# # Competitive Security Assessment

## GoMutual

Mar 20th, 2023

Secure3

secure3.io

# Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

# Overview

### Project Detail

| | |
|---|---|
| **Project Name** | GoMutual |
| **Platform & Language** | Solidity |
| **Codebase** | • https://github.com/GOMUTUAL/go-mutual-contract<br>• audit commit - 8980625660e661ae1efc2f610c214f67c4c9e5fa<br>• final commit - a4086675636171a3d5e64cfa2b60092ff5b13af5 |
| **Audit Methodology** | • Audit Contest<br>• Business Logic and Code Review<br>• Privileged Roles Review<br>• Static Analysis |

### Code Vulnerability Review Summary

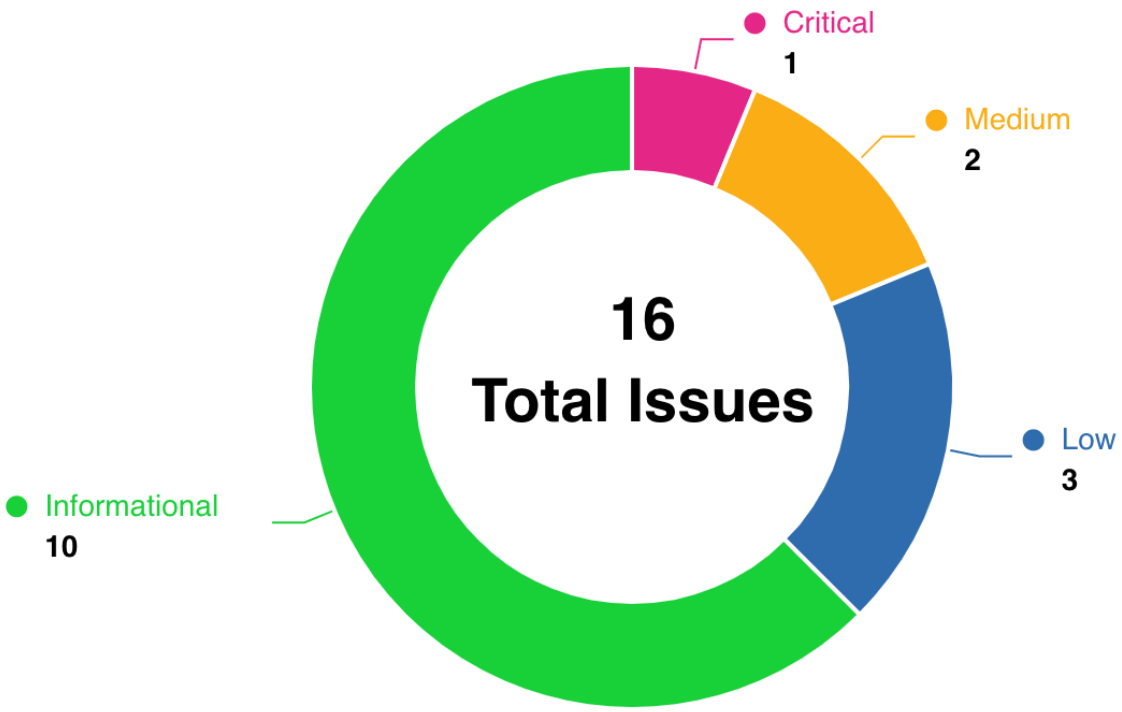
| Vulnerability Level | Total | Reported | Acknowledged | Fixed | Mitigated | Declined |
|---|---|---|---|---|---|---|
| Critical | 1 | 0 | 0 | 1 | 0 | 0 |
| Medium | 2 | 0 | 0 | 2 | 0 | 0 |
| Low | 3 | 0 | 1 | 2 | 0 | 0 |
| Informational | 10 | 0 | 3 | 7 | 0 | 0 |

