Smart Contract Secondary Code Review and Security Analysis Report

Customer: PumaPay
Date: September 21, 2018

HACKEN

HACKEN CYBERSECURITY ECOSYSTEM.

Document:

Name	Smart Contract Code Review and Security Analysis Report for PumaPay
Platform	Ethereum / Solidity
Link	https://github.com/pumapayio/pumapay-token/blob/master/contracts/MasterPullPayment.sol
Date of first audit	12.09.2018
Version of first audit	badb0ec7b30821f3d34f8acab2876ac065dcc238
Date of secondary audit	21.09.2018
Version of secondary audit	3ec35a84163436fb9fedce30622546206c01cf0a

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by PumaPay (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of Customer`s smart contract and its code review conducted between September 4th, 2018 – September 12th, 2018. Secondary audit was conducted between September 18th, 2018 – September 21th, 2018.

Scope

The scope of the project is PumaPay smart contract, which can be found at Github by the link below:

https://github.com/pumapayio/pumapay-token/blob/master/contracts/MasterPullPayment.sol

Commit version – 3ec35a84163436fb9fedce30622546206c01cf0a

We have scanned this smart contract for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that are 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
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

According to the assessment, Customer`s smart contracts are secure.

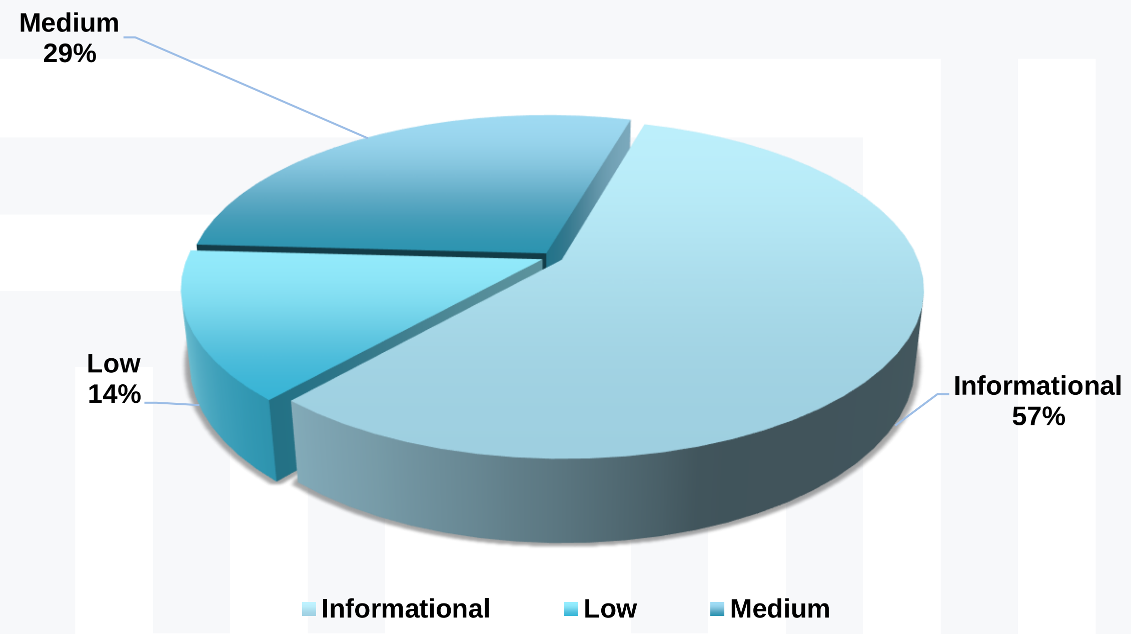
Insecure	Poor secured	Secured	Well-secured

You are here

Our team has performed analysis of code functionality, manual audit and automated checks with solc, Mythril, Slither and remix IDE (see Appendix B pic 1-13). All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in Audit overview section. General overview is presented in AS-IS section and all found issues can be found in Audit overview section.

We have found 2 medium and 1 low vulnerability in smart contract; we also outline 4 informational statements, that can’t have any security effect, but should be presented in the report.

Graph 1. The distribution of vulnerabilities.



Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens lose etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Info	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

AS-IS overview

PumaPayPullPayment contract overview

PumaPayPullPayment contract manages billing systems for businesses. It allows registering, executing and deleting pull payments.

PumaPayPullPayment contract inherits Ownable. It imports SafeMath library for math operations, but doesn't use it.

PumaPayPullPayment contract constructor sets:

- token to _token

PumaPayPullPayment has 8 modifiers:

- isExecutor – checks whether msg.sender is an executor.
- paymentExists – checks whether payment between two specified addresses exists.
- paymentNotCancelled – checks whether payment is cancelled.
- isValidPullPaymentRequest – checks whether pull payment request is valid.
- isValidDeletionRequest – checks whether specified addresses are not 0x0 and paymentID.length is not equal to 0.
- isValidAddress – checks whether specified address is not 0x0 address.
- executorExists – checks whether specified address is an executor.
- executorDoesNotExists – checks whether specified address is not an executor.

PumaPayPullPayment has 13 functions:

- addExecutor is a public function – adds new executor. Has onlyOwner, executorDoesNotExists and isValidAddress modifiers.
- removeExecutor is a public function – removes an executor. Has onlyOwner, executorExists and isValidAddress modifiers.
- setRate is a public function – sets new exchange rate for specified currency. Has onlyOwner modifier.
- registerPullPayment is a public function – creates a new pull payment. Has isExecutor modifier.
- deletePullPayment is a public function – deletes pull payment. Has isExecutor, paymentExists, paymentNotCancelled, isValidDeletionRequest modifiers.
- executePullPayment is a public function – makes a pull payment. Has paymentExists and isValidPullPaymentRequest modifiers.
- getRate is a public view function – returns exchange rate for specified currency.
- calculatePMAFromFiat is an internal view function – returns a number of PMA tokens that can be bought for fiat.
- isValidRegistration is an internal pure function – checks whether registration is valid by comparing signature to specified client address.
- isValidDeletion is an internal view function – returns true if deletion is valid. It compares signature to the specified client address.
- doesPaymentExist is an internal view function – returns true if specified beneficiary address has a pull payment for the specified client address.
- isFundingNeeded is a private view function – returns true if specified address balance is higher than MINIMUM_AMOUN_OF_ETH_FOR_OPARATORS.
- fallback external payable function

Audit overview

Critical

No critical vulnerabilities were found.

High

No high severity vulnerabilities were found.

Medium

1. executePullPayment function could lead to overflow. nextPaymentTimestamp and frequency are specified in contract registerPullPayment function. Only executors can register new payments, but if they provide huge input numbers, nextPaymentTimestamp will cause an overflow and this will result in inability of executing deletePullPayment and executePullPayment, which have a check whether nextPaymentTimestamp < 0 (See Appendix A pic. 1-5 for evidence).

```
pullPayments[_client][msg.sender].nextPaymentTimestamp = pullPayments[_client][msg.sender].nextPaymentTimestamp + pullPayments[_client][msg.sender].frequency;
```

   Not Fixed in 3ec35a8: The possibility of this to happen is very low, however it does exist.

2. calculatePMAFromFiat function doesn't use SafeMath library for math operations. It is a good security practice to use SafeMath for handling math, nevertheless it unlikely that this function will result in overflow. _fiatAmountInCents should be set to 1*10^50 or higher for overflow to happen. (See Appendix A pic. 6 for evidence).

```
return ((ONE_ETHER * DECIMAL_FIXER * _fiatAmountInCents) / exchangeRates[_currency]) / FIAT_TO_CENT_FIXER;
```

Low

3. addExecutor function transfers ether before making changes to the state. It is considered as a bad practice because it violates Checks-Effects-Interactions pattern. However, transfer is safe against reentrancy because it forwards only 2,300 gas stipend (See Appendix A pic. 7 for evidence).

