SmartDec

PumaPay Smart Contracts Security Analysis

This report is private.

Published: October 11, 2019.

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.

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot 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.

Summary

In this report, we considered the security of PumaPay smart contracts. We performed our audit according to the procedure described below.

The audit showed no critical issues. However, one medium severity and a number of low severity issues were found. They do not endanger project security. Nevertheless, we highly recommend addressing them.

Most of the issues were fixed in the latest version of the code.

General recommendations

The contracts code is of high code quality. Thus, we do not have any additional recommendations.

Checklist

Security

The audit showed no vulnerabilities.
Here by vulnerabilities we mean security issues that can be exploited by an external attacker. This does not include low severity issues, documentation mismatches, overpowered contract owner, and some types of bugs.

✓

Compliance with the documentation

The audit showed no discrepancies between the code and the provided documentation.

✓

Tests

The audit showed that the code was covered with tests sufficiently.

✓

The text below is for technical use; it details the statements made in Summary and General recommendations.

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 Ethlint 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 documentation
  - we run tests and check code coverage
- report
  - we reflect all the gathered information in the report

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
- Front running
- DoS with (unexpected) revert
- DoS with block gas limit
- Gas limit and loops
- Locked money
- Integer overflow/underflow
- Unchecked external call
- ERC20 Standard violation
- Authentication with tx.origin
- Unsafe use of timestamp
- Using blockhash for randomness
- Balance equality
- Unsafe transfer of ether
- Fallback abuse
- Using inline assembly
- Short address attack
- Private modifier
- Compiler version not fixed
- Style guide violation
- Unsafe type deduction
- Implicit visibility level
- Use delete for arrays
- Byte array
- Incorrect use of assert/require
- Using deprecated constructions

Project overview

Project description

In our analysis we consider PumaPay specification (”Top Up Pull Payment.md”, sha1sum: d6d332c4313da493693cef09efbad6e58b4c1213) and smart contracts’ code (version on commit 7970e3179bee4fa6d8c62bb41620e3509b9c5e4a).

The latest version of the code

After the initial audit, some fixes were applied and the code was updated to the latest version (commit 18b4d9cd8b5150b28e688e6aa8c936f302842423). Also, the specification was updated to the latest version.

Project architecture

For the audit, we were provided with the truffle project. The project is an npm package and includes tests.

- The project successfully compiles with `truffle compile` command (see Compilation output in Appendix)
- The project successfully passes all the tests with sufficient code coverage

Scope of work

Only **TopUpPullPayment.sol** file was audited.

The total LOC of audited Solidity sources is 353.

Automated analysis

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

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.

False positives are constructions that were discovered by the tools as vulnerabilities but do not consist a security threat.

Cases when these issues lead to actual bugs or vulnerabilities are described in the next section.

Tool	Rule	True positives	False positives
Solhint	Avoid to make time-based decisions in your business logic		8
Total Solhint		0	8
Ethlint	Avoid using ’now’ (alias to ’block.timestamp’).		8
Total Ethlint		0	8
SmartCheck	Prefer external to public visibility level	7	
	Use of SafeMath		1
Total SmartCheck		7	1
Total Overall		**7**	**17**

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 the 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.

Overpowered role

In the smart contract, users with `executor` role have the following power: they choose `_conversionRate` value when calling `executeTopUpPayment()` function.

In the current implementation, payment values depend on executors of the contract. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g. if one of the executors is malicious or if the owner's private keys become compromised. Thus, we recommend designing contracts in a trustless manner.

Comment from the developers: "The executors of the transaction lifecycle are addresses owned by PumaPay and thus we are committed to ensure that we deliver on all registrations, cancellations and executions of all our Pull Payments. It is important to note that we are developing a set of smart contracts with the aim of removing the "overpowered owner nature of our current smart contracts."

Low severity issues

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

Unchecked return value (fixed)

In the code, there are two functions that ignore return value of `transferFrom()` function:

- `executeTopUpPayment()` function, line 324

```
token.transferFrom(payment.customerAddress,
payment.treasuryAddress, amountInPMA);
```

- `executePullPaymentOnRegistration()` function, line 417

```
token.transferFrom(_addresses[0], _addresses[1],
amountInPMA);
```

Despite the fact that according to the ERC20 token standard

```
The function SHOULD throw if the message caller's account
balance does not have enough tokens to spend.
```

not all tokens revert transaction in case of unsuccessful transfer. This case is mentioned in the notes to the standard:

```
Callers MUST handle false from returns (bool success).
Callers MUST NOT assume that false is never returned!
```

We recommend checking return values by using the `require()` function.

The issues have been fixed and are not present in the latest version of the code.