# Audit Scope

| File | Commit Hash |
|---|---|
| src/contracts/Asset/IAssetGateway.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Asset/SurplusProxy.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Asset/ISurplusPool.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Asset/SurplusPool.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Asset/SurplusStorage.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Asset/ERC20AssetGateway.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Plan/IPlan.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Plan/IPlanUtil.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Plan/PlanFactory.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Plan/IPlanFactory.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Plan/Plan.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Plan/PlanUtil.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Common/Registry.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Common/Base.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Common/IRegistry.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/GMEvent/GMEvent.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/GMEvent/GMEventStorage.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/GMEvent/GMEventCreator.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/GMEvent/IGMEvent.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/GMEvent/GMEventFactory.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/GMEvent/PatchCreator.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/GMEvent/IPatch.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/GMEvent/Patch.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/GMEvent/IGMEventFactory.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/GMEvent/GMEventAbstract.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Comptroller/Comptroller.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |

| | |
|---|---|
| src/contracts/Comptroller/EpochBeaconProxy.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Comptroller/Epoch.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Comptroller/IEpoch.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Comptroller/IComptroller.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Comptroller/ComptrollerStorage.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |
| src/contracts/Comptroller/ComptrollerProxy.sol | 8980625660e661ae1efc2f610c214f67c4c9e5fa |

# Code Assessment Findings




| ID | Name | Category | Severity | Status | Contributor |
|---|---|---|---|---|---|
| GMT-1 | Apply Checks-Effects-Interactions defensively to mitigate reentrancy | Reentrancy | Informational | Fixed | yosriady |
| GMT-2 | DoS in `GMEvent` contract | DoS | Medium | Fixed | infinityhacker |
| GMT-3 | ERC20 transfer and transferFrom return values not checked | Logical | Medium | Fixed | 0xzoobi |
| GMT-4 | ERC777 reentrancy in `ERC20AssetGateway` | Reentrancy | Critical | Fixed | BradMoonUESTC, yosriady |

| | | | | | |
|---|---|---|---|---|---|
| GMT-5 | Event are missing indexed fields | Code Style | Informational | Fixed | 0xzoobi |
| GMT-6 | Events not emitted for important state changes | Code Style | Informational | Fixed | 0xzoobi |
| GMT-7 | Incorrect and floating solidity version | Integer Overflow and Underflow | Informational | Fixed | BradMoonUESTC, yosriady |
| GMT-8 | Lack of zero address validation | Logical | Low | Fixed | 0xzoobi |
| GMT-9 | Missing `startTime` and `expiredTime` validation in Plan | Logical | Low | Fixed | yosriady |
| GMT-10 | Missing error message in require statements | Code Style | Informational | Fixed | BradMoonUESTC, 0xzoobi |
| GMT-11 | Missing fields in events | Code Style | Informational | Acknowledged | yosriady |
| GMT-12 | Possible DOS (out-of-gas) on loops | DOS | Low | Acknowledged | 0xzoobi |
| GMT-13 | Remove unused Hardhat imports | Gas Optimization | Informational | Fixed | yosriady |
| GMT-14 | State variables that never be used could be deleted or declared constant | Gas Optimization | Informational | Fixed | BradMoonUESTC |
| GMT-15 | Unfinished TODOs and missing implementation | Code Style | Informational | Acknowledged | BradMoonUESTC, yosriady |
| GMT-16 | Unused return value | Code Style | Informational | Acknowledged | BradMoonUESTC |

# GMT-1:Apply Checks-Effects-Interactions defensively to mitigate reentrancy

| Category | Severity | Code Reference | Status | Contributor |
| --- | --- | --- | --- | --- |
| Reentrancy | Informational | • code/src/contracts/Activity/GoodDriving.sol#L82-L83<br>• code/src/contracts/GMEvent/GMEventAbstract.sol#L113-L114 | Fixed | yosriady |

## Code

```
82:             _getToken().mint(IPlan(_plan).planId(), _plan, name);
83:             isNftDistributed[_plan] = true;

113:         IPlan(planContract()).payForGMEvent(address(this));
114:         isLocked = true;
```

## Description

**yosriady :** The USDC contract is upgradable and could theoretically implement ERC777 token callbacks. In that case, many instances of reentrancy can occur.

## Recommendation

**yosriady :** Use the Checks-Effects-Interactions best practice and make all state changes before calling external contracts. Also, consider using function modifiers such as `nonReentrant` from Reentrancy Guard to prevent re-entrancy at the contract level.

## Client Response

Fixed.We have added `nonReentrant` to the special functions.

