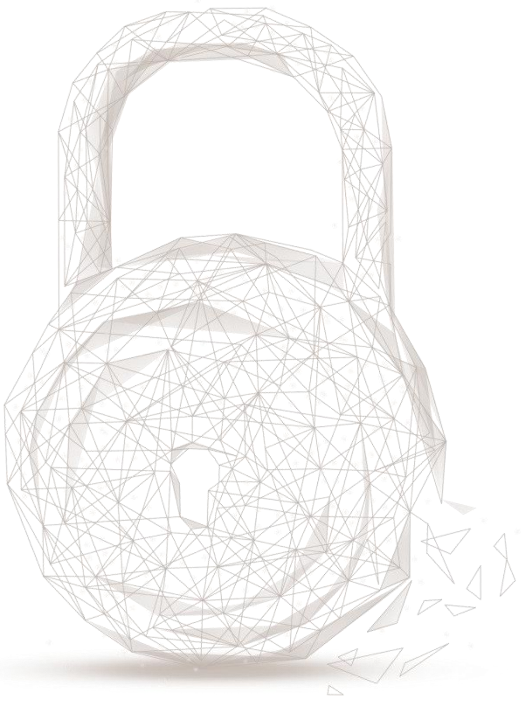
Smart Contract Audit Report for AElf Quadratic Funding



Audit Number: 202112312134

Contract Name: AElf Quadratic Funding

Deployment Platform: AElf

Audit Contract Link Address:

Codebase	https://github.com/AElfProject/aelf-quadratic-funding-contract
Initial Commit	7c22f771c4ba3edc2ff49f72e5b4bbb364466720
final Commit	1c51d5684df306aabd5b20a196e6465820a8d068

Audit Start Date: 2021.12.20

Audit Completion Date: 2021.12.31

Audit Result: Pass

Audit Team: Beosin Technology Co. Ltd.

Audit Results Overview

Beosin Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of AElf Quadratic Funding contract, including Coding Conventions, General Vulnerability and Business Security. **After auditing, the** AElf Quadratic Funding contract **was found to have 10 risk items: 1 High-risk, 1 Medium-risk, 4 Low-risk and 4 Info. As of the completion of the audit, all risk items [part of the risk items] have been fixed or properly handled. The overall result of the AElf Quadratic Funding contract is Pass.** The following is the detailed audit information for this project.

Index	Risk items	Risk level	Fix results status
Funding-1	The logic of token transferring is missing in the *Donate* function	High	Fixed
Funding-2	The input parameters of the function can be negative	Medium	Fixed
Funding-3	The *GetGrantsOf* function does not consider whether the corresponding Project is banned, and the return value may be inaccurate	Low	Fixed
Funding-4	Voted event is not triggered in the *Vote* function	Low	Fixed
Funding-5	The functional authority of the *DangerSetTime* function is slightly higher	Low	Acknowledge
Funding-6	The *Withdraw* function can be called by any user	Low	Fixed
Funding-7	The error message of the *RoundOver* function is inaccurate	Info	Fixed
Funding-8	Lack of an interface to query whether the specified project is banned	Info	Fixed
Funding-9	The collision probability when calculating the project id based on the address	Info	Fixed
Funding-10	The query result of the *GetProjectOf* function is inaccurate		

Table 1. Key Audit Findings

Risk description:

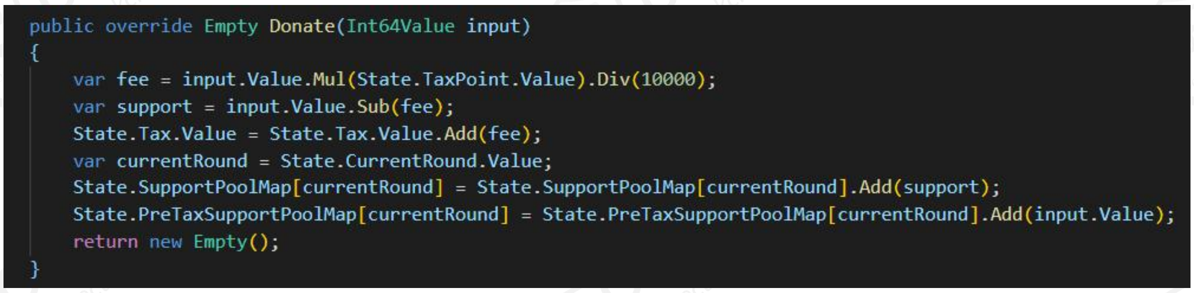
Funding-5 is not repaired, and then the project manager can call the *DangerSetTime* function to modify the start and end time of any round, and re-open the corresponding project and votes.

The project party confirms that this function is a business that must exist in this project, and promises to properly keep the administrator address.