Lowest / Code style / Info

Informational statements

Informational statements are audit team findings that doesn't have any security issues. However, they are presented in report to clarify and outline functionality and business requirements.

4. Private variables in the lines 29-32, 40 are still visible for other people and can be read from blockchain.
5. Contract imports SafeMath library, but doesn't use it.
6. Contract name is PumaPayPullPayment, but file name is MasterPullPayment.
7. isValidRegistration function has the same comment as an isValidDeletion function.

   Fixed in 3ec35a8: Comment was rewritten.

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. For the contract high level description of functionality was presented in As-is overview section of the report.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

Overall quality of reviewed contracts is good; however, it contains 2 medium vulnerabilities and 1 low vulnerability.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

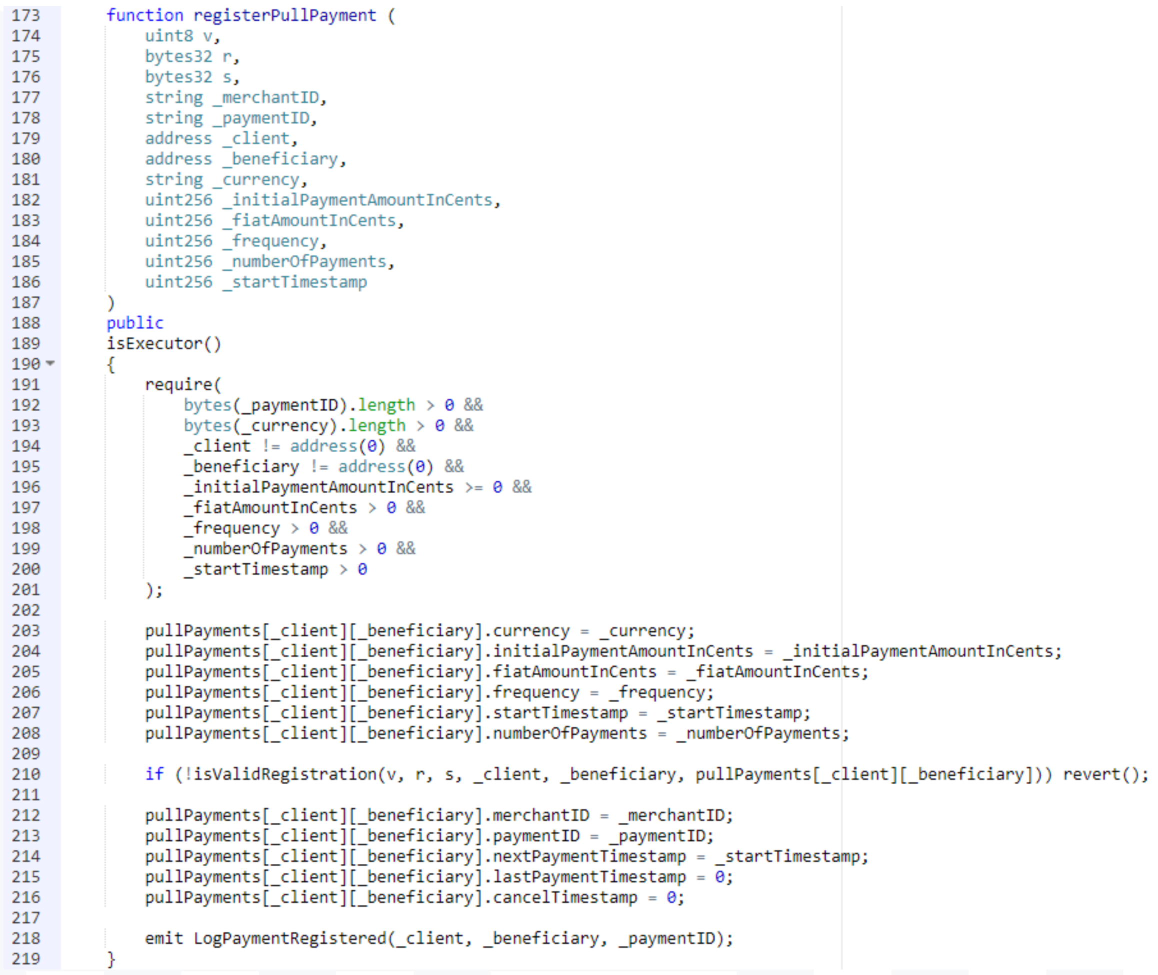
The audit makes no statements or warranties on the security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on blockchain platform. The platform, its programming language, and other software related to the smart contract can have own vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee explicit security of the audited smart contracts.

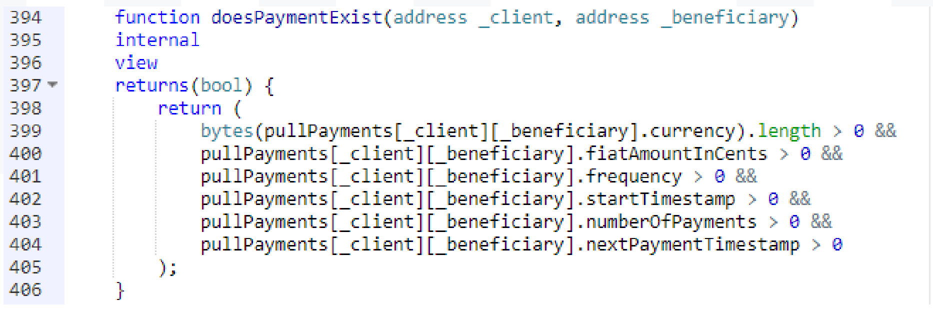
Appendix A. Evidences

Pic 1. registerPullPayment function:



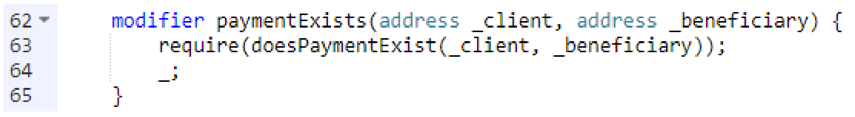
```
function registerPullPayment (
    uint8 v,
    bytes32 r,
    bytes32 s,
    string _merchantID,
    string _paymentID,
    address _client,
    address _beneficiary,
    string _currency,
    uint256 _initialPaymentAmountInCents,
    uint256 _fiatAmountInCents,
    uint256 _frequency,
    uint256 _numberOfPayments,
    uint256 _startTimestamp
)
public
isExecutor()
{
    require(
        bytes(_paymentID).length > 0 &&
        bytes(_currency).length > 0 &&
        _client != address(0) &&
        _beneficiary != address(0) &&
        _initialPaymentAmountInCents >= 0 &&
        _fiatAmountInCents > 0 &&
        _frequency > 0 &&
        _numberOfPayments > 0 &&
        _startTimestamp > 0
    );

    pullPayments[_client][_beneficiary].currency = _currency;
    pullPayments[_client][_beneficiary].initialPaymentAmountInCents = _initialPaymentAmountInCents;
    pullPayments[_client][_beneficiary].fiatAmountInCents = _fiatAmountInCents;
    pullPayments[_client][_beneficiary].frequency = _frequency;
    pullPayments[_client][_beneficiary].startTimestamp = _startTimestamp;
    pullPayments[_client][_beneficiary].numberOfPayments = _numberOfPayments;

    if (!isValidRegistration(v, r, s, _client, _beneficiary, pullPayments[_client][_beneficiary])) revert();

    pullPayments[_client][_beneficiary].merchantID = _merchantID;
    pullPayments[_client][_beneficiary].paymentID = _paymentID;
    pullPayments[_client][_beneficiary].nextPaymentTimestamp = _startTimestamp;
    pullPayments[_client][_beneficiary].lastPaymentTimestamp = 0;
    pullPayments[_client][_beneficiary].cancelTimestamp = 0;

    emit LogPaymentRegistered(_client, _beneficiary, _paymentID);
}
```