# GMT-2:DoS in `GMEvent` contract

| Category | Severity | Code Reference | Status | Contributor |
|---|---|---|---|---|
| DoS | Medium | • code/src/contracts/GMEvent/GMEventAbstract.sol#L228-L229 | Fixed | infinityhacker |

## Code

```
228:        uint256 _balance = _getUsdc().balanceOf(address(this));
229:        return amount - _balance;
```

## Description

**infinityhacker :** In the `Plan` contract, there is a `debt` function to determine if GMEvent amount is larger than the current token balance in contract. the `debt()` function is called in `settlement()` function. Because the contract run in `solidity >=0.7.0 <0.9.0` and directly use `amount - _balance` to get the difference, so if `_balance` is larger than the `amount`, the `settlement` function will revert. A malicious attacker can send unit 1 token to the GMEvent contract after `approve` is called, which will finally break the logic in `debt` function and cause reverts. Btw, because each GMEvent can only approve once, so `beneficiaries` will never get paid.

POC:

```
contract hackGMEvent{
     function DoShack() public {
         // breaks debt function
         IERC20(usdc).transfer(GMEventContract, 1);
     }
}
```

## Recommendation

**infinityhacker :** Consider below fix in the `debt` function

```
function debt() public view override returns (uint256) {
        uint256 _balance = _getUsdc().balanceOf(address(this));
        if (balance > amount) {return 0}
        return amount - _balance; // auditor: anyone can send unit 1 usdc to this contract to break
to check and cause error, becasue each event can only approve once, which will cause infinity ddos
    }
```

## Client Response

Fixed.If the balance more than the `amount` that event needed, we will return `0`.

# GMT-3:ERC20 transfer and transferFrom return values not checked

| Category | Severity | Code Reference | Status | Contributor |
|---|---|---|---|---|
| Logical | Medium | • code/src/contracts/Asset/SurplusPool.sol#L43<br>• code/src/contracts/Asset/ERC20AssetGateway.sol#L84<br>• code/src/contracts/Asset/ERC20AssetGateway.sol#L128<br>• code/src/contracts/GMEvent/GMEventAbstract.sol#L164<br>• code/src/contracts/Asset/ERC20AssetGateway.sol#L169<br>• code/src/contracts/Plan/Plan.sol#L173<br>• code/src/contracts/Comptroller/Epoch.sol#L180<br>• code/src/contracts/GMEvent/GMEventAbstract.sol#L181<br>• code/src/contracts/Comptroller/Epoch.sol#L206<br>• code/src/contracts/Comptroller/Epoch.sol#L244<br>• code/src/contracts/Plan/Plan.sol#L246<br>• code/src/contracts/Plan/Plan.sol#L275<br>• code/src/contracts/Comptroller/Epoch.sol#L317 | Fixed | 0xzoobi |

## Code

```
43:         usdc.transferFrom(msg.sender, address(this), amount);

84:         IERC20(tokenAddress).transferFrom(from, address(this), amount);

128:         IERC20(tokenAddress).transfer(certificate.depositor, amount);

164:             _getUsdc().transferFrom(

169:             IERC20(tokenAddress).transfer(payer, certificate.amount);

173:         _getUsdc().transferFrom(_payerProxy, address(this), planValue);

180:         IERC20(getUsdcTokenAddress()).transferFrom(address(planContract), address(this),
amount);

181:             _usdcContract.transfer(

206:         IERC20(getUsdcTokenAddress()).transferFrom(msg.sender, address(this), amount);

244:         token.transfer(payer, amount);

246:         _getUsdc().transfer(owner, _balance);

275:         _getUsdc().transfer(address(_gmEvent), _gmEventAmount);

317:             IERC20(getUsdcTokenAddress()).transfer(donationValue.user, value);
```

## Description

**0xzoobi :** ERC20 implementations are not always consistent. Some implementations of `transfer` and `transferFrom` could return false on failure instead of reverting.

The impact is that the ERC20 token transfer or USDC transfer (deposit, withdraw and pay via Plan, donate and withdraw via Comptroller etc.,) to the depositor, payer, contract owner may fail but the code executes as if the transfer took place successfully.