Prefer external to public visibility level (fixed)

In the code there are functions with the `public` visibility level that are not called internally:

- `addExecutor()` function
- `removeExecutor()` function
- `registerTopUpPayment()` function
- `executeTopUpPayment()` function
- `cancelTopUpPayment()` function
- `updateTotalLimit()` function
- `retrieveLimits()` function

We recommend changing visibility level of such functions to `external` in order to improve code readability. Moreover, in many cases functions with `external` visibility modifier require less gas comparing to functions with `public` visibility modifier.

The issues have been fixed and are not present in the latest version of the code.

Redundant code (fixed)

The following lines are redundant:

- **TopUpPullPayment.sol**, line 259

  ```
  bytes32[2] memory paymentIDs = _paymentIDs;
  ```

  The line is redundant since `_paymentIDs` array is already stored in memory.

- TopUpPullPayment.sol, lines 59, 61

  ```
  uint256 constant internal FIAT_TO_CENT_FIXER = 100;
  ...
  uint256 constant internal ONE_ETHER = 1 ether;
  ```

  These state variables are unused.

We highly recommend removing redundant code in order to improve code readability and transparency and decrease cost of deployment and execution.

The issues have been fixed and are not present in the latest version of the code.

Code logic (fixed)

When top-up payment is registered, the initial pull payment is executed. However, this payment does not count in `totalSpent` limit.

We recommend either changing code logic or clarifying it in the documentation.

The issue has been fixed and is not present in the latest version of the code.

This analysis was performed by SmartDec.

Sergei Pavlin, Chief Operating Officer
Igor Sobolev, Analyst
Pavel Kondratenkov, Analyst

October 11, 2019

Appendix

Compilation output

```
Compiling your contracts...
===========================
> Compiling ./contracts/Migrations.sol
> Compiling ./contracts/PumaPayPullPayment.sol
> Compiling ./contracts/PumaPayPullPaymentV2.sol
> Compiling ./contracts/SinglePullPayment.sol
> Compiling ./contracts/access/roles/ExecutorRole.sol
> Compiling ./contracts/ownership/PayableOwnable.sol
> Compiling ./contracts/topUp/TopUpPullPayment.sol
> Compiling ./contracts/topUp/TopUpPullPaymentWithExpiration
.sol
> Compiling ./contracts/topUp/TopUpTimeBasedPullPayment.sol
> Compiling ./contracts/topUp/TopUpTimeBasedPullPaymentWithE
xpiration.sol
> Compiling openzeppelin-solidity/contracts/access/Roles.sol
> Compiling openzeppelin-solidity/contracts/math/SafeMath.so
l
> Compiling openzeppelin-solidity/contracts/token/ERC20/IERC
20.sol
> Artifacts written to ./build/contracts
> Compiled successfully using:
- solc: 0.5.10+commit.5a6ea5b1.Emscripten.clang
```

Tests output

```
  Contract: Top Up Pull Payment Smart Contract
Deploying
 PumaPay Pull Payment owner should be the address that was s
pecified on contract deployment
 PumaPay Pull Payment token should be the token address spec
ified on contract deployment
 PumaPay Pull Payment deployment should revert when the toke
n is a ZERO address
Register a top up pull payment
 should add the top up payment in the smart contract mapping
 should transfer the PMA for the initial payment
 should revert if not called by one of the executors
 should revert if you try to register the same payment twice
 should revert if paymentID is empty
 should revert if businessID is empty
 should revert if customer address is ZERO ADDRESS
 should revert if pull payment executor address is ZERO ADDR
ESS
 should revert if treasury address is ZERO ADDRESS
 should revert if initial conversion rate is zero
 should revert if initial conversion rate too high
 should revert if initial payment amount is zero
 should revert if initial payment amount too high
 should revert if top up amount is zero
 should revert if top up amount too high
 should revert if start timestamp is zero
 should revert if start timestamp too high
 should revert if total limit is zero
 should revert if total limit is too high
 should revert if the currency is empty
 should revert if the signature from the customer doesn't ma
tch
 should emit a "LogPaymentRegistered" event
 should emit a "LogPullPaymentExecuted" event
Execute top up pull payment
 should transfer PMA to the treasury wallet
 should update the last payment timestamp
 should update the total spent amount
 should execute even if called after 100 years
 should revert if not called by the pull payment executor
 should revert if the payment does not exists
 should revert if conversion rate is too high - OVERFLOW LIM
ITS
 should revert if conversion rate is ZERO
 should emit a "LogPullPaymentExecuted" event
Execute a top up pull payment - TOTAL LIMITS
 should fail to execute the top up if the total limits have
been reached
Cancel a top up pull payment
 should update the cancel timestamp
 should not allow for a top up to be executed after a paymen
t is cancelled
 should revert if the payment doesn't exist
 should revert if the payment is already cancelled
 should revert if not called by an executor
 should emit a "LogPaymentCancelled" event
```

```
Updating total limits for payment
 should update the total limit for the payment
 should be able to execute a pull payment when the total lim
it is increased
 should not allow for a pull payment to be executed when the
limit is decreased
 should revert if not executed from the customer for that pa
yment
 should revert if the number is zero
 should revert if the number is higher than the overflow lim
it
 should revert if the number is lower or equal to the amount
spent
 should emit "LogTotalLimitUpdated" event
Retrieve limits
 should return the total limit for the top up billing model
based on the payment ID
 should return the total spent for the top up billing model
based on the payment ID
 should return ZERO values if the payment doesn't exists
Executors Funding
 should fund the executor on registration
 should emit a "LogSmartContractActorFunded" event when the
executor is funded on registration
 should fund the executor on cancellation
 should emit a "LogSmartContractActorFunded" event when the
executor is funded on cancellation
 should fund the owner on adding a new executor
 should fund the new executor when is low in ETH
 should emit a "LogSmartContractActorFunded" event when the
owner is funded on adding new executor
 should emit a "LogSmartContractActorFunded" event when the
executor is funded on adding new executor
 should fund the owner on removing an executor
 should emit a "LogSmartContractActorFunded" event when the
owner is funded on removing an executor
```