Findings

[Funding-1 High] The logic of token transferring is missing in the *Donate* function

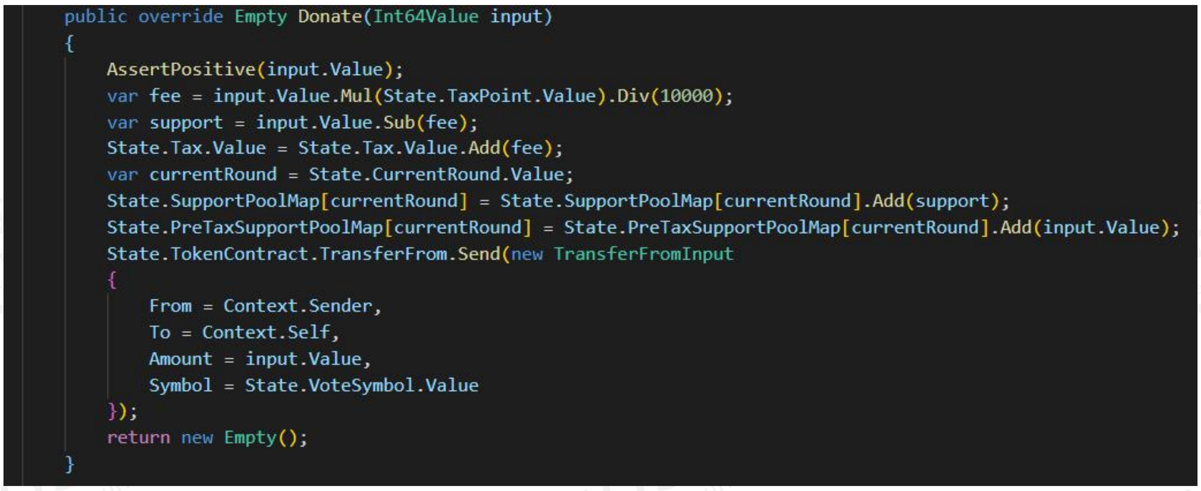
- **Description:** As shown in the figure below, when using the *Donate* function to donate, only the relevant data is updated and the real token transfer is performed.



```
public override Empty Donate(Int64Value input)
{
    var fee = input.Value.Mul(State.TaxPoint.Value).Div(10000);
    var support = input.Value.Sub(fee);
    State.Tax.Value = State.Tax.Value.Add(fee);
    var currentRound = State.CurrentRound.Value;
    State.SupportPoolMap[currentRound] = State.SupportPoolMap[currentRound].Add(support);
    State.PreTaxSupportPoolMap[currentRound] = State.PreTaxSupportPoolMap[currentRound].Add(input.Value);
    return new Empty();
}
```

Figure 1 source code of function *Donate*

- **Fix recommendations:** Add the logic of the token transfer.
- **Fix results: Fixed.** The fix result is shown in the figure below:



```
    public override Empty Donate(Int64Value input)
    {
        AssertPositive(input.Value);
        var fee = input.Value.Mul(State.TaxPoint.Value).Div(10000);
        var support = input.Value.Sub(fee);
        State.Tax.Value = State.Tax.Value.Add(fee);
        var currentRound = State.CurrentRound.Value;
        State.SupportPoolMap[currentRound] = State.SupportPoolMap[currentRound].Add(support);
        State.PreTaxSupportPoolMap[currentRound] = State.PreTaxSupportPoolMap[currentRound].Add(input.Value);
        State.TokenContract.TransferFrom.Send(new TransferFromInput
        {
            From = Context.Sender,
            To = Context.Self,
            Amount = input.Value,
            Symbol = State.VoteSymbol.Value
        });
        return new Empty();
    }
```

Figure 2 source code of function *Donate* (fixed)

[Funding-2 Medium] The input parameters of the function can be negative

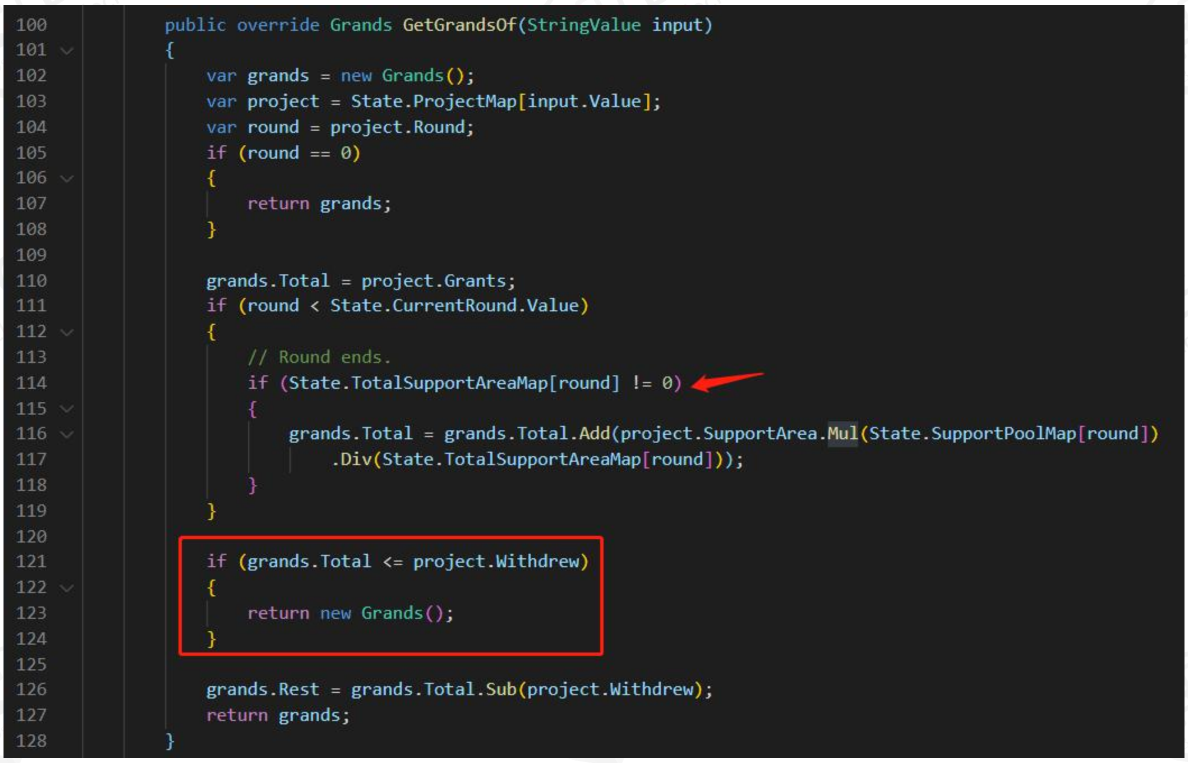
- **Description:** The numeric parameters of many functions in the contract are of type int, so if a negative number is passed in, unexpected results will occur.

