SmartDec

Gladius Smart Contracts Security Audit

Abstract

In this audit we consider the security of the Gladius project. Our task is to find and describe security issues in the Smart Contracts of the platform.

Procedure

In our audit we consider Gladius whitepaper (version on 8 Oct 2017) and Smart Contracts code (version with latest commit 0a66dc2 on October 8th 2017).
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 and Securify (beta version since full version was unavailable at the moment this report was made)
  - 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 deploy contracts and run tests
- report
  - we report all the issues found to the developer during the audit process
  - we check the issues fixed by the developer
  - we reflect all the gathered information in the report

Checked vulnerabilities

We have scanned Gladius 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 x.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

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	Vulnerability	Confirmed issues	Rejected issues
SmartCheck	DOS with throw		2
	ERC20 API violation	2	
	Gas limit and loops		2
	Pragmas version	19	
	Private modifier	3	7
	send instead of transfer		3
	Timestamp dependence	1	16
	Use of tx.origin		1
	Implicit visibility level	35	
Oyente	Fallback requires too much gas	1	
	Gas requirement unknown or not constant		49
	Use of tx.origin		1
Remix	Fallback requires too much gas	1	
	Gas requirement unknown or not constant		49
	Is constant but potentially should not be		1
	No visibility specified	39	
	Potential Violation of Checks-Effects-Interaction	1	1
	Potentially should be constant but is not	1	6
	use of "now"	1	16
	use of "send"		3
	Use of tx.origin		1
	Variables have very similar names		109
Securify*	Transactions May Affect Ether Receiver	3	

Securify* — beta version, full version is unavailable.

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 we describe below.
We have found no serious vulnerabilities. However, the code contained some medium, minor and codestyle issues.

Medium issues

Medium issues can influence Smart Contracts operation in current implementation. We highly recommend addressing them.

Deployment process

In Gladius contracts correct operation crucially depends on deployment process. Our general recommendation is to leave less role for deploy by implementing as much as possible in code itself.
Contracts' logic strongly depends on several parameters that are defined in deployment script. Script injects parameters' values via constructors and special setters. Correctness of such deployment procedure must be ensured with tests.
We deployed contracts and checked deployment process. We ran no tests as they were not present on commit 0a66dc2.
We recommend checking values actually used to initialize contracts.

Bug in code: Crowdsale.sol, line 328

The comment states that the method is supposed to "get invested and refundable balance of _owner". However, the second and the third arguments of the ternary operator are swapped:

```
      return now > crowdsaleEnd && raised < minAmount ? 0 :
balances[_owner];
```

Thus, if `minAmount` was not raised, the owner will not be able to find out his refundable balance since the function will return 0 for any value of balance.

The issue was fixed by the developer and is not present on commit 7f1a941.

ERC20 API violation: Token.sol, line 79

According to ERC20 Token Standard, "transfers of 0 values MUST be treated as normal transfers and fire the Transfer event", but they do not. The `require` at line 85 prohibits transfers of 0 values:

```
      require(balances[_to] + _value > balances[_to]);
```

The issue was fixed by the developer and is not present on commit 7f1a941.

ERC20 API violation: Token.sol, line 105

According to ERC20 Token Standard, "transfers of 0 values MUST be treated as normal transfers and fire the Transfer event", but they don't. The `require` at line 111 prohibits transfers of 0 values:

```
require(balances[_to] + _value > balances[_to]);
```

The issue was fixed by the developer and is not present on commit 7f1a941.

Minor issues

Minor issues can influence smart contracts operation in future versions of code. We recommend considering them.

Warning: Crowdsale.sol, line 253

We recommend to add one more `require` to function `deploy()` to make sure that `setupVolumeMultipliers` was called.

Warning: Crowdsale.sol, line 264

The following line and the `confirmedBy` variable are not needed and can be removed since the value of `confirmedBy` is never checked in the contracts:

```
confirmedBy = msg.sender;
```

The issue was fixed by the developer and is not present on commit 7f1a941.