Pic 2. doesPaymentExist function:



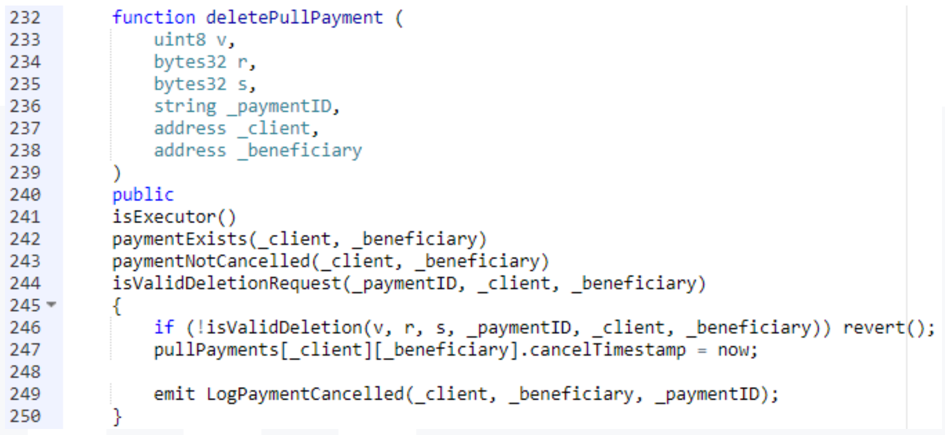
```
function doesPaymentExist(address _client, address _beneficiary)
internal
view
returns(bool) {
    return (
        bytes(pullPayments[_client][_beneficiary].currency).length > 0 &&
        pullPayments[_client][_beneficiary].fiatAmountInCents > 0 &&
        pullPayments[_client][_beneficiary].frequency > 0 &&
        pullPayments[_client][_beneficiary].startTimestamp > 0 &&
        pullPayments[_client][_beneficiary].numberOfPayments > 0 &&
        pullPayments[_client][_beneficiary].nextPaymentTimestamp > 0
    );
}
```

Pic 3. paymentExists modifier:



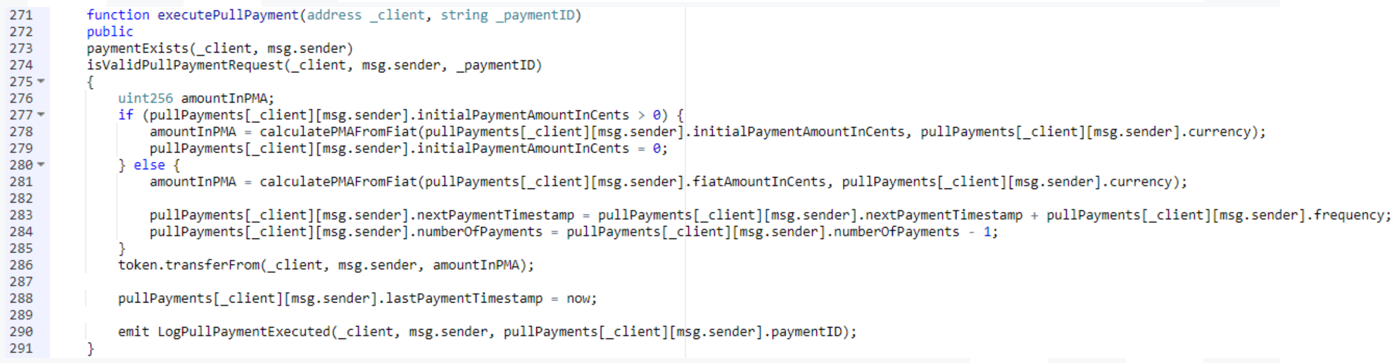
```
modifier paymentExists(address _client, address _beneficiary) {
    require(doesPaymentExist(_client, _beneficiary));
    _;
}
```

Pic 4. deletePullPayment function:



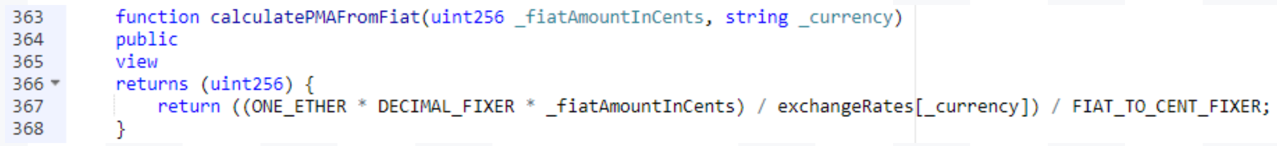
```
function deletePullPayment (
    uint8 v,
    bytes32 r,
    bytes32 s,
    string _paymentID,
    address _client,
    address _beneficiary
)
public
isExecutor()
paymentExists(_client, _beneficiary)
paymentNotCancelled(_client, _beneficiary)
isValidDeletionRequest(_paymentID, _client, _beneficiary)
{
    if (!isValidDeletion(v, r, s, _paymentID, _client, _beneficiary)) revert();
    pullPayments[_client][_beneficiary].cancelTimestamp = now;

    emit LogPaymentCancelled(_client, _beneficiary, _paymentID);
}
```

Pic 5. executePullPayment function:



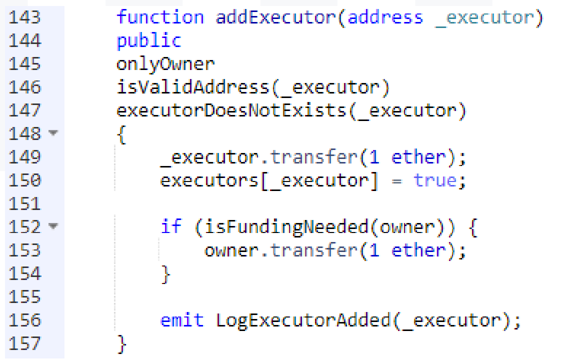
```
function executePullPayment(address _client, string _paymentID)
public
paymentExists(_client, msg.sender)
isValidPullPaymentRequest(_client, msg.sender, _paymentID)
{
    uint256 amountInPMA;
    if (pullPayments[_client][msg.sender].initialPaymentAmountInCents > 0) {
        amountInPMA = calculatePMAFromFiat(pullPayments[_client][msg.sender].initialPaymentAmountInCents, pullPayments[_client][msg.sender].currency);
        pullPayments[_client][msg.sender].initialPaymentAmountInCents = 0;
    } else {
        amountInPMA = calculatePMAFromFiat(pullPayments[_client][msg.sender].fiatAmountInCents, pullPayments[_client][msg.sender].currency);

        pullPayments[_client][msg.sender].nextPaymentTimestamp = pullPayments[_client][msg.sender].nextPaymentTimestamp + pullPayments[_client][msg.sender].frequency;
        pullPayments[_client][msg.sender].numberOfPayments = pullPayments[_client][msg.sender].numberOfPayments - 1;
    }
    token.transferFrom(_client, msg.sender, amountInPMA);

    pullPayments[_client][msg.sender].lastPaymentTimestamp = now;

    emit LogPullPaymentExecuted(_client, msg.sender, pullPayments[_client][msg.sender].paymentID);
}
```

