SmartDec

PumaPay Smart Contracts Security Analysis

This report is public.
Report version 1.1
Published: May 03, 2018

Abstract

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

Procedure

In our audit, we consider the following crucial features of the smart contract code:

1. Whether the code is secure.
2. Whether the code corresponds to the documentation (including whitepaper).
3. Whether the code meets best practices in efficient use of gas, code readability, etc.

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 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 check ERC20 compliance
  - we run tests and check code coverage
- report
  - we report all the bugs to the developer during the audit
  - we check the issues fixed by the developer
  - 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 with several independent audits and a public bug bounty program to ensure the security of smart contracts. Besides, security audit is not an investment advice.

Checked vulnerabilities

We have scanned PumaPay 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 (Unexpected) revert
- 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
- Address hardcoded
- Using delete for arrays
- Integer overflow/underflow
- Locked money
- Private modifier
- Revert/require functions
- Using var
- Visibility
- Using blockhash
- Using SHA3
- Using suicide
- Using throw
- Using inline assembly

Project overview

Project description

In our analysis we consider PumaPay whitepaper (pumapay_whitepaper.pdf, sha1sum c3873a5c4eb1cf58ebbda6b7dc6f407df45c2fbd) and smart contracts code (version on commit ec178ec).

The latest versions of the code and the whitepaper

We have performed the check of the fixed issues in the latest version of the code (version on commit b55fa75) and the latest version of the whitepaper (pumapay_whitepaper.pdf, sha1sum 2c03c39259fecf9a1f5693cbd22ba159a0349f5b).
The whitepaper has changed significantly since the time of the initial audit. We have performed only the check of the issues found before, not the full check of the whitepaper.

Project architecture

For the audit, we have been provided with the following set of solidity files:

- PumaPayToken.sol – MintableToken contract from OpenZeppelin library with additional functionality

The project contains 19 lines of Solidity code. The files are the part of the truffle project and an npm package. The project also contains tests.

- The project successfully compiles with `truffle compile` command (see Compilation output in Appendix).
- The project successfully passes all the tests (`truffle test` command, see Tests output in Appendix).

Code logic

PumaPayToken is ERC20 compatible MintableToken contract with modified functionality: transfers (both `transfer` and `transferFrom` functions are allowed only after minting is finished, i.e. after `PumaPayToken.finishMinting()` function is manually called by the contract's owner).
The token has following parameters:

- token name: "PumaPayToken"
- token symbol: "PUM"
  token symbol has been changed to "PMA" in the latest version of the code
- token decimals: 18

Automated analysis

We used several publicly available automated Solidity analysis tools. Here are the combined results of SmartCheck, Solhint, and Remix. Oyente has found no issues.
All the issues found by tools were manually checked (rejected or confirmed). False positives are constructions that were discovered by the tools as vulnerabilities but do not consist a security threat. True positives are constructions that were discovered by the tools as vulnerabilities and can actually be exploited by attackers or lead to incorrect contracts operation. Cases when these issues lead to actual bugs or vulnerabilities are described in the next section.

Tool	Rule	False positives	True positives
Remix	Gas requirement of function high	17	
	Potentially should be constant but is not	3	
	Variables have very similar names		1
Total Remix		20	1
SmartCheck	Erc20 Transfer Should Throw	2	
	No Payable Fallback	1	
Total SmartCheck		3	0
Total Overall		**23**	**1**

Manual analysis

The contracts were completely manually analyzed, their logic was checked and compared with the one described in the documentation. Besides, the results of the 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.

The audit showed no critical issues.

Medium severity issues

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

Discrepancies with the whitepaper

1. According to the whitepaper (page 23):

   ```
   "The token will initially be developed as an ERC223 compatible
   token over the public Ethereum blockchain (PMA V1.0)."
   ```

   The issue has been fixed by the developer and is not present in the latest version of the whitepaper.
   However, PMA is ERC20 compatible token and not ERC223 compatible token.
2. According to the whitepaper (page 23):

   ```
   "symbol PMA"
   ```

   However, in contract's code:

   ```
   string public symbol = "PUM";
   ```

   The issue has been fixed by the developer and is not present in the latest version of the code.
3. According to the whitepaper (page 24):

   ```
   "PumaPay Tokens will be available for purchase on our sale
   site https://ico.PumaPay.io commencing April 26th, 2018 at
   12:00:00 am (UTC) and for a period of 7 days."
   ```

   However, minting is available until `finishMinting()` function is manually called.
   The issue has been fixed by the developer and is not present in the latest version of the whitepaper.

