BLOCKSTARS TECHNOLOGY

PowerPlay

SMART CONTRACT SECURITY AUDIT

ANALYSIS TYPE (STANDARD)

VERSION 2

Blockstars Technology

InvestorVesting: Moderate Vulnerable (7/10)
BillBuster: Low Vulnerable (9/10)
REMI: Secured (10/10)

31st August, 2022

Table of Contents

1. Introduction 4

2. Project Context 4

4. Definitions 6

4.1. Security Severity 6

4.2. Project Quality 7

5. Audit Summary 8

5.1. Project Quality 8

5.2. Smart contract security 9

5.2.1. Security score for **REMI**.sol 9

5.2.2. Security score for **InvestorVesting**.sol 9

5.2.3. Security score for **BillBuster**.sol 10

6. Analytic Statistics 11

6.1. Programming Issues 11

6.2. Code Specifications and Best practices 12

6.3. Gas optimization 12

6.4 Risk to attacks 13

7. Manual Audit process 14

7.1. Functions Overview of REMI.sol 14

7.2. Functions Overview of InvestorVesting.sol 14

7.3. Functions Overview of BillBuster.sol 15

8.Audit Findings in Detail 16

8.1. REMI.sol 16

8.2. InvestorVesting.sol 16

8.2.1. Moderate 16

8.2.2. Low 17

8.2.3. Informational 17

8.3. BillBuster.sol 18

8.3.1. Low 18

8.3.2. Informational 19

9. Conclusion 20

10. Appendix 21

10.1. Functional Flow Chart (REMI, InvestorVesting, and BillBuster) 21

10.2. Score checklists 22

10.2.1. Documentation Quality Checklist 22

10.2.2. Code Quality Checklist 23

10.2.3. Architecture Quality Checklist 24
10.2.4. Project Quality Score Calculations 25
10.3. Smart Contracts diagrams 26
10.3.1. REMI contract Flow 26
10.3.2. InvestorVesting contract 27
10.3.3. BillBuster contract 28

Declaration

This document is Blockstars smart contract security audit report for the REMI, BillBuster and InvestorVesting contracts from PowerPlay Project.

Information contained within this document has confidential information regarding these contracts.

Analysis results of the smart contracts are supplied containing lists of vulnerabilities and malicious code which could be used to malform the project. Upon receiving this report and until the issues are resolved or mitigated, this report is kept private between both Blockstars Technology and our respected client.

1. Introduction

This audit report contains confidential information of smart contract programs running in the Power Play project. It analyses security vulnerabilities, smart contract best practices, and possible attacks, using popular automated tests and manual audits. We outlined our systematic approach to evaluate potential security issues in core smart contracts in Power Play project and provide audit summary with remedies for mitigate the vulnerability findings.

2. Project Context

This section explains the client's project in detail with scope.

Item	Description
Issuer	DLTX
Website	https://www.dltx.io/
Source	Smart contract programs
Language	Solidity
Blockchain	Ethereum
Git Repository	N/A
Audit type	Standard
Analysis Methods	Static, Dynamic and Manual
Audit Team	Pura, Faizin, and Marcus
Approved By	Pura, and Nilanga
Timeline	From: *17th August 2022* To: *26th August 2022*
Change logs	Version 2

3. Audit Scope

Scope of this project is to identify smart contract vulnerabilities to improve the coding practice in the given three smart contract programs, that are REMI.sol, InvestorVesting.sol, and BillBuster.sol.

Audit Method: **Standard**

- Automated testing using analysis tools which includes static and dynamic analysis methods.
- Manual audit with code review for each function. It requires business documents from the client.

Repository	N/A
Commit Id	N/A
Branch	N/A
Technical Documentation	Flowchart and business logics are given
REMI.sol	https://kovan.etherscan.io/token/0xd9bacc5bccad9a380001d41cd234b4d5f33ece76#code
InvestorVesting.sol	https://kovan.etherscan.io/address/0xBDfEbE14E08A04C11Ba4F155043DAA0002831b96#code
BillBuster.sol	https://kovan.etherscan.io/address/0xBeF3AEB5ADCD13a4781dc6f63E3351d27f875832#code

