4th DECEMBER 2018

-

version 1.0.0

BApp(Blockchain Application) Smart Contract Security Audit and General Analysis

# **Table of Contents**

*2 Handled, 13 Unhandled (1 Critical, 2 Major, 7 Minor, 3 Informative) Issues Found*

[Table of Contents](#_kz6tr9rj4yoi)

[01. Introduction](#_1fob9te)

[02. Contracts subject to audit](#_3znysh7)

[03. About HAECHI LABS](#_6uooawayhyh4)

[04. Issues Found](#_xepd3vhbus6d)

[04.a. Handled Issues](#_1sn5lelldd7r)

[HANDLED : Re-Entrancy](#_z48i1k3m14c5)

[HANDLED : Issue of inflow of unexpected fund into the contract](#_be459tm53rdf)

[04.b. Unhandled Issues](#_3mrwbbirwaob)

[CRITICAL : BlasqUser#createAdminServicePolicy() cannot guarantee type-policy one-to-one correspondence and QnaPolicy may stop operating. [Unintended Behavior]](#_hwfmweo5x23v)

[MAJOR : Fix length of policy array to 12 in QnaPolicy#constructor() and QnaPolicy#createPolicy() . [Unintended Behavior]](#_2n0x0mug9ad6)

[MAJOR : Wrong type casting exists in QnaPolicy#createSelectHistory(). [Wrong Type Casting]](#_1mcy7wz1f8wj)

[MINOR : Intege](#_3wzwgkc9gc3r)[#heading=h.1mcy7wz1f8wj](https://docs.google.com/document/d/1begqe9o3aU7pvpXS0SSZdcMwPkP7jZHKwQeQKYDR4wc/edit#heading=h.1mcy7wz1f8wj)[r Overflow/Underflow may occur in Qna#\_createAnswer() and Qna#\_deleteAnswer (). [Arithmetic Over/Underflow]](#_3wzwgkc9gc3r)

[MINOR : Integer Overflow/Underflow may occur at BlasqUser. [Arithmetic Over/Underflow]](#_chcilk6kq4vo)

[MINOR : Integer Underflow may occur in QnaService#withdrawKlay() and if it occurs, escrowed Klay can also be withdrawn. [Arithmetic Over/Underflow]](#_4wa36yx79ue4)

[MINOR : BlasqUser#deleteUser() and BlasqUser#changeUserState() use different method for deleting users. [Unintended Behavior]](#_dgzhzx9xt)

[MINOR : require() of BlasqUser#migrateUserReputation() cannot block when input userId is 0. [Unintended Behavior]](#_bmqxwt24k3s2)

[MINOR : constructor(), createAdminService(), and createAdminBlasqUser() of QnaPolicy may accept zero address. [Unintended Behavior]](#_dr5bk08ikotq)

[MINOR : User’s Klay could be bound to Qna#\_escrowKlayForQuestion(). [Unintended Behavior]](#_mvwsg5ggylg0)

[INFORMATIVE : At QnaService#constructor(), contract is still created even when the address of newQnaPolicy and newBlasqUser are zero addresses. [Wrong Argument]](#_hqc6duxmplok)

[INFORMATIVE : QnaPolicy address could be modified even when newQnaPolicy, a parameter of QnaService#createQnaPolicy(), is zero address. [Wrong Argument]](#_nghd3ki6nhzb)

[INFORMATIVE : BlasqUser address could be modified even when newBlasqUser, a parameter of QnaService#createBlasqUser(), is zero address. [Wrong Argument]](#_q1886m5u3k6l)

[05. Tech Consulting](#_b6h42mmpau2p)

[05.a. Gas Optimization](#_17dp8vu)

[GAS OPTIMIZATION : It is recommended to record log data as an Event, instead of storing in the storage.](#_xx7vyyo3suj0)

[GAS OPTIMIZATION : It is recommended to not store the array of Array index in storage.](#_ocj4lpbqpxef)

[GAS OPTIMIZATION : Comprehensive Application](#_yinag3cfvvxf)

[05.b. Best Practices](#_zay4801un0xk)

[BEST PRACTICES : Deploy Management](#_rq39mbo2p0m0)

[BEST PRACTICES : ECverify](#_qwjxryavqpqj)

[BEST PRACTICES : Encoding](#_phb7pv2xd7l8)

[BEST PRACTICES : Enum](#_is7z7ix89fl5)

[BEST PRACTICES : General Function Structure](#_57taz54kpi6u)

[BEST PRACTICES : Migration](#_flp40erdm4q1)

[BEST PRACTICES : BlasqUser#migrateUserReputation() which should only be called during migration is written to be able to be called anytime.](#_f9oqe7jx3n4c)

[BEST PRACTICES : As UpdateReputation, an event of BlasqUser.sol:140 only records changes in point, restoration process for reputation is inefficient.](#_qgllzh5rjg9h)

[05.c. Tips](#_lnxbz9)

[TIPS : There is an unused argument in Qna#\_voting().](#_cvqbge11ahc6)

[TIPS : There are typos in Qna#\_cancleUpVoting() and Qna#\_cancleDownVoting().](#_2emhfsin2ajn)

[TIPS : Larger values could be stored when uint type is used for user.giveReward and user.takeReward, instead of int type.](#_2jtr3wa47wil)

[TIPS : There is a typo in the name of BlasqUser#\_createNewAdminSerivcePolicy().](#_he0l333cwlvd)

[TIPS : Restricting the access to index at outside Array range is not required.](#_g6imcqe6tnch)

[TIPS : Remove unused variable at QnaPolicy.](#_n432zzjvle0b)

[TIPS : Change the name of policyAddr at QnaPolicy#constructor().](#_axkgtxkriub0)

[TIPS : There is an unused input argument at QnaPolicy#createSelectHistory().](#_yt27b51cc89n)

[06. Upgradeable Smart Contract Strategy](#_z3egukwnnng9)

[06.a. Develop Upgradeable Smart Contract with the Unstructured Storage pattern](#_m7sr8c5ida1p)

[07. Conclusion](#_1v8my7o3zbg6)

[08. Test Results](#_ujed9fpg5t8s)

[09. Disclaimer](#_z337ya)

# 

# **01. Introduction**

This audit report covers the smart contract security of the blasq smart contract created by Ground X team. “HAECHI LABS” audited the smart contract codes to ensure that the codes were designed for the purpose outlined in the white paper and other published materials, and review whether the codes are secure.

The codes used for the audit can be found at Github repository of “ground-x/BLASQ-contract-audit” (<https://github.com/ground-x/BLASQ-contract-audit>). The last commit of the code used for the audit is “eb01c20a0b3747ef7010d7e440ec8874903ada50”.

# **02. Contracts subject to audit**

* BlasqUser.sol
* ECVerify.sol
* Qna.sol
* QnaPolicy.sol
* QnaPolicyStandard.sol
* QnaService.sol
* QnaUser.sol
* ServiceAdmin.sol

# 

# **03. About HAECHI LABS**

HAECHI LABS is a leading technology company in blockchain industry, providing the best quality security audits and the powerful tech solutions for blockchain startups and enterprises.

HAECHI LABS has provided blockchain tech solutions that allow you to seamlessly develop services on blockchain and enhance your blockchain development experience. The representative solutions include ‘VVISP’, a powerful open source CLI tool to easily develop, deploy and test smart contract systems and ‘Query Layer’, an effective middleware solution to query blockchain data in real time more easily and faster.

With our mission to deliver our clients the highest quality services and tech solutions, many Fortune 100 corporations, leading blockchain startups and foundations have been greatly satisfied. Our notable clients are Kakao’s blockchain subsidiary (Ground X), SK telecom, Carry Protocol, Bit-Z and so on.

Contact : [hello@haechi.io](mailto:hello@haechi.io)  
Website : <https://haechi.io>

# 

# **04. Issues Found**

The found issues are largely divided into **Handled Issue** and **Unhandled Issue**.

**Handled Issue** is a type of issue that has already been resolved in the written contract, and general information regarding that issue is provided. Analysis on how current build is preventing the issue is also provided so that the customer can prevent the similar weaknesses in the future occasions.

**Unhandled Issue** is a weakness the written contract has missed, and the corresponding issues are categorized into, , , and , according to the gravity of the issues.

| . | Critical issue is fatal security weakness which may cause damage or loss to the user and should be solved. |
| --- | --- |
|  | Major issue is a problem which may cause a security problem or a problem of implementation that is different from the original design intention. |
|  | Minor issue requires correction as it could potentially cause problems. |
| . | Informative issue is a point which may further improve usability or efficiency of the code when modified, although it would not cause any major security issue. |

“HAECHI LABS” recommends Ground X team to resolve all the issues found.

The following descriptions use the format of *{File Name}:{Line Number}* and *{Contract Name}#{Function/Variable Name}*, to specify the code. For example, *Qna.sol:20* refers to the 20th line of ‘Qna.sol’ file and *ContractA#transfer()* refers to the transfer() function of the ‘ContractA’ contract.

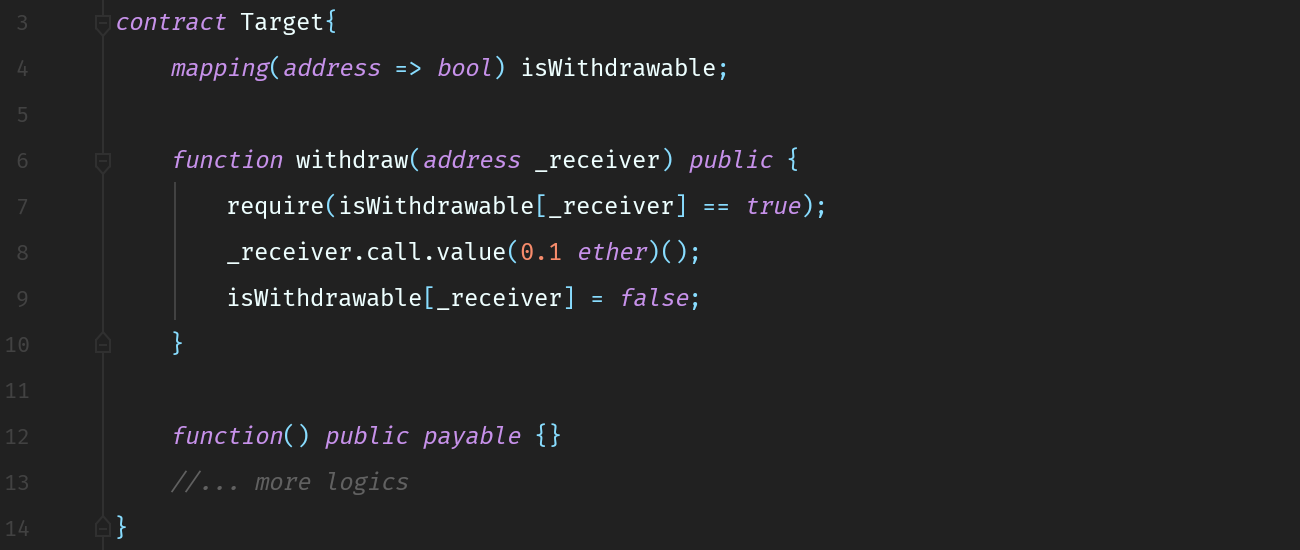
# 04.a. Handled Issues

### 

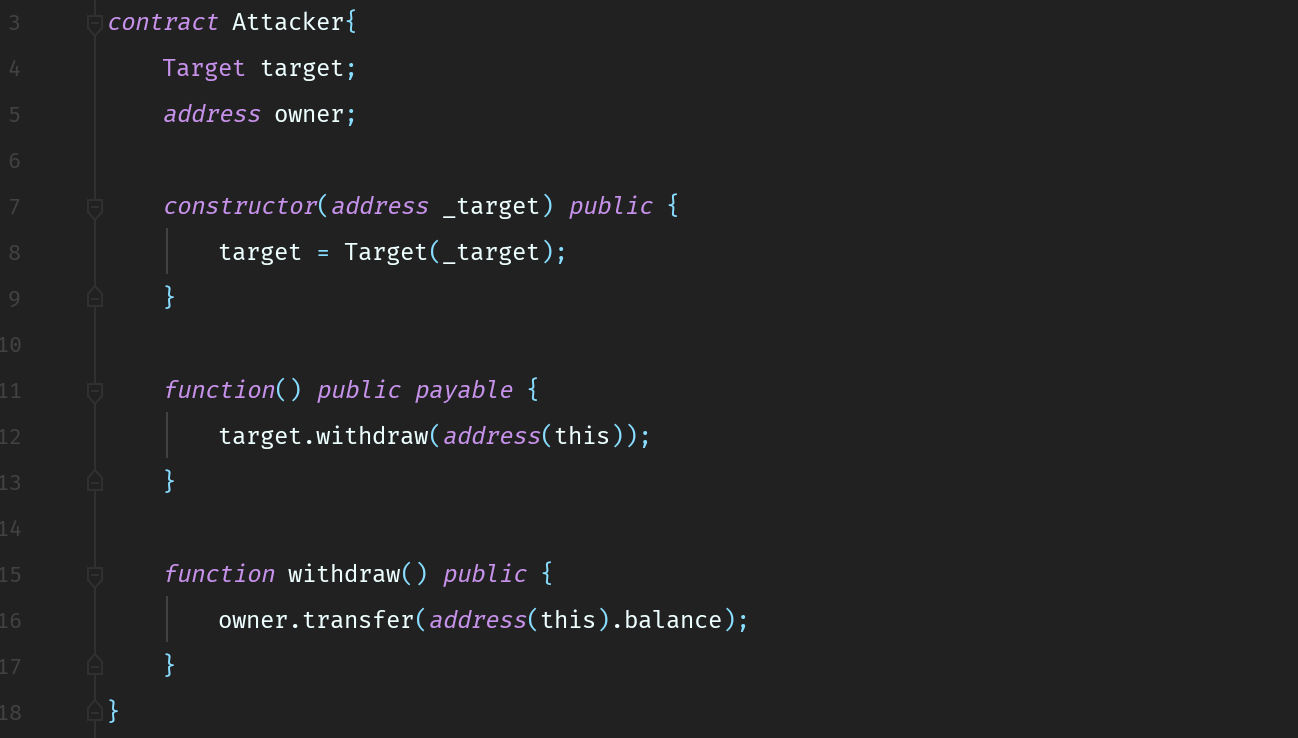
### HANDLED : Re-Entrancy

#### Description

Re-entrancy attack usually occurs when the contract sends money to an unknown address. If the receiving address is not the personal wallet’s address but attacker’s contract address, the attacker may use *fallback()* of the contract to conduct re-entrancy and steal fund from the contract.

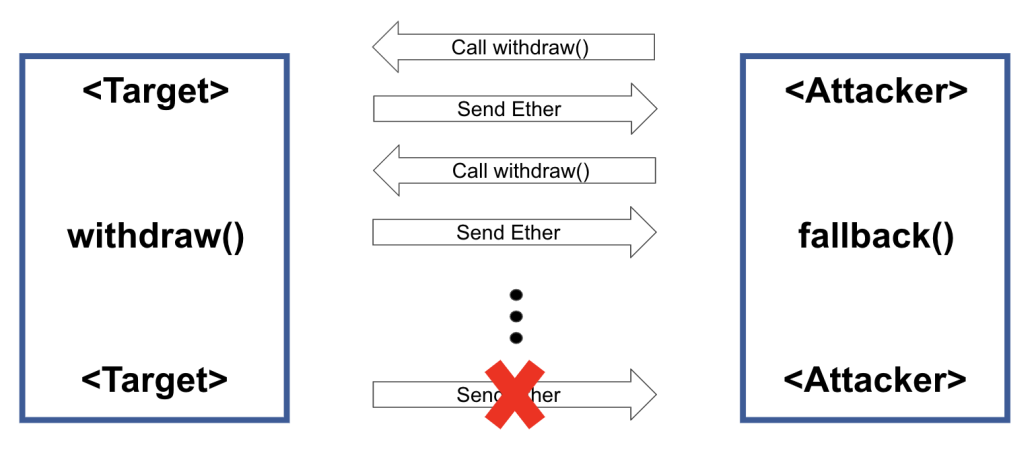
  
[Figure 1] Re-Entrancy Target Contract

Assume that there is a contract called *Target* in which an address whose *isWithdrawable* value is *true* can withdraw 0.1ether, like Figure 1. Once withdrawal is made, *isWithdrawable* is changed into *false* so that only a single withdrawal could be made. However, the attacker can take advantage of that logic to steal the fund from *Target*.

  
[Figure 2] Re-Entrancy Attacker Contract

Attacker creates a contract as shown at [Figure 2] and sets the value of *isWithdrawable* to *true* in *Target* contract. Once *withdraw()* in *Target* contract is invoked in regard to the address of *Attacker* contract, the code runs as follows.

1. *Target* sends 0.1 ethers to *Attacker*.
2. *fallback()* of *Attacker* is run and calls *withdraw()* of *Target*.
3. Because *isWithdrawable* has not been turned *false* yet as it still remains at Target.sol:8, it passes *require* condition at Target.sol:7 and sends another 0.1 ethers to *Attacker*.
4. Phase from 2 to 3 is repeated.

