Published in SmartDec Cybersecurity Blog

Valerie Kim Follow
Oct 23, 2019 · 7 min read · Listen

PumaPay Smart Contracts Security Analysis

SmartDec

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

The initial 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.

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

General recommendations

The contracts code is of medium code quality. The audit did not reveal any issues that endanger project security.

In addition, if the developers decide to improve the code, we recommend fixing Coverage issue. However, mentioned above is minor issue. It does not affect code operation.

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 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.

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


- 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 (docs folder from repository) and smart contracts' code (version on commit 5eb99b1a94d9e5d98873fb4338b97943b9821569).

The latest version of the code

After the initial audit, some fixes were applied and the code was updated to the latest version (commit c248be94bb00ac19eb0d53629a4698f86da3d3f9).

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

Automated analysis

We used several publicly available automated Solidity analysis tools. Here are the combined results of SmartCheck, Solhint, and Remix scanning. All the issues found by tools were manually checked (rejected or confirmed). Cases when these issues lead to actual bugs or vulnerabilities are described in the next section.

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 owner

PumaPayPullPayment.sol, line 212:

```
function setRate(string memory _currency, uint256 _rate)
```

This function allows the owner of the contract to change the exchange rate at any time. Thus, the owner can change the rate when a user's payment is in process, which can lead to unexpected outcome of the payment for the user. **PumaPayPullPaymentV2.sol** contract also contains this issue, line 428:


Executor of `pullPayment` can change the rate while the payment is in process.

Moreover, since executors register and execute pull payments, they can postpone or ignore certain payments processing.

In the current implementation, the system depends heavily on the owner of the contract. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g. if the owner's private keys become compromised. Thus, we recommend designing contracts in a trustless manner.

> *Comment from the developers: "PumaPay always places our merchants-first approach in regards to our Pull Payment Protocol. When listing their products/services for purchase, merchants set their prices in fiat currency and not in PMA. It is a necessity to have the conversion rate placed on the smart contract, to ensure both merchant and customer are in agreement on the PMA price following conversion. As the initial PMA price is not listed by the merchant, the conversion rate placed on the smart contract, gives the final price in PMA that the customer will be paying. In our V1 smart contracts, the conversion rate is a global variable that we refresh every 10 minutes, however in the V2 smart contracts, the conversion rate will be produced live at the moment of execution giving the most accurate rate available. Currently, the executors of the transaction lifecycle are addresses owned by PMA 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.

Incorrect check (fixed)

There are incorrect checks:

- **PumaPayPullPayment.sol,** lines 130, 263, 264:


- **PumaPayPullPaymentV2.sol**, lines 147, 163, 264–266:

```
require(_paymentID.length != 0, "Invalid deletion request - Payment
ID is empty.");
...
require(_paymentType.length > 0, "Payment Type is empty.");
...
SmartDec Smart Contracts Security Analysis
https://smartcontracts.smartdec.net 9
require(_paymentDetails[0].length > 0, "Payment ID is empty.");
require(_paymentDetails[1].length > 0, "Business ID is empty.");
require(_paymentDetails[2].length > 0, "Unique Reference ID is
empty.");
```

The length of `bytes32` type is always 32. Therefore, we recommend implementing the following check instead:

```
bytes32 constant internal EMPTY_BYTES32 = "";
...
require(someVariable != EMPTY_BYTES32);
```

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

Coverage

The coverage script provided with the code failed. Testing is crucial for the code security. We highly recommend fixing the code so that the coverage script will run successfully.

Misleading comments (fixed)

The following comments are misleading:

1. According to the comment (**PumaPayPullPayment.sol**, line 535):

```
/// The minimum amount the owner/executors should always have is
```


```
uint256 constant private MINIMUM_AMOUNT_OF_ETH_FOR_OPERATORS = 0.15
ether; /// min amount of ETH for owner/executor
```

2. According to the comment (**PumaPayPullPaymentV2.sol**, line 607):

```
/// The minimum amount the owner/executors should always have is
0.001 ETH
```

However, it is 0.15 ETH, **PumaPayPullPaymentV2.sol**, line 56:

```
uint256 constant private MINIMUM_AMOUNT_OF_ETH_FOR_OPERATORS = 0.15
ether; /// min amount of ETH for owner/executor
```

We recommend fixing these comments in order to avoid confusion and improve code readability.