We recommend either improving contract's functionality so that it matches the whitepaper or modifying the whitepaper in order to avoid any discrepancies.

Low severity issues

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

ERC20 standard violation

According to the ERC20 standard, `decimals` should be `uint8`. However, in the code it is `uint` (PumaPayToken.sol, line 12):

```
uint public decimals = 18;
```

We highly recommend strictly following the standard.
The issue has been fixed by the developer and is not present in the latest version of the code.

Not implemented functionality

The following functionality described in the whitepaper has not been implemented:

1. The PumaPay Token will be sold at a fixed price denominated in ETH.
   The issue has been fixed by the developer and is not present in the latest version of the whitepaper.
2. The total number of tokens will depend on the quantity of PumaPay Tokens sold.
   The issue has been fixed by the developer and is not present in the latest version of the whitepaper.
3. Our TGE will be soft-capped at 20K ETH (Twenty Thousand Ether) and hard-capped at 150M USD (One Hundred and Fifty Million US Dollars).
   The issue has been fixed by the developer and is not present in the latest version of the whitepaper.
4. During the TGE, we will offer a special 5% (Five Percent) bonus to a person/entity who contributed at least 6 ETH (Six Ether).
   The issue has been fixed by the developer and is not present in the latest version of the whitepaper.
5. The PumaPay Token Contract will be the ledger used by PumaPay to keep track of the balances associated with each address, as well as keeping track of PullContracts and Limiters associated with the Accounts.
   The issue has been fixed by the developer and is not present in the latest version of the whitepaper.

We highly recommend improving smart contract’s code by adding this functionality in new versions.

Conclusion

In this report we have considered the security of PumaPay smart contracts. We performed our audit according to the procedure described above.
The initial audit showed no critical issues. However, several medium and low severity issues were found. All the issues has been fixed by the developer in the latest versions of the code and the whitepaper.

This analysis was performed by SmartDec.

Igor Sobolev, Analyst
Evgeniy Marchenko, Lead Developer
Alexander Seleznev, Chief Business Development Officer
Ivan Ivanitskiy, Chief Analytics Officer
Alexander Drygin, Analyst

Sergey Pavlin, Chief Operating Officer 

May 03, 2018

Appendix

Code coverage

```
------------------|----------|----------|----------|----------|----------------|
File              |  % Stmts | % Branch |  % Funcs |  % Lines |Uncovered Lines |
------------------|----------|----------|----------|----------|----------------|
 contracts/       |      100 |      100 |      100 |      100 |                |
  PumaPayToken.sol |      100 |      100 |      100 |      100 |                |
------------------|----------|----------|----------|----------|----------------|
All files         |      100 |      100 |      100 |      100 |                |
------------------|----------|----------|----------|----------|----------------|
```

Compilation output

```
$ truffle compile
Compiling .\contracts\Migrations.sol...
Compiling .\contracts\PumaPayToken.sol...
Compiling .\node_modules\zeppelin-
solidity\contracts\math\SafeMath.sol...
Compiling .\node_modules\zeppelin-
solidity\contracts\ownership\Ownable.sol...
Compiling .\node_modules\zeppelin-
solidity\contracts\token\ERC20\BasicToken.sol...
Compiling .\node_modules\zeppelin-
solidity\contracts\token\ERC20\ERC20.sol...
Compiling .\node_modules\zeppelin-
solidity\contracts\token\ERC20\ERC20Basic.sol...
Compiling .\node_modules\zeppelin-
solidity\contracts\token\ERC20\MintableToken.sol...
Compiling .\node_modules\zeppelin-
solidity\contracts\token\ERC20\StandardToken.sol...
Writing artifacts to .\build\contracts
```