  
[Figure 3] Re-Entrancy Attack

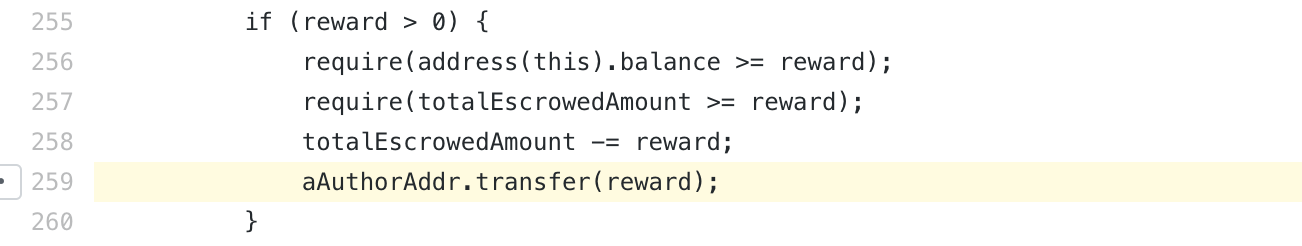
As shown at [Figure 3], *call* and *send* are repeated, and when the sent amount exceeds the amount of total gas, the last created internal transaction fails because of Out of Gas. At that moment, all the transferred ether while making internal transaction would be considered as gas consumption. Even if the execution halts in an exceptional case( e.g. out of gas), no gas is refunded to caller. Therefore the attacker can steal a large amount of fund.

The aforementioned re-entrancy attack could be prevented through the following method.

1. Use *transfer()* or *send()* instead of *call.value(x)()*. As the maximum gas for those functions is limited at 2,300 gas, using those functions will prevent the running of other logics during inter-contract fund transfer.
2. When *call.value(x)()* has to be used inevitably, you must change the state before calling the function. As the state has already been changed, repeated transfer of fund could be prevented.

[The DAO](https://en.wikipedia.org/wiki/The_DAO_(organization)) of Ethereum is the representative example of re-entrancy attack.

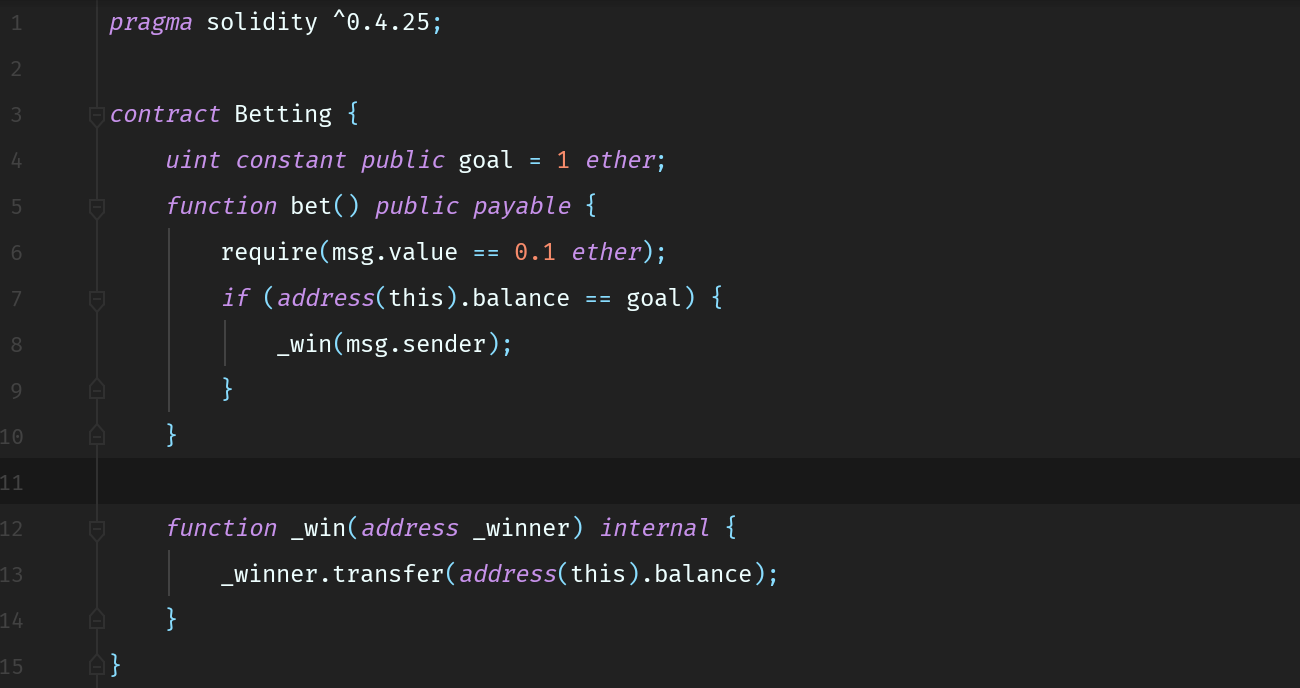
#### How BLASQ Handled the Issue

  
(Qna.sol -  
<https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/Qna.sol#L106>  
<https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/Qna.sol#L259>)  
  
In the current implementation of Qna.sol, there is no risk of re-entrancy attack. First of all, as *transfer()* has been used instead of *call.value(x)()*, any action that exceeds 2,300 gas could be prevented. In addition, in Qna.sol:106, all state changes were made before calling *transfer()* which transfers Klay. Moreover, *QnaState* is set to *Deleted* and also it is always checked whether *QnaState* is *Active*. Lastly, in Qna.sol:259, state changes occur even after the *transfer* function. Nevertheless, *QnaState* is changed into *Complete* in prior to the *transfer* function and always checked. Based on these factors, therefore, re-entrancy attack does not occur.

### HANDLED : Issue of inflow of unexpected fund into the contract

#### Description

This issue mainly occurs when an attacker sends fund to the contract in an unintended way.

  
[Figure 4] Betting Contract

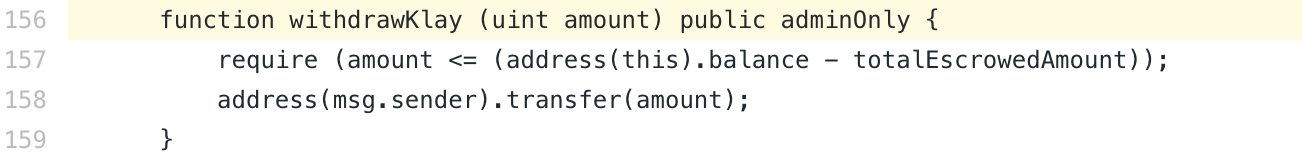
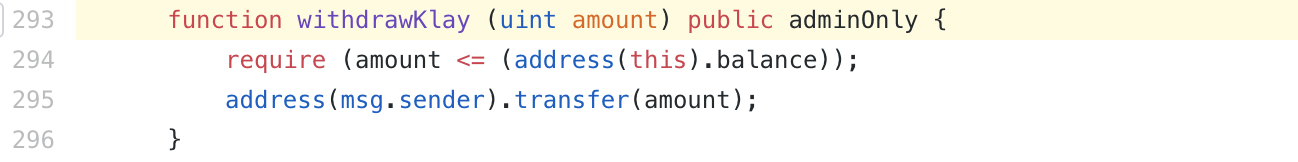
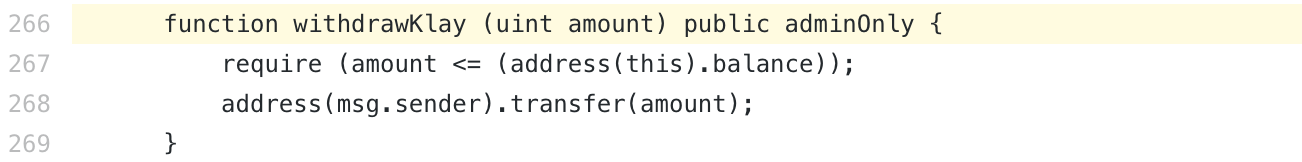
Assume that there is a game contract in which the user deposits ETH and receives accumulated deposit when the deposit reaches a predefined goal, like [Figure 4]. It is a game in which user may bet 0.1 ether through *bet()* and when the accumulated amount reaches 1 ether, the last user wins accumulated 1 ether. While designing this contract, the developer assumes that fund would be deposited only through *bet()* and fixed the unit of betting at 0.1 ether.

However, an attacker may transfer fund in ways other than calling *bet()* function of the contract, and if the attacker transfers 0.05 ether, not 0.1 ether, the game cannot be operated according to the intended way.

First, an attacker may send fund by predicting the address of the contract before the contract is generated. That is because it is easy to know the contract's address in advance since it is derived from *address = keccak256(rlp.encode([account\_address, transaction\_nonce]))*.

Second, even when the contract already has been deployed, an attacker can forcibly transfer ETH through *selfdestruct(address)* function. *selfdestruct(address)* function is a built-in function of Solidity that any contract can use. This function deletes every bytecode of the contract and sends the entire amount of fund in the contract to the *address* which is given as an argument. If an attacker generates a random contract (A), transfers fund to the contract (A) and inputs an address of the target contract (B) as a parameter of *selfdestruct(address)* function, then fund are forcibly transferred to the target contract (B) without running *fallback* function. (Usually, when fund are transferred to the address of the target contract (B) without function call, *fallback* function of the target contract (B) is run, and if *fallback* function is not explicitly defined in the target contract, the transfer is reverted.)

#### How BLASQ Handled the Issue

  
(QnaService.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/QnaService.sol#L156>)  
  
(Qna.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/Qna.sol#L65>)  
  
(BlasqUser.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/BlasqUser.sol#L293>)  
  
(QnaPolicy.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/QnaPolicy.sol#L266>)

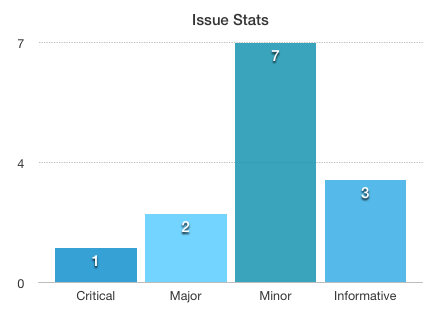
In the current implementation, Blasq is handling this issue in largely two ways.  
The first is *withdrawKlay()* function. As BlasqUser, QnaPolicy, and QnaService, the deploy targets, all have *withdrawKlay()* function, contracts can respond to unexpected increase in deposit by withdrawing Klay.

The second way is using *totalEscrowedAmount* instead of *this.balance*. Because the value of *totalEscrowedAmount* variable only changes within the designated function, it is not affected by the above mentioned attack. Hence, possible bugs regarding fund could be prevented.

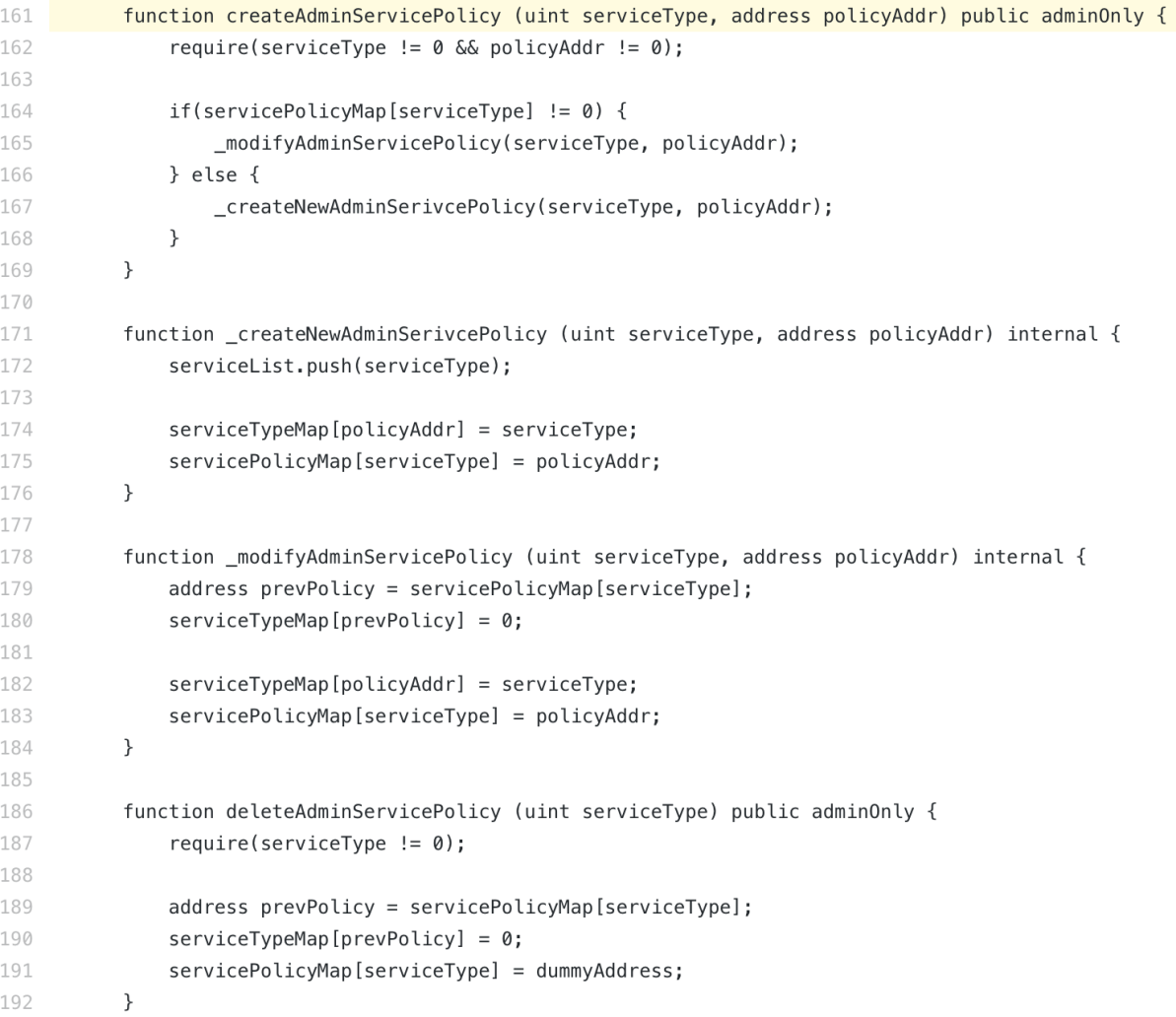
Because of centralization of authority over fund, using *withdrawKlay()* function like BlasqUser and QnaPolicy is not recommended. However, in QnaService.sol:157, only the received fund that was sent in unintended way could be withdrawn by using *totalEscrowedAmount* variable, therefore, it resolves both the attack issue and the authority issue.

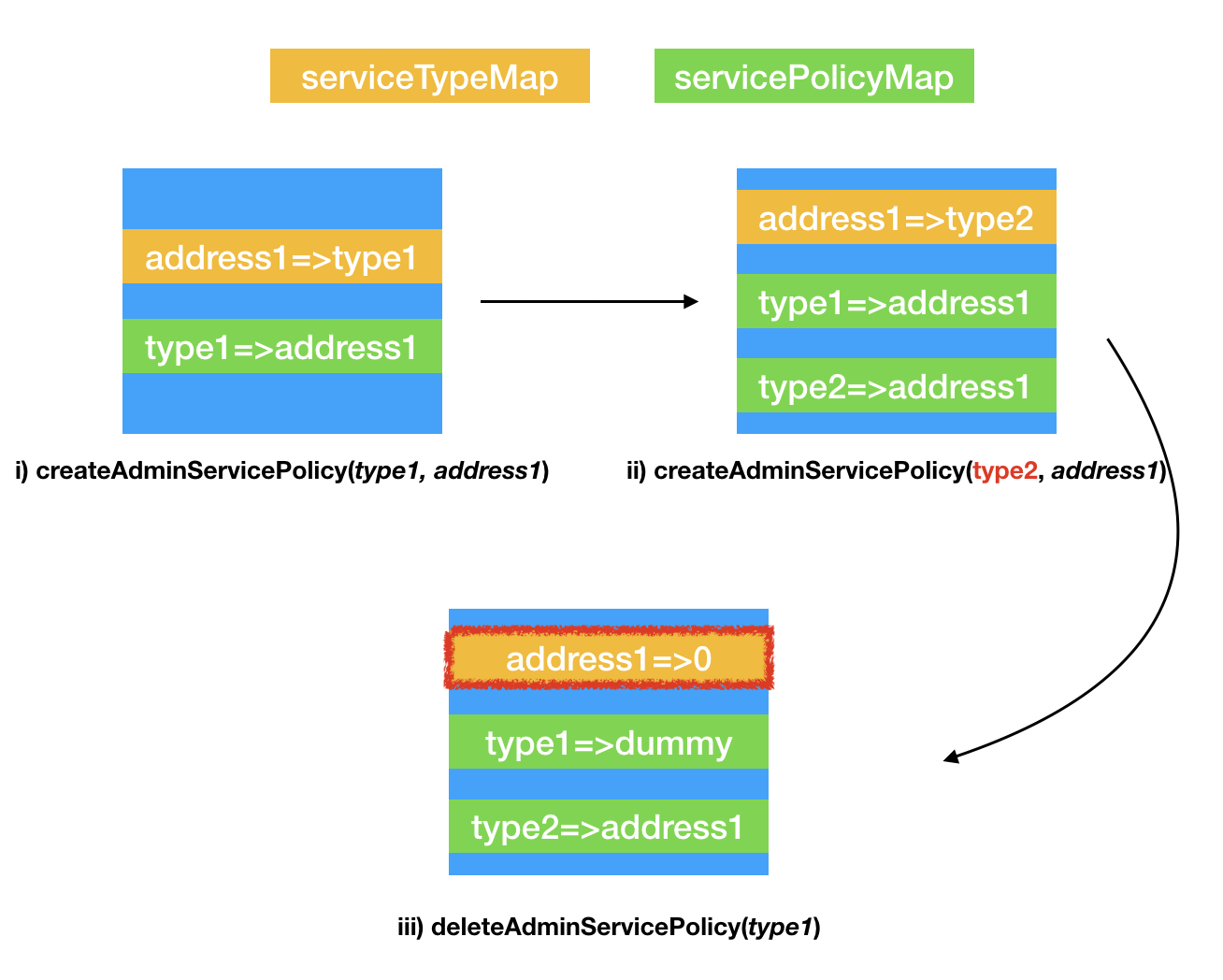
# 

# 04.b. Unhandled Issues

****[그림 5] Issue Stats

### CRITICAL : *BlasqUser#createAdminServicePolicy()* cannot guarantee *type*-*policy* one-to-one correspondence and *QnaPolicy* may stop operating. [Unintended Behavior]

  
(BlasqUser.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/BlasqUser.sol#L161>)

  
[Figure 6] Exploit Scenario

#### Exploit Scenario

1. Call *createAdminServicePolicy(address1, type1).*
2. Call *createAdminServicePolicy(address1, type2).*  
   As described in [Figure 6], both *type1* and *type2* are mapping with *address1*. That is, although policy type and its address should be in one-to-one correspondence relations, that relations collapses in this scenario.
3. Call *deleteAdminServicePolicy(type1)*.  
   Assume that after ii), now you are trying to delete *type1* policy. In that case, as you can see at [Figure 6], the corresponding value of *address1* is set to 0 in *serviceTypeMap*. As a result, the type data corresponding to *address1* is erased and Dapp does not properly operate.

