VERILOG X FUZZLAND
JOINT AUDIT

2024

GEMNIFY:
SECURITY REVIEW REPORT

Jun 13, 2024

Contents

1 | Introduction

Figure 1.1: Gemnify Report Cover

This report presents our engineering engagement with the Gemnify team on their decentralized derivative platform tailored for pegged assets.

Project Name	Gemnify Protocol
Repository Link	https://github.com/GMX-For-NFT/gemnify-contract
First Commit Hash	First: 405889f;
Final Commit Hash	Final: d7008d5;
Language	Solidity
Chain	Ethereum

2 | About Verilog Solutions

Founded by a group of cryptography researchers and smart contract engineers in North America, Verilog Solutions elevates the security standards for Web3 ecosystems by being a full-stack Web3 security firm covering smart contract security, consensus security, and operational security for Web3 projects.
Verilog Solutions team works closely with major ecosystems and Web3 projects and applies a quality above quantity approach with a continuous security model. Verilog Solutions onboards the best and most innovative projects and provides the best-in-class advisory services on security needs, including on-chain and off-chain components.

3 | Service Scope

3.1 | Service Stages

Our auditing service includes the following two stages:

- Smart Contract Auditing Service

3.1.1 | Smart Contract Auditing Service

The Verilog Solutions team analyzed the entire project using a detailed-oriented approach to capture the fundamental logic and suggested improvements to the existing code. Details can be found under Findings And Improvement Suggestions.

3.2 | Methodology

- **Code Assessment**
  - We evaluate the overall quality of the code and comments as well as the architecture of the repository.
  - We help the project dev team improve the overall quality of the repository by providing suggestions on refactorization to follow the best practices of Web3 software engineering.
- **Code Logic Analysis**
  - We dive into the data structures and algorithms in the repository and provide suggestions to improve the data structures and algorithms for the lower time and space complexities.
  - We analyze the hierarchy among multiple modules and the relations among the source code files in the repository and provide suggestions to improve the code architecture with better readability, reusability, and extensibility.

3.3 | Audit Scope

Our auditing for Gemnify covered the Solidity smart contracts under the folder 'contracts' in the repository (https://github.com/GMX-For-NFT/gemnify-contract) with commit hash **405889f**.

4 | Project Summary

Gemnify is a decentralized derivative exchange tailored for pegged assets that aims to offer:

- amplified leveraged trading via basis points incrementation.
- capital-efficient support for open interest via taking advantage of supporting a basket of relatively pegged assets and modifications made to the on-chain perpetual model tailored for permissionless capital-efficiency price speculation and hedging..

Below is the protocol architecture:

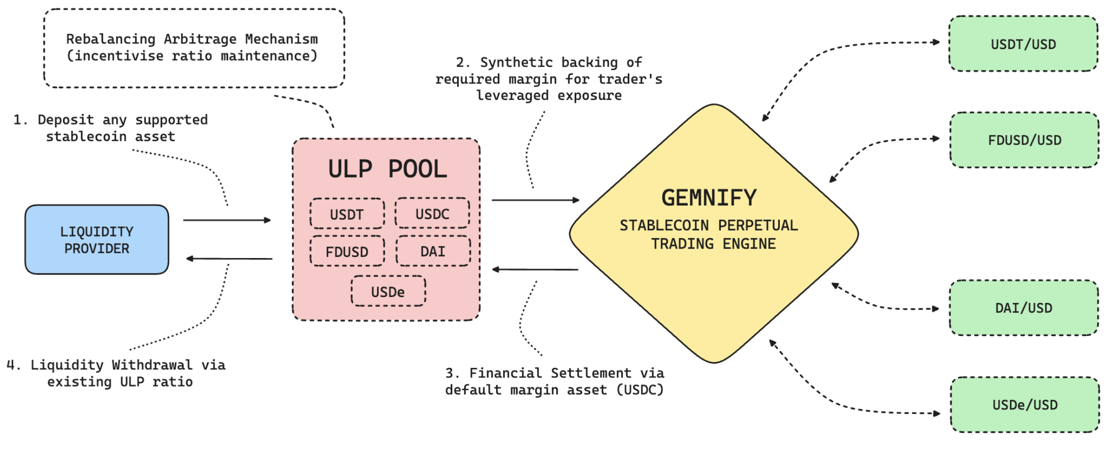
Rebalancing Arbitrage Mechanism (incentivise ratio maintenance)
1. Deposit any supported stablecoin asset
LIQUIDITY PROVIDER
ULP POOL
USDT
USDC
FDUSD
DAI
USDe
4. Liquidity Withdrawal via existing ULP ratio
2. Synthetic backing of required margin for trader's leveraged exposure
3. Financial Settlement via default margin asset (USDC)
GEMNIFY
STABLECOIN PERPETUAL TRADING ENGINE
USDT/USD
FDUSD/USD
DAI/USD
USDe/USD

