CoinFabrik

YieldYak Audit

November 2022

By CoinFabrik

v202211

Executive Summary

CoinFabrik was asked to audit some contracts for the YieldYak project. The audited files are from the git repository located at https://github.com/yieldyak/smart-contracts.git. The audit is based on the commit a818f34b553763c0c25d63fbb9c2c2ada7280def.

Fixes were checked on commit `93645e8732daf92472ac19c867c9cce55806b1bc`.

The YieldYak project provides a series of investment strategies on the Avalanche blockchain that can be used by any investor and pool resources for its execution.

This audit is focussed on analyzing the `GmxStrategyForGLP` contract and its interactions with the `GmxProxy` contract and the `GmxDepositor` contract, and the interactions of those contracts with the GMX contracts.

During this audit we found one critical issue, no medium issues and several minor issues. The critical issue was resolved. Of the minor issues, four were resolved and eight were acknowledged.

Scope

The scope for this audit includes and is limited to the following files:

- `contracts/strategies/GmxDepositor.sol`: Holder of the funds handled by the `GmxProxy` contract. This file was moved to `contracts/strategies/avalanche/gmx/GmxDepositor.sol` for the review commit.
- `contracts/strategies/GmxProxy.sol`: Contract that interacts with the GMX tokens on behalf of the strategies. This file was moved to `contracts/strategies/avalanche/gmx/GmxProxy.sol` for the review commit.
- `contracts/strategies/GmxStrategyForGLP.sol`: Investment strategy for Staked GLP tokens. This file was moved to `contracts/strategies/avalanche/gmx/GmxStrategyForGLP.sol` for the review commit.
- `contracts/YakERC20.sol`: YRT token implementation.
- `contracts/YakStrategyV2.sol`: Base contract for yield-farming strategies.
- `contracts/interfaces/IERC20.sol`: Interface definition for ERC20 token contracts.
- `contracts/interfaces/IGmxDepositor.sol`: Interface definition of the `GmxDepositor` contract. This file was moved to `contracts/strategies/avalanche/gmx/interfaces/IGmxDepositor.sol` for the review commit.

- `contracts/interfaces/IGmxProxy.sol`: Interface definition of the `GmxProxy` contract. This file was moved to `contracts/strategies/avalanche/gmx/interfaces/IGmxProxy.sol` for the review commit.
- `contracts/interfaces/IGmxRewardRouter.sol`: Interface definition of the `RewardRouterV2` contract[1]. This file was moved to `contracts/strategies/avalanche/gmx/interfaces/IGmxRewardRouter.sol` for the review commit.
- `contracts/interfaces/IGmxRewardTracker.sol`: Interface definition of the `RewardTracker` contract[2]. This file was moved to `contracts/strategies/avalanche/gmx/interfaces/IGmxRewardTracker.sol` for the review commit.
- `contracts/interfaces/IWAVAX.sol`: Interface definition of the WAVAX token contract.
- `contracts/interfaces/IYakStrategy.sol`: Interface definition for the YieldYak investment strategies.
- `contracts/lib/Permissioned.sol`: Abstract contract that contains the logic to handle depositors. Used by `YakStrategyV2`.
- `contracts/lib/SafeMath.sol`: Library used to avoid silent underflows and overflows on integer operations..

The following files were taken from the OpenZeppelin git repository (https://github.com/OpenZeppelin/openzeppelin-contracts), tag `v3.4.2-solc-0.7`, commit `04695aecbd4d17dddfd55de766d10e3805d6f42f`. The only changes made were changing the `pragma solidity` statement, adjusting import paths and changing whitespace to reformat the source code.

- `contracts/lib/Address.sol`: Located in `/contracts/utils/Address.sol`.
- `contracts/lib/Context.sol`: Located in `/contracts/utils/Context.sol`.
- `contracts/lib/Ownable.sol`: `Located in /contracts/access/Ownable.sol`.
- `contracts/lib/SafeERC20.sol`: Located in `/contracts/token/ERC20/SafeERC20.sol`.

No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.