#### Problem Statement

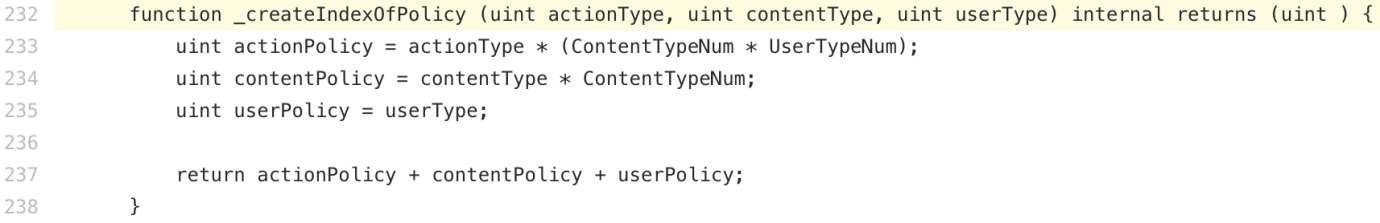
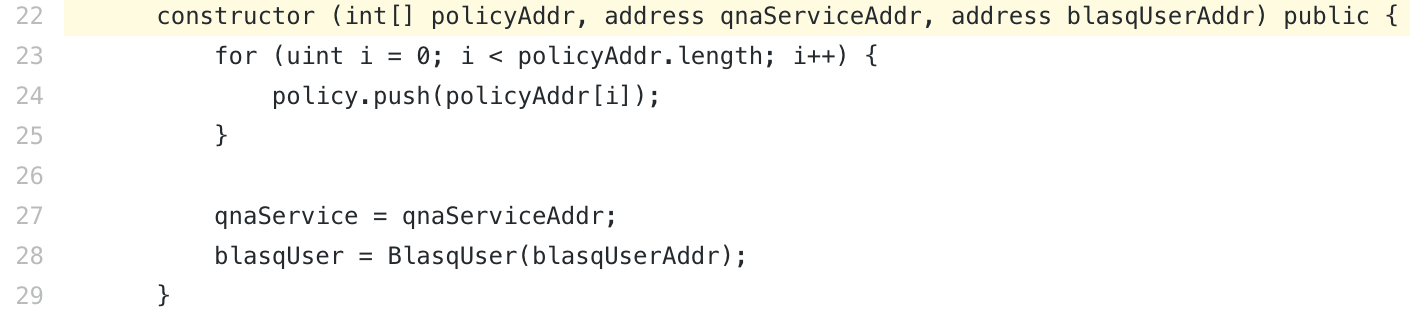
When *createAdminServicePolicy()* is executed twice with the same *address* but different *type*, *\_modifyAdminServicePolicy()* will not be executed. Meanwhile, *\_createNewAdminPolicy* will be executed, breaking one-to-one correspondence relations between *policyAddr* and *serviceType*. Hence, type loaded through *serviceTypeMap* and *address* loaded through *servicePolicyMap* do not correspond one-to-one.

If *type* overwritten by *deleteAdminServicePolicy* is deleted, the *address* which should be at *servicePolicyMap* is deleted. In that case, as the *address* cannot be called through *servicePolicyMap* and *QnaPolicy* Contract, which is the deleted *address*, cannot pass *serviceOnly*, most of the contract’s functions will stop operating.

#### Recommendation

1. In BlasqUser.sol:164, modify *if(servicePolicyMap[serviceType] != 0)*  
   to *if(servicePolicyMap[serviceType] != 0 || serviceTypeMap[policyAddr] != 0)*.
2. Run the initialization for the *address* that was done for the *type* at BlasqUser.sol:179~180.

### MAJOR : Fix length of *policy* array to 12 in *QnaPolicy#constructor()* and *QnaPolicy#createPolicy()* . [Unintended Behavior]

****(QnaPolicy.sol -   
<https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/QnaPolicy.sol#L22>  
<https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/QnaPolicy.sol#L232>)

#### 

#### Problem Statement

*QnaPolicy#\_createIndexOfPolicy()* is a function that returns an index of reputation point which will be used at *policy*. Arguments of this function, *actionType*, *contentType*, and *userType*, are enums of *ActionType*, *ContentType*, and *UserType*, respectively, and the length of each enums is 3, 2, and 2, respectively. Also, *ContentTypeNum* and *UserTypeNum* are constants whose value is 2. Hence, an index returned by *\_createIndexOfPolicy()* has a fixed range between 0 and 11.

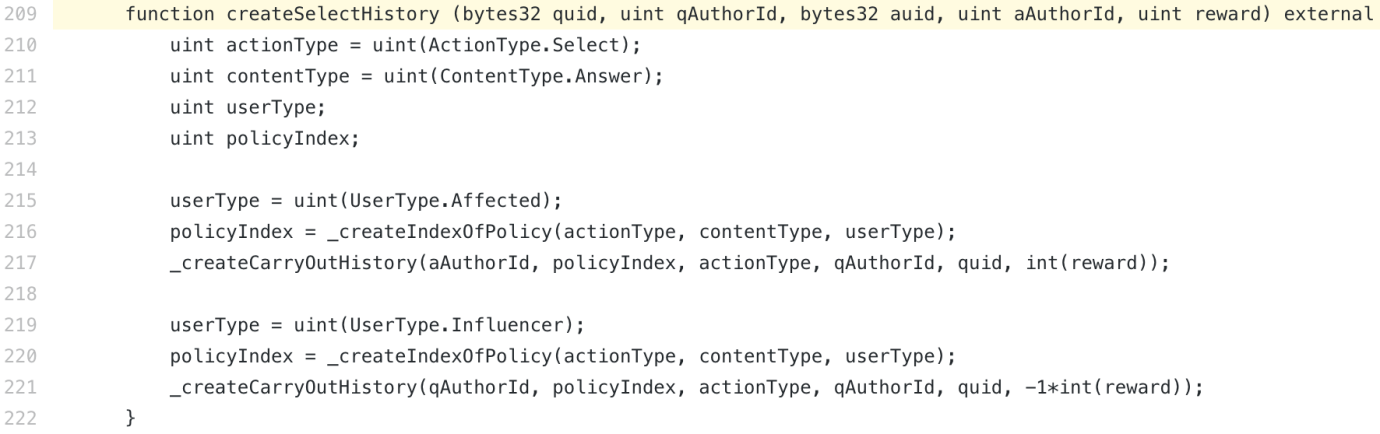
If the length of *policy* is shorter than 12, the contract may not access the index, for example whose value is 11, and the functions do not operate properly.

#### 

#### Recommendation

Add *require()* to fix the length of *policy* array to 12, at *constructor()* and *createPolicy()*, respectively.

### MAJOR : Wrong type casting exists in *QnaPolicy#createSelectHistory(*). [Wrong Type Casting]

****(QnaPolicy.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/QnaPolicy.sol#L209>)

#### 

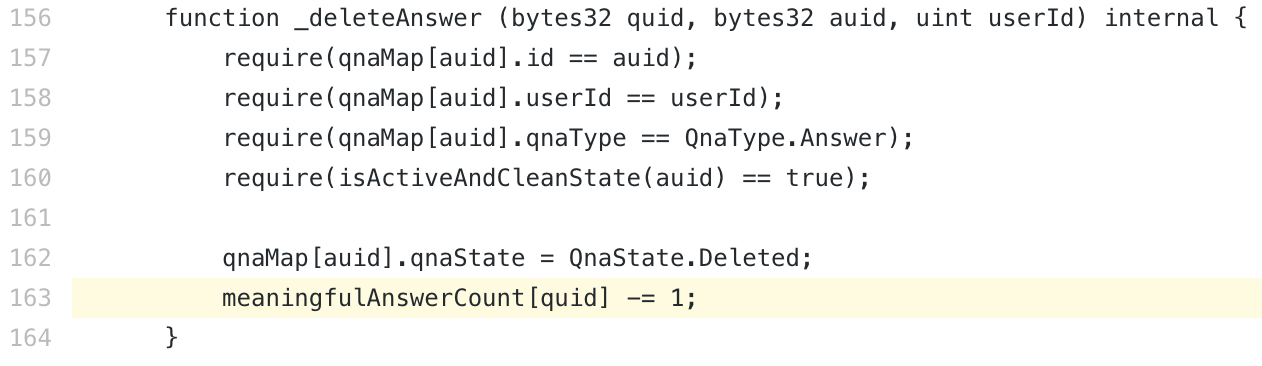
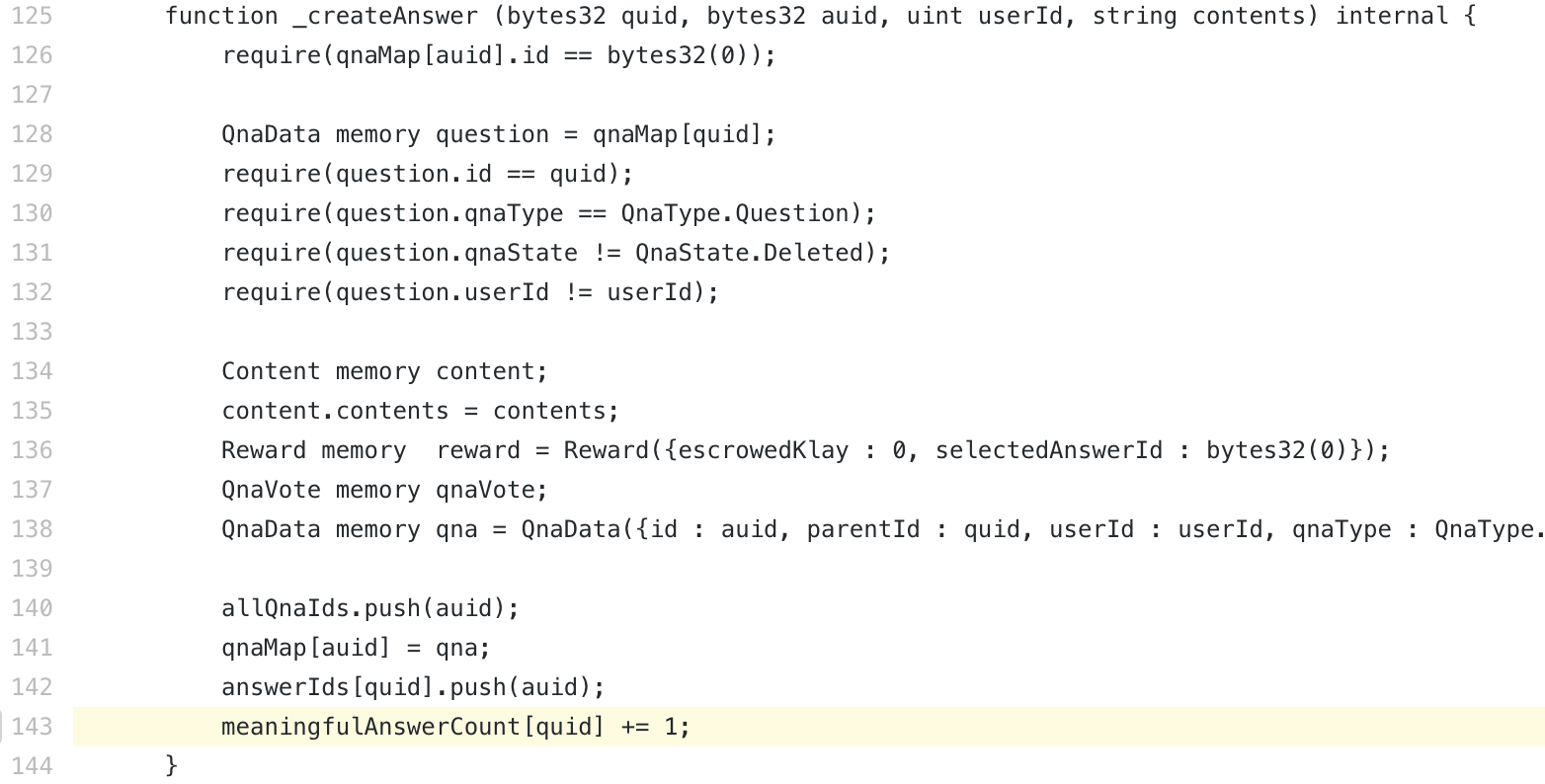
#### Problem Statement

*reward*, the fifth argument of *createSelectHistory()*, is transformed from *uint* type to *int* type at QnaPolicy.sol:217, 221. Range of *uint* type is 0 ~ (2256 -1) but the range of *int* type is -2255 ~ (2255 -1). If the value of *reward* exceeds 2255, Integer Overflow occurs.

#### Recommendation

Limit the range of *reward* to 2255-1 or under 2255-1, or use the uint type that is same or smaller than *uint128* as the *reward* type instead of using *uint* (*uint256*).

### MINOR : Integer Overflow/Underflow may occur in *Qna#\_createAnswer()* and *Qna#\_deleteAnswer ()*. [Arithmetic Over/Underflow]

  
(Qna.sol -  
<https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/Qna.sol#L125>  
<https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/Qna.sol#L156>)

#### Problem Statement

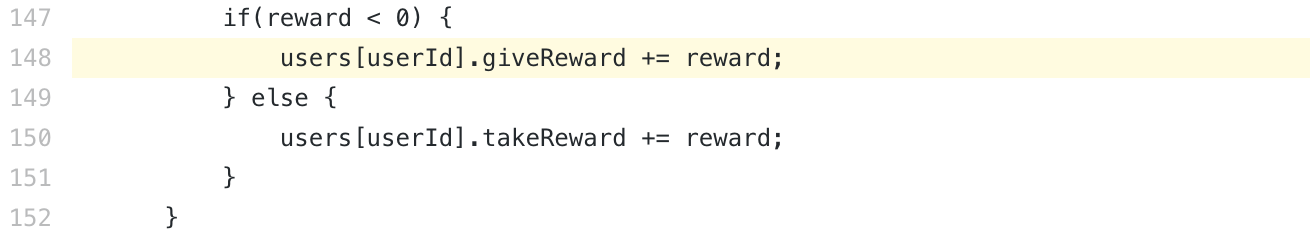
Integer Overflow/Underflow may occur in the logic which modifies the value of *meaningfulAnswerCount* at *\_createAnswer()* and *\_deleteAnswer()*.

#### 

#### Recommendation

Use [SafeMath.sol](https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/math/SafeMath.sol) library of openzeppelin-solidity in each functions, instead of using “+=” and “-=”.

### MINOR : Integer Overflow/Underflow may occur at *BlasqUser*. [Arithmetic Over/Underflow]

****(BlasqUser.sol -   
<https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/BlasqUser.sol#L148>  
<https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/BlasqUser.sol#L150>)

#### 

#### Problem Statement

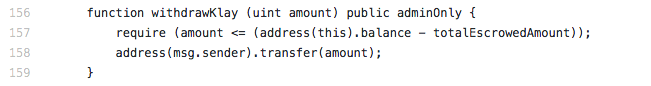
BlasqUser.sol:148, 150 has possibility of Integer Overflow or Underflow as the network grows.

#### 

#### Recommendation

For BlasqUser.sol:148, 150, apply “*Consulting - TIPS : Larger values could be stored when uint type is used for user.giveReward and user.takeReward, instead of int type.*” at first and then use [SafeMath.sol](https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/math/SafeMath.sol) library of openzeppelin-solidity.

### MINOR : Integer Underflow may occur in *QnaService#withdrawKlay()* and if it occurs, escrowed Klay can also be withdrawn. [Arithmetic Over/Underflow]

****(QnaService.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/QnaService.sol#L156>)

#### 

#### Problem Statement

*withdrawKlay()* is designed to withdraw the received Klay which a user sent by mistake. *require()* at QnaService.sol:157 checks whether the *amount* of Klay required to withdraw is smaller than the amount of Klay sent by mistake.