4. Definitions

4.1. Security Severity

Severity	Value	Description
Critical	0 - 1.9	Critical vulnerabilities are easily exploited by attackers, and they lead to potential assets loss (Example: Tokens, Cryptocurrency)
High	2 - 3.9	High level vulnerabilities are difficult to exploit, however they also have significant impact on smart contract execution due to lack of secured access control. (Example: Public access to crucial functions and data)
Medium	4 - 5.9	Medium vulnerabilities do not lead to loss of assets or data, but it is important to fix those issues.
Moderate	6 - 7.9	Moderate vulnerabilities lead to potential risks of errors when external programs call the contract. Otherwise, the contract works as intended.
Low	8 - 9.9	Low level vulnerabilities are related to out-dated, unused code snippets, and they don't have a significant impact on contract execution.
Informational / Secure	10	Not in association to vulnerabilities. It requires best practices, code standards and documentary code. Contract is not vulnerable.

4.2. Project Quality

Quality	Value	Description
Very Poor	0 - 1.9	Extremely little or no documentation provided. Very Poor architecture and no best-practices used
Poor	2 - 3.9	Very little amount of documentation provided. Majority of important information is missing or not included. Poor architecture and a bit of best-practices used.
Good	4 - 5.9	Good amount of documentation provided. Good architecture and good use of best practices.
Very Good	6 - 7.9	Satisfactory amount of documentation. Very good architecture and very good use of best practices.
Excellent	8 - 10	Required amount of documentation provided. Architecture is excellent and excellent use of best practice.

5. Audit Summary

5.1. Project Quality

5.1.1 **Documentation Quality:** **4/10**

It covers the quality of the business documentations provided and how it matches with code implementations.

(Appendix 10.2.1)

5.1.2. **Code Quality:**

REMI: **10/10**
InvestorVesting: **8/10**
BillBuster: **9/10**

This includes the use of best practices, coding standards, coding readability, and proper comments in the code.

(Appendix 10.2.2)

5.1.3. **Architecture Quality:**

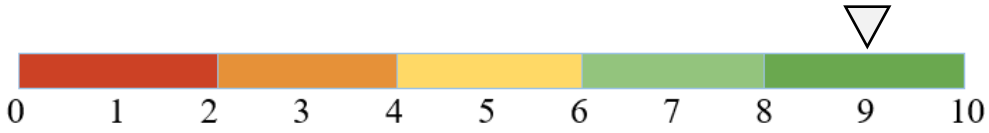
REMI: **10/10**
InvestorVesting: **10/10**
BillBuster: **10/10**

The process of execution of the contracts in detail such as development environment, and compilation, deployment, and execution detail.

(Appendix 10.2.3)

5.1.4. **Summary Score:** **9/10**

According to the audit result, we summarize that in association with the Power Play project, the project quality is "**Excellent**".



(Appendix 10.2.4)

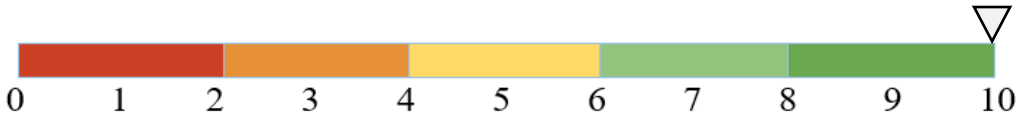
5.2. Smart contract security

5.2.1. Security score for **REMI**.sol

0 critical / 0 medium / 0 low level vulnerabilities found in REMI.sol

Security Score: 10/10

This summarizes the security audit results from automated and manual audits.
According to the audit result, we summarize that in association to contract security, the REMI contract is "Secure" (safe for deployment)



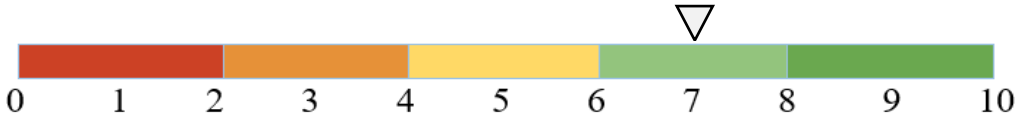
(REMI manual audit)

5.2.2. Security score for **InvestorVesting**.sol

1 moderate / 1 low level / 3 informational vulnerabilities found in InvestorVesting.sol

Security Score: 7/10

This summarizes the security audit results from automated and manual audits.
According to the audit result, we summarize that in association to contract security, the BillBuster contract is "Moderate Vulnerable**"**



(InvestorVesting manual audit)

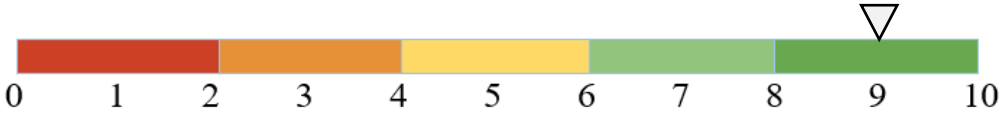
5.2.3. Security score for **BillBuster**.sol

1 low, 1 informational level vulnerability found in BillBuster.sol

Security Score: **9/10**

This summarizes the security audit results from automated and manual audits.