Methodology

CoinFabrik was provided with the project source code. Our auditors spent two weeks auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function,

[1] See https://github.com/gmx-io/gmx-contracts/blob/master/contracts/staking/RewardRouterV2.sol.
[2] See https://github.com/gmx-io/gmx-contracts/blob/master/contracts/staking/RewardTracker.sol.

understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. Without being limited to them, The audit process included the following analyses:

- Arithmetic errors
- Outdated version of Solidity compiler
- Race conditions
- Reentrancy attacks
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

After delivering a report with our findings, the development team had the opportunity to comment on every finding and fix the issues they considered convenient. Once fixed and/or commented, our team ran a second review process to verify that the changes to the code effectively solve the issues found and do not unintentionally add new ones. This report includes the final status after the second review.

Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below.

ID	Title	Severity	Status
CR-01	`GmxStrategyForGLP` Missing Gains	Critical	Resolved
MI-01	Owner May Block Reinvest Process	Minor	Resolved
MI-02	Only EOA Check Bypass	Minor	Acknowledged
MI-03	Possible Overflow in `YakStrategyV2` While Mapping Deposit Tokens to Shares	Minor	Acknowledged
MI-04	Solidity Compiler Version	Minor	Resolved
MI-05	Missing Zero-Checks in `GmxProxy`	Minor	Resolved

ID	Title	Severity	Status
MI-06	Possible Reentrancies in `GmxStrategyForGLP`	Minor	Acknowledged
MI-07	Guard Against Deposit to Shares Ratio Change	Minor	Acknowledged
MI-08	Excessive Rights for Strategies in `GmxProxy`	Minor	Acknowledged
MI-09	Deposit Token Must be `StakedGlp`	Minor	Acknowledged
MI-10	Ignored Minimum Rescued Funds	Minor	Acknowledged
MI-11	`depositWithPermit()` Does Not Work	Minor	Acknowledged
MI-12	Missing Safe Transfer	Minor	Resolved

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. They must be fixed **immediately**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of, but might be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit has one of the following statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code, but is a result of an intentional decision.
- **Resolved**: Adjusted program implementation to eliminate the risk.

- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code, but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk.

Critical Severity Issues

CR-01 `GmxStrategyForGLP` Missing Gains

Location:

- `contracts/strategies/GmxStrategyForGLP.sol: 132`
- `contracts/strategies/GmxProxy.sol: 139`

When the `StakedGlp` contract makes a transfer[3] it interacts with the staked GLP tracker and the fee GLP tracker but the `GmxStrategyForGLP` contract, via the `GmxProxy` contract only claims rewards awarded by the fee GLP tracker. A similar situation occurs when the reward router mints and stakes GLP[4]. The unclaimed rewards can be claimed by the `GmxStrategyForGMX` contract, which is outside of the scope of this audit.

Recommendation

Isolate the `GmxStrategyForGMX` and `GmxStrategyForGLP` strategies by using different depositors and claim rewards on both trackers in the reinvestment procedure.

Status

Resolved.

It now uses the reward router's `handleRewards()` function to claim the rewards. It must also be noticed that the GmxStrategyForGMX will not be able to obtain any rewards at all given the line 146 in the `contracts/strategies/avalanche/gmx/GmxProxy.sol` file in the review commit.

```
if (rewardTracker == feeGmxTracker) return;
```

It must be noted that while this audit was made, the development team began to solve this issue, prior to it being reported by us. They informed us that they intend to only use the `GmxStrategyForGLP` and claim rewards for both trackers. The merge request with the fixes is being developed[5].

[3] See https://snowtrace.io/address/0x5643f4b25e36478ee1e90418d5343cb6591bcb9d#code#F1#L73.
[4] See https://snowtrace.io/address/0x82147c5a7e850ea4e28155df107f2590fd4ba327#code#F1#L133.
[5] See https://github.com/yieldyak/smart-contracts/pull/149.

Medium Severity Issues

No issues found.

Minor Severity Issues

MI-01 Owner May Block Reinvest Process

Location:

- `contracts/YakStrategyV2.sol: 197,207,217`

If the sum of `DEV_FEE_BIPS`, `ADMIN_FEE_BIPS` and `REINVEST_REWARD_BIPS` is bigger than `BIPS_DIVISOR` then the reinvest process will revert when it runs out of funds while transferring fees. This is triggered in the `GmxStrategyForGLP._reinvest()` function. This will also apply to other strategies that inherit from `YakStrategyV2`.

Recommendation

Extract a function to check that the fee settings are consistent and use that function instead of the 3 `require` statements.

Status

Resolved. `ADMIN_FEE_BIPS` is forced to be 0 in line 216 of `contracts/YakStrategyV2.sol`.