Pic 6. calculatePMAFromFiat function:



```
function calculatePMAFromFiat(uint256 _fiatAmountInCents, string _currency)
public
view
returns (uint256) {
    return ((ONE_ETHER * DECIMAL_FIXER * _fiatAmountInCents) / exchangeRates[_currency]) / FIAT_TO_CENT_FIXER;
}
```

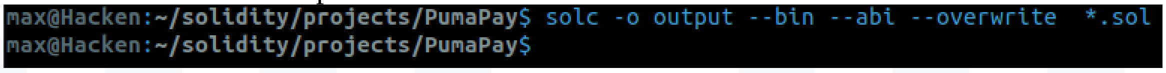
Pic. 7 addExecutor function:



```
function addExecutor(address _executor)
public
onlyOwner
isValidAddress(_executor)
executorDoesNotExists(_executor)
{
    _executor.transfer(1 ether);
    executors[_executor] = true;

    if (isFundingNeeded(owner)) {
        owner.transfer(1 ether);
    }

    emit LogExecutorAdded(_executor);
}
```

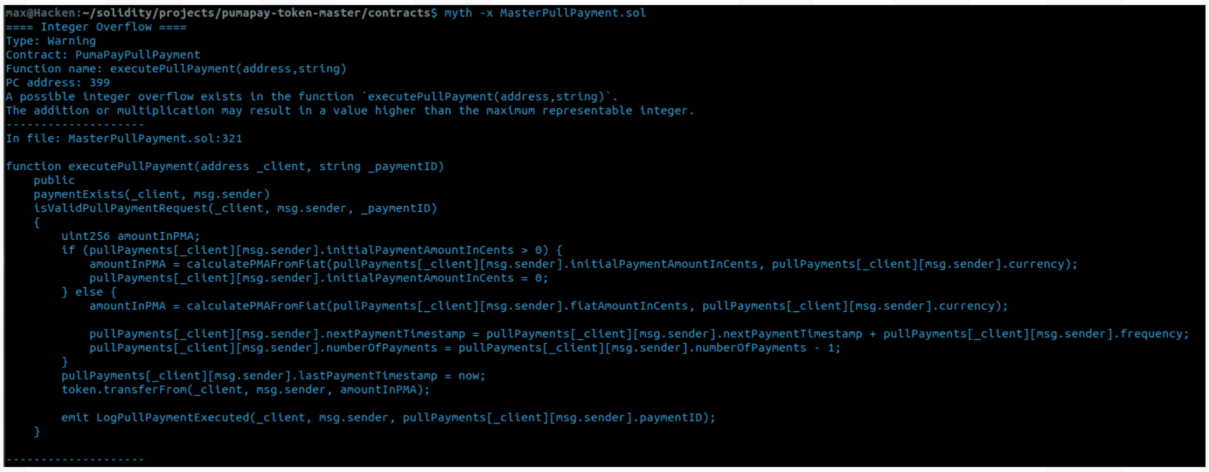
Appendix B. Automated tools reports

Pic 1. Solc automated report:



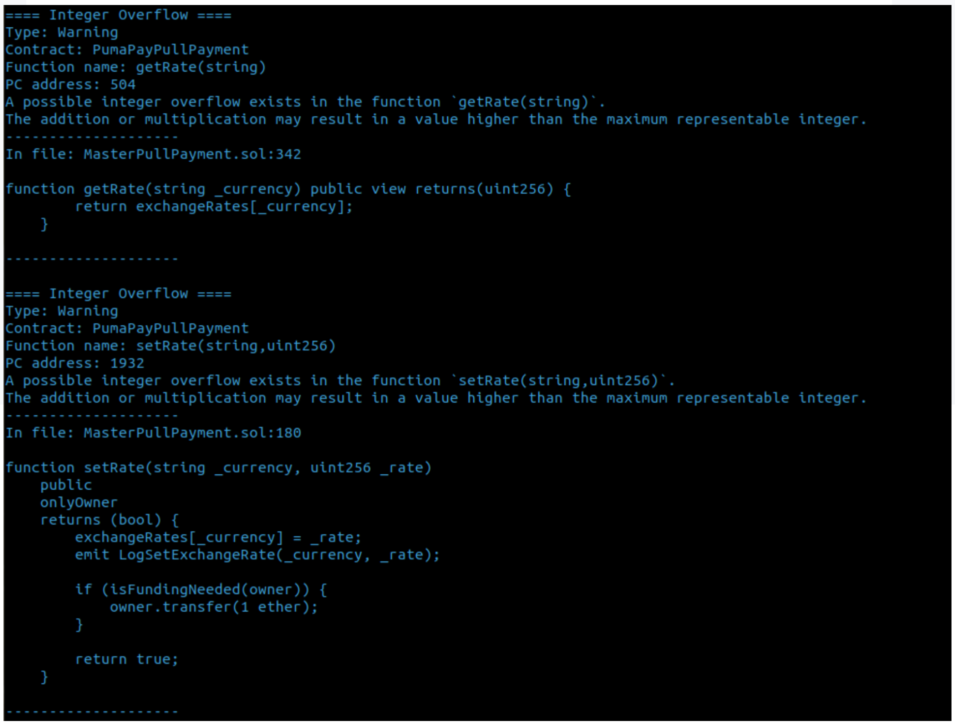
```
max@Hacken:~/solidity/projects/PumaPay$ solc -o output --bin --abi --overwrite  *.sol
max@Hacken:~/solidity/projects/PumaPay$
```

Pic 2. Mythril automated report 1:



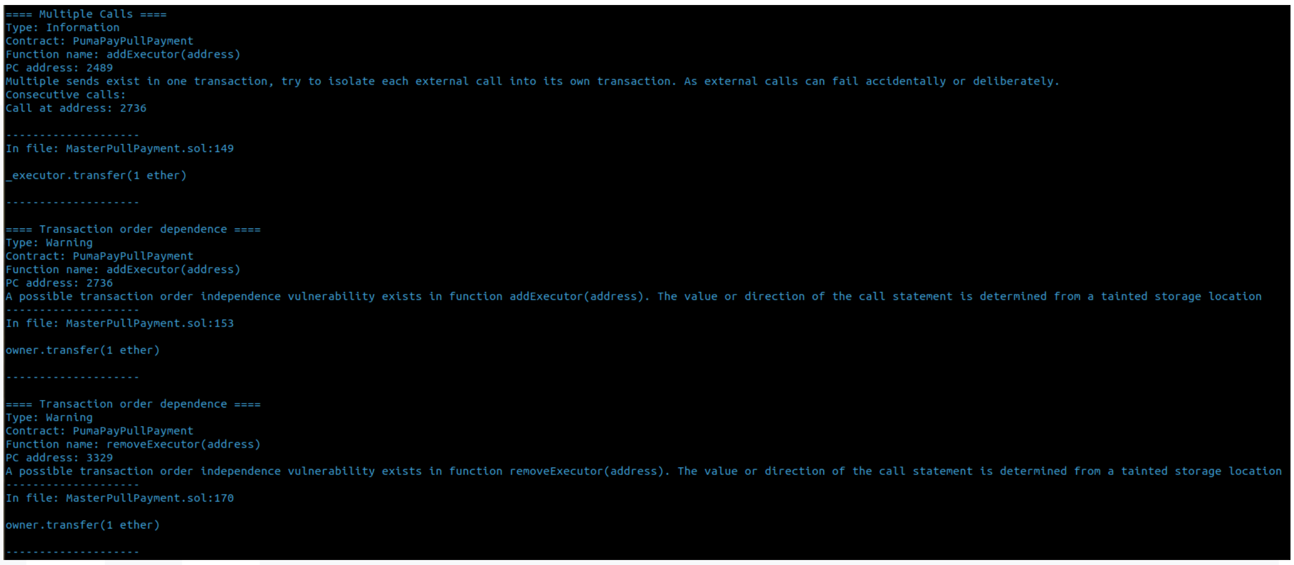
```
max@Hacken:~/solidity/projects/pumapay-token-master/contracts$ myth -x MasterPullPayment.sol
==== Integer Overflow ====
Type: Warning
Contract: PumaPayPullPayment
Function name: executePullPayment(address,string)
PC address: 399
A possible integer overflow exists in the function `executePullPayment(address,string)`.
The addition or multiplication may result in a value higher than the maximum representable integer.
--------------------
In file: MasterPullPayment.sol:321

function executePullPayment(address _client, string _paymentID)
    public
    paymentExists(_client, msg.sender)
    isValidPullPaymentRequest(_client, msg.sender, _paymentID)
    {
        uint256 amountInPMA;
        if (pullPayments[_client][msg.sender].initialPaymentAmountInCents > 0) {
            amountInPMA = calculatePMAFromFiat(pullPayments[_client][msg.sender].initialPaymentAmountInCents, pullPayments[_client][msg.sender].currency);
            pullPayments[_client][msg.sender].initialPaymentAmountInCents = 0;
        } else {
            amountInPMA = calculatePMAFromFiat(pullPayments[_client][msg.sender].fiatAmountInCents, pullPayments[_client][msg.sender].currency);

            pullPayments[_client][msg.sender].nextPaymentTimestamp = pullPayments[_client][msg.sender].nextPaymentTimestamp + pullPayments[_client][msg.sender].frequency;
            pullPayments[_client][msg.sender].numberOfPayments = pullPayments[_client][msg.sender].numberOfPayments - 1;
        }
        pullPayments[_client][msg.sender].lastPaymentTimestamp = now;
        token.transferFrom(_client, msg.sender, amountInPMA);

        emit LogPullPaymentExecuted(_client, msg.sender, pullPayments[_client][msg.sender].paymentID);
    }

--------------------
```