- **Fix recommendations:** Check that the input numeric parameters is greater than 0.

- **Fix results: Fixed.**

[Funding-3 Low] The *GetGrantsOf* function does not consider whether the corresponding Project is banned, and the return value may be inaccurate

- **Description:** In the *GetGrandsOf* function shown in the following figure, the situation where the corresponding project is banned is not considered; in addition, when grands.Total <= project.Withdrew, it should not directly return empty Grands, because grands.Total at this time is not 0.



```
        public override Grands GetGrandsOf(StringValue input)
        {
            var grands = new Grands();
            var project = State.ProjectMap[input.Value];
            var round = project.Round;
            if (round == 0)
            {
                return grands;
            }

            grands.Total = project.Grants;
            if (round < State.CurrentRound.Value)
            {
                // Round ends.
                if (State.TotalSupportAreaMap[round] != 0)
                {
                    grands.Total = grands.Total.Add(project.SupportArea.Mul(State.SupportPoolMap[round])
                        .Div(State.TotalSupportAreaMap[round]));
                }
            }

            if (grands.Total <= project.Withdrew)
            {
                return new Grands();
            }

            grands.Rest = grands.Total.Sub(project.Withdrew);
            return grands;
        }
```

Figure 3 source code of function *GetGrandsOf*

- **Fix recommendations:** Update the code logic of this function.
- **Fix results: Fixed.** The fix result is shown in the figure below:

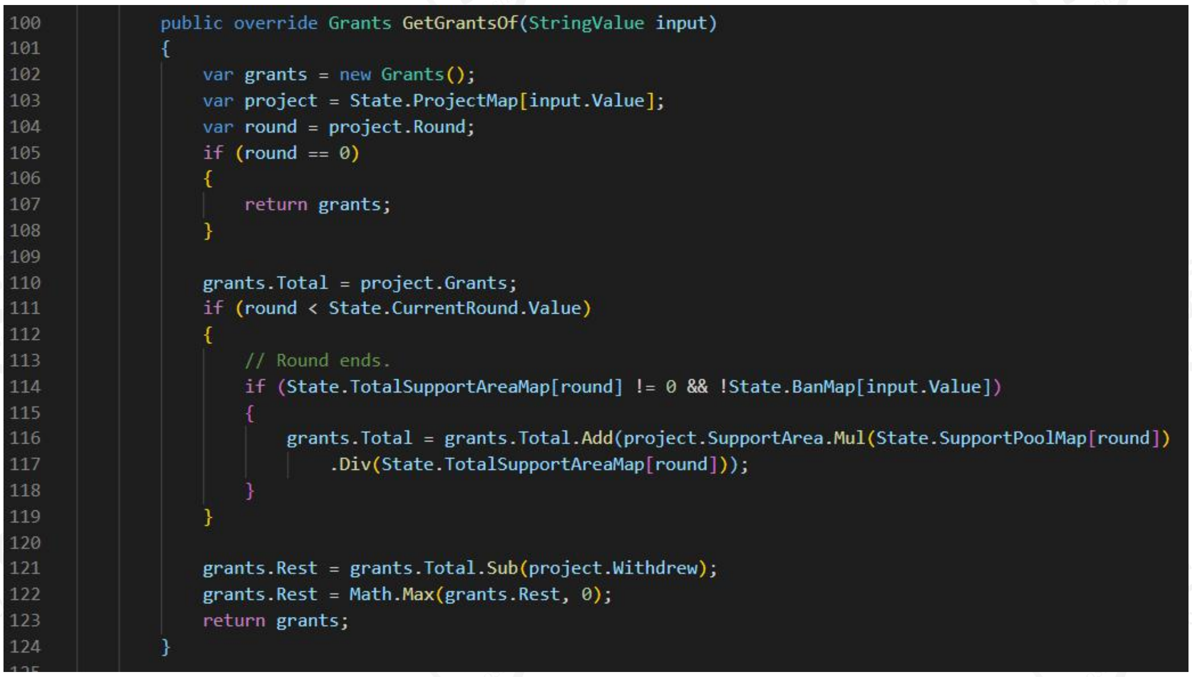
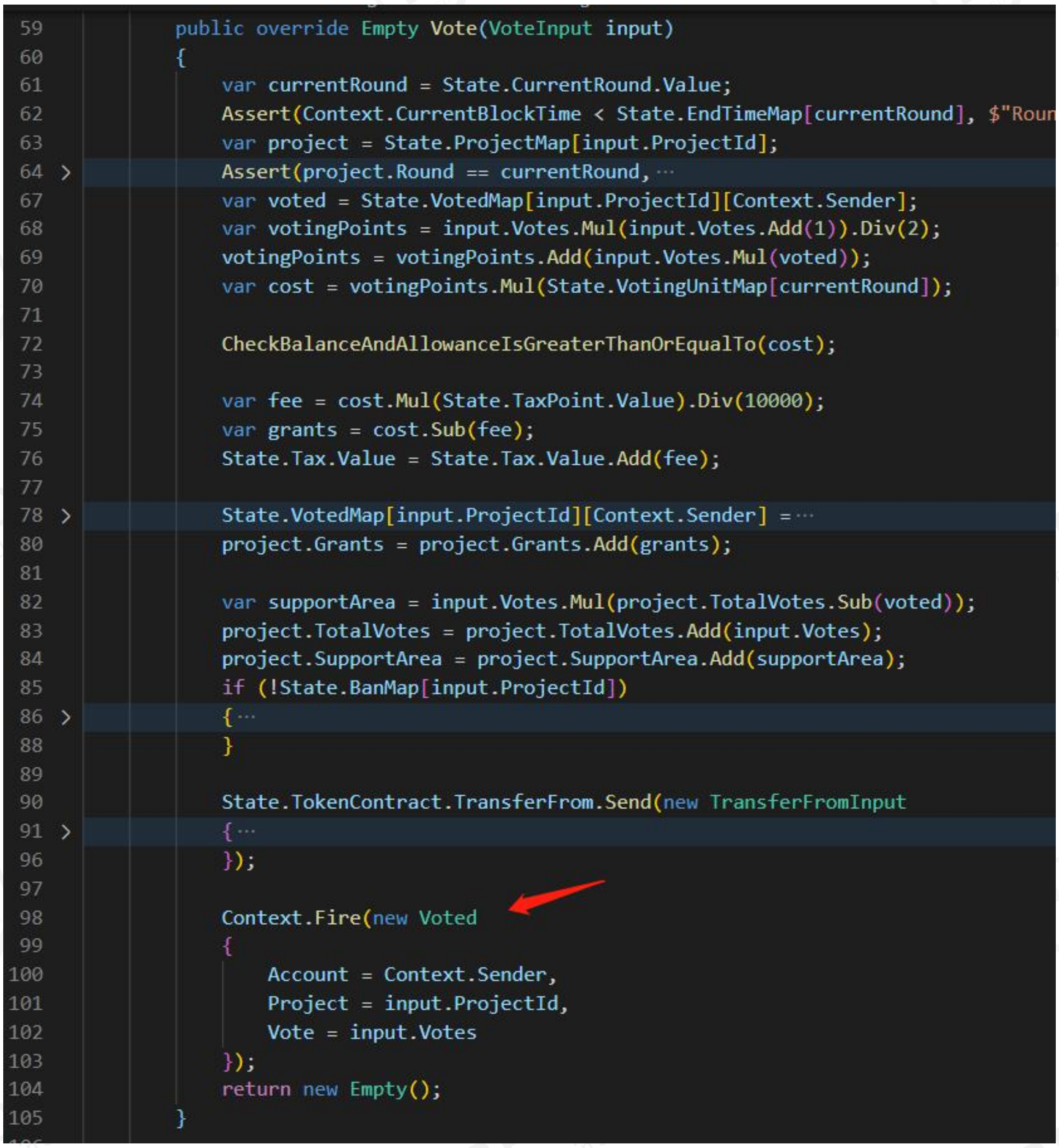

```
public override Grants GetGrantsOf(StringValue input)
{
    var grants = new Grants();
    var project = State.ProjectMap[input.Value];
    var round = project.Round;
    if (round == 0)
    {
        return grants;
    }

    grants.Total = project.Grants;
    if (round < State.CurrentRound.Value)
    {
        // Round ends.
        if (State.TotalSupportAreaMap[round] != 0 && !State.BanMap[input.Value])
        {
            grants.Total = grants.Total.Add(project.SupportArea.Mul(State.SupportPoolMap[round])
                .Div(State.TotalSupportAreaMap[round]));
        }
    }

    grants.Rest = grants.Total.Sub(project.Withdrew);
    grants.Rest = Math.Max(grants.Rest, 0);
    return grants;
}
```