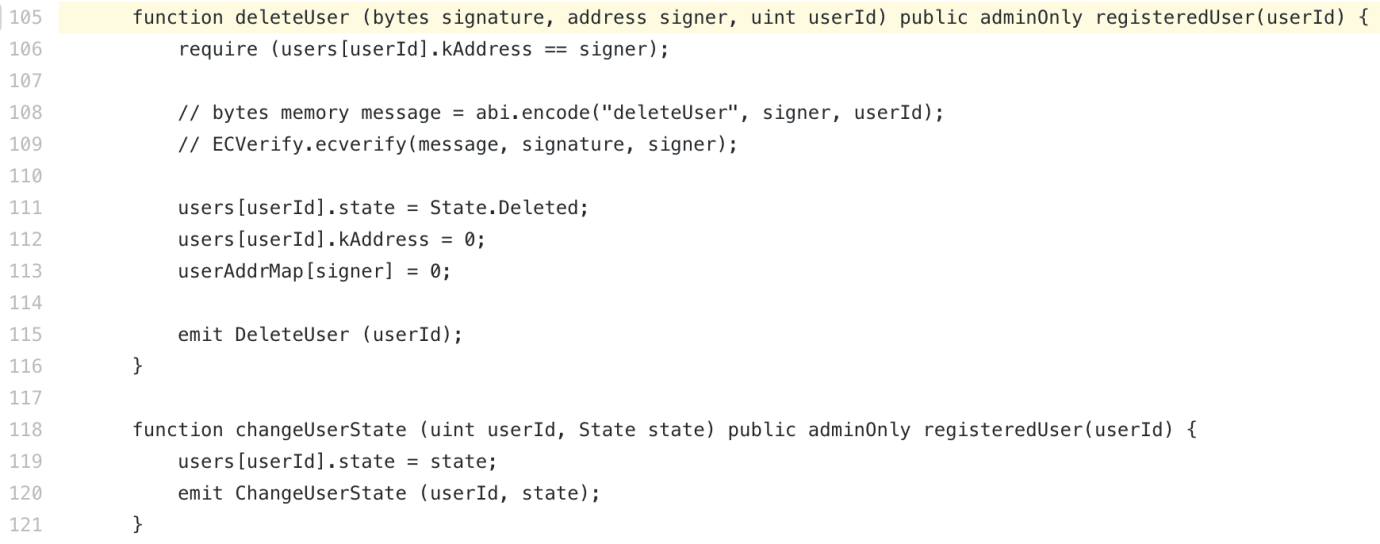
At that moment, if *totalEscrowedAmount* is larger than *address(this).balance*, Integer Underflow may occur. If Integer Underflow occurs, not just funds deposited by user mistakes but also escrowed amounts could be withdrawn, as *require()* always passes. (However, possibility of *totalEscrowedAmount* being larger than *balance* is rare in BLASQ service logic. Nevertheless, critical defect could be resulted if this context is not recognized when the logics are modified during switch to production level or built as upgradable smart contract.)

#### 

#### Recommendation

It is recommended to prevent Integer Overflow/Underflow by using [SafeMath.sol](https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/math/SafeMath.sol) library of openzeppelin-solidity.

### MINOR : *BlasqUser#deleteUser()* and *BlasqUser#changeUserState()* use different method for deleting users. [Unintended Behavior]

****(BlasqUser.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/BlasqUser.sol#L105>)

#### 

#### Problem Statement

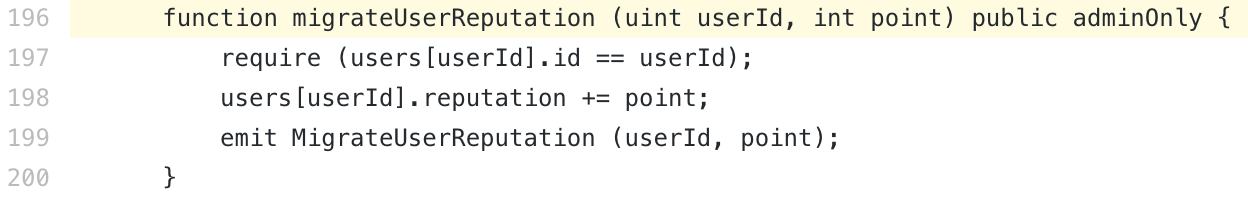
*changeUserState()* as well as *deleteUser()* can also change the user state into *Deleted*. However, not only arguments structure is different as *changeUserState()* does not take *signature* as the argument, but also the emitted events and result states are different.

#### 

#### Recommendation

It is recommended to allow that only *deleteUser()* can delete users. Add *require()* so that *State.Deleted* is not entered as the input for *changeUserState()*.

### MINOR : *require()* of *BlasqUser#migrateUserReputation()* cannot block when input *userId* is 0. [Unintended Behavior]

****(BlasqUser.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/BlasqUser.sol#L196>)

#### 

#### Problem Statement

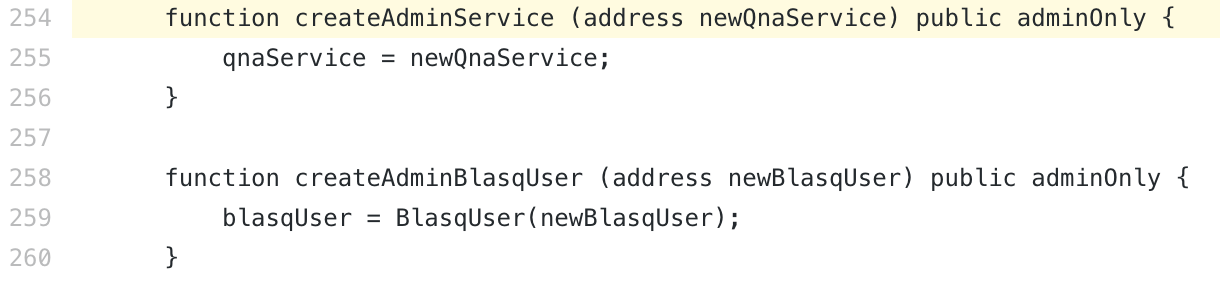
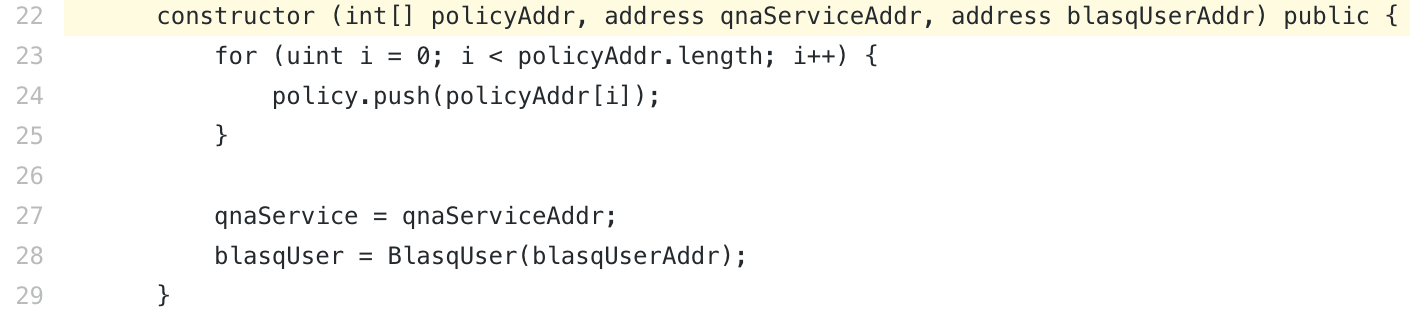
Although this function has to work only with registered *userId*, when *userId* is 0 (erroneous input), the process can pass the require statement.

#### 

#### Recommendation

It is recommended to delete BlasqUser.sol:197 and add *registeredUser* modifier in the contract to the function.

### MINOR : *constructor()*, *createAdminService()*, and *createAdminBlasqUser()* of *QnaPolicy* may accept zero address. [Unintended Behavior]

****(QnaPolicy.sol -   
<https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/QnaPolicy.sol#L22>  
<https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/QnaPolicy.sol#L254>)

#### 

#### Problem Statement

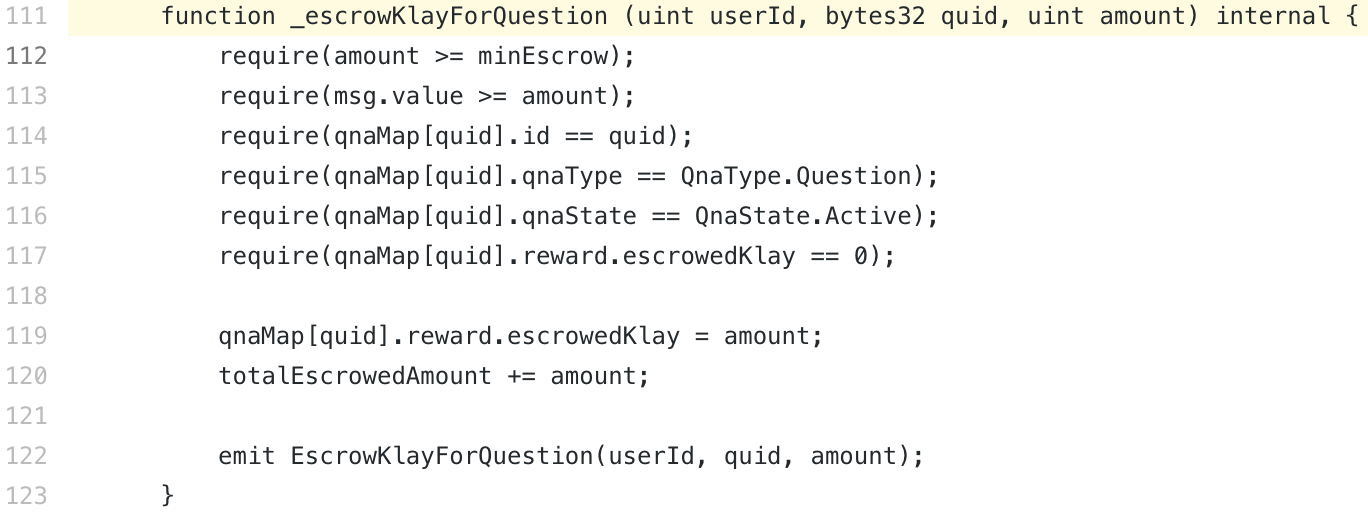
If zero address is given as an argument in *QnaPolicy#constructor()*, *QnaPolicy#createAdminService()*, and *QnaPolicy#createAdminBlasqUser()*, the function does not operate properly, as *qnaService* and *blasqUser* are the global variables.

#### 

#### Recommendation

Add *require()* to check whether if input value is zero address.

### MINOR : User’s Klay could be bound to *Qna#\_escrowKlayForQuestion()*. [Unintended Behavior]

****(Qna.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/Qna.sol#L111>)

#### 

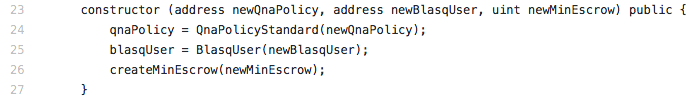
#### Problem Statement

If *msg.value* is larger than *amount* while calling *\_escrowKlayForQuestion()*, the difference between *msg.value* and *amount* is bound to the contract. Although the bound Klay could be withdrawn through *withdrawKlay()*, preliminary prevention may provide a better UX.

#### Recommendation

It is recommended to modify Qna.sol:113 into *require(msg.value == amount);*, or remove *amount* in the function entirely. When you choose to remove *amount*, it is recommended to convert all *amount*s into *msg.value* and then remove Qna.sol:113.

### INFORMATIVE : At *QnaService#constructor()*, contract is still created even when the address of *newQnaPolicy* and *newBlasqUser* are zero addresses. [Wrong Argument]

****(QnaService.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/QnaService.sol#L23>)

#### 

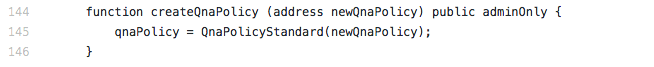
#### Problem Statement

Even when the address of *newQnaPolicy* or *newBlasqUser* which is given as a parameter is not a proper contract address but a zero address, the *QnaService* contract is well generated.

#### Recommendation

It is recommended to add a logic to check if the addresses of *newQnaPolicy*, and *newBlasqUser* are zero addresses, like *require(newQnaPolicy != address(0))* and *require(newBlasqUser != address(0))*, on the top line of the constructor.

### INFORMATIVE : *QnaPolicy* address could be modified even when *newQnaPolicy*, a parameter of *QnaService#createQnaPolicy()*, is zero address. [Wrong Argument]

****(QnaService.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/QnaService.sol#L144>)

#### 

#### Problem Statement

Even when *newQnaPolicy*, a parameter of *createQnaPolicy* is zero address, QnaPolicy address could be modified. In that case, *QnaService* contract cannot be connected to proper *QnaPolicy* contract.

#### Recommendation

It is recommended to add a logic to check if the address of *newQnaPolicy* is zero address, like *require(newQnaPolicy != address(0))*, on the top line of *createQnaPolicy* function.

### INFORMATIVE : *BlasqUser* address could be modified even when *newBlasqUser*, a parameter of *QnaService#createBlasqUser()*, is zero address. [Wrong Argument]

****(QnaService.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/QnaService.sol#L148>)

#### 

#### Problem Statement

Even when *newBlasqUser*, a parameter of *createBlasqUser* is zero address, BlasqUser address could be modified. In that case, QnaService contract cannot be connected to proper BlasqUser contract.

#### 

#### Recommendation

It is recommended to add a logic to check if the address of *newBlasqUser* is zero address, like *require(newBlasqUser != address(0))*, on the top line of *createBlasqUser* function.

# **05. Tech Consulting**

## 05.a. Gas Optimization

In EVM, the parts that consume most of gas are usually parts using storage. (Gas consumption by EVM opcode could be found at Appendix G table of Yellowpaper[[1]](#footnote-0) of Ethereum.) Hence, you should reduce storage use as much as possible when executing functions at smart contracts, in order to save gas. In Blasq Smart Contract, Ground X team has made great efforts for gas optimization, for example, the utilization of memory. HAECHI LABS, furthermore, recommends Ground X team to handle variables which do not necessarily have to be stored in storage according to the business logic, in the following two ways.

### GAS OPTIMIZATION : It is recommended to record log data as an Event, instead of storing in the storage.

#### Problem Statement

Log data has the following characteristics.

* Variable not referred by any function other than view function
* Variable that does not require indexing

If a certain variable is not referred by any function other than public view function, that variable is highly likely to be log data. For example, *createQuestion* of QnaService.sol:37 calls *\_createActivity* of QnaUser.sol:32 and store various information in a variable called *userMap*. However, *userMap* is not referred by any function other than a view function called *readQnaUserActivity* at QnaUser.sol:49.

Of course, when there is a variable called *userMap*, user’s history could be easily read through *userId*, as the variable is a mapping variable. However, it costs much gas. Hence, you have to consider whether the gas consumption is essential and required.

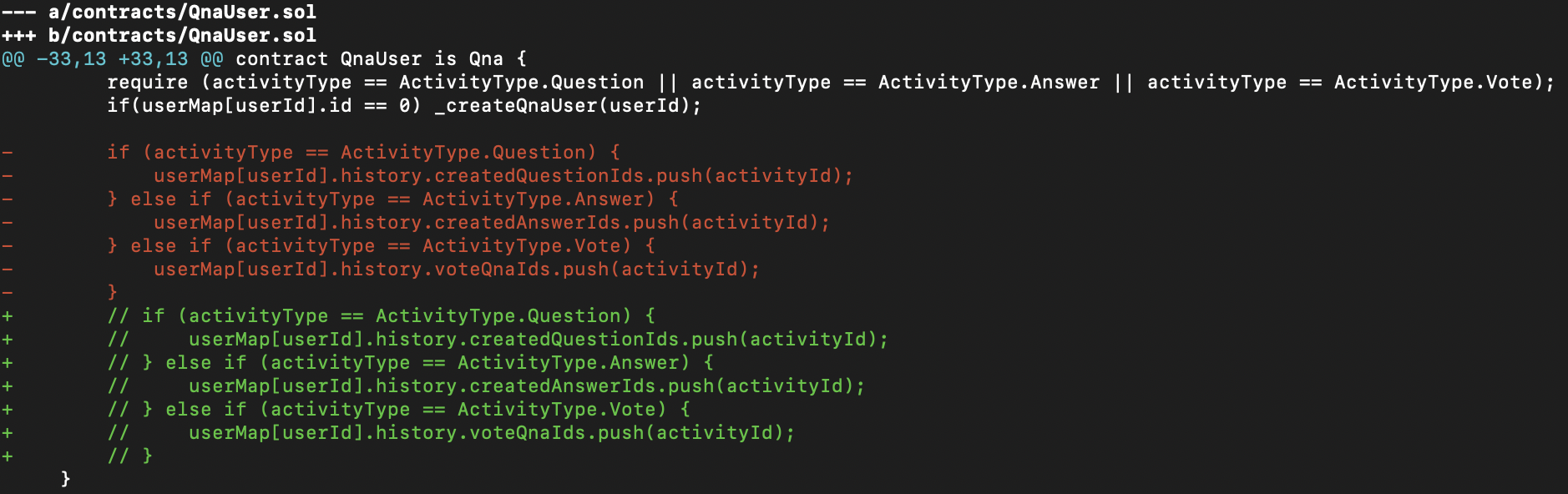
#### Recommendation

After comprehensively considering the consumed gas cost and utility gained from *userMap*, HAECHI LABS concluded that using *userMap* is not desirable. It is because *userMap* is frequently modified whenever activities occur, resulting in large gas consumption. In contrast, constructing a DB which provides the same functionality as *userMap* through event is easy. Hence, it is recommended to exclude the logic that store log data in the storage from *\_createActivity* function. In fact, even when data are not recorded in *userMap*, the data could be easily restored through *CreateQuestion*, *CreateAnswer*, and *Voting* Events. We recommend constantly receiving above mentioned Events from the server and indexing those Events in DB, rather than using *userMap*.