Pic 3. Mythril automated report 2:



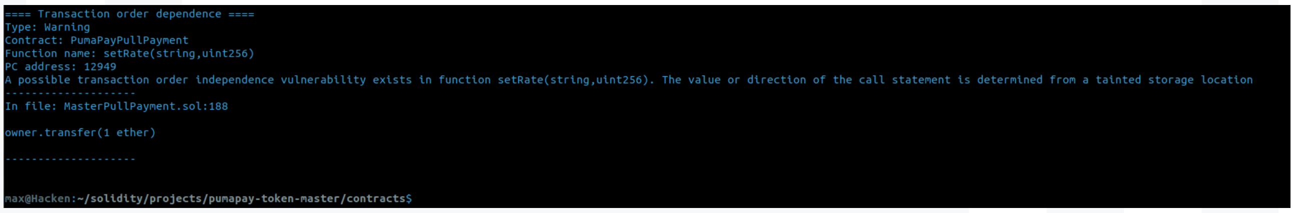
```
==== Integer Overflow ====
Type: Warning
Contract: PumaPayPullPayment
Function name: getRate(string)
PC address: 504
A possible integer overflow exists in the function `getRate(string)`.
The addition or multiplication may result in a value higher than the maximum representable integer.
--------------------
In file: MasterPullPayment.sol:342

function getRate(string _currency) public view returns(uint256) {
        return exchangeRates[_currency];
    }

--------------------

==== Integer Overflow ====
Type: Warning
Contract: PumaPayPullPayment
Function name: setRate(string,uint256)
PC address: 1932
A possible integer overflow exists in the function `setRate(string,uint256)`.
The addition or multiplication may result in a value higher than the maximum representable integer.
--------------------
In file: MasterPullPayment.sol:180

function setRate(string _currency, uint256 _rate)
    public
    onlyOwner
    returns (bool) {
        exchangeRates[_currency] = _rate;
        emit LogSetExchangeRate(_currency, _rate);

        if (isFundingNeeded(owner)) {
            owner.transfer(1 ether);
        }

        return true;
    }

--------------------
```

Pic 4. Mythril automated report 3:



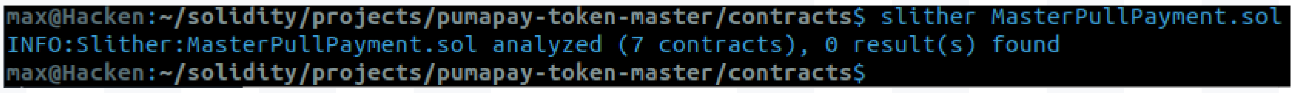
```
==== Multiple Calls ====
Type: Information
Contract: PumaPayPullPayment
Function name: addExecutor(address)
PC address: 2489
Multiple sends exist in one transaction, try to isolate each external call into its own transaction. As external calls can fail accidentally or deliberately.
Consecutive calls:
Call at address: 2736

--------------------
In file: MasterPullPayment.sol:149

_executor.transfer(1 ether)

--------------------

==== Transaction order dependence ====
Type: Warning
Contract: PumaPayPullPayment
Function name: addExecutor(address)
PC address: 2736
A possible transaction order independence vulnerability exists in function addExecutor(address). The value or direction of the call statement is determined from a tainted storage location
--------------------
In file: MasterPullPayment.sol:153

owner.transfer(1 ether)

--------------------

==== Transaction order dependence ====
Type: Warning
Contract: PumaPayPullPayment
Function name: removeExecutor(address)
PC address: 3329
A possible transaction order independence vulnerability exists in function removeExecutor(address). The value or direction of the call statement is determined from a tainted storage location
--------------------
In file: MasterPullPayment.sol:170

owner.transfer(1 ether)

--------------------
```

Pic 5. Mythril automated report 4:



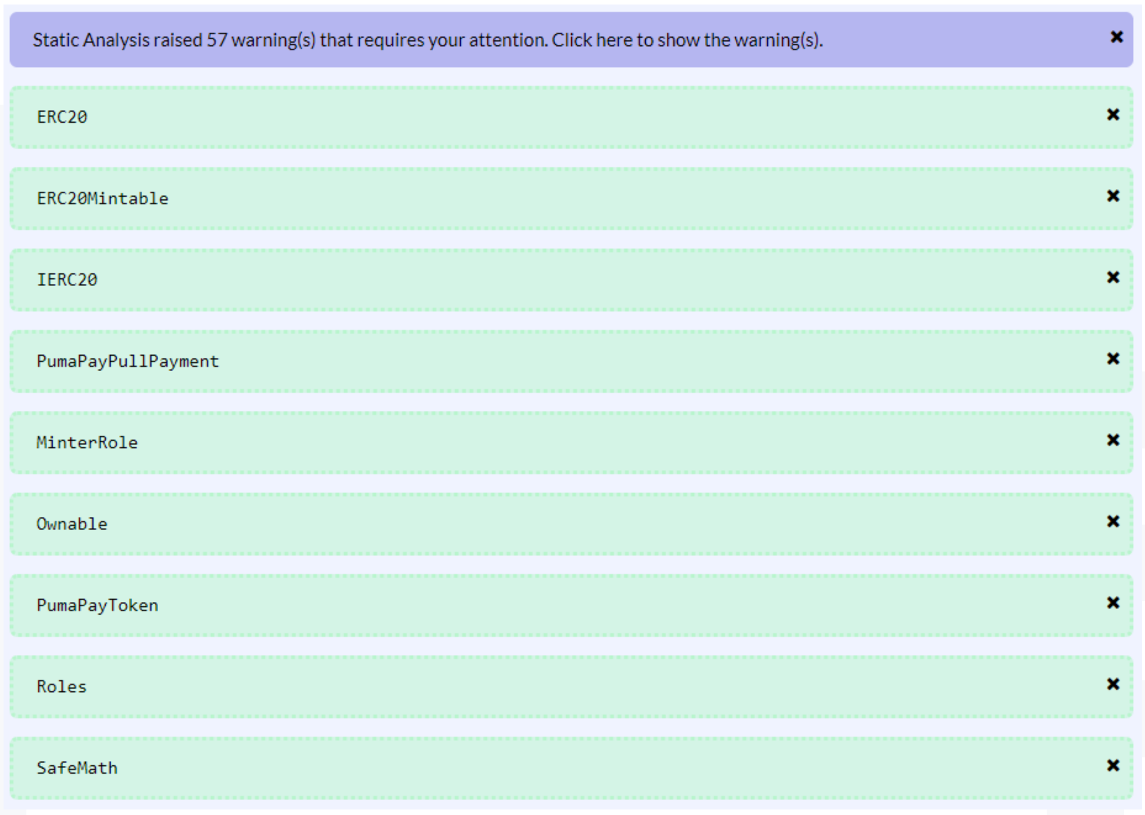
```
==== Transaction order dependence ====
Type: Warning
Contract: PumaPayPullPayment
Function name: setRate(string,uint256)
PC address: 12949
A possible transaction order independence vulnerability exists in function setRate(string,uint256). The value or direction of the call statement is determined from a tainted storage location
--------------------
In file: MasterPullPayment.sol:188

owner.transfer(1 ether)

--------------------


max@Hacken:~/solidity/projects/pumapay-token-master/contracts$
```