Figure 4 source code of function *GetGrantsOf* (fixed)

[Funding-4 Low] Voted event is not triggered in the *Vote* function

- **Description:** The Voted event is declared in the contract, but it is not triggered in the *Vote* function.
- **Fix recommendations:** In the *Vote* function, the Voted event is triggered after the vote is completed.
- **Fix results: Fixed.**

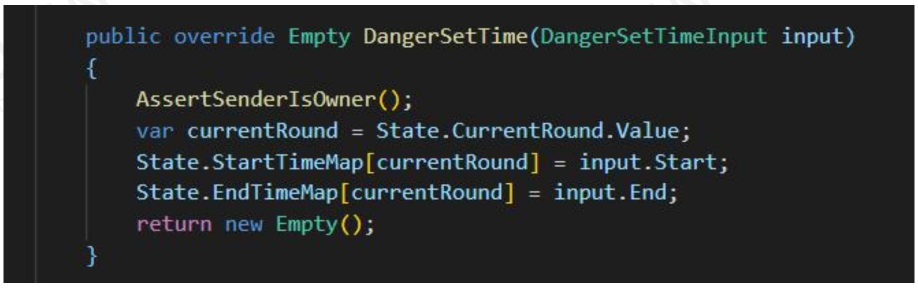


```
public override Empty Vote(VoteInput input)
{
    var currentRound = State.CurrentRound.Value;
    Assert(Context.CurrentBlockTime < State.EndTimeMap[currentRound], $"Roun
    var project = State.ProjectMap[input.ProjectId];
    Assert(project.Round == currentRound,···
    var voted = State.VotedMap[input.ProjectId][Context.Sender];
    var votingPoints = input.Votes.Mul(input.Votes.Add(1)).Div(2);
    votingPoints = votingPoints.Add(input.Votes.Mul(voted));
    var cost = votingPoints.Mul(State.VotingUnitMap[currentRound]);

    CheckBalanceAndAllowanceIsGreaterThanOrEqualTo(cost);

    var fee = cost.Mul(State.TaxPoint.Value).Div(10000);
    var grants = cost.Sub(fee);
    State.Tax.Value = State.Tax.Value.Add(fee);

    State.VotedMap[input.ProjectId][Context.Sender] =···
    project.Grants = project.Grants.Add(grants);

    var supportArea = input.Votes.Mul(project.TotalVotes.Sub(voted));
    project.TotalVotes = project.TotalVotes.Add(input.Votes);
    project.SupportArea = project.SupportArea.Add(supportArea);
    if (!State.BanMap[input.ProjectId])
    {···
    }

    State.TokenContract.TransferFrom.Send(new TransferFromInput
    {···
    });

    Context.Fire(new Voted
    {
        Account = Context.Sender,
        Project = input.ProjectId,
        Vote = input.Votes
    });
    return new Empty();
}
```

Figure 5 source code of function *Vote* (fixed)

[Funding-5 Low] The functional authority of the *DangerSetTime* function is slightly higher

- **Description:** The project manager can call the *DangerSetTime* function to modify the start and end time of any round, and re-open the corresponding project and votes.



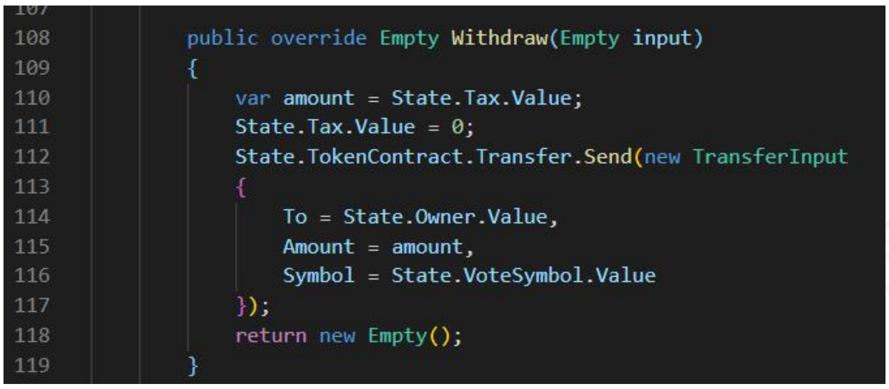
```
public override Empty DangerSetTime(DangerSetTimeInput input)
{
    AssertSenderIsOwner();
    var currentRound = State.CurrentRound.Value;
    State.StartTimeMap[currentRound] = input.Start;
    State.EndTimeMap[currentRound] = input.End;
    return new Empty();
}
```

Figure 6 source code of function *DangerSetTime*

- **Fix recommendations:** Delete this function.
- **Fix results: Acknowledge.** The project party confirms that this function is a business that must exist in this project, and promises to properly keep the administrator address.