#### 

#### Gas Comparison

*experiment setting*s

[Figure 7] Difference between before and after codes of Gas Optimization 1

* *before*: The original of QnaService.sol
* *after*: Source code in which the logic that record log data of *\_createActivity* in storage has been removed from QnaService.sol
* *testcase*: test/QnaService.test.js
* *environments*
  + ganache-cli
  + mocha reporter: eth-gas-reporter

*experiment result*

| Contract.method | before | after | gas diff |
| --- | --- | --- | --- |
| QnaService.createAnswer | 377398 | 336930 | **-10.7%** |
| QnaService.createQuestion | 424993 | 384574 | **-9.5%** |
| QnaService.voting | 394766 | 354257 | **-10.3%** |

[Table1] Gas Optimization 1

### GAS OPTIMIZATION : It is recommended to not store the array of Array index in storage.

#### Problem Statement

*allQnaUserIds* declared at QnaUser.sol:21 is a variable to store id’s of every user. This is also not referred by any function other than public view functions including *\_createQnaUser*, and *readAllQnaUserIds* in QnaUser.sol:45.   
Usually, array of indice is stored in storage is when another contract need to directly compute the array through for calculation. For example, there could be a logic that transfer token to a certain address through for calculation. However, *allQnaUserIds* does not have such purpose.

#### Recommendation

It is recommended to load *userId* through predefined Event (including *CreateQuestion), instead of allQnaUserIds*.

#### 

#### Gas Comparison

*experiment setting*s

  
[Figure 8] Difference between before and after codes of Gas Optimization 2

* *before*: The original of QnaUser.sol
* *after*: Code in which *allQnaUserIds* has been removed from QnaUser.sol
* *testcase*: test/QnaService.test.js
* *environments*
  + ganache-cli
  + mocha reporter: eth-gas-reporter

*experiment result*

| Contract.method | before | after | gas diff |
| --- | --- | --- | --- |
| QnaService.createAnswer | 377398 | 352162 | **-6.69%** |
| QnaService.createQuestion | 424993 | 384757 | **-9.47%** |
| QnaService.voting | 394766 | 369530 | **-6.39%** |

[Table2] Gas Optimization 2

### GAS OPTIMIZATION : Comprehensive Application

When both the above mentioned methods are applied simultaneously, 19.0% of gas could be saved.

*experiment result*

| Contract.method | before | after | gas diff |
| --- | --- | --- | --- |
| QnaService.createAnswer | 377398 | 311694 | **-17.4%** |
| QnaService.createQuestion | 424993 | 344338 | **-19.0%** |
| QnaService.voting | 394766 | 329021 | **-16.7%** |

[Table3] Total Gas Optimization

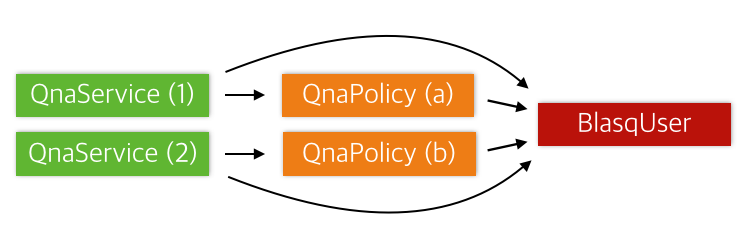
## 

## 

## 05.b. Best Practices

Best Practices consist of items recommended by HAECHI LABS for a better structure of smart contract.

### BEST PRACTICES : Deploy Management

******[Figure 9] Smart Contract Architecture

#### Overview

Blasq Service largely consists of three types of smart contracts. Firstly, at QnaService, one can create/modify/delete questions and answers, *vote* on questions and answers and *select* good answers. Secondly, at QnaPolicy, policy on reputation and reward according to vote and select are handled. Lastly, in BlasqUser, wallet address and user information are matched and reputation and reward of the users according to vote and select are recorded. Each contract is subject to change under certain circumstances. For example, if you want to change the policies regarding reputation and reward, you can write and deploy a new QnaPolicy and connect it to QnaService and BlasqUser.

#### 

#### Deployment Problem

To deploy contracts in the from of the architecture shown in [Figure 9] above, deployment process is very important. Let’s divide the deployment process into the situation where the initial deployment of the entire contract takes place and where service update takes place through additional deployment of specific contract.

* First Deployment

| Order | Transaction |
| --- | --- |
| 1 | Deploy BlasqUser. |
| 2 | Deploy QnaPolicy. At this moment, enter the address of BlasqUser and the *address of QnaService that has not yet been deployed.* (Since QnaService is not yet deployed, the address is still unknown. Enter zero address at the time of deployment) |
| 3 | Call *BlasqUser#createAdminServicePolicy()* function to register the QnaPolicy address and type. To register various policies on BlasqUser, type and QnaPolicy address are registered in pair. (Only the registered QnaPolicy can call *BlasqUser#createUserHistory()*.) |
| 4 | Deploy QnaService. Then BlasqUser address and QnaPolicy address are passed as the parameters to the constructor of QnaService. |
| 5 | Since the QnaService has not been deployed nor the address was properly passed as the parameter to the constructor of QnaPolicy, call *QnaPolicy#createAdminService()* function to pass the address of QnaService to connect with QnaPolicy. (Only the registered QnaService address can call *QnaPolicy#createVoteHistory()* and *QnaPolicy#createSelectHistory()*.) |

[Table 4] Flow of Service Deployment

When you deploy contracts as the architecture described in [Figure 9], you have to create 5 transactions with current architecture. At that moment, for QnaPolicy and QnaService, Cyclic Dependency is formed as the counterparty’s contract address should be entered in constructor. Hence, as undeployed QnaService address is entered during QnaPolicy deployment, there is a possibility of error and additional transactions are required for the entire deployment.

* Update

Deploying contract for a service update can be divided into three cases which are modifying QnaService, modifying QnaPolicy and modifying BlasqUSer.

As each contract refer to each other’s addresses as variables, when you update service, you should not only deploy new contracts but also modify the address value each contract is referring to. Among the aforementioned three cases, let’s take a look at the process of service update through re-deployment of QnaPolicy.

| Order | Transaction |
| --- | --- |
| 1 | Deploy QnaPolicy. At the same time, deliver addresses of deployed BlasqUser and QnaService to the constructor. |
| 2 | Modify QnaPolicy that QnaService refers to by calling the *createQnaPolicy* function of QnaService which will be using newly deployed QnaPolicy. |
| 3 | Register newly deployed QnaPolicy on BlasqUser by calling the *createAdminServicePolicy* function. |

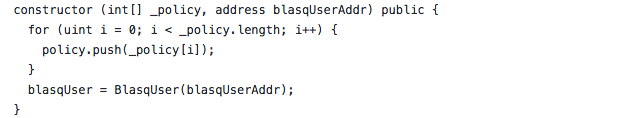
[Table 5] Flow of Service Update through Distribution of QnaPolicy

When QnaPolicy is newly deployed in order to change policy, 3 transactions should be made to properly conduct the service update. As those three transactions are not processed atomically, the service operation might stop for a while. Let’s take a look at some examples on what kind of situations may occur. Assume that a user called *QnaService#voting()* and *voting* function is calling newly deployed *QnaPolicy#createVoteHistory()*. If only the transaction 1~2 at [Table 5] are processed while transaction 3 is not processed yet, QnaPolicy will not have the authority to modify reputation of BlasqUser. Thus, *QnaService#voting()* might revert and the service may not operate. Update without downtime is required as a service update method since service interruption will disrupt users of the service.

#### 

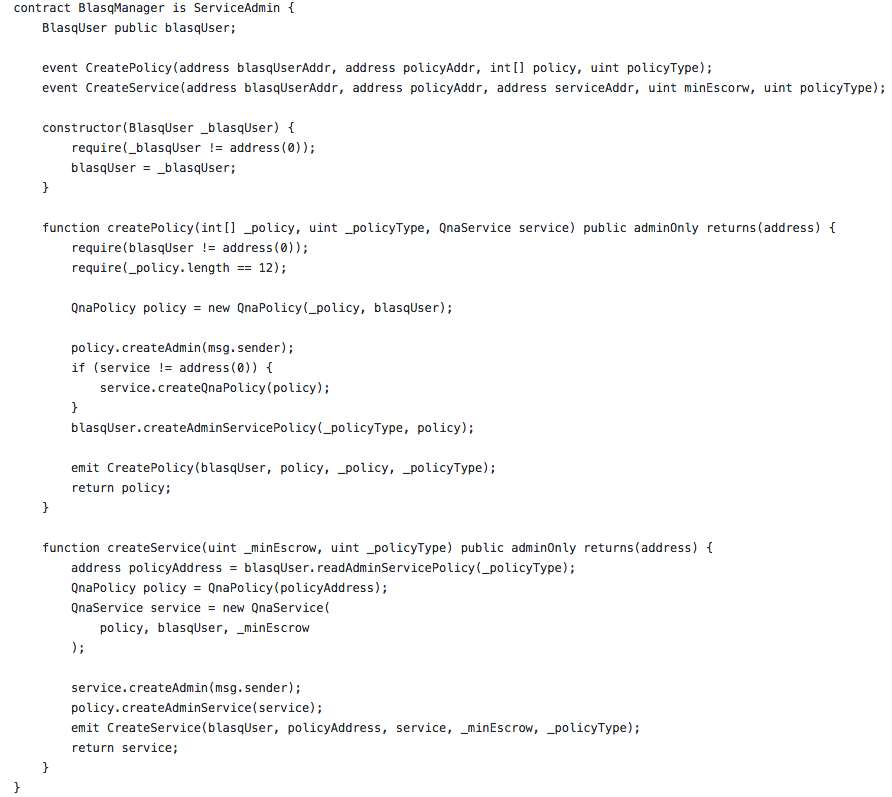
#### Deployment Solution

Firstly, let’s take a look on the solutions of Cyclic Dependency. Cyclic Dependency occurs constructors of both QnaService and QnaPolicy receive each other’s address. Therefore, it could be resolved when one constructor does not receive the counterparty’s address. In order to design the architecture described in [Figure 9], the best order of deployment is BlasqUser → QnaPolicy → QnaService. In this regard, it is recommended not to include the address of QnaService in the constructor of QnaPolicy that will be deployed afterwards.

[Figure 10] QnaPolicy Modified constructor

Thus, it is recommended to remove the part that receives the address of QnaService from the constructor as shown in the code above. Instead, the address of QnaService can be received through *QnaPolicy#createAdminService().* The address of QnaService is only used in the *qnaServiceOnly* modifier of QnaPolicy, but revert keeps occurring at *qnaServiceOnly* until the address of QnaService is set by calling *QnaPolicy#createAdminService()*. Such revert is not an issue since the revert in the initial deployment occurs due to incomplete initialization that entails improper operation of contract, not unintended operation.

Lastly, it is recommended to create BlasqManager type of contract to support service update without downtime. BlasqManager helps deploy contracts more conveniently and handle multiple transactions into a single transaction which supports atomic deployment and updates.

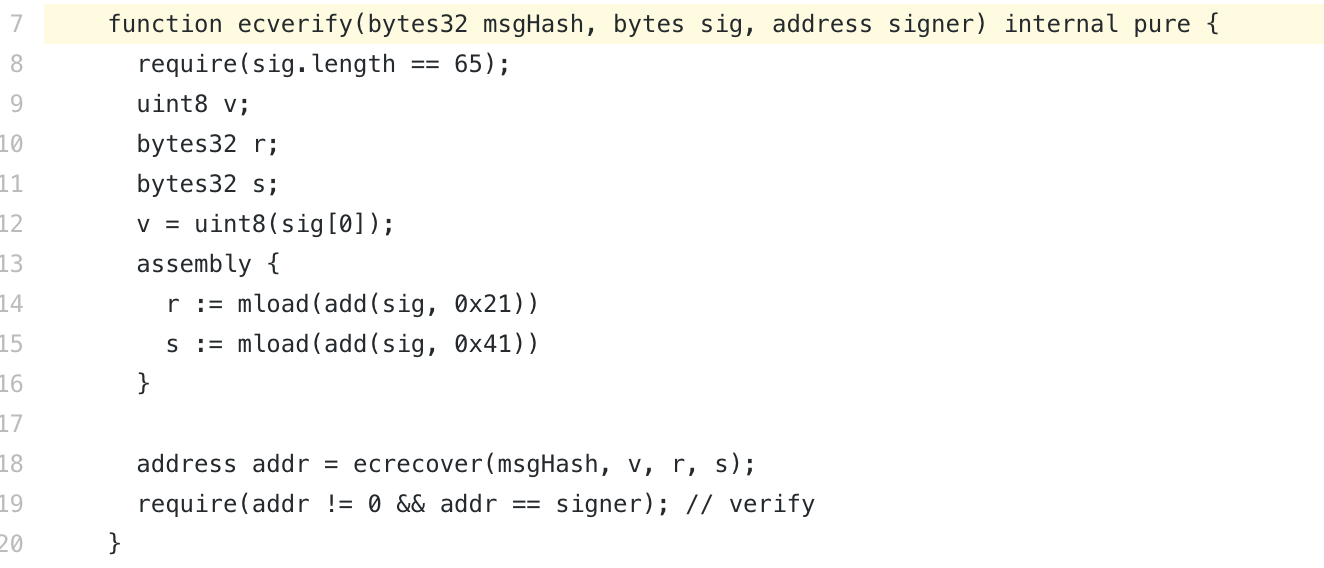
  
[Figure 11] BlasqManager.sol

The above BlasqManager is a sample contract to show how QnaPolicy and QnaService could be atomically deployed. As BlasqUser is storing user data of the service, BlasqUser was not handled assuming that it won’t updated. In deploying BlasqManager, pass the address of BlasqUser to constructor.

*createPolicy* function was created to atomically deploy QnaPolicy. *createPolicy* function handles 3 transactions as a single transaction which is described at [Table 5]. By handling transaction 2~3 at [Table 5] as an Internal Transaction of *createPolicy*, QnaPolicy is deployed atomically.

Similarly, *createService* function is also used when deploying QnaService. In the same manner as the case of *createPolicy*, QnaService can be atomically deployed by handling the deployment of QnaService contract and connecting with other contracts as an Internal Transaction.

### BEST PRACTICES : ECverify

****(ECVerify.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/ECVerify.sol#L7>)

#### Problem Statement

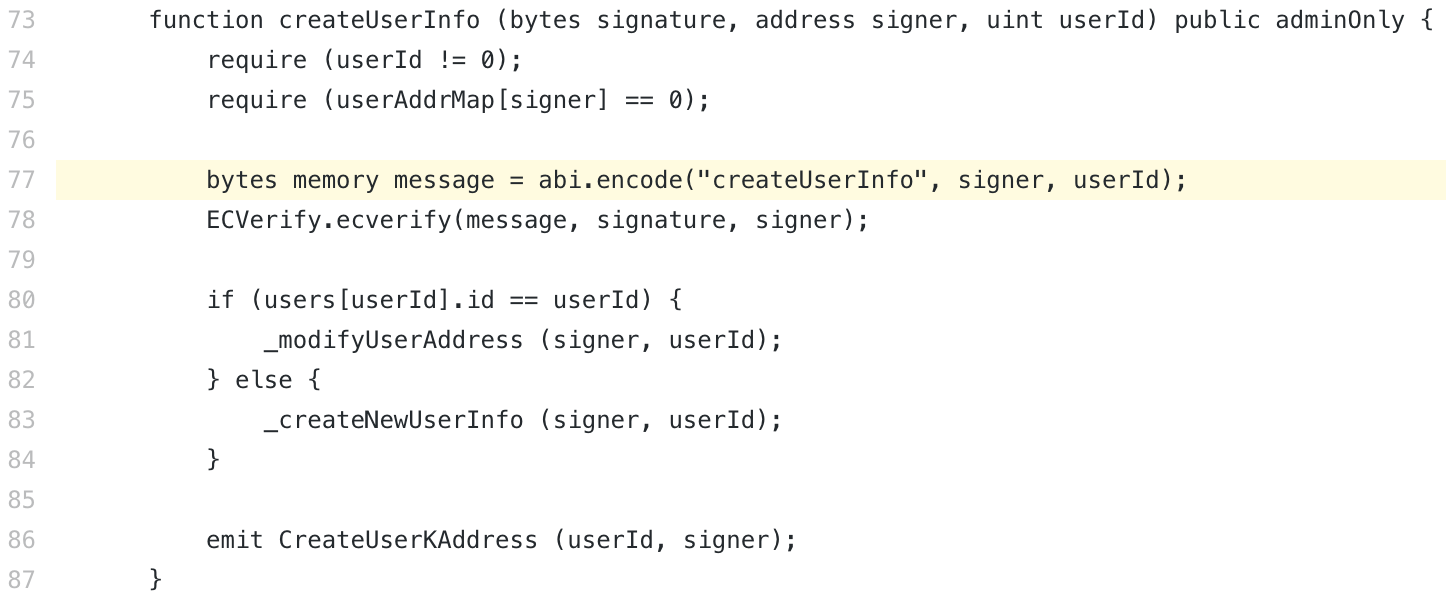
*ECVerify#ecverify()* is different from the ordinary signature generating library protocol. Although the most of signature generating library parse/concatenate signatures in the order of r, s, and v, the current code is implemented to operate in the order of v, r, and s. Additionally, when using the old version of signature generating library (e.g. web3 0.2.x of Metamask), the library generates v to 0 or 1. Therefore, *ecverify()* of the current version may collide with external signature generating library.