Figure 4.1: Gemnify Architecture

Gemnify is a collection of smart contracts that consists of five folders. Each folder handles specific areas of the protocol, the folders and their respective contracts are described below:

- Oracle

  Gemnify leverages accurate stablecoin price feeds from a mix of centralized and decentralized exchanges to provide reliable references and minimize the risk of price manipulation.

  For the decentralized sources, chainlink is considered.

  For the centralized sources, the FastPriceFeed contract allows setting up the price according to the source.

- Tokens

  ULP is the liquidity provider token for V1. Liquidity providers earn fees from leverage trading, borrowing fees, and swaps.

  YieldToken is a token used for internal accounting.

  TimeDistributor allows the distribution of tokens.

- Staking

  The RewardTracker contract contains the token that users will get when they stake their LP tokens.

  The RewardRouter contract allows users to seamlessly provide liquidity and stake their LP tokens in Gemnify. Also allows to unstake and redeem LP tokens.

  The RewardDistributor contract allows to distribute the staked tokens.

- Peripherals

  The Reader contract allows reading information associated with the fees. This includes the deposit, withdrawal, swap, and borrowing fees.

- Core

  The ULPManager contract contains the logic to provide and remove liquidity.

  The Router contract contains the logic to perform swaps.

  The OrderBook contract contains the logic to create orders.

  The Vault contract will keep track of the positions and the logic associated with them.

5 | Findings and Improvement Suggestions

Severity	Total	Acknowledged	Resolved
High	3	3	3
Medium	5	5	5
Low	6	6	6
Informational	4	4	3

5.1 | High

5.1.1 | Incorrect addition operation in `_mint()` function

Severity	High
Source	contracts/tokens/BaseToken.sol#L193;
Commit	06a3f51;
Status	Resolved in commit 5f999e0;

- **Description**

  In the `_mint` function, there is an incorrect addition operation in the conditional block for non-staking accounts.

  `nonStakingSupply += nonStakingSupply;` should be changed to `nonStakingSupply += _amount;` as it is intended to increase the non-staking supply by the amount issued, not by its previous value.

```
function _mint(address _account, uint256 _amount) internal {
    require(_account != address(0), "BaseToken: mint to the zero address");

    _updateRewards(_account);

    totalSupply += _amount;
    balances[_account] += _amount;

    if (nonStakingAccounts[_account]) {
        nonStakingSupply += nonStakingSupply;//@audit
    }

    emit Transfer(address(0), _account, _amount);
}
```

- **Exploit Scenario**

  - The `_mint()` gets called during the minting of tokens;
  - The accounting of the token is wrong since the wrong variable was used to update;
  - The information associated with the total supply is wrong and therefore the token distribution doesn't work as expected.

- **Recommendations**

  Correct the addition operation in `nonStakingSupply += _amount;`;.

- **Results**

  Resolved in commit 5f999e0.

  The suggestion was implemented.

5.1.2 | `<address>.call()` should be used instead of `<address>.transfer()` to transfer native tokens.

Severity	**High**
Source	contracts/tokens/Token.sol#L79;
Commit	06a3f51;
Status	Resolved in commit 5f999e0;

- **Description**
  The current smart contract uses the `transfer()` function to transfer the native tokens:

  ```
  payable(msg.sender).transfer(amount);
  ```

  However, using this method has some limitations such as the gas limit of 2,300.

- **Exploit Scenario**

  - Alice calls the `withdraw()` function to redeem tokens;
  - The transaction reverts because of the gas limit of the `transfer()` function;
  - Now the distribution gets affected because the tokens weren't able to be redeemed.

- **Recommendations**
  Use the `call` method to transfer ETH and check the return value.

  ```
  (bool success, ) = msg.sender.call{value: amount}('');
  require(success, 'send value failed');
  ```

- **Results**
  Resolved in commit 5f999e0.