According to the audit result, we summarize that in association to contract security, the InvestorVesting contract is "Low Vulnerable" (safe for deployment)



(BillBuster manual audit)

6. Analytic Statistics

6.1. Programming Issues

#	Description	Type	Severity	REMI	Investor Vesting	BillBuster
1	Unchecked return value from low-level external calls	SWC-104	Medium	N/A	Passed	Passed
3	Floating pragma is set	SWC-103	Low	Passed	Passed	Passed
4	Error Handling and logging are implemented	Custom	Medium	N/A	Passed	Passed
5	Spot prices should not be used as a source for price oracles	Custom	Medium	N/A	Passed	Passed
6	State variable should not be used without being initialized	Custom	Medium	N/A	Failed	Failed
7	Is inheritance used properly	SWC-125	High	Passed	Passed	Passed
8	External components used insecurely	Custom	High	N/A	Passed	Passed
9	Functions that loop over unbounded data structures	Custom	Critical	N/A	Passed	Passed
10	Msg.value should not be used in a loop	Custom	Critical	Passed	Passed	Passed

6.2. Code Specifications and Best practices

#	Description	Type	Severity	REMI	Investor Vesting	BillBuster
1	Use of the "constant" state mutability modifier is deprecated.	SWC-111	Low	N/A	Passed	Passed
2	Use of the "throw" state mutability modifier is deprecated.	SWC-111	Low	N/A	Passed	Passed
3	Strict equalities should not render the function to be unusable	Custom	Moderate	N/A	Passed	Passed
4	Use of best Practices	Custom	Low	Passed	Failed	Failed
5	Business logic is implemented as per the documents provided	Custom	High	Passed	Passed	Passed

6.3. Gas optimization

#	Description	Type	Severity	REMI	Investor Vesting	BillBuster
1	Message call with hardcoded gas amount	SWC-134	Medium	N/A	Passed	Passed
2	Check for gas usage and minimize gas consumption.	Custom	Low	N/A	Passed	Passed

6.4 Risk to attacks

#	Description	Type	Severity	REMI	Investor Vesting	BillBuster
1	Code contains suicidal, greedy, and prodigal instructions	SWC-106	High	Passed	Passed	Passed
2	Contract is Haltable	Custom	Medium	N/A	Passed	Passed
3	Adopt checks-effects-interactions patterns for any transactions of value	Custom	Critical	N/A	Passed	Passed
4	Reduce and remove unnecessary code to reduce attack surface area.	Custom	Low	Passed	Passed	Passed
5	Timestamps should not be used to execute critical functions.	SWC-116	Medium	Passed	Failed	Passed
6	Sensitive data in normal form should not be stored on-chain	Custom	Medium	N/A	Passed	Passed
7	Vulnerable to Integer overflow and under-flow	SWC-101	High	Passed	Passed	Passed

7. Manual Audit process

7.1. Functions Overview of REMI.sol

#	Function	Type	Observation	Status
1	constructor()	Constructor	It mints 10bn REMI tokens at the contract creation using SafeERC20	Safe

7.2. Functions Overview of InvestorVesting.sol

#	Function	Type	Observation	Status
1	constructor()	constructor	It sets token address, operator and startDate	Safe
2	onlyOperator()	modifier	checks the user is an operator	Safe
3	setVesting()	write	Set vestingDetail with required parameters. This function holds the token for vestor	Safe (Confirmed with the business requirement)
4	claim()	write	Beneficiary is able to claim his vesting Amount gradually. It uses block.timestamp value to calculate the amountToClaim.	Not-recommended (block.timestamp can be manipulated by miners)
5	getOperator()	read	Returns the operator address	Safe
6	getBeneficiaryVesting()	read	Returns beneficiary's vesting detail	Safe
7	setOperator()	write	Owner can set the operator address. But there is no event emitted	Safe
8	setStartDate()	write	Owner can set startDate	Safe (It is not recommended since it leads to centralization)