[Funding-6 Low] The *Withdraw* function can be called by any user

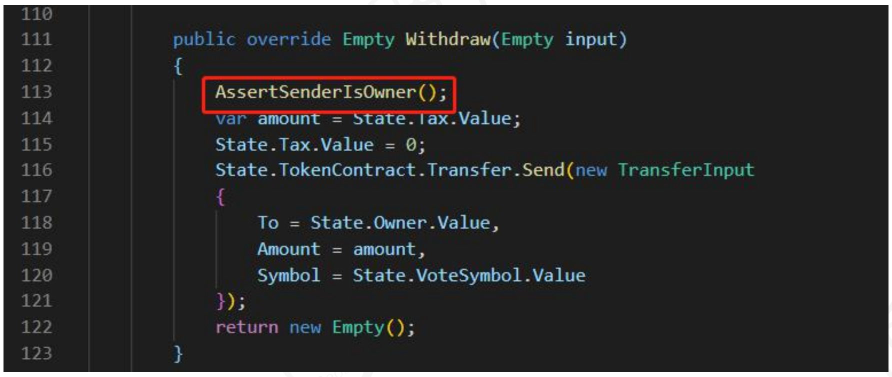
- **Description:** As shown in the figure below, the *Withdraw* function can be used to extract the tax fee, but this function can be called by any user to extract the tax fee to the owner address, which is inconsistent function permission control with the solidity implementation version。



```
public override Empty Withdraw(Empty input)
{
    var amount = State.Tax.Value;
    State.Tax.Value = 0;
    State.TokenContract.Transfer.Send(new TransferInput
    {
        To = State.Owner.Value,
        Amount = amount,
        Symbol = State.VoteSymbol.Value
    });
    return new Empty();
}
```

Figure 7 source code of function *Withdraw*

- **Fix recommendations:** Restrict the caller of this function must be the owner.
- **Fix results: Fixed.**



```
public override Empty Withdraw(Empty input)
{
    AssertSenderIsOwner();
    var amount = State.Tax.Value;
    State.Tax.Value = 0;
    State.TokenContract.Transfer.Send(new TransferInput
    {
        To = State.Owner.Value,
        Amount = amount,
        Symbol = State.VoteSymbol.Value
    });
    return new Empty();
}
```

Figure 8 source code of function *Withdraw* (fixed)

[Funding-7 Info] The error message of the *RoundOver* function is inaccurate

- **Description:** As shown in the figure below, the variable here should not be Context.CurrentBlockTime, but currentRound.

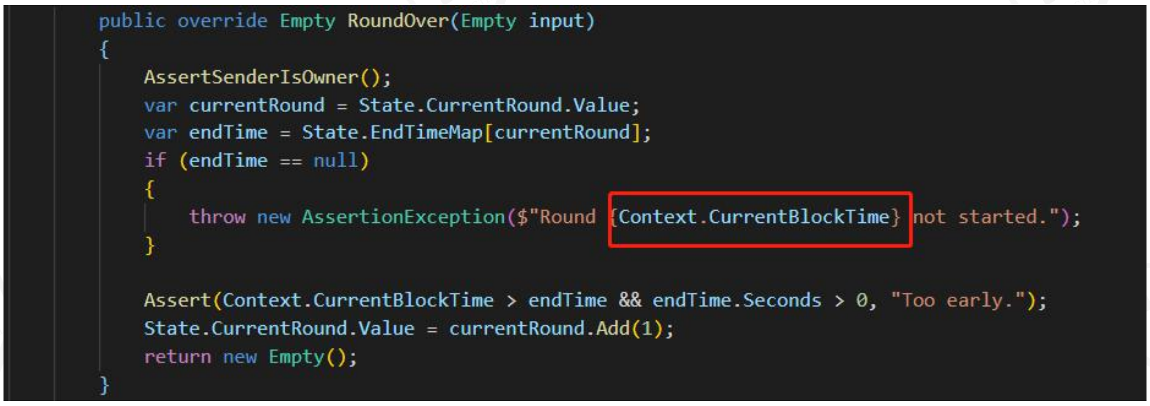

```
public override Empty RoundOver(Empty input)
{
    AssertSenderIsOwner();
    var currentRound = State.CurrentRound.Value;
    var endTime = State.EndTimeMap[currentRound];
    if (endTime == null)
    {
        throw new AssertionException($"Round {Context.CurrentBlockTime} not started.");
    }

    Assert(Context.CurrentBlockTime > endTime && endTime.Seconds > 0, "Too early.");
    State.CurrentRound.Value = currentRound.Add(1);
    return new Empty();
}
```

Figure 9 source code of function *RoundOver*

- **Fix recommendations:** Modify Context.CurrentBlockTime to currentRound.
- **Fix results: Fixed.**

[Funding-8 Info] Lack of an interface to query whether the specified project is banned

- **Description:** The contract lacks an interface to query whether the specified project is banned.
- **Fix recommendations:** If there are related business requirements, add the corresponding query interface.
- **Fix results: Fixed**