  The suggestion was implemented.

5.1.3 | Issue with the `claimFundingFees()` function

Severity	**High**
Source	contracts/core/Vault.sol#L681;
Commit	62a81d7;
Status	Resolved in commit 024b6b7;

- **Description**
  As the code stands, the `claimFundingFees()` function is an external function which inputs are the account holder and the receiver:

```
function claimFundingFees(
    address _account,
    address _receiver
) external returns (uint256) {
    return FundingFeeLogic.claimFundingFees(_account, _receiver);
}
```

  This means that any user can claim funding fees from any address and set the receiver as themselves to get those rewards.

- **Exploit Scenario**

  - Mallory calls the `claimFundingFees()` function with input parameters account of Bob and receiver as her address;
  - Mallory claims the fees from Bob;
  - When Bob wants to claim fees, he is not able to do so.

- **Recommendations**
  Refactor the function so that users are not able to claim fees from other users.

- **Results**
  Resolved in commit 024b6b7.
  The function was refactored to the following:

```
function claimFundingFees() external returns (uint256) {
    DataTypes.FeeStorage storage fs = StorageSlot.getVaultFeeStorage();
    DataTypes.AddressStorage storage addrs = StorageSlot
        .getVaultAddressStorage();

    uint256 claimableFundingAmount = fs.claimableFundingAmount[msg.sender];

    if (claimableFundingAmount > 0) {
        IERC20Upgradeable(addrs.collateralToken).safeTransfer(
            msg.sender,
            claimableFundingAmount
        );

        fs.claimableFundingAmount[msg.sender] = 0;

        emit ClaimFundingFee(
            msg.sender,
            addrs.collateralToken,
            claimableFundingAmount
        );
    }
    return claimableFundingAmount;
}
```

5.2 | Medium

5.2.1 | Use of deprecated Chainlink function `latestAnswer`

Severity	**Medium**
Source	contracts/core/VaultPriceFeed.sol#L261;
Commit	06a3f51;
Status	Resolved in commit 5f999e0;

- **Description**
  According to Chainlink's documentation, the `latestAnswer()` function is deprecated. This function does not error if no answer has been reached but returns 0:

```
function getLatestPrimaryPrice(
   address_token
) public view override returns (uint256) {
   address priceFeedAddress = priceFeeds[_token];
   require(
       priceFeedAddress != address(0),
       "VaultPriceFeed: invalid price feed"
   );

   IPriceFeed priceFeed = IPriceFeed(priceFeedAddress);

   int256 price = priceFeed.latestAnswer();
   require(price > 0, "VaultPriceFeed: invalid price");

   return uint256(price);
}

function getAssetPriceFromChainlink(address asset) public view returns (uint256) {
uint256 price;

AggregatorV2V3Interface sourceAgg = assetChainlinkAggregators[asset];
require(address(sourceAgg) != address(0), Errors.ASSET_AGGREGATOR_NOT_EXIST);

int256 priceFromAgg = sourceAgg.latestAnswer();//@audit Use of deprecated Chainlink
price = uint256(priceFromAgg);

require(price > 0, Errors.ASSET_PRICE_IS_ZERO);
return price;
}
```

- **Exploit Scenario**

  - The `getAssetPrice()` function gets called to get the price of a token;
  - The answer is not reached;
  - The return value is zero since the `latestAnswer()` function doesn't perform a check on the stale price.

- **Recommendations**
  Use the `latestRoundData()` function to get the price instead. Add checks on the return data with proper revert messages if the price is stale or the round is incomplete, for example.

- **Results**
  Resolved in commit 5f999e0.

  The `latestRoundData()` function is used and the time is checked.

5.2.2 | Fixed `ethTransferGasLimit` that could cause transactions to fail

Severity	**Medium**
Source	contracts/core/BasePositionManager.sol#L308;
Commit	06a3f51;
Status	Resolved in commit 5f999e0;