7.3. Functions Overview of BillBuster.sol

#	Function	Type	Observation	Status
1	constructor()	constructor	Sets the REMI token address and tax fee. However, there is no validation.	Safe
2	setTaxFee()	write	Sets the tax fee with fee and decimals by the owner	Safe
3	toggleTransactionFees()	write	waiveFees is set to true or false by owner	Safe
4	updateTaxAddress()	write	Owner can update taxAddress value	Safe
5	calculateFee()	read	Calculate taxFee according to the percentage and returns the value	Safe
6	transferTax()	write	If waiveFee is not true, it calculates tax and transfers it to the taxAddress.	Safe
7	getBalance()	read	returns the balance of the token that an address has.	Safe
8	totalStaked()	read	Returns the total staked amount of tokens in the contract	Safe
9	deposit()	write	Deposit the tokens from the depositor address to the contract address.	Safe
10	withdraw()	write	User is able to withdraw tokens from the contract if he has enough token balance.	Safe

8.Audit Findings in Detail

8.1. REMI.sol

No vulnerabilities detected. It is **safe** to deploy this contract.

8.2. InvestorVesting.sol

8.2.1. Moderate

1. **Issue:** Use of block timestamp value in critical calculations
   Function name: claim()

```
amountToClaim +=
  ((block.timestamp - lastClaimedTime) *
    (vestingDetails[beneficiary].vestingAmount -
      vestingDetails[beneficiary].initialAmount)) /
  vestingDetails[beneficiary].duration;
```

Description:
The claim function uses block.timestamp value to calculate the amountToClaim value. Malicious miners can alter the timestamp of their blocks if they want to gain advantages by doing so. (SWC-166)

Resolution:
It is recommended to avoid relying on block timestamps in the critical calculations or operations. Block values are not precise, and the use of them can lead to unexpected effects.
(Acknowledged by the client)

8.2.2. Low

1. **Issue:** Missing events emitting.
   Function name: setOperator()

```
function setOperator(address _operator) external onlyOwner {
require(_operator != address(0), "Address cannot be zero");
require(_operator != msg.sender, "Cannot set self as operator");
require(_operator != operator, "Already set");
operator = _operator;
}
```

Description:
function should emit an event after setting the state variable operator = _operator.

Resolution:
It is recommended to Emit an event for state variable changes or critical operations.
(Acknowledged by the client)

8.2.3. Informational

1. **Suggestion:** It works as per the logic provided
   Function name: setVesting(), claim()

Description:
These functions are working as per the logic provided by client.

According to the business requirements provided, InvestorVesting contract holds the tokens for investors to allow them to vest tokens gradually over time. The totalVestingAmount is manually transferred to the contract and then use the setVesting() to add the investors vesting details.

The user gets back the exact amount that they have been set to vesting gradually.
(This function is designed and worked as per the business logic is provided by the client)

2. **Suggestion:** Be aware of the functions that have onlyOwner modifier
   Function name: renounceOwnership

   Description: The contract is Ownable. Renouncing ownership will leave the contract without an owner, thereby removing any functionality that is only available to the owner.
   (Acknowledged by the client)

3. **Suggestion:** Uninitialized local variable.
   Function name: claim()

```
uint256 amountToClaim;  – Line no 183
```

   Description:
   amountToClaim is a local variable that is not initialized.

   Resolution:
   It is best practice to initialize the variable. If amountToClaim is meant to be initialized to zero, explicitly set it to zero to improve code readability.
   (Acknowledged by the client)

8.3. BillBuster.sol

8.3.1. Low

1. **Issue:** Missing zero address validation
   Function name: constructor

```
constructor(address _token) {
  token = _token;
  // set the Tax fee to be 0.5%
  setTaxFee(5, 1);
}
```

   Description:
   The constructor parameter _token is not checked for zero address validation before setting it to the state variable.
   - token = _token

Resolution:
Check that the address is not zero before setting the value to the state variable.
Eg: require(_token != address(0), "Address cannot be zero");
(Acknowledged by the client)

8.3.2. Informational

1. **Suggestion:** Be aware of the functions that have onlyOwner modifier
   Function name: renounceOwnership

   Description: The contract is Ownable. Renouncing ownership will leave the contract without an owner, thereby removing any functionality that is only available to the owner.
   (Acknowledged by the client)

9. Conclusion

Blockstars received three smart contracts for **STANDARD** auditing from DLTX.
We have done automation tests and manual audits for REMI, InvstorVesting, and BillBuster contracts. The audit used static, dynamic analysis, and manual code reviews for each function in the contracts.