Consider below POC contract

```
contract Token {
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool success);
}
contract MyBank{
    mapping(address => uint) balances;
    Token token;
    function deposit(uint amount) public{
        token.transferFrom(msg.sender, address(this), amount); //returns false instead of revert
        balances[msg.sender] += amount;
    }
}
```

## Recommendation

**0xzoobi :** use OpenZeppelin's safeERC20's `safeTransfer()` and `safeTransferFrom()` function. The SafeERC20 implementation validates the transfer calls via `_callOptionalReturn` function.

```
/**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract),
relaxing the requirement
     * on the return value: the return value is optional (but if data is returned, it must not be
false).
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function _callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking
mechanism, since
        // we're implementing it ourselves. We use {Address-functionCall} to perform this call,
which verifies that
        // the target address contains contract code and also asserts for success in the low-level
call.

        bytes memory returndata = address(token).functionCall(data, "SafeERC20: low-level call faile
d");
        if (returndata.length > 0) {
            // Return data is optional
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
```

## Client Response

Fixed.We have changed all `transfer` and `transferFrom` functions to `safeTransfer` and `safeTransferFrom`

# GMT-4:ERC777 reentrancy in `ERC20AssetGateway`

| Category | Severity | Code Reference | Status | Contributor |
| --- | --- | --- | --- | --- |
| Reentrancy | Critical | • code/src/contracts/Asset/ERC20AssetGateway.sol#L84-L86<br>• code/src/contracts/Asset/ERC20AssetGateway.sol#L118<br>• code/src/contracts/Asset/ERC20AssetGateway.sol#L128-L129 | Fixed | BradMoonUESTC, yosriady |

## Code

```
84:         IERC20(tokenAddress).transferFrom(from, address(this), amount);
85:
86:         address originPayer = depositionCertificates[planId].depositor;

118:     function withdraw(uint256 planId) public returns (uint256) {

128:         IERC20(tokenAddress).transfer(certificate.depositor, amount);
129:         certificate.amount = 0;
```

## Description

**BradMoonUESTC :** When ERC20 token address support ERC777 standard with `tokenReceived()` function, attacker may make reentrancy call to the `withdraw` and `deposit` function to implement an attack.

In the `withdraw()` because the state variable update `certificate.amount = 0;` is after the external call, this is problematic. similar in `deposit()` function

**yosriady :** There is a potential reentrancy in the `withdraw()` function if `tokenAddress` is an ERC777 token. This is because the internal state `certificate.amount = 0` is only updated AFTER the external call instead of BEFORE.

Consider below POC contract

```
    function withdraw(uint256 planId) public returns (uint256) {
        DepositionCertificate storage certificate = depositionCertificates[
            planId
        ];
        address payer = certificate.depositor;
        require(payer == msg.sender, "EAG_04");
        require(certificate.amount > 0, "EAG_03");

        uint256 amount = certificate.amount;

        IERC20(tokenAddress).transfer(certificate.depositor, amount);
        certificate.amount = 0;

        emit UserWithdrawed(msg.sender, planId, amount);
        return amount;
    }
```

This allows the caller to re-enter and withdraw more than they should.

## Recommendation

**BradMoonUESTC :** use `nonReentrant` modifier in `ReentrancyGuard` contract. more details please refer to https://docs.openzeppelin.com/contracts/4.x/api/security#ReentrancyGuard
**yosriady :** Use the Checks-Effects-Interactions.

## Client Response

Fixed.We added `nonReentrant()` modifier.

# GMT-5:Event are missing indexed fields

| Category | Severity | Code Reference | Status | Contributor |
|---|---|---|---|---|
| Code Style | Informational | • code/src/contracts/Asset/SurplusStorage.sol#L4<br>• code/src/contracts/Comptroller/IComptroller.sol#L7<br>• code/src/contracts/GMEvent/GMEventFactory.sol#L13<br>• code/src/contracts/Comptroller/IComptroller.sol#L16<br>• code/src/contracts/Comptroller/IComptroller.sol#L17<br>• code/src/contracts/Common/Registry.sol#L19<br>• code/src/contracts/Comptroller/IEpoch.sol#L19<br>• code/src/contracts/Comptroller/IEpoch.sol#L20<br>• code/src/contracts/Plan/PlanFactory.sol#L20<br>• code/src/contracts/Common/Registry.sol#L21<br>• code/src/contracts/GMEvent/GMEventFactory.sol#L21<br>• code/src/contracts/Comptroller/IEpoch.sol#L22<br>• code/src/contracts/Asset/ERC20AssetGateway.sol#L32 | Fixed | 0xzoobi |