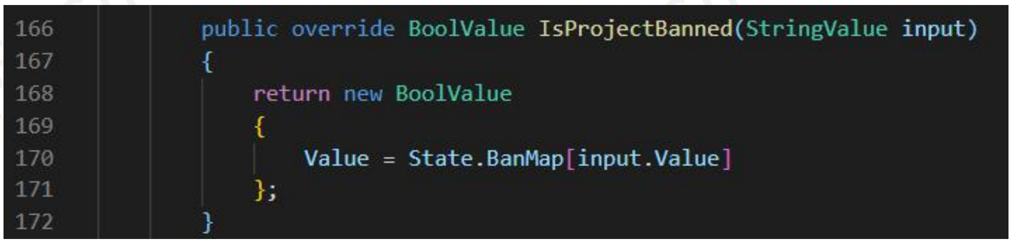

```
        public override BoolValue IsProjectBanned(StringValue input)
        {
            return new BoolValue
            {
                Value = State.BanMap[input.Value]
            };
        }
```

Figure 10 source code of function *IsProjectBanned*

[Funding-9 Info] The collision probability when calculating the project id based on the address.

- **Description:** The method of determining the project id used in the current contract may have the possibility of collision

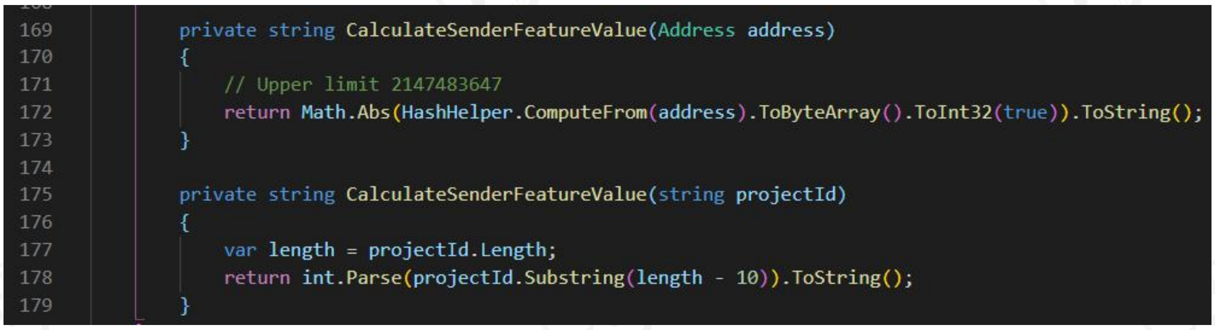
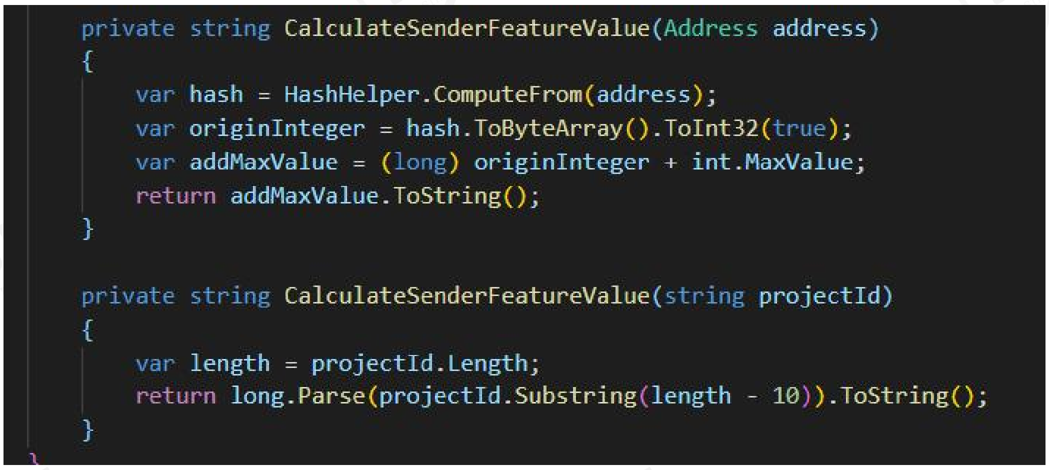

```
        private string CalculateSenderFeatureValue(Address address)
        {
            // Upper limit 2147483647
            return Math.Abs(HashHelper.ComputeFrom(address).ToByteArray().ToInt32(true)).ToString();
        }

        private string CalculateSenderFeatureValue(string projectId)
        {
            var length = projectId.Length;
            return int.Parse(projectId.Substring(length - 10)).ToString();
        }
```

Figure 11 source code of functions *CalculateSenderFeatureValue*

- **Fix recommendations:** Use the new method of generating project id.
- **Fix results: Fixed.**

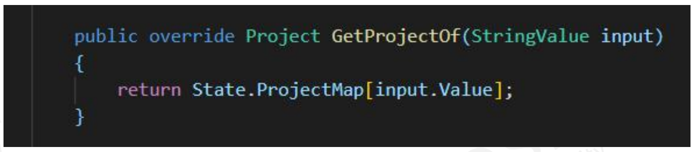


```
    private string CalculateSenderFeatureValue(Address address)
    {
        var hash = HashHelper.ComputeFrom(address);
        var originInteger = hash.ToByteArray().ToInt32(true);
        var addMaxValue = (long) originInteger + int.MaxValue;
        return addMaxValue.ToString();
    }

    private string CalculateSenderFeatureValue(string projectId)
    {
        var length = projectId.Length;
        return long.Parse(projectId.Substring(length - 10)).ToString();
    }
}
```

Figure 12 source code of functions *CalculateSenderFeatureValue* (fixed)

[Funding-10 Info] The query result of the *GetProjectOf* function is inaccurate

➢ **Description:** The *GetProjectOf* function does not consider whether the project is banned during the query, and it is still the returned stored SupportArea value.