- **Description**
  Gas limits should be adjusted for different situations and network environments. A fixed limit may cause transactions to fail or execute inefficiently under certain circumstances. For example, during times of network congestion, gas limits may need to be increased to ensure transactions are processed in a timely manner. If the gas limit is set too low, it may cause the transaction to fail because in some cases the transaction needs to consume more gas to successfully execute.

```
function _transferOutETHWithGasLimitFallbackToWeth(
     uint256 _amountOut,
     address payable_receiver
 ) internal {
     IWETH _weth = IWETH(weth);
     _weth.withdraw(_amountOut);

     (bool success /* bytes memory data */, ) = _receiver.call{
         value: _amountOut,
         gas: ethTransferGasLimit//@audit
     }("");

     if (success) {
         return;
     }

     // if the transfer failed, re-wrap the token and send it to the receiver
     _weth.deposit{value: _amountOut}();
     _weth.transferFrom(address(this), address(_receiver), _amountOut);
 }
```

- **Exploit Scenario**
  - The network is congested and the `_transferOutETHWithGasLimitFallbackToWeth()` function gets called;
  - The fixed gas limit prevents the transaction from going through;
  - The transfer failed.

- **Recommendations**
  Dynamically determine the amount of gas required.

- **Results**
  Resolved in commit 5f999e0.

  The fixed gas limit was removed.

5.2.3 | Unchecked transfers

Severity	**Medium**
Source	contracts/tokens/Token.sol#L69; contracts/core/BasePositionManager.sol#L326;
Commit	06a3f51;
Status	Resolved in commit 5f999e0;

- **Description**
  The `withdrawToken()` function makes a call to the `transfer()` function which is known not to check the return value of the transfer. This means that a transfer might fail while the transaction is being successful.

- **Exploit Scenario**

  - The `withdrawToken()` function gets called by a user;
  - The transaction goes through but the user does not receive any tokens;
  - The user gets confused as to what is going on with the protocol.

- **Recommendations**
  Use the `SafeTransfer` library or add a `require()` statement to check that the transfers actually go through.

- **Results**
  Resolved in commit 5f999e0.

  The `SafeTransfer` library is now being used.

5.2.4 | `setShortsTrackerAveragePriceWeight` verification parameter error

Severity	Medium
Source	contracts/core/UlpManager.sol#L90;
Commit	06a3f51;
Status	Resolved in commit 5f999e0;

Description

```
function setShortsTrackerAveragePriceWeight(
     uint256 _shortsTrackerAveragePriceWeight
 ) external override onlyOwner {
     require(
         shortsTrackerAveragePriceWeight <= Constants.PERCENTAGE_FACTOR,
         "UlpManager: invalid weight"
     );
     shortsTrackerAveragePriceWeight = _shortsTrackerAveragePriceWeight;
 }
```

The purpose of this function is to set a new value for "shortsTrackerAveragePriceWeight". The `require()` statement is intended to ensure that the value being set does not exceed a predefined maximum value (assumed to be 100 %, given the variable name PERCENTAGE_FACTOR).

But the function incorrectly checks with the current state variable "shortsTrackerAveragePriceWeight" instead of the parameter "_shortsTrackerAveragePriceWeight". This means it checks the existing value against the maximum value allowed, rather than the new value to be set. Therefore, it does not effectively prevent "shortsTrackerAveragePriceWeight" from being set to a higher value than "Constants.PERCENTAGE_FACTOR".

Exploit Scenario

- The `setShortsTrackerAveragePriceWeight` function gets called with the wrong weights;
- Since the checks are performed incorrectly, the weights get updated;
- The transactions associated with the average price weight will be incorrect.

Recommendations

Change the following piece of code

```
require(
     _shortsTrackerAveragePriceWeight <= Constants.PERCENTAGE_FACTOR,
     "UlpManager: invalid weight"
);
```

Results

Resolved in commit 5f999e0.

The suggestion was implemented.

5.2.5 | Precision Loss

Severity	Medium
Source	contracts/core/UlpManager.sol#L199;
Commit	62a81d7;
Status	Resolved in commit 024b6b7;

- **Description**
  The following piece of code performs a division before a multiplication. This means that there will be some decimals truncated when performing the division first.

```
uint256 tokenAum = (tokenInUsdgAmount *
                Constants.PRICE_PRECISION) / 10 ** Constants.USDG_DECIMALS;
            uint256 amountOut = (tokenAum * 10 ** tokenInfo.tokenDecimal) /
                price;
```

  Therefore some precision is lost when performing the operation.

- **Exploit Scenario**
  - The `getTokenInUsdg()` function gets called to calculated the equivalent USDG tokens;
  - The operation performs a division before a multiplication and therefore incurs a loss of precision.