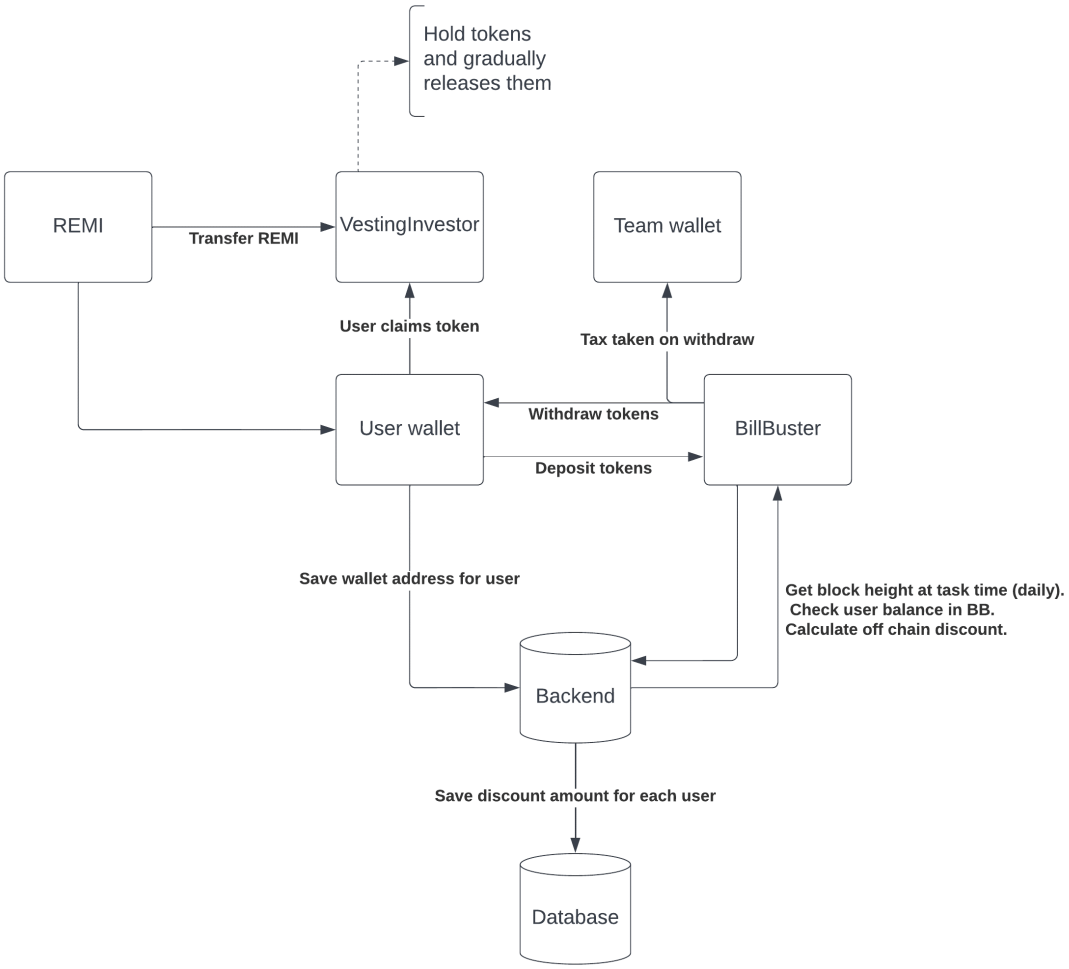
There are no major vulnerabilities in the REMI token contract. Safe (safe for deployment)

There were 1 moderate, 1 low, and 3 informational levels of vulnerabilities found in the InvestorVesting contract. Moderate Vulnerable

And 1 low, and 1 informational level of vulnerabilities identified in the BillBuster contract. Low Vulnerable (safe for deployment)