MI-02 Only EOA Check Bypass

Location:

- `contracts/strategies/GmxStrategyForGLP.sol: 121`

The `GmxStrategyForGLP.reinvest()` method is guarded by the `onlyEOA` modifier, stopping a contract from reinvesting. But if the unclaimed rewards exceed `MAX_TOKENS_TO_DEPOSIT_WITHOUT_REINVEST` a contract may call `deposit()` and `withdraw()` in a single transaction and do a reinvestment while keeping all the deposit tokens involved.

Recommendation

Decouple reinvesting from depositing tokens, do not check if the actor doing the reinvest is an EOA for consistency or both.

Status

Acknowledged. The development team informed us that the intent of the EOA check is to protect against flash-loan attacks. This is effective if `MIN_TOKENS_TO_REINVEST` is significantly less than `MAX_TOKENS_TO_DEPOSIT_WITHOUT_REINVEST`, always giving the EOAs the opportunity to run the reinvest procedure and reap the rewards. Little clarification regarding the onlyEOA finding. While it's correct what you found, the solution as is, serves its purpose. It's a protection against flash loan attacks stealing rewards. Since the reinvestment during deposit occurs before shares are minted we don't need it there and we wouldn't like to lock out gnosis users or protocols building on top of us.

MI-03 Possible Overflow in `YakStrategyV2` While Mapping Deposit Tokens to Shares

Location:

- `contracts/YakStrategyV2.sol: 156,159,168,171`

An integer overflow may occur while running `YakStrategyV2.getSharesForDepositTokens()` (line 156) and `YakStrategyV2.getDepositTokensForShares()` (line 168) if `totalSupply * totalDeposits() >=` 2^{256}. The offending and repeating code is

```
if (totalSupply.mul(totalDeposits()) == 0) {
```

After each check, multiplication followed by a division is made on each of the functions (lines 159 and 171). And the multiplication may overflow the 256 bits of the Solidity `uint` type, even if the final result after the division is less than 2^{256}.

Given that `YakStrategyV2.getSharesForDepositTokens()` will be used in the deposit process and `YakStrategyV2.getDepositTokensForShares()` wil be used in the withdrawal process this issue may effectively block them both. And given that the overflow is generated using only persistent state, the block may be permanent.

Recommendation

Check individually for zero `totalSupply` and `totalDeposits()` in lines 156 and 168 like this:

```
if (totalSupply == 0 || totalDeposits() == 0) {
```

And do the multiplication and division in lines 159 and 171 taking into account that the intermediate multiplication may overflow the 256 bytes of the solidity's `uint`.

Status

Acknowledged.

MI-04 Solidity Compiler Version

All audited files use the `pragma solidity 0.7.3;` statement. This implies that an old solidity version is being used and also adds risks because bugs may be introduced by using a different solidity compiler.

Recommendation

It is better to lock to a specific compiler version (for example, `pragma solidity 0.8.17;`) and keep it up to date. Also, when updating to 0.8 take into account the new semantics for safe math operations.

Status

Resolved. All files use `pragma solidity 0.8.13`.

MI-05 Missing Zero-Checks in GmxProxy

Location:

- `contracts/strategies/GmxProxy.sol: 69-71,80`

If an address is mistakenly set as zero (default value for solidity variables) you may lock-up the functionality of the contract.

Recommendation

Add `require` statements for all the addresses passed as parameters to the `constructor` and the `updateDevAddr()` function checking that each address is not zero.

Status

Resolved. The `require` guards were added.

MI-06 Possible Reentrancies in `GmxStrategyForGLP`

Location:

- `contracts/strategies/GmxStrategyForGLP.sol: 67,79,90,108,174`

Reentrancy attacks are difficult to predict in contracts with complex logic, especially when the invocation chains are long.

Recommendation

While all the contracts invoked directly or indirectly when running operations in the `GmxStrategyForGLP` strategy seem to be safe, given the complexity of the interactions we recommend to use the `nonReentrant` guard defined in the `contracts/lib/ReentrancyGuard.sol` file for the following functions of the `GmxStrategyForGLP` contract:

- `deposit()`
- `depositWithPermit()`
- `depositFor()`
- `withdraw()`
- `rescueDeployedFunds()`

Status
Acknowledged.