| | | | | |
|---|---|---|---|---|
| | | • code/src/contracts/Comptroller/IEpoch.sol#L32<br>• code/src/contracts/Comptroller/IEpoch.sol#L34<br>• code/src/contracts/Comptroller/IEpoch.sol#L35<br>• code/src/contracts/Comptroller/IEpoch.sol#L36<br>• code/src/contracts/Asset/ERC20AssetGateway.sol#L40<br>• code/src/contracts/GMEvent/GMEventAbstract.sol#L40<br>• code/src/contracts/Plan/Plan.sol#L45<br>• code/src/contracts/Plan/Plan.sol#L62 | | |

## Code

```
4:    event Deposited(uint epoch, uint amount);

7:    event PlanPayed(

13:    event NewGMEvent(

16:    event EpochLocked(uint256 epoch, address epochAddress);

17:    event EpochStart(uint256 epoch);

19:    event GMEventRecorded(uint epoch, string gmEventId, address addr, uint amount);

19:    event NewContractAddress(string name, ContractDetail detail);

20:    event PlanDeducted(uint epoch, uint planId, address plan, uint amount);

20:    event NewPlan(

21:    event NewPatch(

21:    event RemoveContractAddress(string name);

22:    event GapDonationDeducted(uint epoch, address member, uint index, uint amount);

32:    event UserDeposited(address user, uint256 planId, uint256 amount);

32:    event GapDonateDeducted(uint epoch, address member, uint index, uint donateAmount, uint
finalAmount);

34:    event GMEventWithdrawn(uint epoch, string gmEventId, address gmEventAddress, uint
finalAmount);

35:    event GapDonationStart(uint epoch, uint amount);

36:    event EpochFinished(uint epoch, uint surplus, uint256 totalDailyDoanteValue, uint256
totalCalimValue, uint256 totalGapDonateValue);

40:    event UserWithdrawed(address user, uint256 planId, uint256 amount);

40:    event StateChanged(GMEventState oldState, GMEventState newState);

45:    event BalanceChanged(
```

```
62:     event NewGMEvent(address contractAddress);
```

## Description

**0xzoobi :** Indexed parameters are searchable parameters and help to query events. Non-Indexed parameters are regular parameters passed to an event that is not searchable and are only used to log the messages to the blockchain.
Each event can use three indexed fields.

## Recommendation

**0xzoobi :** use the `indexed` keyword for the parameters you want the be searchable via indexing protocols like the Graph.
Consider below fix

```
event Deposited(uint256 indexed epoch, uint256 indexed amount);

function deposit()  external {
        emit Deposited(epoch, amount);
}
```

## Client Response

Fixed.We've added indexed for the `variable` that we need to index.

# GMT-6:Events not emitted for important state changes

| Category | Severity | Code Reference | Status | Contributor |
|---|---|---|---|---|
| Code Style | Informational | • code/src/contracts/Common/Base.sol#L27<br>• code/src/contracts/Asset/SurplusPool.sol#L61<br>• code/src/contracts/Asset/ERC20AssetGateway.sol#L147 | Fixed | 0xzoobi |

## Code

```
27:    function _setRegistryAddress(address _registryAddress) internal {

61:    function setTokenPrice(uint256 _tokenPrice, uint256 _tokenPriceDecimal) onlyRegisteredRole("MAINTENANCE_ROLE") public {

147:    function pay(
```

## Description

**0xzoobi :** When changing state variables events are not emitted. Emitting events allows monitoring activities with off-chain monitoring tools. It also provides transparency to the users when some important changes are made to the protocol.

## Recommendation

**0xzoobi :** Emit an event to track the event..

Sample Example for Fix

```
event Deposited(uint256 epoch, uint256 amount);

function deposit()  external {
        emit Deposited(epoch, amount);
}
```

## Client Response

Fixed.We've added two events: `PlanPayed` and `TokenPriceChanged`.