Solhint output

```
contracts/topUp/TopUpPullPayment.sol
146:8     warning    Line exceeds the limit of 145 character
s.
```

```
180:68    warning    Avoid using 'now' (alias to 'block.time
stamp').
185:63    warning    Avoid using 'now' (alias to 'block.time
stamp').
201:63    warning    Avoid using 'now' (alias to 'block.time
stamp').
274:16    error      Only use indent of 12 spaces.
275:16    error      Only use indent of 12 spaces.
276:16    error      Only use indent of 12 spaces.
277:16    error      Only use indent of 12 spaces.
278:16    error      Only use indent of 12 spaces.
291:69    warning    Avoid using 'now' (alias to 'block.time
stamp').
318:51    warning    Avoid using 'now' (alias to 'block.time
stamp').
321:69    warning    Avoid using 'now' (alias to 'block.time
stamp').
354:39    warning    Avoid using 'now' (alias to 'block.time
stamp').
413:39    warning    Avoid using 'now' (alias to 'block.time
stamp').

 5 errors, 9 warnings found.
```

Ethlint output

```
./contracts/topUp/TopUpPullPayment.sol
10:2  error  Line length must be no more than 120 but curren
t length is 121  max-line-length
12:2  error  Line length must be no more than 120 but curren
t length is 132  max-line-length
13:2  error  Line length must be no more than 120 but curren
t length is 154  max-line-length
16:2  error  Line length must be no more than 120 but curren
t length is 128  max-line-length
17:2  error  Line length must be no more than 120 but curren
t length is 129  max-line-length
59:2  error  Line length must be no more than 120 but curren
t length is 129  max-line-length
61:2  error  Line length must be no more than 120 but curren
t length is 137  max-line-length
62:2  error  Line length must be no more than 120 but curren
```

```
t length is 124  max-line-length
63:2  error  Line length must be no more than 120 but curren
t length is 124  max-line-length
126:2  error  Line length must be no more than 120 but curre
nt length is 123  max-line-length
130:2  error  Line length must be no more than 120 but curre
nt length is 122  max-line-length
146:2  error  Line length must be no more than 120 but curre
nt length is 165  max-line-length
209:2  error  Line length must be no more than 120 but curre
nt length is 132  max-line-length
217:2  error  Line length must be no more than 120 but curre
nt length is 123  max-line-length
299:2  error  Line length must be no more than 120 but curre
nt length is 134  max-line-length
330:2  error  Line length must be no more than 120 but curre
nt length is 132  max-line-length
332:2  error  Line length must be no more than 120 but curre
nt length is 121  max-line-length
336:2  error  Line length must be no more than 120 but curre
nt length is 131  max-line-length
337:2  error  Line length must be no more than 120 but curre
nt length is 132  max-line-length
397:2  error  Line length must be no more than 120 but curre
nt length is 123  max-line-length
427:2  error  Line length must be no more than 120 but curre
nt length is 121  max-line-length
438:2  error  Line length must be no more than 120 but curre
nt length is 123  max-line-length
445:2  error  Line length must be no more than 120 but curre
nt length is 123  max-line-length

 23 problems (23 errors, 0 warnings)
```