MI-07 Guard Against Deposit to Shares Ratio Change

Location:

- `contracts/strategies/GmxStrategyForGLP.sol: 67,79,90,108`
- `contracts/YakStrategyV2.sol: 80,91,99,105`

When a user does a deposit or a withdrawal, the deposit token is converted to or from shares handled by the strategy. This leaves the user vulnerable to price fluctuations after the transaction is put in the mempool, but before it is added to a block to be processed.

This may happen in, at least, two situations:

1. on emergency withdrawal of the deposit tokens.
2. if the reinvestment procedure, or some other external occurrence, makes the `totalDeposits()` to go down without a call to the `withdraw()` function.

The development team informed us that the second condition cannot happen in the `GmxStrategyForGLP` contract.

Recommendation
Add a parameter where the caller can state the minimum number of shares to be obtained in the `deposit()`, `depositWithPermit()` and `depositFor()` functions. Also add a parameter to the `withdraw()` function with the minimum amount of deposit token to be obtained. This should apply to all the strategies.

If you need to keep retrocompatibility, it is possible to add extra functions with the additional parameter. The current functions should be functionally equivalent to passing 0 as the limit.

Status
Acknowledged.

MI-08 Excessive Rights for Strategies in GmxProxy

Location:

- `contracts/strategies/GmxProxy.sol: 138`

All the strategies can invoke the `claimReward()` function in the `GmxProxy` contract with any `rewardTracker`, effectively making it possible for a rogue strategy to steal the rewards of the other strategy.

It must be noted that this issue is possible because of the aliasing generated by sharing a `GmxProxy` instance between several strategies.

Recommendation
Keep track of the `rewardTracker` of each strategy, and only allow the strategy to claim the rewards using its assigned `rewardTracker`.

Status
Acknowledged.

MI-09 Deposit Token Must be `StakedGlp`

Location:
- `contracts/strategies/GmxStrategyForGLP.sol: 37`

While in the `GmxStrategyForGLP` contract the address of the reward is hardcoded to the WAVAX token, the deposit token is expected to be the `StackedGLP` contract but any address is accepted by the constructor as the deposit token. Using another address will lead to undefined behavior.

Recommendation
Also hardcode the address of the deposit token, as is with the reward token already.

Status
Acknowledged.

MI-10 Ignored Minimum Rescued Funds

Location:
- `contracts/strategies/GmxStrategyForGLP.sol: 37`

The `GmxStrategyForGLP.rescueDeployedFunds()` function does not respect the `minReturnAmountAccepted` parameter, documented in `contracts/YakStrategyV2.sol`, line 138. This may lead to rescuing less funds than expected by the caller.

Recommendation
Revert if the rescued funds do not meet the minimum required. If a caller intends to recover any funds available, they will call the function passing 0 as the first parameter.

Status
Acknowledged.

MI-11 `depositWithPermit()` Does Not Work

Location:
- `contracts/strategies/GmxStrategyForGLP.sol: 86`

The `GmxStrategyForGLP.depositWithPermit()` function does not work, because the `StakedGLP` contract does not have the `permit()` function.

Recommendation

Remove the `GmxStrategyForGLP.depositWithPermit()` function. If not possible, raise a descriptive error instead of failing to execute the `permit()` function.

Status

Acknowledged.

MI-12 Missing Safe Transfer

Location:

- `contracts/YakStrategyV2.sol: 248`

The transfer invocation in line 248 of `contracts/YakStrategyV2.sol` will not work properly if the `transfer()` function does not return any value, which may happen with some old token implementations.

Recommendation

Use the `safeTransfer()` implementation in `contracts/lib/SafeERC20.sol`.

Status

Resolved.

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest to other stakeholders of the project, including users of the audited contracts, token holders or project investors.

Centralization

The dev and the owner of each of the analyzed contracts have privileged capabilities.

The development team informed us that they are using a community multisig to help mitigate centralization risks.

Upgrades

There are no provided methods to upgrade the analyzed contracts but:

- All the belongings of the GmxStrategyForGLP contract can be recovered using the `rescueDeployedFunds()`, `recoverERC20()` and `recoverAVAX()` functions, and then a new strategy may be deployed.

- A different `GmxProxy` contract can be used with the same `GmxDepositor` contract, making the upgrade transparent to the GMX project contracts.

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

GmxDepositor

This contract is derived from the `Ownable` contract and inherits all its privileged roles.