Pic 6. Slither automated report:



```
max@Hacken:~/solidity/projects/pumapay-token-master/contracts$ slither MasterPullPayment.sol
INFO:Slither:MasterPullPayment.sol analyzed (7 contracts), 0 result(s) found
max@Hacken:~/solidity/projects/pumapay-token-master/contracts$
```

Pic 7. Remix IDE automated report part 1:



Static Analysis raised 57 warning(s) that requires your attention. Click here to show the warning(s).

ERC20

ERC20Mintable

IERC20

PumaPayPullPayment

MinterRole

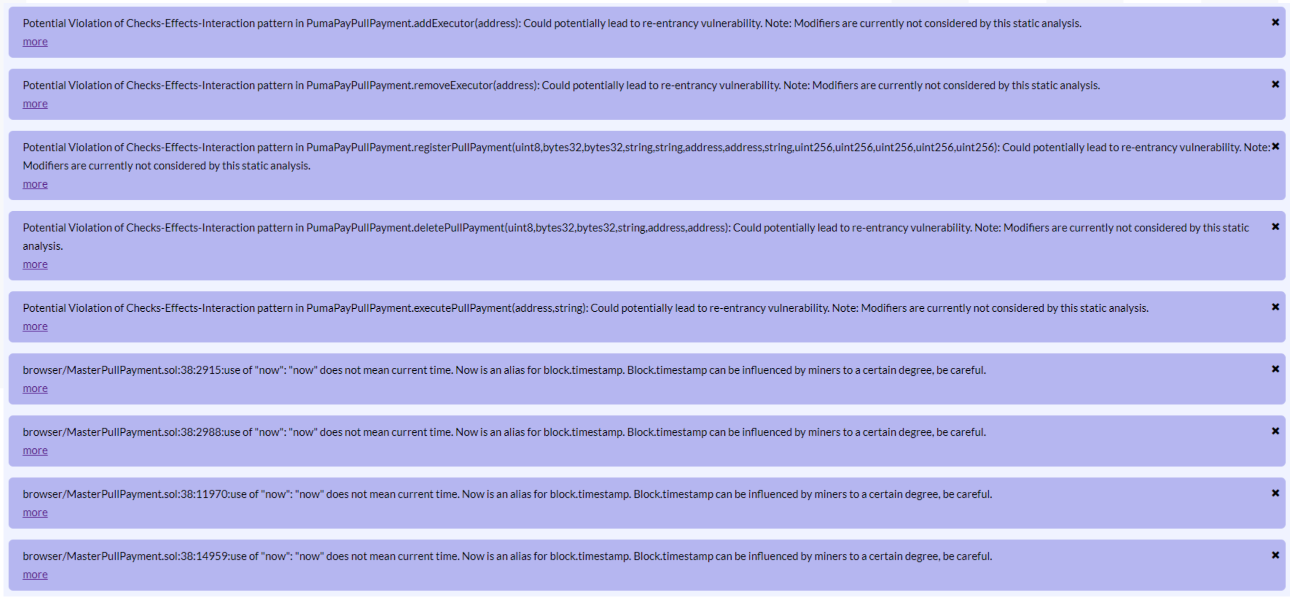
Ownable

PumaPayToken

Roles

SafeMath

Pic 8. Remix IDE automated report part 2:



Potential Violation of Checks-Effects-Interaction pattern in PumaPayPullPayment.addExecutor(address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more

Potential Violation of Checks-Effects-Interaction pattern in PumaPayPullPayment.removeExecutor(address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more

Potential Violation of Checks-Effects-Interaction pattern in PumaPayPullPayment.registerPullPayment(uint8,bytes32,bytes32,string,string,address,address,string,uint256,uint256,uint256,uint256,uint256): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more

Potential Violation of Checks-Effects-Interaction pattern in PumaPayPullPayment.deletePullPayment(uint8,bytes32,bytes32,string,address,address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more

Potential Violation of Checks-Effects-Interaction pattern in PumaPayPullPayment.executePullPayment(address,string): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more

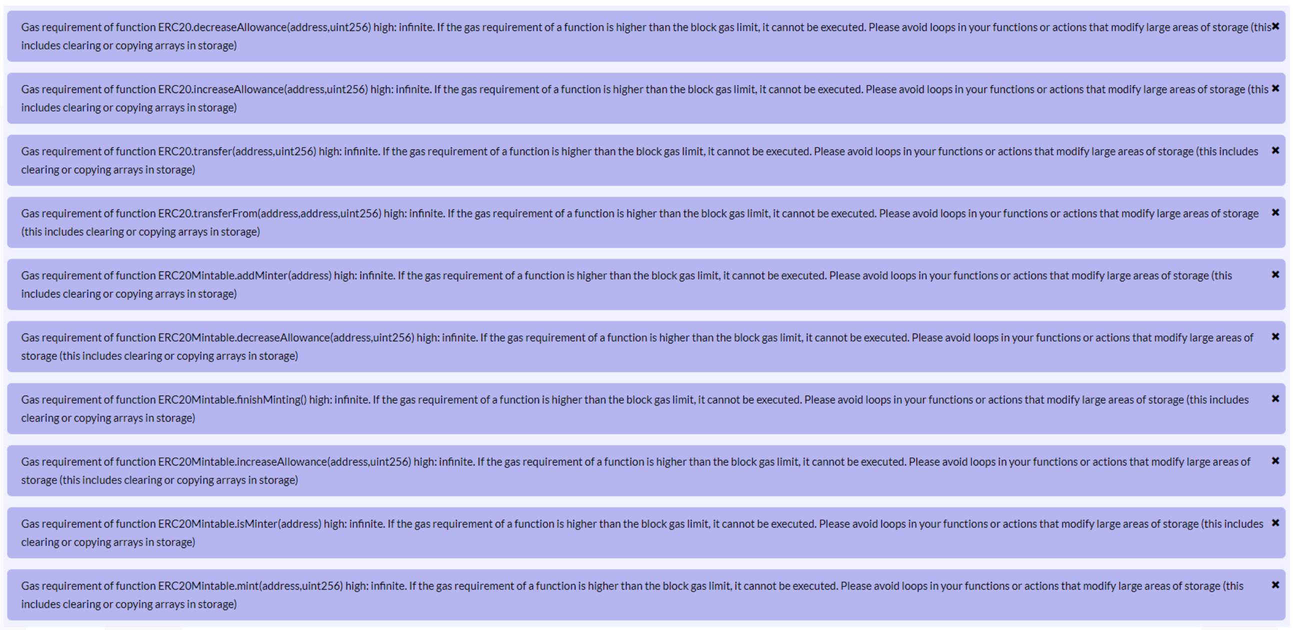
browser/MasterPullPayment.sol:38:2915:use of "now": "now" does not mean current time. Now is an alias for block.timestamp. Block.timestamp can be influenced by miners to a certain degree, be careful.
more

browser/MasterPullPayment.sol:38:2988:use of "now": "now" does not mean current time. Now is an alias for block.timestamp. Block.timestamp can be influenced by miners to a certain degree, be careful.
more

browser/MasterPullPayment.sol:38:11970:use of "now": "now" does not mean current time. Now is an alias for block.timestamp. Block.timestamp can be influenced by miners to a certain degree, be careful.
more