- **Recommendations**
  Arrange the operation to the following:

```
uint256 amountOut = (tokenInUsdgAmount *
               Constants.PRICE_PRECISION *
               10 ** tokenInfo.tokenDecimal) /
               (10 ** Constants.USDG_DECIMALS * price);
```

- **Results**
  Resolved in commit 024b6b7.

  The suggestion was implemented.

5.3 | Low

5.3.1 | Inconsistent inequality

Severity	**Low**
Source	contracts/core/OrderBook.sol#L889;
Commit	62a81d7;
Status	Resolved in commit 024b6b7;

- **Description**
  The following check is performed in the `createDecreaseOrder()` of the `OrderBook` contract

```
require(
            msg.value > minExecutionFee,
            "OrderBook: insufficient execution fee"
        );
```

  If we look at other functions, the inequality looks like the following:

```
require(
            _executionFee >= minExecutionFee,
            "OrderBook: insufficient execution fee"
        );
```

- **Exploit Scenario**
  N/A.

- **Recommendations**
  Fix the inequality.

- **Results**
  Resolved in commit 024b6b7.

  The inequality was fixed.

5.3.2 | The `executeDecreaseOrder()` function not using the `onlyPositionManager` modifier

Severity	**Low**
Source	contracts/core/OrderBook.sol#L669;
Commit	62a81d7;
Status	Resolved in commit 024b6b7;

- **Description**
  The `executeIncreaseOrder()` function uses the `onlyPositionManager` modifier which allows the `PositionManager` contract to be the only caller of this function. The same does not apply to the `executeDecreaseOrder()` function.

- **Exploit Scenario**
  N/A.

- **Recommendations**
  Add the `onlyPositionManager` modifier to the `executeDecreaseOrder()` function.

- **Results**
  Resolved in commit 024b6b7.

  The modifier was added.

5.3.3 | Recommendation of using `Ownable2Step` and `Ownable2StepUpgradable`

Severity	**Low**
Source	Global;
Commit	06a3f51;
Status	Resolved in commit 5f999e0;

- **Description**
  Just like the `TimeDistributor` contract, we recommend considering the `Ownable2Step` and `Ownable2StepUpgradable` contracts in other contracts like `BaseToken`, `RewardTracker`, etc.
- **Exploit Scenario**
  N/A.
- **Recommendations**
  Consider the he `Ownable2Step` and `Ownable2StepUpgradable` contracts.
- **Results**
  Resolved in commit 5f999e0.

  The suggestion was implemented.

5.3.4 | Lack of a two-step process for critical operations

Severity	**Low**
Source	contracts/tokens/TimeDistributor.sol#L45;
Commit	06a3f51;
Status	Resolved in commit 5f999e0;

- **Description**
  The `gov` address is the only address allowed to set the distribution. This address can be changed directly when calling the `setGov()` function.

```
function setGov(address _gov) external onlyGov {
    gov = _gov;
}
```

  Since this address is critical, it should consider a two-step process when changing it.

- **Exploit Scenario**
  N/A.

- **Recommendations**
  Consider a two step-process for changing the `gov` address:

- **Results**
  Resolved in commit 5f999e0.

  The suggestion was implemented.

5.3.5 | Lack of zero address checks

Severity	**Low**
Source	contracts/core/OrderBook.sol#179;
Commit	06a3f51;
Status	Resolved in commit 5f999e0;

- **Description**
  The `initialize()` function of the `OrderBook` contract and other contracts lack zero address checks when setting up the addresses. This is a critical check to ensure correct initialization.

- **Exploit Scenario**
  N/A.

- **Recommendations**
  Add zero address checks to the `initialize()` functions.

- **Results**
  Resolved in commit 5f999e0.

  The checks were added.

5.3.6 | Lack of event emissions for critical operations

Severity	**Low**
Source	contracts/core/Router.sol#L48-96;
Commit	06a3f51;
Status	Resolved in commit 5f999e0;

- **Description**
  Events are vital aids in monitoring contracts and detecting suspicious behavior. The functions specified in the source section perform state changes and thus they should consider the emission of events.

- **Exploit Scenario**
  N/A.

- **Recommendations**
  Add events to the functions that perform state changes.

- **Results**
  Resolved in commit 5f999e0.

  The events were added.