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

Excessive gas consumption (fixed)

PumaPayPullPayment.sol, lines 380–384:

```
if (pullPayments[_customer][msg.sender].initialPaymentAmountInCents
> 0)
{
     amountInPMA = calculatePMAFromFiat(
     pullPayments[_customer]
[msg.sender].initialPaymentAmountInCents,
     pullPayments[_customer]   [msg.sender].currency );
```

`pullPayments[_customer][msg.sender].initialPaymentAmountInCents` variable is read twice. We recommend reading this variable once and using addition `memory` variable in both places in order to reduce gas consumption.

 

There are no events emitted when ether is sent to the owner or to an executor. We recommend adding such an event and emitting it in all appropriate places.

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

Redundant check (fixed)

There is a redundant check, **PumaPayPullPaymentV2.sol**, line: 163–171:

```
require(_paymentType.length > 0, "Payment Type is empty."); require(
 (
     _paymentType == TYPE_SINGLE_PULL_PAYMENT ||
     _paymentType == TYPE_RECURRING_PULL_PAYMENT ||
     _paymentType == TYPE_RECURRING_PULL_PAYMENT_WITH_INITIAL ||
     _paymentType == TYPE_PULL_PAYMENT_WITH_FREE_TRIAL ||
     _paymentType == TYPE_PULL_PAYMENT_WITH_PAID_TRIAL),
 "Payment Type provided not supported");
```

The first check is then duplicated in the second check. Thus, the first `require` statement is redundant and we recommend removing it.

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

Unused import (fixed)

There are unused imports:

- **PumaPullPayment.sol**, line 6

```
import
"openzeppelinsolidity/contracts/token/ERC20/ERC20Mintable.sol";
```

ERC20Mintable contract is imported, however it is not used anywhere in the code.

- **PumaPayPullPaymentV2.sol**, line 4


We recommend removing unused imports in order to improve code readability.

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

> This article was created by SmartDec, a security team specialized in static code analysis, decompilation and secure development.
>
> Feel free to use SmartCheck, our smart contract security tool for Solidity and Vyper, and follow us on Medium, Telegram and Twitter. We are also available for smart contract development and auditing work.

SmartDec

More from SmartDec Cybersecurity Blog Follow

Security tutorials, tools, and ideas

Polina Grimalo · Oct 17, 2019

SmartDec: Digitex Futures Exchange Development Status Report #8

Blockchain 2 min read

Polina Grimalo · Oct 3, 2019

SmartDec: Digitex Futures Exchange Development Status Report #7

Tornado Mixer Security Audit

Ethereum 3 min read

Polina Grimalo · Sep 19, 2019

SmartDec: Digitex Futures Exchange Development Status Report #6

Blockchain 3 min read

Yelizaveta Kharlamova · Sep 13, 2019

SmartDec Scanner 3.2.0 Release Notes

Smartdec Scanner 3 min read

Read more from SmartDec Cybersecurity Blog

Recommended from Medium

Rebecca Iannilli

Guide to choosing the best security lock

Fox Pass

Explain tips to achieve business goals with zero trust security?

PrivacySwap

PrivacyCard giveaway| Join now and get a chance to win over $3000 worth of PrivacyCards|

Knownsec Blockchain Lab

Knownsec Blockchain Lab | Poly Network cross-chain project security incident

Wu-Tang-TRAM

Hotspots: When that hotspot is not so hot!

David Maximilian Güth in Cryption Network

Cryption Network: Weekly Update (25th October—31st October)

Cloudflare in Cloudflare

Introducing the Cloudflare Geo Key Manager