10. Appendix

10.1. Functional Flow Chart (REMI, InvestorVesting, and BillBuster)



10.2. Score checklists

10.2.1. Documentation Quality Checklist

#	Documentation Quality	Status
1	UML diagram containing all use cases	Not Provided
2	System flow chart containing functionalities	Passed
3	System Requirements and Specifications (SRS)	Not Provided
4	Business logic documentation	Partially Passed

Score = ((1 + 0.5)/4)*10 = **4/10**

10.2.2. Code Quality Checklist

#	Code Quality	REMI	Investor Vesting	BillBuster
1	Code readability	Passed	Passed	Passed
2	Use of comments with function explanations (What logic, input parameters, expected outputs, accessibilities)	Passed	Passed	Passed
3	Error handling and logging (Use assert(), require(), revert() properly	N/A	Passed	Failed
4	Use modifiers only for checks	N/A	Passed	Passed
5	Beware rounding with integer division	Passed	Passed	Passed
6	Be aware of the tradeoffs between abstract contracts and interfaces	Passed	Passed	Passed
7	Keep fallback functions simple	N/A	N/A	N/A
8	Check data length in fallback functions	N/A	N/A	N/A
9	Explicitly mark payable functions	N/A	Passed	Passed
10	Explicitly mark visibility in functions and state variables	N/A	Passed	Passed
11	Lock pragmas to specific compiler version	Passed	Passed	Passed
12	Use events to monitor contract activity	N/A	Failed	Passed
13	Be aware that 'Built-ins' can be shadowed	Passed	Passed	Passed
14	Avoid using tx.origin	Passed	Passed	Passed
15	Avoid using block Timestamp manipulation	Passed	Failed	Passed
16	Use interface type instead of the address for type safety	Passed	Passed	Passed

Score (REMI) = (9/9)*10 = 10/10
Score (Investor Vesting) = (12/14)*10 = 8/10
Score (Bill Buster) = (13/14)*10 = 9/10

10.2.3. Architecture Quality Checklist

#	Architecture Quality	REMI	Investor Vesting	BillBuster
1	Check if the contract is upgradable	N/A	N/A	N/A
2	If upgradable, is proper access control used?	N/A	N/A	N/A
3	Use of latest compiler version	Passed	Passed	Passed
4	Avoid nightly build compiler version	Passed	Passed	Passed
5	Check if contract is haltable	Passed	Passed	Passed
6	If contract is haltable, make sure the accessibility of other emergency functions is available	N/A	N/A	N/A
7	Use of proper constructor	Passed	Passed	Passed
8	Proper access control used	Passed	Passed	Passed
9	Proper beneficiary management	N/A	Passed	Passed

Score (REMI) = (6/6)*10 = 10/10
Score (Investor Vesting) = (6/6)*10 = 10/10
Score (Bill Buster) = (6/6)*10 = 10/10

10.2.4. Project Quality Score Calculations

#	Score Type	Weight	REMI	InvestorVesting	BillBuster
1	Documentation Quality	10%	4/10	4/10	4/10
2	Code Quality	40%	10/10	8/10	9/10
3	Architecture Quality	50%	10/10	10/10	10/10

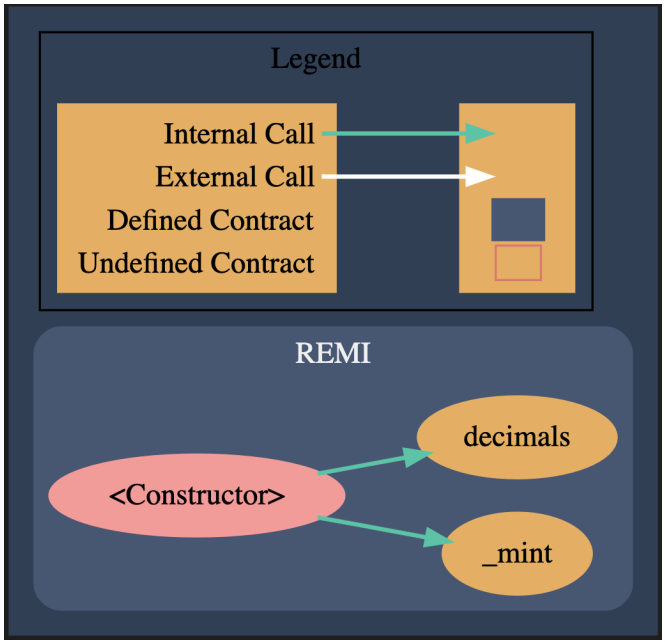
Final Summary Calculation

Documentation = 10% = 0.1*4/10 = 0.04 = **4%**
Code Quality = 40% = 0.4*((1 + 0.8 + 0.9) / 3) / 10)= 0.36 = **36%**
Architecture Quality = 50% = 0.5*((1 + 1 + 1) /3) / 10 = 0.5 = **50%**