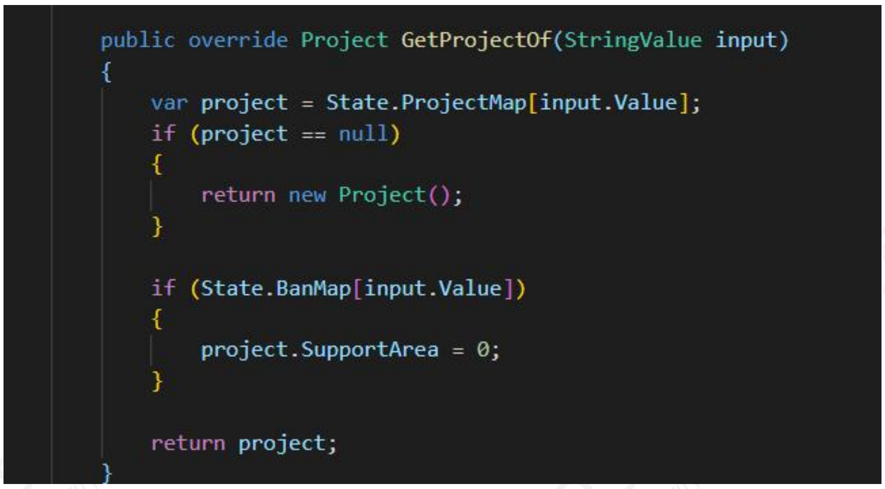


```
public override Project GetProjectOf(StringValue input)
{
    return State.ProjectMap[input.Value];
}
```

Figure 13 source code of function *GetProjectOf*

➢ **Fix recommendations:** When the corresponding Project is banned, the return value of SupportArea is 0.

➢ **Fix results: Fixed.**



```
public override Project GetProjectOf(StringValue input)
{
    var project = State.ProjectMap[input.Value];
    if (project == null)
    {
        return new Project();
    }

    if (State.BanMap[input.Value])
    {
        project.SupportArea = 0;
    }

    return project;
}
```

Figure 14 source code of function *GetProjectOf*（fixed）

OtherAuditItemsDescriptions

This project uses the Quadratic Funding model. At present, relevant studies have shown that this mode is vulnerable to multiple attack vectors (https://medium.com/block-science/how-to-attack-and-defend-quadratic-funding-a10f0152f069). The most prominent among them are sybil attacks, where the attacker creates many fake accounts to game the system and collusion, where malicious real users secretly coordinate among themselves to game the system.

These problems cannot be prevented from the contract. It is recommended that the project take preventive measures.

Appendix 1 Vulnerability Severity Level

Vulnerability Level	Description	Example
Critical	Vulnerabilities that lead to the complete destruction of the project and cannot be recovered. It is strongly recommended to fix.	Malicious tampering of core contract privileges and theft of contract assets.
High	Vulnerabilities that lead to major abnormalities in the operation of the contract due to contract operation errors. It is strongly recommended to fix.	Unstandardized docking of the USDT interface, causing the user's assets to be unable to withdraw.
Medium	Vulnerabilities that cause the contract operation result to be inconsistent with the design but will not harm the core business. It is recommended to fix.	The rewards that users received do not match expectations.
Low	Vulnerabilities that have no impact on the operation of the contract, but there are potential security risks, which may affect other functions. The project party needs to confirm and determine whether the fix is needed according to the business scenario as appropriate.	Inaccurate annual interest rate data queries.
Info	There is no impact on the normal operation of the contract, but improvements are still recommended to comply with widely accepted common project specifications.	It is needed to trigger corresponding events after modifying the core configuration.

Appendix 2 Disclaimer

This report is made in response to the project code. No description, expression or wording in this report shall be construed as an endorsement, affirmation or confirmation of the project. This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin. Due to the technical limitations of any organization, this report conducted by Beosin still has the possibility that the entire risk cannot be completely detected. Beosin disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin Technology.

Appendix 3 About Beosin

BEOSIN is a leading global blockchain security company dedicated to the construction of blockchain security ecology, with team members coming from professors, post-docs, PhDs from renowned universities and elites from head Internet enterprises who have been engaged in information security industry for many years. BEOSIN has established in-depth cooperation with more than 100 global blockchain head enterprises; and has provided security audit and defense deployment services for more than 1,000 smart contracts, more than 50 blockchain platforms and landing application systems, and nearly 100 digital financial enterprises worldwide. Relying on technical advantages, BEOSIN has applied for nearly 50 software invention patents and copyrights.

BEOSIN

Blockchain Security

Official Website

https://beosin.com

Twitter

https://twitter.com/Beosin_com