#### 

#### Recommendation

Implement the *bytes sig* to parse in the order of r, s and v. Moreover, if the value of v is smaller than 27, then there is a possibility that an old library has generated v as 0 or 1. Thus, add 27 to v if v < 27.

### BEST PRACTICES : Encoding

****(BlasqUser.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/BlasqUser.sol#L77>)

#### 

#### Problem Statement

Message encoding method that is used in *BlasqUser#createUserInfo()* is different from the intended method. The ordinary way of encoding functions and variables in Ethereum is as follows. Enter string *“createUserInfo(address,uint256)”* as an input of the hash function *keccak256*, and take the first 4 bytes from the result.

| bytes4(keccak256(“createUserInfo(address,uint256)”)) |
| --- |

That value is called function selector. If you concatenate argument, then you can see the string as below.

| “{**function selector**}{argument1}...{argumentN}” |
| --- |

However, *abi.encode* used at BlasqUser.sol:77 is a function that simply concatenates the string. That is, as it is encoded below, different from the ordinary method.

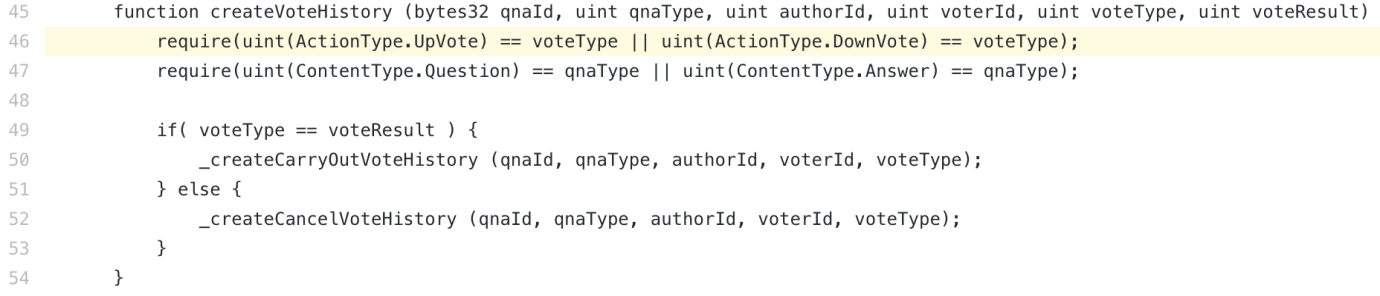
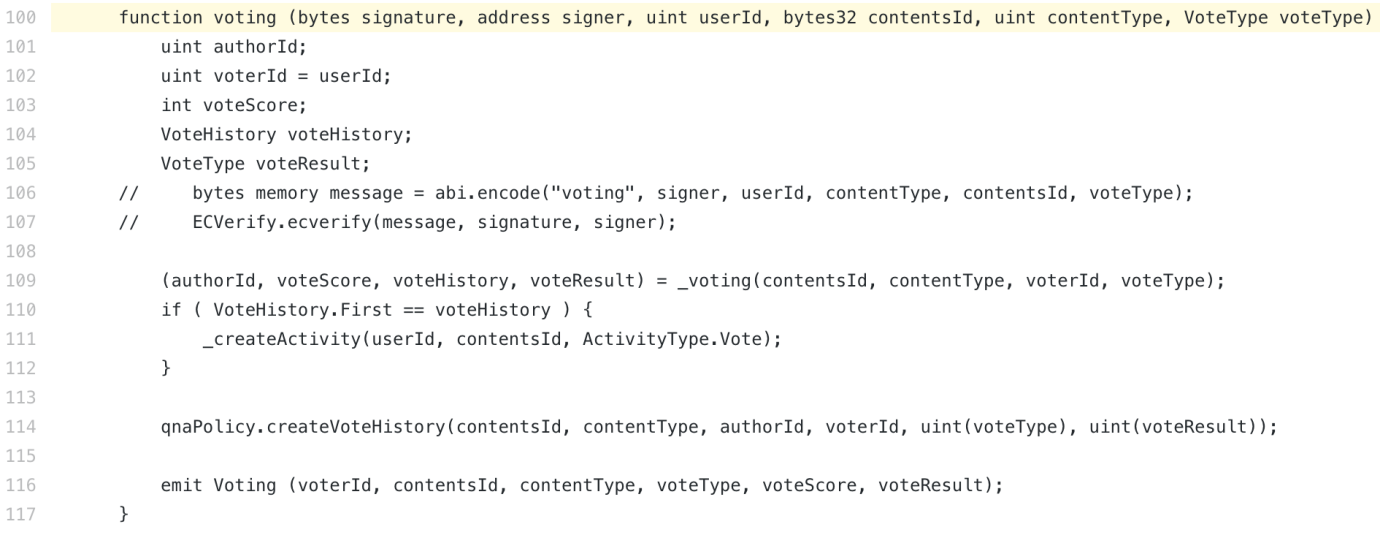
| “{**function name**}{argument1}...{argumentN}” |
| --- |

Of course, as long as those standards are newly defined and strictly kept within Blasq Smart Contract, it won’t be a major issue. Nevertheless, it is recommended to adopt a well-known method instead of encoding with other methods unless there is a specific reason.

#### Recommendation

Instead of *abi.encode(“createUserInfo”, signer, userId)*, concatenate the function selector generated from *bytes4(keccak256(“createUserInfo(address,uint256)”))* with *abi.encode(signer, userId)* and use it as a message.

### BEST PRACTICES : Enum

****(QnaPolicy.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/QnaPolicy.sol#L46>)  
  
(QnaService.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/QnaService.sol#L100>)

#### 

#### Problem Statement

Match enum used at QnaPolicy.sol and QnaService.sol. The fifth argument of *QnaPolicy#createVoteHistory()* function is enum of *ActionType* in the contract. However, in QnaService, when calling the *createVoteHistory* function of *QnaPolicy*, the *VoteType* enum is used as the enum of the fifth argument. (*QnaType* and *ContentType* are the same issues).

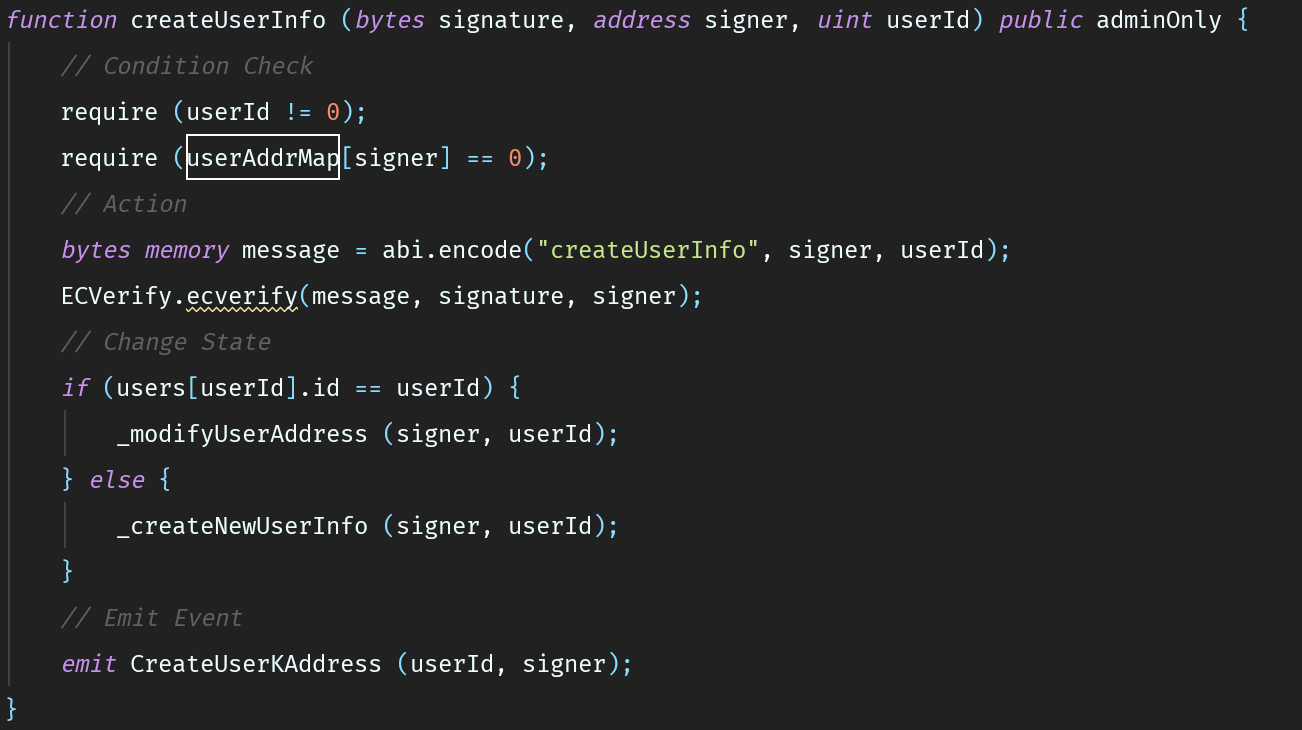
There is no issue in operation because they are the same results at the end. Still, implementing enum in different names at separate files is not desirable since it increases the chance of mistakes for an upgrade and might confuse other developers.

#### Recommendation

It is recommended to store a constant that is repeatedly used in multiple contracts such as enum in the *Constants* type of contract and make each contract that uses enum to inherit *Constants.*

### BEST PRACTICES : General Function Structure

It is recommended to write functions of smart contract in the order described in the following table. Below is a good practice written at BlasqUser.sol:73.

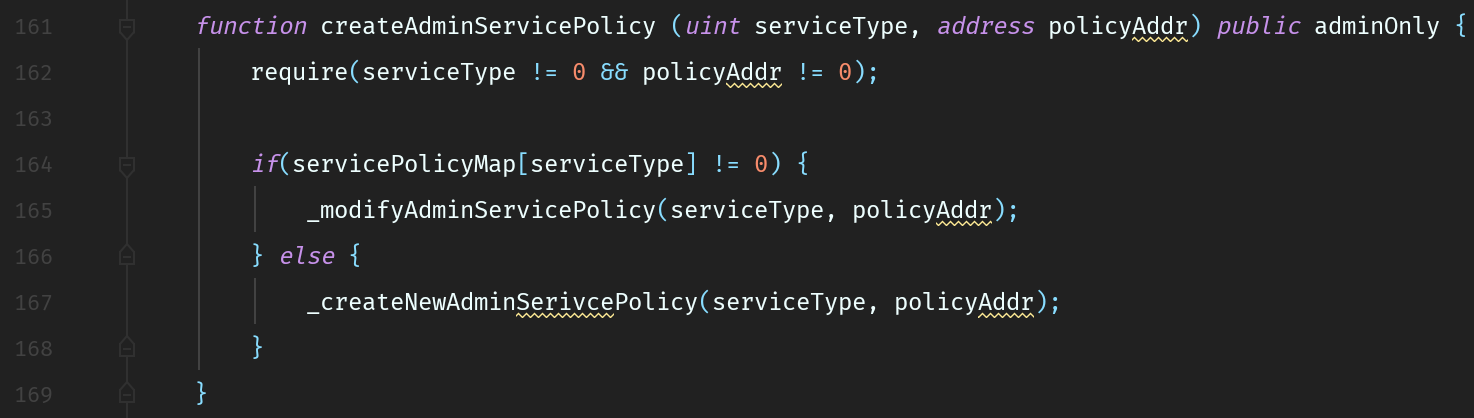
  
[Figure 12] General Function Structure Example1

| 1 | Condition Check | Operation condition of the function is defined here. The purpose is to prevent utilization of this function in a wrong place. (it could be replaced by Modifier) |
| --- | --- | --- |
| 2 | Action | Operate key logic of a function here. (If the logic that calls random external contract is included , then it is recommended to change the state first to prevent Re-entrancy issue) |
| 3 | Change State | Operate logic to modify the state of the contract here. |
| 4 | Emit Event | Make log data you wish to store occur as Event in this step. If you use Events well, then you will be able to easily conduct data tracking or data migration in the future. |

[Table 6] General Function Structure

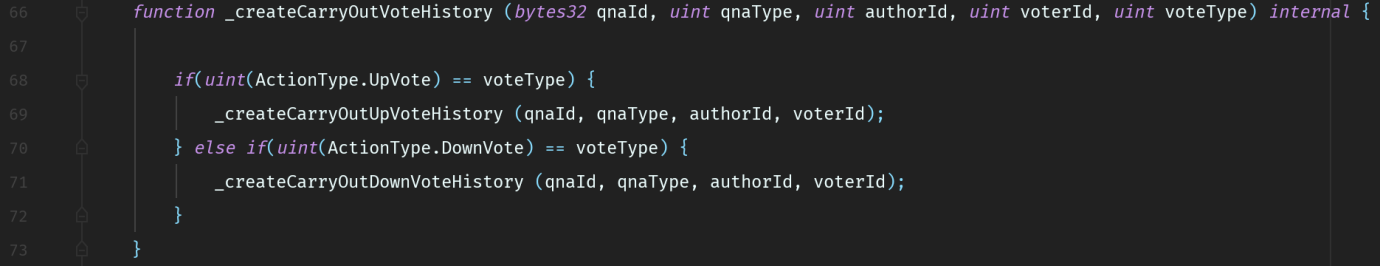
Below are the undesirable examples.

* BlasqUser.sol:161 *createAdminServicePolicy()*

  
[Figure 13] General Function Structure Example2

Data tracking could be difficult for this function because there is no Event occurring logic.

* QnaPolicy:56*createCarryOutVoteHistory()*

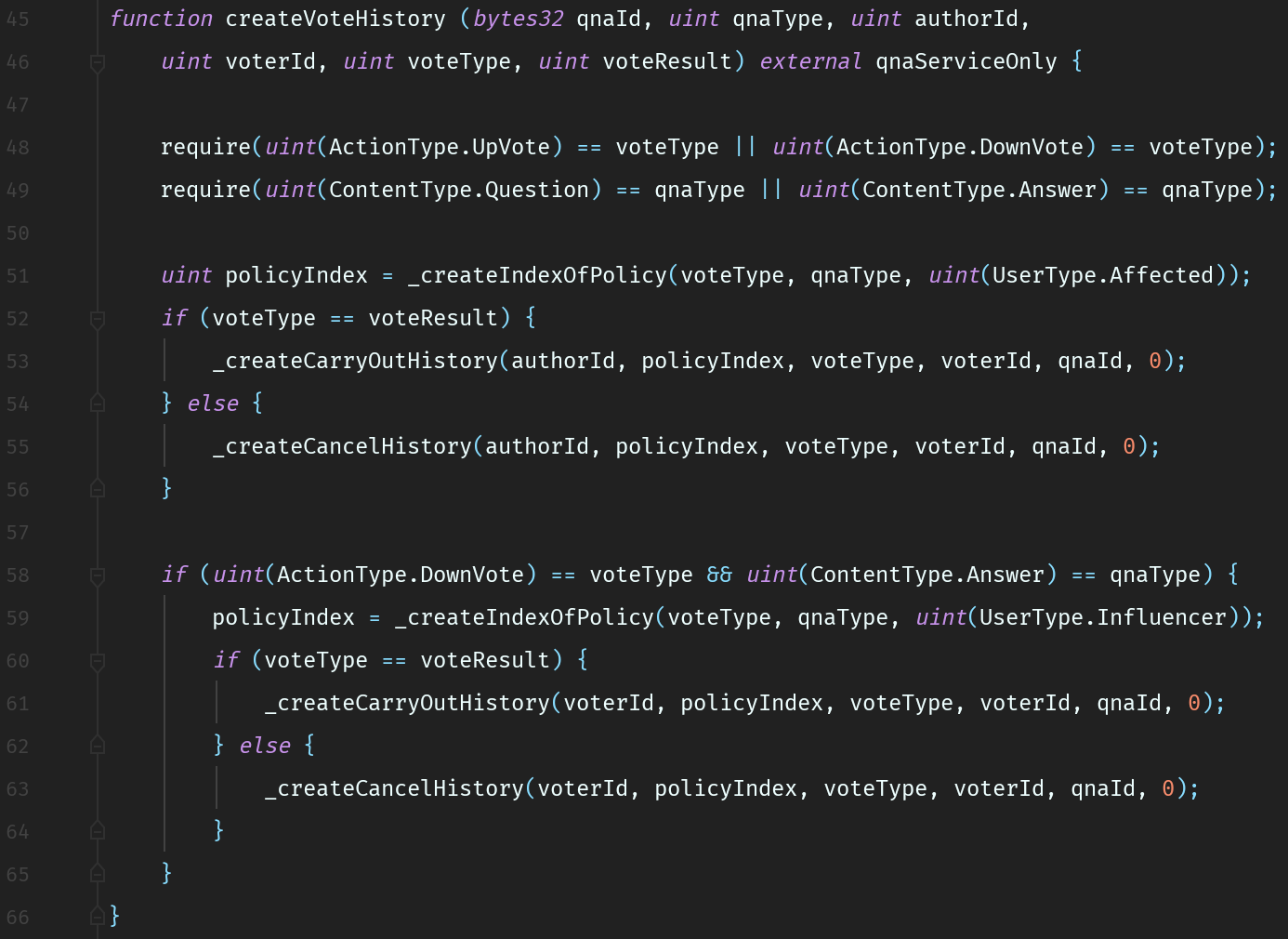
  
[Figure 14] General Function Structure Eample3

Since there is no Condition Check process in this function, it is hard to know conditions under which is safe to call function or what kind of condition was expected when the function was designed.