Warning: Crowdsale.sol, line 338

`getCurrentPhase` function returns `(bool found, uint phase)`. First return value `(bool found)` is never used and thus can be removed.

The issue was fixed by the developer and is not present on commit 7f1a941.

Warning: Crowdsale.sol, line 517

The `refund` function formally violates "Checks-Effects-Interactions" pattern.

```
balances[msg.sender] = 0;
if (receivedAmount > 0 && !msg.sender.send(receivedAmount)) {
     balances[msg.sender] = receivedAmount;
}
```

Some contract can fail to get refund and still get its balance set to zero. We recommend following "Checks-Effects-Interactions" pattern.

Potentially should be constant: GLACrowdsale.sol, line 68

Function `isAcceptedContributor` is actually constant but does not have `constant` modifier. We recommend to explicitly specify modifiers to increase code readability and avoid mistakes.

The issue was fixed by the developer and is not present on commit 7f1a941.

Fallback function requires too much gas: GLACrowdsale.sol

Fallback function is inherited from Crowdsale.sol, line 554 and is big. If the fallback function requires more than 2300 gas, the contract cannot receive Ether by some of standard ways. We recommend avoiding creating big fallback functions. However, since in this case it is needed, we recommend mentioning in API documentation that contract might not be able to accept Ether in some cases.

Warning: ManagedToken.sol, line 110

Function `issue` does not allow issuing zero tokens:

```
require(balances[_to] + _value > balances[_to]);
```

The contract developer might want to enable minting zero tokes. In addition, ERC20 recommendation will be followed (it is not required since the function is outside ERC20).

The issue was fixed by the developer and is not present on commit 7f1a941.

Codestyle issues

Codestyle issues influence code readability and in some cases may lead to bugs in future. We recommend considering them.

Timestamp dependence: Crowdsale.sol

`now` is used in `endCrowdsale()` function:

```
_allocateStakeholdersTokens(totalTokenSupply * p.percentage / percentageDenominator, now + p.vestingPeriod);
```

The timestamp of the block can be manipulated by the miner, and so should not be used for critical components of the contract. It is quite safe to use `block.timestamp` to change stages. Still, we recommend using `block.number` (and average block time) instead. Other contracts can use `block.number` to stage changes.
Besides, we recommend adding an event for stage change so that it will be more transparent for users.

Private modifier: Crowdsale.sol, lines 87, 88

Contrary to a popular misconception, the `private` modifier does not make a variable invisible. The developer should take into account, that `allocated` and `allocatedIndex` private variables will be visible.

Private modifier: GLACrowdsale.sol, line 24

Contrary to a popular misconception, the `private` modifier does not make a variable invisible. The developer should take into account, that `whitelist` will be visible.

Code quality: Crowdsale.sol, line 150

`isAcceptedContributor` method is not implemented in Crowdsale.sol; so Crowdsale is neither “interface” nor “contract”.

Code quality: Crowdsale.sol, line 221

We recommend using `i` instead of `stakeholderPercentagesIndex.push(_stakeholders[i]) - 1`.

The issue was fixed by the developer and is not present on commit 7f1a941.

Implicit visibility level

In many places in the code, there are functions with implicit visibility level. We recommend specifying visibility level explicitly or following strict notation on when to specify it and when not to.

The issue was fixed by the developer and is not present on commit 7f1a941.

Conclusion

In this report, we have considered the security of Gladius Smart Contracts. We performed our audit according to the procedure described above.
We checked Smart Contracts logic and compared it with the one described in the whitepaper. No discrepancies were found.
The audit showed high code quality and security of the project. No serious vulnerabilities were found. However, few medium, minor and codestyle issues were found and reported to the developer. In commit 7f1a941 most of them (and all of the important ones) were fixed.

This audit was performed by SmartDec.

COO Sergey Pavlin 

October 13, 2017