# GMT-7:Incorrect and floating solidity version

| Category | Severity | Code Reference | Status | Contributor |
| --- | --- | --- | --- | --- |
| Integer Overflow and Underflow | Informational | • code/src/contracts/GMEvent/GMEventAbstract.sol#L2<br>• code/src/contracts/Comptroller/Comptroller.sol#L162<br>• code/src/contracts/Comptroller/Comptroller.sol#L179<br>• code/src/contracts/Comptroller/Comptroller.sol#L188<br>• code/src/contracts/Comptroller/Epoch.sol#L205<br>• code/src/contracts/Comptroller/Epoch.sol#L217<br>• code/src/contracts/Comptroller/Epoch.sol#L238<br>• code/src/contracts/Comptroller/Epoch.sol#L241 | Fixed | BradMoonUESTC, yosriady |

## Code

```
2:pragma solidity >=0.7.0 <0.9.0;

162:        uint256 _endPoint = epochValue.currentPlanIndex + batch;

179:                totalDonateAmount += amount;

188:        epochValue.totalDonateAmount += totalDonateAmount;

205:        uint value = donationValue.value + amount;

217:        totalGapDonationValue += value;

238:        value = value - amount;

241:        totalGapDonationValue -= amount;
```

## Description

**BradMoonUESTC :** The versions of most contracts are limited to 0.7-0.9. Even though the deployment script is restricted, if you don't pay attention, this may cause some logic to generate integer overflow
**yosriady :** Contracts should be deployed with the same compiler version and flags that they have been tested the most with. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs. Contracts may also be deployed by others and the pragma indicates the compiler version intended by the original authors.
https://swcregistry.io/docs/SWC-103

## Recommendation

**BradMoonUESTC :** limit the version of contracts to 0.8-0.9
**yosriady :** Instead of:

```
pragma solidity >=0.7.0 <0.9.0;
```

Do:

```
pragma solidity 0.8.18;
```

## Client Response

Fixed.The solidity version is `0.8.9` now.

# GMT-8:Lack of zero address validation

| Category | Severity | Code Reference | Status | Contributor |
|---|---|---|---|---|
| Logical | Low | • code/src/contracts/GMEvent/Patch.sol#L33<br>• code/src/contracts/GMEvent/GMEvent.sol#L40<br>• code/src/contracts/Asset/ERC20AssetGateway.sol#L61 | Fixed | 0xzoobi |

## Code

```
33:         gmEventAddress = _gmEventAddress;

40:         _planContract = _planAddress;

61:         tokenAddress = _tokenAddress;
```

## Description

**0xzoobi :** The parameters that are used in the constructor are to initialize the state variable. Some of them lacks zero address validation, it will be problematic if there is error in these state variable. Some of the function will loss their functionality which can cause the redeployment of contract.

## Recommendation

**0xzoobi :** Add a require condition which validates zero address.

Consider below fix

```
require(address(tokenAddress) != address(0));
```

## Client Response

Fixed.We've added `ERC165` supports for some contract and validated the `address(0)`

# GMT-9:Missing `startTime` and `expiredTime` validation in Plan

| Category | Severity | Code Reference | Status | Contributor |
| --- | --- | --- | --- | --- |
| Logical | Low | • code/src/contracts/Plan/PlanFactory.sol#L36<br>• code/src/contracts/Plan/Plan.sol#L108 | Fixed | yosriady |

## Code

```
36:    function create(

108:    function setInfo(
```

## Description

**yosriady :** In `PlanFactory`, currently, all that is checked when creating a plan is that the plan doesn't already exist in the planById mapping. Other checks should be done, such as verifying that the input startTime is not before the current block timestamp and that the amount is greater than 0.

Likewise in `Plan.setInfo` function, it allows expired plans to be created by accident.

## Recommendation

**yosriady :** Validate that `startTime < expiredTime` and any other relevant validations.

## Client Response

Fixed

# GMT-10:Missing error message in require statements

| Category | Severity | Code Reference | Status | Contributor |
| --- | --- | --- | --- | --- |
| Code Style | Informational | • code/src/contracts/Comptroller/Comptroller.sol#L56<br>• code/src/contracts/Plan/Plan.sol#L65<br>• code/src/contracts/Comptroller/Comptroller.sol#L72<br>• code/src/contracts/Plan/Plan.sol#L77<br>• code/src/contracts/Comptroller/Epoch.sol#L101<br>• code/src/contracts/Comptroller/Comptroller.sol#L143<br>• code/src/contracts/Plan/Plan.sol#L145<br>• code/src/contracts/Comptroller/Comptroller.sol#L226 | Fixed | BradMoonUESTC, 0xzoobi |