#### Internal Functions

In the current implementation, there are too many internal functions generated. The internal functions that are run by *QnaPolicy#createVoteHistory()* are used to branch the following actions. Such function makes the purpose of the function’s usage unclear,which results in lower readability of code during maintenance code and unnecessarily increases the gas cost for deployment.

For example, 1 public function and 14 internal functions written at QnaPolicy.sol:45~207 can be expressed in a single public function as written below.

[Figure 15] Improved createVoteHistory.sol

Through eliminating code repetition, it has the advantages on readability, maintenance and the gas cost for deployment as it has been reduced from 1,395,535 gases to 1,278,690 gases.

Internal Functions are usually written to avoid code repetition, but there are many unnecessary functions written in the current implementation. Also, each function does not have Condition Check and Emit Event steps of General Function Structure. This hinders knowing what conditions are required to use the function and make data tracking through Events difficult. As it could be an obstacle for cooperation and maintenance, it is recommended to follow the General Function Structure above.

### BEST PRACTICES : Migration

If you are considering contract data Migration, re-designing the migration function and Event is required.

#### Smart Contract Migration[[2]](#footnote-1)

As existing services migrate backups stored in database slave to master in the case of fatal defects in service, smart contract also can conduct migration when adopt a strategy that supports upgradeability. The process of smart contract migration processing could be divided into two parts.

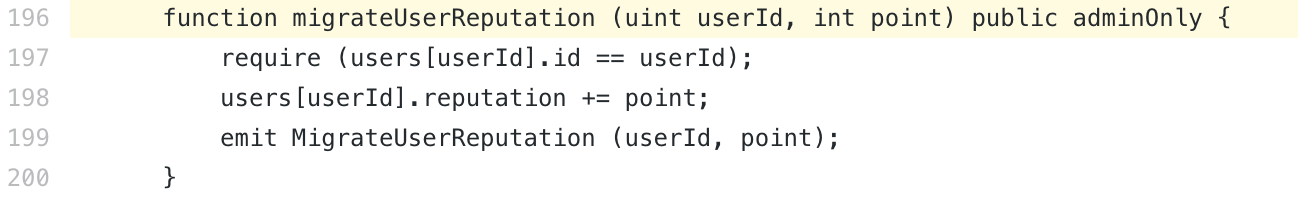
1. Extract data stored in smart contract and take a snapshot of them.
2. Record the data of snapshot you want in the newly deployed smart contract.

The part you have to consider while writing a smart contract is (1). If (1) is largely divided into data extraction work and data storage work, the process of data storage work is relatively clear since it is about recording it in the existing database, but you have to be cautious about the data extraction work. First of all, different extraction strategy is required depending on the types of smart contract variable. For example, public variable could be easily extracted with getter and private variable could be extracted through either *getStorageAt* of *web3* after calculating the memory offset or by subscribing the Event log. If the Array variable could also be extracted through those methods as long as its length is known.

However, data extraction through above mentioned methods is difficult for mapping variable because key value regarding mapping is not stored in blockchain intact (stored as hash value). To solve it, a well-designed Event log system is required. That is, the strategy of determining when to generate Event log is important to facilitate smart contract data migration.

Considering *BlasqUser#migrateUserReputation()*, it seems like the Ground X team was also considering about migration in Blasq. However, the current implementation code is hard to tell that migration was adequately considered for the following reasons.

### BEST PRACTICES : *BlasqUser#migrateUserReputation()* which should only be called during migration is written to be able to be called anytime.

****(BlasqUser.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/BlasqUser.sol#L196>)

#### 

#### Problem Statement

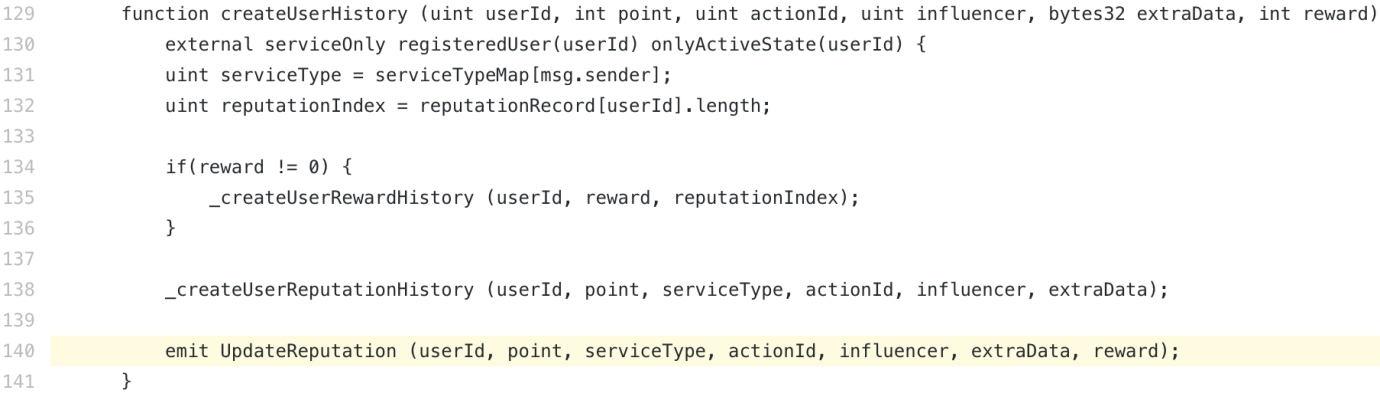
Usually, migration is a one-time function which only runs in the process (2). However, *BlasqUser#migrateUserReputation()* is able to be called anytime. Also, it is written to increase *reputation* of *point*, such as *users[userId].reputation += point*. In that case, if the concerned function is repeatedly run during migration, more *point* could be transferred than the intended amount.

#### Recommendation

It is recommended to modify the function so that *BlasqUser#migrateUserReputation()* is called only once per a *userId*.

### 

### BEST PRACTICES : As *UpdateReputation*, an event of BlasqUser.sol:140 only records changes in *point*, restoration process for reputation is inefficient.

****(BlasqUser.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/BlasqUser.sol#L140>)

#### 

#### Problem Statement

The simplest method of data migration through Event is as follows:

1. As for Event, always record the latest value of mapping key
2. Later, read the last Event from numbers in the target block and enter that into the newly deployed contract

However, *UpdateReputation* Event is only recording the changing amount of *point* not the current value. In that case, there is an inconvenience of reading every Event to restore the value. If there is a need to restore data of multiple users, then it requires inefficient calculation in all parts including the api call to request to the blockchain and to integrating values from offchain DB.

#### Recommendation

Implement the Event regarding mapping variables to record the mapping value corresponding to the key. It is recommended to modify *UpdateReputation* Event so that it records the value ultimately stored in *users[userId]*, not the changing amount of changing values.

## 05.c. Tips

Tips consist of additional recommendations suggested by HAECHI LABS to implement better smart contract.

### TIPS : There is an unused argument in *Qna#\_voting().*

****(Qna.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/Qna.sol#L167>)

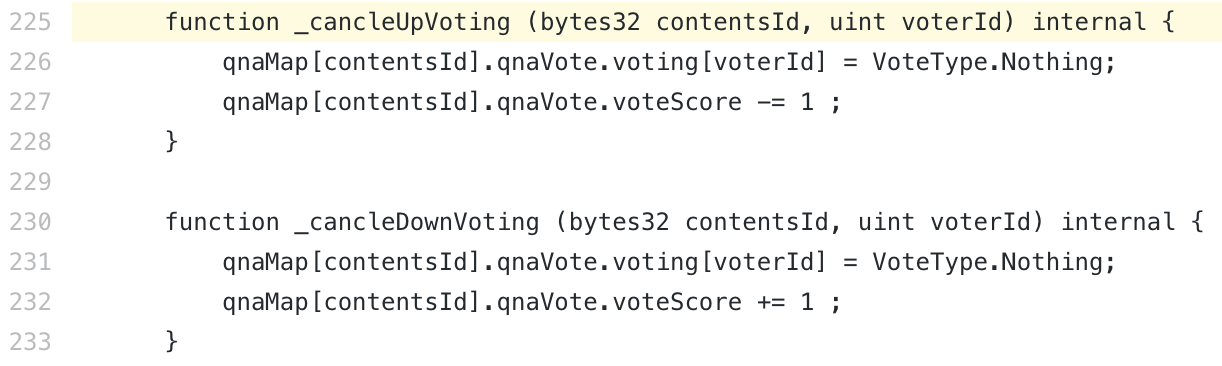
#### Problem Statement

*contentType*, the second argument of *\_voting*, is not used in the function.

#### Recommendation

It is recommended to remove the concerned argument.

### TIPS : There are typos in *Qna#\_cancleUpVoting()* and *Qna#\_cancleDownVoting()*.

****(Qna.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/Qna.sol#L225>)

#### Problem Statement

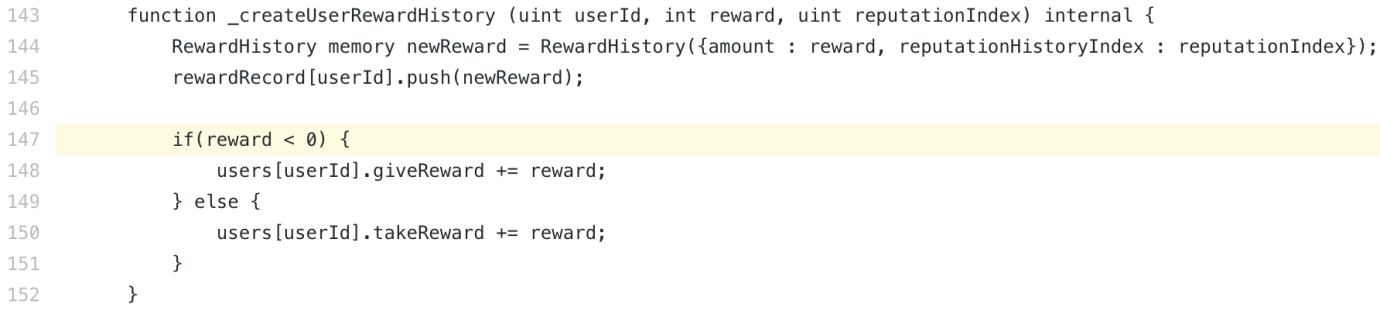
There are typos in the *\_cancleUpVoting* and *\_cancleDownVoting* functions.

#### 

#### Recommendation

It is recommended to change the names of those functions to *\_cancelUpVoting* and *\_cancelDownVoting*.

### TIPS : Larger values could be stored when uint type is used for *user.giveReward* and *user.takeReward*, instead of int type.

****(BlasqUser.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/BlasqUser.sol#L147>)

#### Problem Statement

Currently, setting of *giveReward* and *takeReward* at *User* structure is *int* type. However, when storing data, *giveReward* is always negative and *takeReward* is always positive. Therefore, the half of storage capacity of *int* type is not being used.

#### Recommendation

It is recommended to change the variable type of*giveReward* and *takeReward* to *uint*. Then, cast *uint* values according to the sign of *reward* at the code above and sum the casted values.

### TIPS : There is a typo in the name of *BlasqUser#\_createNewAdminSerivcePolicy()*.

****(BlasqUser.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/BlasqUser.sol#L171>)

#### Problem Statement

There is a typo in the function name.

#### Recommendation

It is recommended to change the function name to *\_createNewAdminServicePolicy*.

### TIPS : Restricting the access to index at outside Array range is not required.

****(BlasqUser.sol -   
<https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/BlasqUser.sol#L249>  
<https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/BlasqUser.sol#L260>)

#### Problem Statement

As EVM automatically reverts in the case of access to index at outside Array range, the concerned require statement is not essential which even consumes additional gas.

#### 

#### Recommendation

If you prefer checking through error messages or having explicit require statements, it is recommended to at least make inequality signs in the two require statements be same (in “>”). Currently, require statement at BlasqUser.sol:249 cannot prevent access to index with the same value as array length.

If an explicit require statement is not required, it is recommended to remove the require statement for gas optimization.

### TIPS : Remove unused variable at *QnaPolicy*.

****(QnaPolicy.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/QnaPolicy.sol#L13>)

#### Problem Statement

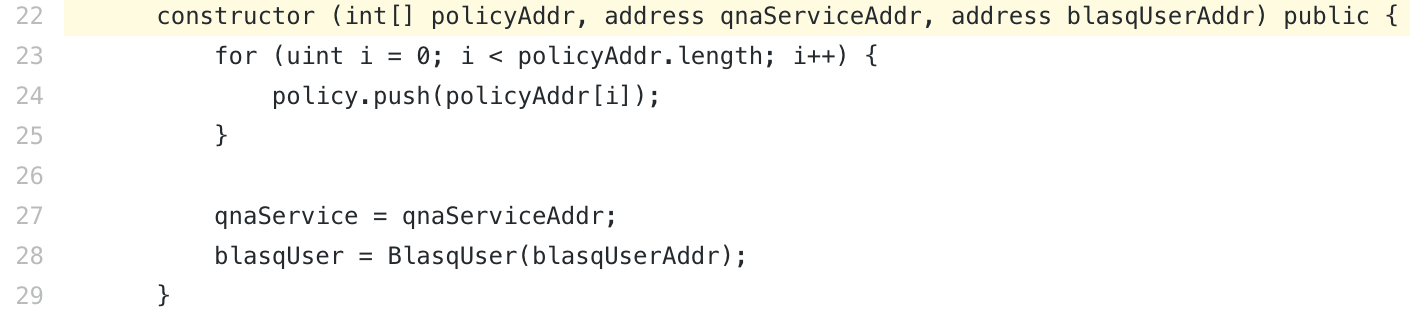
Unused constant is allocated in *QnaPolicy* contract.

#### 

#### Recommendation

It is recommended to remove the concerned constant to reduce unnecessary gas consumption during deployment. .

### TIPS : Change the name of *policyAddr* at *QnaPolicy#constructor()*.

****(QnaPolicy.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/QnaPolicy.sol#L22>)

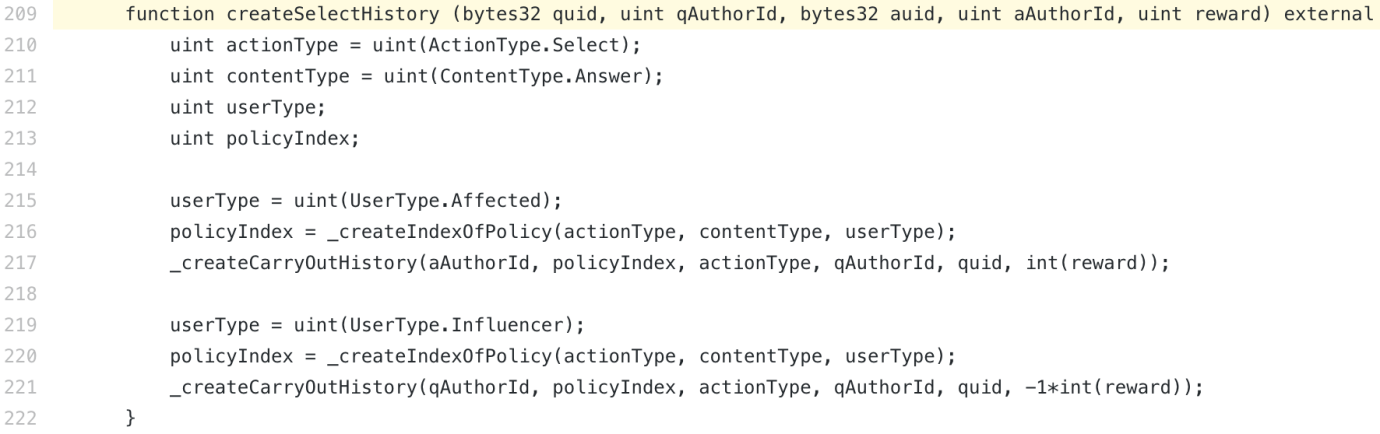
#### Problem Statement

The name of *policyAddr* used at the constructor of*QnaPolicy* contract does not match its purpose. The concerned variable is the value to store reputation point granted according to the policy. It does not match with the name *policyAddr*, which means address, and such discord harms readability of the code.

#### Recommendation

It is recommended to change the name of the concerned variable to *\_policy* or *policyPoints*, to match the purpose.

### TIPS : There is an unused input argument at *QnaPolicy#createSelectHistory()*.

****(QnaPolicy.sol - <https://github.com/ground-x/BLASQ-contract-audit/blob/master/contract/QnaPolicy.sol#L209>)

#### Problem Statement

*auid* variable, the third argument of *createSelectHistory* function is not used.

#### 

#### Recommendation

It is recommended to remove the concerned argument.

# 

# **06. Upgradeable Smart Contract Strategy**

There are two main strategies for Upgradeable Smart Contract. Unstructured Storage is a method which stores data in the storage of Proxy contract, through memory layout structure of EVM and Delegate Call. External Storage is a method which first deploys the storage contract in key-value type at an externally run storage and other contracts refer to the deployed storage contract as if they request query to a database. The table below describes the pros and cons of the two main strategies.

|  | Unstructured | External |
| --- | --- | --- |
| Common | * Data migration is not required for every upgrade. | |
| Pros | * No change in address after upgrade. * It is quite similar to previous solidity development experience. | * It has high degree of freedom in changing logics during upgrades |
| Cons | * Understanding on memory layout and delegate call is required. | * Change in address after upgrade. * There is an inconvenience that acquiring keys is required to access storage during development. |

[Table 7] Comparison of Upgradeable Strategies

For the following reasons, “HAECHI LABS” recommends the Unstructured method.