5.4 | Informational

5.4.1 | Empty contract

Severity	**Informational**
Source	contracts/core/libraries/helpers/Events.sol;
Commit	06a3f51;
Status	Resolved in commit 5f999e0;

- **Description**
  The `Events` file is an empty library that is not used anywhere.
- **Exploit Scenario**
  N/A.
- **Recommendations**
  Remove the empty contract.
- **Results**
  Resolved in commit 5f999e0.

  The contract was removed

5.4.2 | Unused import

Severity	**Informational**
Source	contracts/core/Vault.sol#L19;
Commit	62a81d7;
Status	Resolved in commit 024b6b7;

- **Description**
  The `IWETH` interface is imported into the `Vault` contract but the interface is no longer being used
- **Exploit Scenario**
  N/A.
- **Recommendations**
  Remove unused imports.
- **Results**
  Resolved in commit 024b6b7.

  The import was removed.

5.4.3 | Floating solidity pragma version

Severity	**Informational**
Source	Global;
Commit	62a81d7;
Status	Acknowledged;

- **Description**
  Current smart contracts use ^0.8.19. And compilers within versions $\geq$ 0.8.19 and <0.9.0 can be used to compile those contracts. Therefore, the contract may be deployed with a newer or latest compiler version which generally has higher risks of undiscovered bugs.

  It is a good practice to fix the solidity pragma version if the contract is not designed as a package or library that will be used by other projects or developers.

- **Exploit Scenario**
  N/A.

- **Recommendations**
  Use the fixed solidity pragma version.

- **Results**
  Acknowledged.

5.4.4 | Unused `pendingReceivers` mapping

Severity	**Informational**
Source	contracts/core/RewardRouter.sol#L69;
Commit	62a81d7;
Status	Resolved in commit 024b6b7;

- **Description**
  The `pendingReceivers` mapping in the `RewardRouter` contract is not used. The rewards are directly distributed as the code stands.

- **Exploit Scenario**
  N/A.

- **Recommendations**
  Removed the unused mapping.

- **Results**
  Resolved in commit 024b6b7.

  The mapping was deleted.

6 | Use Case Scenarios

Gemnify allows users to perform the following actions with whitelisted stablecoins:

- Provide liquidity to earn fees and allows them to remove it anytime.
- Stake the LP tokens to earn additional rewards
- Create and execute orders following an order book model.
- Perform spot and amplified leverage trading.

Just like GMX's GLP (LP token), ULP supports a basket of selected USD-pegged stablecoins with different risk profiles depending on the nature of the stablecoin. The risk profile of each stablecoin is determined via (1) pegged mechanism, (2) volatility risk, and (3) liquidity depth.

7 | Access Control Analysis

There are mainly two privileged roles in the Gemnify protocol. A description of each of the actions for the roles can be found below:

7.1 | owner

- Set important addresses for the functionality of the protocol, for example, the address of the weth, collateral token, vault, etc.
- Set important parameters such as the buffer amount, borrowing rate, token configuration, fees, etc.
- Set the information of the yield token.
- Add plugins to allow trading from accounts different than the actual sender of the transaction.
- Set the admin role.
- Set the price feed.
- Add handlers to manage staking from different accounts.

7.2 | admin

- Withdraw fees.
- Set important addresses such as the order keeper, liquidator, position keeper, etc.
- Set important parameters such as the gas limit, min execution fee, delay values, etc.

8 | Appendix

8.1 | Appendix I: Severity Categories

Severity	Description
High	Issues that are highly exploitable security vulnerabilities. It may cause direct loss of funds / permanent freezing of funds. All high severity issues should be resolved.
Medium	Issues that are only exploitable under some conditions or with some privileged access to the system. Users' yields/rewards/information is at risk. All medium severity issues should be resolved unless there is a clear reason not to.
Low	Issues that are low risk. Not fixing those issues will not result in the failure of the system. A fix on low severity issues is recommended but subject to the clients' decisions.
Information	Issues that pose no risk to the system and are related to the security best practices. Not fixing those issues will not result in the failure of the system. A fix on informational issues or adoption of those security best practices-related suggestions is recommended but subject to clients' decision.

8.2 | Appendix II: Status Categories

Severity	Description
Unresolved	The issue is not acknowledged and not resolved.
Partially Resolved	The issue has been partially resolved
Acknowledged	The Finding / Suggestion is acknowledged but not fixed / not implemented.
Resolved	The issue has been sufficiently resolved

9 | Disclaimer