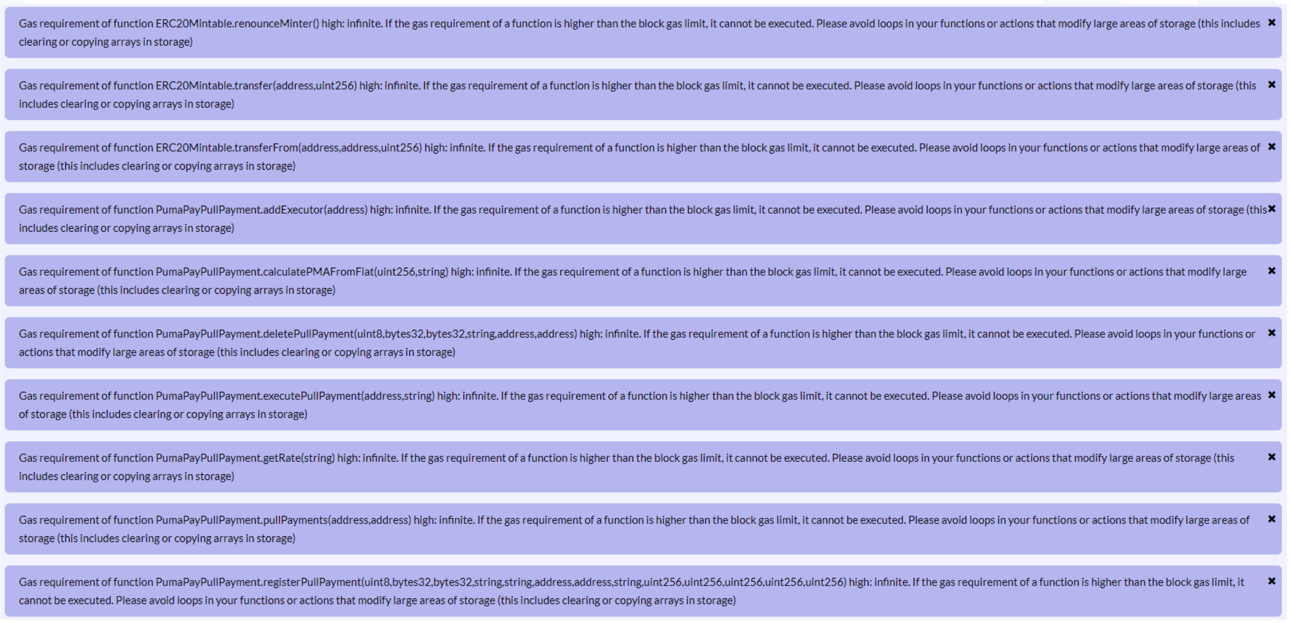
browser/MasterPullPayment.sol:38:14959:use of "now": "now" does not mean current time. Now is an alias for block.timestamp. Block.timestamp can be influenced by miners to a certain degree, be careful.
more

Pic 9. Remix IDE automated report part 3:



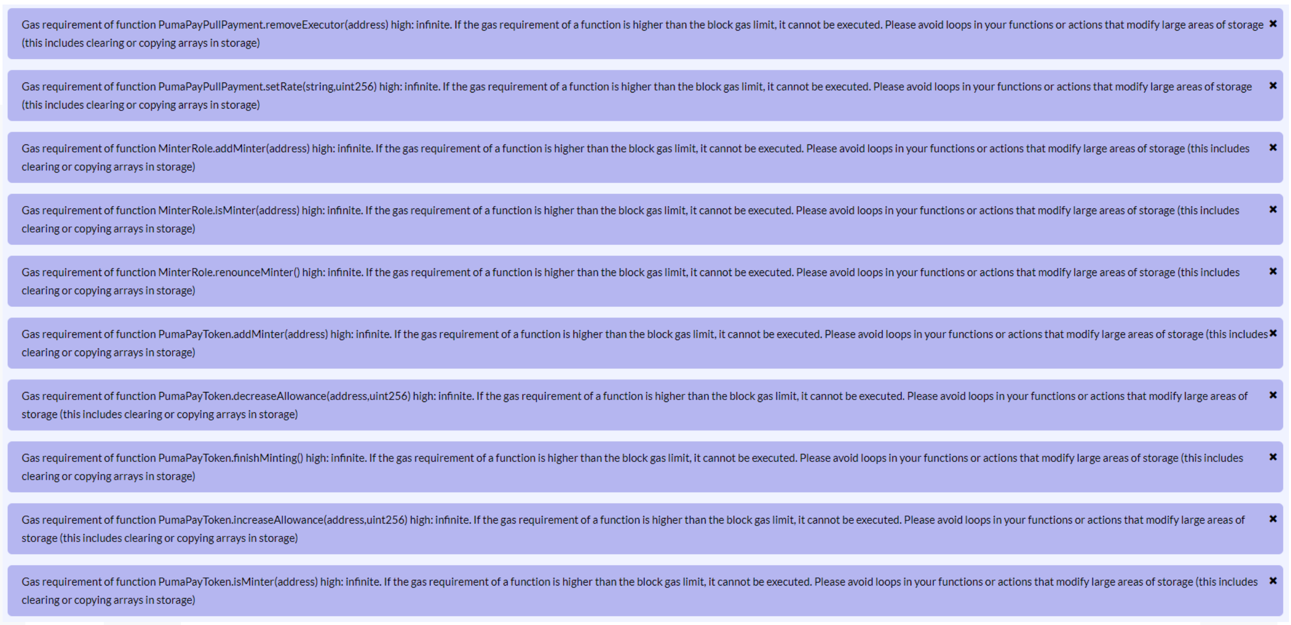
```
Gas requirement of function ERC20.decreaseAllowance(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Gas requirement of function ERC20.increaseAllowance(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Gas requirement of function ERC20.transfer(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Gas requirement of function ERC20.transferFrom(address,address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Gas requirement of function ERC20Mintable.addMinter(address) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Gas requirement of function ERC20Mintable.decreaseAllowance(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Gas requirement of function ERC20Mintable.finishMinting() high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Gas requirement of function ERC20Mintable.increaseAllowance(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Gas requirement of function ERC20Mintable.isMinter(address) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Gas requirement of function ERC20Mintable.mint(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
```