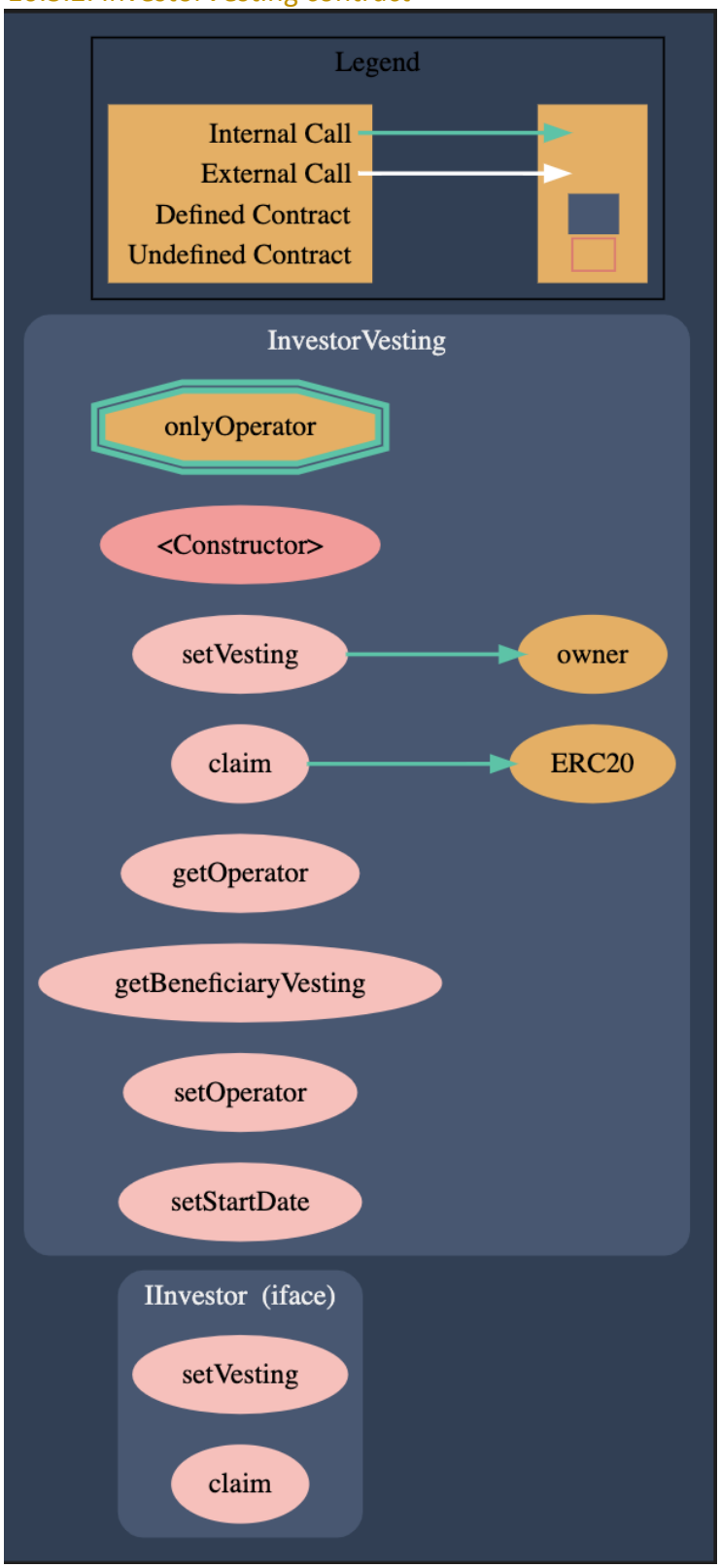
Total Project Quality = 5% + 34% + 50% = **90%**

10.3. Smart Contracts diagrams

10.3.1. REMI contract Flow



10.3.2. InvestorVesting contract



10.3.3. BillBuster contract