1. You can maintain a fixed address.

This can provide a better UX compared to the External Storage pattern in which the users are required to renew contract addresses in every update.

1. It has lower entry barrier for developers.

In case of the External storage pattern, the access to the storage is different from existing contract development methods so developer can face difficulties. However, as the Unstructured pattern can separate upgrade library from the business logic, Unstructured pattern does not show a significant difference from previous development methods , resulting in easier application.

## 06.a. Develop Upgradeable Smart Contract with the Unstructured Storage pattern

Unstructured storage is same with ordinary contract writing methods in almost all parts. One thing to consider is that data structure has to be designed to preserve memory layout order. For example, User and Activity structure used at QnaUser.sol could be expressed as follows.

[Figure 16] Mapping Data Structure

Data preservation is possible when variables are listed as the code above instead of adopting struct data structure. In Unstructured method, as a new version of contract inherits the older version of contract and adds new variables, if there was struct data structure in the older version of contract, new data field cannot be added to the structure.

If upgrade strategy through above mentioned structure is considered, development and deployment could be easily done through [Upgradeable Smart Contract Framework](https://github.com/HAECHI-LABS/HAECHI-CLI), developed by “HAECHI LABS”.

# 

# 

# **07. Conclusion**

This report covered 2 Handled Issues and 13 Unhandled Issues in Blasq contract, and pointed out 16 Consulting Points.

Handled Issue provides general knowledge and guide for security weaknesses handled in the contract to allow for effective handling of similar situations. Unhandled Issues are items which may cause security vulnerabilities and 1 Critical issue, 2 Major issues, 7 Minor issues and 3 Informative issues were found in Blasq contract. Critical issue is a fatal security vulnerability from which a wide range of users can be damaged and therefore the resolution of the corresponding issue is essential. Major issue is an item which either includes security vulnerability or requires modification because of unintended implementation. Minor issue is an item which requires modification because it can potentially cause problems. Lastly, Informative issue is an item modification of which can improve usability and efficiency of code although there is minimal possibility of security vulnerability. “HAECHI LABS” recommends Ground X team to modify all found issues.

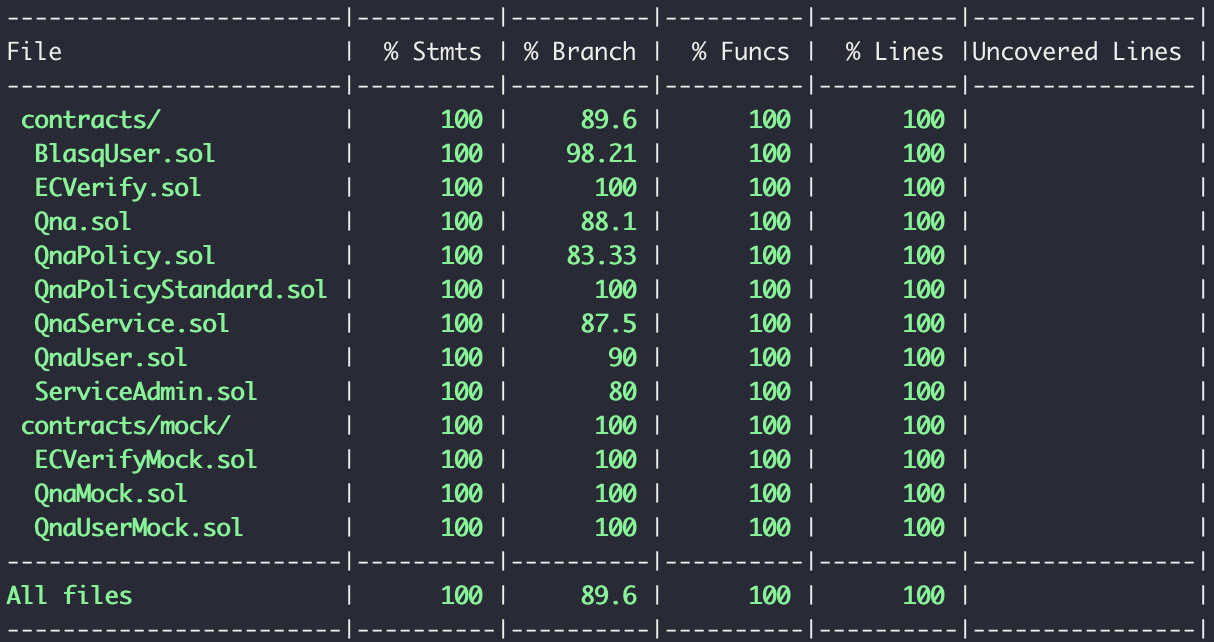
Consulting Points are structural improvements for better usability and efficiency, and 2 Gas Optimizations, 6 Best Practices, and 8 Tips were suggested in this report. Gas Optimization is a suggestion for optimization of gas consumption, with which fees for using services could be reduced. Best Practice is a suggestion on contract design, which helps improve contract in general. Tips are helpful items in improving contract when applied.

# 

# **08. Test Results**

The below data shows the result of unit test which covers major logics of the smart contract subject to security audit. The parts that are marked red are the test cases which did not pass the test due to present issue.

| Contract: BlasqUser  Control User Information  createUserInfo()  ✓ should create a new user (911ms)  ✓ should create multiple users (437ms)  ✓ should modify user's information (332ms)  deleteUser()  ✓ should delete user (231ms)  ✓ disallow to create deleted user (145ms)  changeUserState()  1) should set states Deleted  ✓ should set states LongBann (579ms)  ✓ should set states ShortBann (130ms)  ✓ should set states Sleep (352ms)  ✓ should set states Lock (119ms)  Control Policy Information  createAdminServicePolicy()  ✓ should apply address as service (265ms)  ✓ should modify service address (191ms)  ✓ should apply multiple addresses as service (271ms)  2) should keep 1:1 relationship between policy address and policy type  deleteAdminServicePolicy()  ✓ should delete service policy (125ms)  Control User Reputation  createUserHistory()  ✓ should createUserHistory without reward (584ms)  ✓ should update reputation without reward (421ms)  ✓ should give minus reputation without reward (235ms)  ✓ should give reputation with reward (359ms)  ✓ should give reputation with minus reward (249ms)  ✓ should update reputation with reward (379ms)  migrateUserReputation()  ✓ should increase reputation of user (153ms)  ✓ should decrease reputation of user (108ms)  3) should reject 0 userId  Control Klay Flow  fallback()  ✓ should receive Klay from anyone (172ms)  withdrawKlay()  ✓ should withdraw Klay from contract (170ms)  Contract: ECVerify  ecverify() with string message  ✓ should verify signature with given message (49ms)  ✓ should reject wrong signature length (55ms)  ✓ should reject when signer is different (44ms)  ecverify() with bytes message  ✓ should verify signature with given message (46ms)  ✓ should reject wrong signature length (54ms)  ✓ should reject when signer is different (44ms)  Contract: Qna  Question  createQuestion()  ✓ should create a question (255ms)  ✓ disallow to create a question, if the quid already exist (126ms)  modifyQuestion()  ✓ should modify the question (169ms)  ✓ disallow to modify the question, if the quid does not match  ✓ disallow to modify the question, if the userId does not match  ✓ disallow to modify the question, if the QnaType is not QnaType.Question (128ms)  ✓ disallow to modify the question, if the question is deleted (191ms)  ✓ disallow to modify the question, if someone already voted (143ms)  ✓ disallow to modify the question, if someone already answered (162ms)  deleteQuestion()  ✓ should delete the question, if the escrowedKlay amount is zero (157ms)  ✓ should delete the question, if the escrowedKlay amount is not zero (229ms)  ✓ disallow to delete the question, if the quid does not match  ✓ disallow to delete the question, if the userId does not match  ✓ disallow to delete the question, if the QnaType is not QnaType.Question (120ms)  ✓ disallow to delete the question, if the question is deleted (123ms)  ✓ disallow to delete the question, if someone already voted (209ms)  ✓ disallow to delete the question, if someone already answered (186ms)  Escrow  createMinEscrow()  ✓ should change the minimum escrow amount (60ms)  escrowKlayForQuestion()  ✓ should escrow the klay amount for the question (183ms)  ✓ disallow to escrow the klay, if the klay is less than the minimum escrow amount  ✓ disallow to escrow the klay, if msg.value is less than the amount (160ms)  ✓ disallow to escrow the klay, if quid does not match  ✓ disallow to escrow the klay, if QnaType is not QnaType.Question (128ms)  ✓ disallow to escrow the klay, if QnaState is not QnaState.Active (122ms)  ✓ disallow to escrow the klay, if escrowedKlay is not zero (186ms)  4) should return the klay, if msg.value is bigger than amount  Answer  createAnswer()  ✓ should create an answer (376ms)  ✓ disallow to create an answer, if the quid already exist (125ms)  ✓ disallow to create an answer, if the question is not QnaType.Question (116ms)  ✓ disallow to create an answer, if the question is QnaType.Deleted (106ms)  ✓ disallow to create an answer, if the userId is same as question (47ms)  modifyAnswer()  ✓ should modify the answer (148ms)  ✓ disallow to modify the answer, if the auid does not match  ✓ disallow to modify the answer, if the parentId is not quid  ✓ disallow to modify the answer, if the userId does not match  ✓ disallow to modify the answer, if the QnaType is not QnaType.Answer  ✓ disallow to modify the answer, if the answer is already deleted (128ms)  ✓ disallow to modify the answer, if someone already voted (138ms)  deleteAnswer()  ✓ should delete the answer (149ms)  ✓ disallow to delete the answer, if the auid does not match  ✓ disallow to delete the answer, if the QnaType is not QnaType.Answer  ✓ disallow to delete the answer, if the answer is already deleted (114ms)  ✓ disallow to delete the answer, if someone already voted (134ms)  select()  ✓ should reject ZERO\_ADDRESS  ✓ should select answer without reward (109ms)  ✓ should select answer with reward (394ms)  ✓ disallow to select answer, if the answer was already selected (152ms)  ✓ disallow to select answer, if the answer was deleted (91ms)  ✓ disallow to select answer, if the answer and question have different QnaType  ✓ disallow to select unregistered userId  Vote  carryoutVoting()  ✓ should carry out up voting (50ms)  ✓ should carry out down voting (49ms)  createFirstVoteForContents()  ✓ should create the first vote history (74ms)  carryoutUpVoting()  ✓ should carry out up voting (85ms)  carryoutDownVoting()  ✓ should carry out down voting (87ms)  cancelVoting()  ✓ should cancel the voting (121ms)  ✓ disallow to cancel the voting, if the vote type does not match (169ms)  cancelUpVoting()  ✓ should cancel the up voting (141ms)  cancelDownVoting()  ✓ should cancel the down voting (139ms)  voting()  ✓ should vote the content (115ms)  ✓ should cancel previous up vote (226ms)  ✓ should cancel previous down vote (237ms)  ✓ disallow to vote the content, if the content id does not match  ✓ disallow to vote the content, if the voter id is same as the content writer  ✓ disallow to vote the content, if the QnaState is QnaState.Deleted (155ms)  ✓ disallow to vote the content, if the VoteType is VoteType.Noting  Contract: QnaPolicy  Creating Contract: constructor()  ✓ should create contract (365ms)  5) should reject policy with wrong length  6) should reject zero address of service  7) should reject zero address of BlasqUser  Creating Vote History: createVoteHistory()  Rejecting invalid pre-condition  ✓ should reject when policy is not registered at BlasqUser (168ms)  ✓ should reject when user is not registered at BlasqUser (76ms)  ✓ should reject invalid caller (87ms)  ✓ should reject invalid ContentType  ✓ should reject invalid ActionType (52ms)  \_createCarryOutVoteHistory()  \_createCarryOutUpVoteHistory()  \_createCarryOutUpVoteForQuestionHistory()  ✓ should create Vote History (259ms)  \_createCarryOutUpVoteForAnswerHistory()  ✓ should create Vote History (257ms)  \_createCarryOutDownVoteHistory()  \_createCarryOutDownVoteForQuestionHistory()  ✓ should create Vote History (253ms)  \_createCarryOutDownVoteForAnswerHistory()  ✓ should create Vote History (477ms)  \_createCancelVoteHistory()  \_createCancelUpVoteHistory()  \_createCancelUpVoteForQuestionHistory()  ✓ should create Cancel History (262ms)  \_createCancelUpVoteForAnswerHistory()  ✓ should create Cancel History (255ms)  \_createCancelDownVoteHistory()  \_createCancelDownVoteForQuestionHistory()  ✓ should create Cancel History (253ms)  \_createCancelDownVoteForAnswerHistory()  ✓ should create Cancel History (454ms)  Creating Select History: createSelectHistory()  ✓ should reject invalid caller  ✓ should create reward history (460ms)  8) should create reward history with large reward  Changing Global Variables  createPolicy()  ✓ should change policy (412ms)  9) should reject policy with wrong length  createAdminService()  ✓ should change service address (68ms)  10) should reject zero address  createAdminBlasqUser()  ✓ should change BlasqUser address (69ms)  11) should reject zero address  Control Klay Flow  fallback()  ✓ should receive Klay from anyone (157ms)  withdrawKlay()  ✓ should withdraw Klay from contract (161ms)  Contract: QnaService  Question  createQuestion()  ✓ should send a transaction about creating a question by the owner (122ms)  ✓ disallow for the non-active user to create a question (51ms)  ✓ disallow to send a transaction about creating a question by other than owner  modifyQuestion()  ✓ should send a transaction about modifying the question by the owner (105ms)  ✓ disallow for non-active user to modify the question (45ms)  ✓ disallow to send a transaction about modifying the question by other than the owner  deleteQuestion()  ✓ should send a transaction about deleting the question without escrowdKlay by the owner (113ms)  ✓ should send a transaction about deleting the question with escrowdKlay by the owner (408ms)  ✓ disallow for non-active user to delete the question (43ms)  ✓ disallow to send a transaction about deleting the question by other than owner  escrowKlayForQuestion()  ✓ should escrow the klay for the question by the active user (185ms)  ✓ disallow to escrow the klay for the question by the non-active user (41ms)  Answer  createAnswer()  ✓ should send a transaction about creating an answer by the owner (147ms)  ✓ disallow for the non-active user to create an answer (48ms)  ✓ disallow to send a transaction about creating an answer by other than owner  modifyAnswer()  ✓ should send a transaction about modifying the answer by the owner (102ms)  ✓ disallow for non-active user to modify the answer (46ms)  ✓ disallow to send a transaction about modifying the answer by other than owner  deleteAnswer()  ✓ should send a transaction about deleting the question by the owner (93ms)  ✓ disallow for non-active user to delete the question (45ms)  ✓ disallow to send a transaction for deleting the question by other than owner  Vote  voting()  ✓ should send a transaction about the voting by the owner (241ms)  ✓ disallow for the non-active user to vote (40ms)  ✓ should send a transaction about the voting by the owner  Select the Answer  select()  ✓ should send a transaction about selecting the answer by the owner (307ms)  ✓ disallow for non-active user to select the answer (97ms)  ✓ disallow to sending a transaction about selecting the answer by other than owner  User  createBlasqUser()  ✓ should set the address as blasq user contract (134ms)  12) disallow to set the ZERO\_ADDRESS as blasq user contract  Policy  createQnaPolicy()  ✓ should set the address as the policy contract (170ms)  13) disallow to set the ZERO\_ADDRESS as policy contract  Klay Management  createMinEscrow()  ✓ should set the new minimum escrow amount by the owner (68ms)  ✓ disallow to set the new minimum escrow amount by other than owner  withdrawKlay()  ✓ should withdraw the klay from qnaService contract by the owner (237ms)  ✓ disallow to withdraw the klay from qnaService contract by other than owner (149ms)  ✓ disallow to withdraw the klay from qnaService contract by the owner, if the amount > contract balance - totalEscrowedAmount  Contract: QnaUser  Create QnaUser  createQnaUser()  ✓ should create a QnaUser (136ms)  Create Activity  createActivity()  ✓ disallow invalid activityType  When the user does not exist  ✓ should create an activity about the question (63ms)  ✓ should create an activity about the answer (64ms)  ✓ should create an activity about the vote (60ms)  When the user exist  ✓ should create an activity for question (52ms)  ✓ should create an activity for answer (54ms)  ✓ should create an activity for vote (56ms)  Contract: ServiceAdmin  Constructor  constructor()  ✓ should assign msg.sender as admin.  Admin Ownership Control  createAdmin()  ✓ should assign new address as admin by admin (75ms)  ✓ disallow to assign new address as admin by other than admin (38ms)  ✓ disallow to assign the already registered admin address as a new admin (88ms)  ✓ disallow to create more than 5 admins (319ms)  deleteAdmin()  ✓ should remove the address from admin addresses by the admin (91ms)  ✓ disallow to remove the address from admin addresses by other than admin (39ms)  Admin Ownership Read  checkAdmin()  ✓ should return true, if the address is assigned as admin  ✓ should return false, if the address is not assigned as admin  readAllAdmin()  ✓ should return the addresses that is assigned ad admin (50ms)  167 passing (2m)  13 failing |
| --- |

[Figure17] Test case code coverage

# **09. Disclaimer**

This report does not guarantee any advice on investment, suitability of business model, nor safety of code without any bug. This report is only used for the sake of discussion on known technical issues and there could be unidentified or unfound issues including defects in Ethereum or Solidity. In order to create a secure smart contract, modification on the found issues and adequate testing are required.

1. https://ethereum.github.io/yellowpaper/paper.pdf [↑](#footnote-ref-0)
2. <https://blog.trailofbits.com/2018/10/29/how-contract-migration-works/> [↑](#footnote-ref-1)