Tests output

```
Contract: StandardToken
    total supply
```

```
      √ returns the total amount of tokens
    balanceOf
      when the requested account has no tokens
        √ returns zero
      when the requested account has some tokens
        √ returns the total amount of tokens
    transfer
      when the recipient is not the zero address
        when the sender does not have enough balance
          √ reverts
        when the sender has enough balance
          √ transfers the requested amount
          √ emits a transfer event
      when the recipient is the zero address
        √ reverts

  Contract: DetailedERC20
    √ has a name
    √ has a symbol
    √ has an amount of decimals

  Contract: Mintable
    minting finished
      when the token is not finished
        √ returns false
      when the token is finished
        √ returns true
    finish minting
      when the sender is the token owner
        when the token was not finished
          √ finishes token minting
          √ emits a mint finished event
        when the token was already finished
          √ reverts
      when the sender is not the token owner
        when the token was not finished
          √ reverts
        when the token was already finished
          √ reverts
    mint
      when the sender is the token owner
        when the token was not finished
          √ mints the requested amount
          √ emits a mint finished event
        when the token minting is finished
          √ reverts
      when the sender is not the token owner
```

```
        when the token was not finished
          √ reverts
        when the token was already finished
          √ reverts

  Contract: Ownable
    √ should have an owner
    √ changes owner after transfer
    √ should prevent non-owners from transfering
    √ should guard ownership against stuck state

  Contract: PumaPayToken
    Deploying
      √ Token name should be PumaPayToken
      √ Token symbol should be PUM
      √ Decimals is set to 18
      √ Owner is set
      √ Minting is enabled uppon deployment
      √ Initial total supply is empty
    Minting functionalities
      √ Owner can mint
      √ Rejects notowner from minting
      √ Minting is cumulative
      √ Owner can call finishMinting
      √ Rejects notowner from calling finishMinting
      √ Calling finishMinting changes the mintingFinished to true
      √ Rejects minting attepts after minting finished
    Token functionalities during minting stage
      √ Can't transfer
    Token functionalities after minting stage
      √ Can transfer
      √ Can approve
      √ Can transfer from
      √ Can increase approve
      √ Can decrease approve

  Contract: SafeMath
    √ multiplies correctly
    √ adds correctly
    √ subtracts correctly
    √ should throw an error if subtraction result would be negative
    √ should throw an error on addition overflow
    √ should throw an error on multiplication overflow

  Contract: StandardToken
    total supply
      √ returns the total amount of tokens
```

```
    balanceOf
      when the requested account has no tokens
        √ returns zero
      when the requested account has some tokens
        √ returns the total amount of tokens
    transfer
      when the recipient is not the zero address
        when the sender does not have enough balance
          √ reverts
        when the sender has enough balance
          √ transfers the requested amount
          √ emits a transfer event
      when the recipient is the zero address
        √ reverts
    approve
      when the spender is not the zero address
        when the sender has enough balance
          √ emits an approval event
          when there was no approved amount before
            √ approves the requested amount
          when the spender had an approved amount
            √ approves the requested amount and replaces the
previous one
        when the sender does not have enough balance
          √ emits an approval event
          when there was no approved amount before
            √ approves the requested amount
          when the spender had an approved amount
            √ approves the requested amount and replaces the
previous one
      when the spender is the zero address
        √ approves the requested amount
        √ emits an approval event
    transfer from
      when the recipient is not the zero address
        when the spender has enough approved balance
          when the owner has enough balance
            √ transfers the requested amount
            √ decreases the spender allowance
            √ emits a transfer event
          when the owner does not have enough balance
            √ reverts
        when the spender does not have enough approved balance
          when the owner has enough balance
            √ reverts
          when the owner does not have enough balance
            √ reverts
```

```
      when the recipient is the zero address
        √ reverts
    decrease approval
      when the spender is not the zero address
        when the sender has enough balance
          √ emits an approval event
          when there was no approved amount before
            √ keeps the allowance to zero
          when the spender had an approved amount
            √ decreases the spender allowance subtracting the
requested amount
        when the sender does not have enough balance
          √ emits an approval event
          when there was no approved amount before
            √ keeps the allowance to zero
          when the spender had an approved amount
            √ decreases the spender allowance subtracting the
requested amount
      when the spender is the zero address
        √ decreases the requested amount
        √ emits an approval event
    increase approval
      when the spender is not the zero address
        when the sender has enough balance
          √ emits an approval event
          when there was no approved amount before
            √ approves the requested amount
          when the spender had an approved amount
            √ increases the spender allowance adding the requested
amount
        when the sender does not have enough balance
          √ emits an approval event
          when there was no approved amount before
            √ approves the requested amount
          when the spender had an approved amount
            √ increases the spender allowance adding the requested
amount
      when the spender is the zero address
        √ approves the requested amount
        √ emits an approval event

  89 passing (17s)
```

Solhint output

```
Migrations.sol
   1:17  warning  Compiler version must be
fixed                            compiler-fixed
   3:1   error    Definition must be surrounded with two blank line
indent  two-lines-top-level-separator
  23:30  error    Function param name must be in
mixedCase                   func-param-name-mixedcase

PumaPayToken.sol
  14:36  warning  Code contains empty block                      no-
empty-blocks
  28:1   error    Mixed tabs and spaces. Allowed only spaces     no-
mix-tabs-and-spaces
  28:2   error    Expected indentation of 8 spaces but found
1  indent
```