Pic 10. Remix IDE automated report part 4:



```
Gas requirement of function ERC20Mintable.renounceMinter() high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Gas requirement of function ERC20Mintable.transfer(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Gas requirement of function ERC20Mintable.transferFrom(address,address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Gas requirement of function PumaPayPullPayment.addExecutor(address) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Gas requirement of function PumaPayPullPayment.calculatePMAFromFiat(uint256,string) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Gas requirement of function PumaPayPullPayment.deletePullPayment(uint8,bytes32,bytes32,string,address,address) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Gas requirement of function PumaPayPullPayment.executePullPayment(address,string) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Gas requirement of function PumaPayPullPayment.getRate(string) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Gas requirement of function PumaPayPullPayment.pullPayments(address,address) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
Gas requirement of function PumaPayPullPayment.registerPullPayment(uint8,bytes32,bytes32,string,string,address,address,string,uint256,uint256,uint256,uint256,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)
```

Pic 11. Remix IDE automated report part 5:



Gas requirement of function PumaPayPullPayment.removeExecutor(address) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function PumaPayPullPayment.setRate(string,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function MinterRole.addMinter(address) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function MinterRole.isMinter(address) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function MinterRole.renounceMinter() high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function PumaPayToken.addMinter(address) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

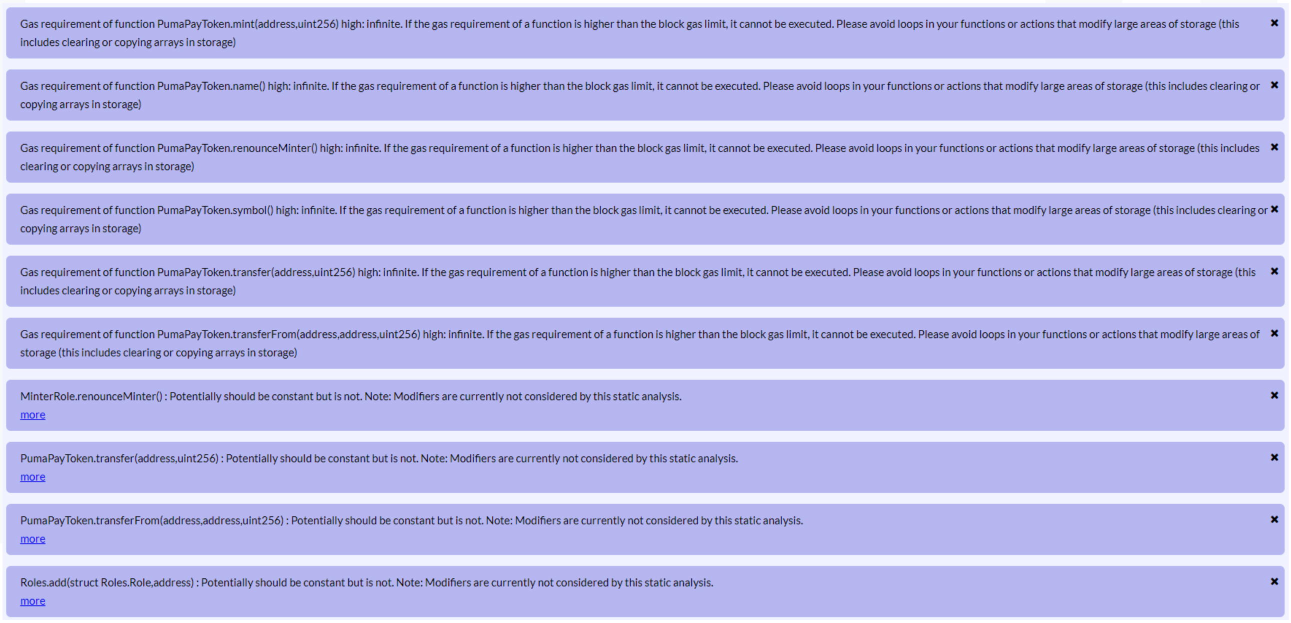
Gas requirement of function PumaPayToken.decreaseAllowance(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function PumaPayToken.finishMinting() high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function PumaPayToken.increaseAllowance(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function PumaPayToken.isMinter(address) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pic 12. Remix IDE automated report part 6:



Gas requirement of function PumaPayToken.mint(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function PumaPayToken.name() high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function PumaPayToken.renounceMinter() high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function PumaPayToken.symbol() high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function PumaPayToken.transfer(address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Gas requirement of function PumaPayToken.transferFrom(address,address,uint256) high: infinite. If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

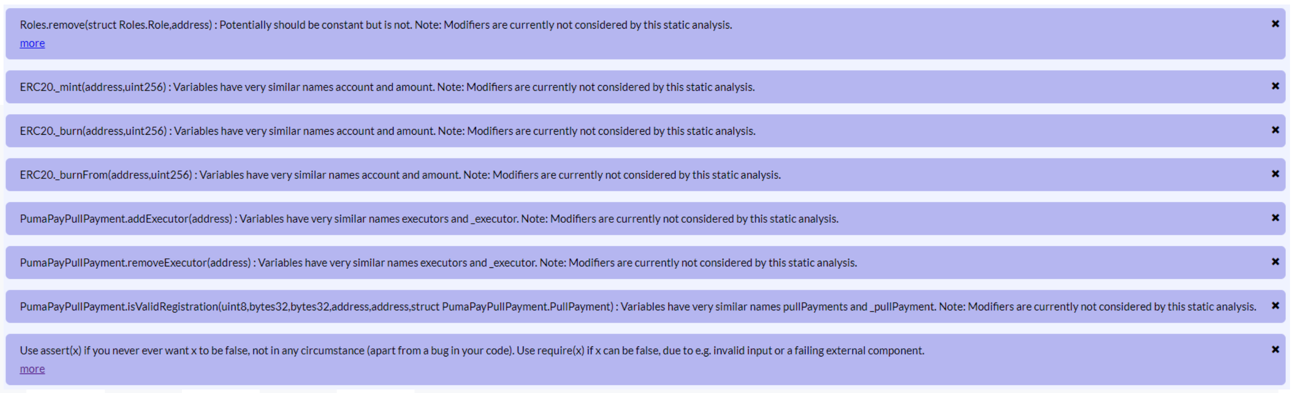
MinterRole.renounceMinter() : Potentially should be constant but is not. Note: Modifiers are currently not considered by this static analysis.
more

PumaPayToken.transfer(address,uint256) : Potentially should be constant but is not. Note: Modifiers are currently not considered by this static analysis.
more

PumaPayToken.transferFrom(address,address,uint256) : Potentially should be constant but is not. Note: Modifiers are currently not considered by this static analysis.
more

Roles.add(struct Roles.Role,address) : Potentially should be constant but is not. Note: Modifiers are currently not considered by this static analysis.
more

Pic 13. Remix IDE automated report part 7:



Roles.remove(struct Roles.Role,address) : Potentially should be constant but is not. Note: Modifiers are currently not considered by this static analysis.
more

ERC20._mint(address,uint256) : Variables have very similar names account and amount. Note: Modifiers are currently not considered by this static analysis.

ERC20._burn(address,uint256) : Variables have very similar names account and amount. Note: Modifiers are currently not considered by this static analysis.

ERC20._burnFrom(address,uint256) : Variables have very similar names account and amount. Note: Modifiers are currently not considered by this static analysis.

PumaPayPullPayment.addExecutor(address) : Variables have very similar names executors and _executor. Note: Modifiers are currently not considered by this static analysis.

PumaPayPullPayment.removeExecutor(address) : Variables have very similar names executors and _executor. Note: Modifiers are currently not considered by this static analysis.

PumaPayPullPayment.isValidRegistration(uint8,bytes32,bytes32,address,address,struct PumaPayPullPayment.PullPayment) : Variables have very similar names pullPayments and _pullPayment. Note: Modifiers are currently not considered by this static analysis.

Use assert(x) if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use require(x) if x can be false, due to e.g. invalid input or a failing external component.
more