## Code

```
56:         require(planState.plan == address(0));

65:         require(msg.sender == owner, "ROLE_000003");

72:         require(code == uint256(Error.NO_ERROR));

77:         require(_isMyGMEvent, "ROLE_000004");

101:         require(msg.sender == getComptroller());

143 :         require(epochValue.state == EpochState.Start);

145:         require(_gmEventAddress != address(0));

226:         require(plan != address(0) && isValidPlan(plan), "COMP_000003");
```

## Description

**BradMoonUESTC :** require can be used to check for conditions and throw an exception if the condition is not met, in which case the error message provided by the developer will appear. This is why a very descriptive error message is needed.
**0xzoobi :** An error message in the require statement helps both user and dev to understand why an execution has failed.

## Recommendation

**BradMoonUESTC :** Adding an error message describing the failed condition.
**0xzoobi :** Adding error messages

## Client Response

Fixed

# GMT-11:Missing fields in events

| Category | Severity | Code Reference | Status | Contributor |
|---|---|---|---|---|
| Code Style | Informational | • code/src/contracts/GMEvent/GMEventFactory.sol#L90-L102 | Acknowledged | yosriady |

## Code

```
90:         address _gmEventAddress = _creator.create(
91:             _plan.owner(),
92:             _planContract,
93:             _id,
94:             _amount,
95:             _gmEventTime,
96:             address(registry)
97:         );
98:
99:         _plan.report(_gmEventAddress);
100:          _getStorage().add(_id, _gmEventAddress);
101:
102:          emit NewGMEvent(_gmEventAddress, _planContract, _id, _amount, _gmEventTime);
```

## Description

**yosriady :** The `NewGMEvent` is missing the `owner` parameter. It should have three parameters after `gmEventId`: `address owner, uint256 amount, uint256 time`.

## Recommendation

**yosriady :** Emit the plan owner address in the event.

## Client Response

Acknowledged.We will change the event in the next release.

# GMT-12:Possible DOS (out-of-gas) on loops

| Category | Severity | Code Reference | Status | Contributor |
|---|---|---|---|---|
| DOS | Low | • code/src/contracts/GMEvent/GMEventAbstract.sol#L180 | Acknowledged | 0xzoobi |

## Code

```
180:        for (uint256 i = 0; i < _beneficiaries.length; i++) {
```

## Description

**0xzoobi :** The settlement function where in we transfer USDC to each beneficiary addresses is iterating the for loop. There is possible to get an out of gas issue or one of the transactions may revert resulting in whole transaction being reverted. In some cases, there is a possibility of Denial of Service, if the _beneficiaries.wallet is a contract address and there is a missing fallback function or forced revert.

You can refer to this Ethernaut Challenge for better understanding - https://ethernaut.openzeppelin.com/level/0x725595BA16E76ED1F6cC1e1b65A88365cC494824

Consider below POC contract

```
        // transfer to each beneficiary addresses
        BeneficiaryInfo[] memory _beneficiaries = beneficiaries();
        for (uint256 i = 0; i < _beneficiaries.length; i++) {
            _usdcContract.transfer(
                _beneficiaries[i].wallet,
                _beneficiaries[i].amount
            );
            payments.push(
                PaymentInfo(
                    _beneficiaries[i].amount,
                    block.timestamp,
                    _beneficiaries[i].wallet
                )
            );
        }
```

## Recommendation

**0xzoobi :** Use a `pull-over-push` design pattern i.e., create a state variable to store the mapping of _beneficiaries wallet and amount. Each beneficiary will call the function on their own. This fixes the issue of the contract owner spending too much of gas on the transaction and will also prevent unnecessary reverts.

Consider below fix

```
mapping(address=> uint256) public beneficiariesBalance;

BeneficiaryInfo[] memory _beneficiaries = beneficiaries();

for (uint256 i = 0; i < _beneficiaries.length; i++) {
          beneficiariesBalance.push(
                _beneficiaries[i].wallet,
                _beneficiaries[i].amount
        )
};
```

## Client Response

Acknowledged.We did not fix it this time because we will remove the function in the feature release.The business has been changed.

