines.
```