# GMT-13:Remove unused Hardhat imports

| Category | Severity | Code Reference | Status | Contributor |
| --- | --- | --- | --- | --- |
| Gas Optimization | Informational | • code/src/contracts/Comptroller/Epoch.sol#L21 | Fixed | yosriady |

## Code

```
21:import "hardhat/console.sol";
```

## Description

**yosriady :** Several contracts in the project includes the following unnecessary import:

```
import "hardhat/console.sol";
```

## Recommendation

**yosriady :** Remove the import statement to save on deployment gas costs.

## Client Response

Fixed.We've removed the unused imports.

# GMT-14:State variables that never be used could be deleted or declared constant

| Category | Severity | Code Reference | Status | Contributor |
| --- | --- | --- | --- | --- |
| Gas Optimization | Informational | • code/src/contracts/Asset/SurplusStorage.sol#L13<br>• code/src/contracts/Comptroller/Epoch.sol#L62<br>• code/src/contracts/Comptroller/Epoch.sol#L77 | Fixed | BradMoonUESTC |

## Code

```
13:    uint256 public withdrawAmounts;

62:    bool public needDonate;

77:    uint256 public currentGapDonationIndex;
```

## Description

**BradMoonUESTC :** Variable withdrawAmounts, needDonate, currentGapDonationIndex that never be used could be deleted or declared constant

## Recommendation

**BradMoonUESTC :** Add the constant attributes to state variables that never change.

## Client Response

Fixed

# GMT-15:Unfinished TODOs and missing implementation

| Category | Severity | Code Reference | Status | Contributor |
| --- | --- | --- | --- | --- |
| Code Style | Informational | • code/src/contracts/Asset/SurplusPool.sol#L54-L59<br>• code/src/contracts/Asset/SurplusPool.sol#L57-L60<br>• code/src/contracts/Comptroller/Comptroller.sol#L274-L279 | Acknowledged | BradMoonUESTC, yosriady |

## Code

```
54:     /**
55:      * TODO: complete it.
56:      */
57:     function withdraw(uint amount, bytes memory reason) external returns (uint) {
58:         return amount;
59:     }

57:     function withdraw(uint amount, bytes memory reason) external returns (uint) {
58:         return amount;
59:     }
60:

274:     /**
275:      * TODO: need complete this.
276:      */
277:     function isValidMember(address) public pure returns (bool) {
278:         return true;
279:     }
```

## Description

**BradMoonUESTC :** The function withdraw has unfinished logic
**yosriady :** There are unfinished functions, such as in `SurplusPool`:

```
    /**
     * TODO: complete it.
     */
    function withdraw(uint amount, bytes memory reason) external returns (uint) {
        return amount;
    }
```

## Recommendation

**BradMoonUESTC :** Complete the logic of withdraw
**yosriady :** Find and complete all TODOs in the code before shipping. New code should also be audited.
Remove unused code.

## Client Response

Acknowledged.We did not fix it this time because we will complete the function in the next release.

# GMT-16:Unused return value

| Category | Severity | Code Reference | Status | Contributor |
| --- | --- | --- | --- | --- |
| Code Style | Informational | • code/src/contracts/Comptroller/Comptroller.sol#L62-L66<br>• code/src/contracts/GMEvent/GMEventAbstract.sol#L113<br>• code/src/contracts/Comptroller/Comptroller.sol#L180<br>• code/src/contracts/Comptroller/Epoch.sol#L325 | Acknowledged | BradMoonUESTC |

## Code

```
62:             gateway.pay(
63:                 planId,
64:                 planContract.planValue(),
65:                 address(planContract)
66:             );

113:         IPlan(planContract()).payForGMEvent(address(this));

180:                 fundPool.routinelyDonate(address(plan), amount, periodValue);

325:         token.mint(donationValue.user, amount);
```

## Description

**BradMoonUESTC :** Either the return value of an external call is not stored in a local or state variable, or the return value is declared but never used in the function body.

## Recommendation

**BradMoonUESTC :** Ensure the return value of external function calls is used. Remove or comment out the unused return function parameters.

## Client Response

Acknowledged.We removed all the unused state variables except the `payForGMEvent(address(this))` function,because we will remove this function in the next release

# Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3's prior written consent in each instance.

This report is not an "endorsement" or "disapproval" of any particular project or team. This report is not an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3's position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.