Owner

Besides all the functionality defined in the `Ownable` contract, the contract owner can set the address that has the `GmxProxy` role using the `setGmxProxy()` function. It is important to notice that the owner address is passed as an argument upon construction and does not necessarily coincide with the address that deploys the contract, as one would expect from an `Ownable` contract.

GmxProxy

The proxy address for the `GmxProxy` can be set only by the contract owner. Once defined, the GmxProxy address is the only one which can use the `execute()` function. There is no proxy address defined by default.

GmxProxy

Dev

The Dev address is initialized when passed as an argument to the constructor of the `GmxProxy` contract. It is responsible for approving strategies with the `approveStrategy()` function. The Dev can set a new Dev using the `updateDevAddr()` function.

Strategy

Strategy addresses are approved by the Dev and can claim rewards using the `claimReward()` function. They are classified in two types: GLPStrategy and GMXStrategy.

GLPStrategy

GLPStrategy addresses are a particular type of Strategy. Besides being approved by Dev and being able to claim rewards using the `claimReward()` function, they can:

- Buy and stake GLP with the `buyAndStakeGlp()` function.
- Withdraw GLP using the `withdrawGlp()` or `emergencyWithdrawGlp()` functions.

GMXStrategy

GLPStrategy addresses are also approved by Dev and are able to claim rewards using the `claimReward()` function. Furthermore, they can:

- Stake GMX with the `stakeGmx()` function
- Withdraw GMX using `withdrawGmx()` or `emergencyWithdrawGmx()` functions.

GmxStrategyForGLP

This contract is derived from the `YakStrategyV2` contract and inherits all its privileged roles. It must be noted that it does not use the depositor role defined in the Permissioned abstract contract, that is a parent of `YakStrategyV2`.

Owner

The owner is the deployer address of this contract. It inherits from the `YakStrategyV2` contract, which is Ownable, all the functionality defined in the Ownable contract as well. Furthermore, it can set the GmxProxy using the `setProxy()` function and retrieve deployed funds without a minimum amount being required using the `rescueDeployedFunds()` function.

EOA

Externally owned accounts, EOA, are the only ones allowed to reinvest. The reinvest action can be triggered using the `reinvest()` function.

Ownable

Owner

The owner of the contract can:

- renounce its ownership.
- transfer the ownership to another address.

Derived contracts may define external or public functions using the `onlyOwner` modifier to make those actions available just to the owner of the contract.

The address doing the deployment is the initial owner of a contract derived from `Ownable`.

Permissioned

This contract is derived from the `Ownable` contract and inherits all its privileged roles.

Owner

The owner is the deployer address of this contract. Besides all the functionality defined in the `Ownable` contract, the contract owner can allow new depositors into the allowedDepositors mapping using the `allowDepositor()` function. It can also remove existing depositors using the `removeDepositor()` function.

Depositor

While this contract has the ability to track the depositors using the `allowDepositor()` and `removeDepositor()` functions, and make a function only executable by a depositor using the `onlyAllowedDeposits()` modifier, it does not define any functionality for the role.

Also take note that this role is not related to the `GmxDepositor` contract or the `IGmxDepositor` interface.

YakStrategyV2

This abstract contract is derived from the `YaKERC20`, `Ownable` and `Permissioned` contracts. Since `YaKERC20` has no privileged roles, `YaKStrategyV2` inherits only the privileged roles of the last two contracts.

Owner

The owner is the deployer address of this contract. In addition to all the functionality defined in the `Ownable` contract, the contract owner can:

- Set the allowance of a spending address to zero through the `revokeAllowance()` function.
- Set the minimum amount of tokens required to reinvest through the `updateMinTokensToReinvest()` function
- Set the maximum tokens to deposit without forcing a reinvest in the deposit process using the `updateMaxTokensToDepositWithoutReinvest()` function.
- Set the dev fee using the `updateDevFee()` function.
- Set the admin fee using the `updateAdminFee()` function.
- Set the reinvest reward using the `updateReinvestReward()` function.
- Enable or disable deposits using the `updateDepositsEnabled()` function.
- Retrieve ERC20 tokens from an address using the `recoverERC20()` function.
- Retrieve AVAX from the contract using the `recoverAVAX()` function.

Dev

The Dev can set a new Dev using the `updateDevAddr()` function. There is no Dev address defined by default, and it must be set initially by the derived contract, as is in the `GmxStrategyForGLP` contract.

Contract Interactions

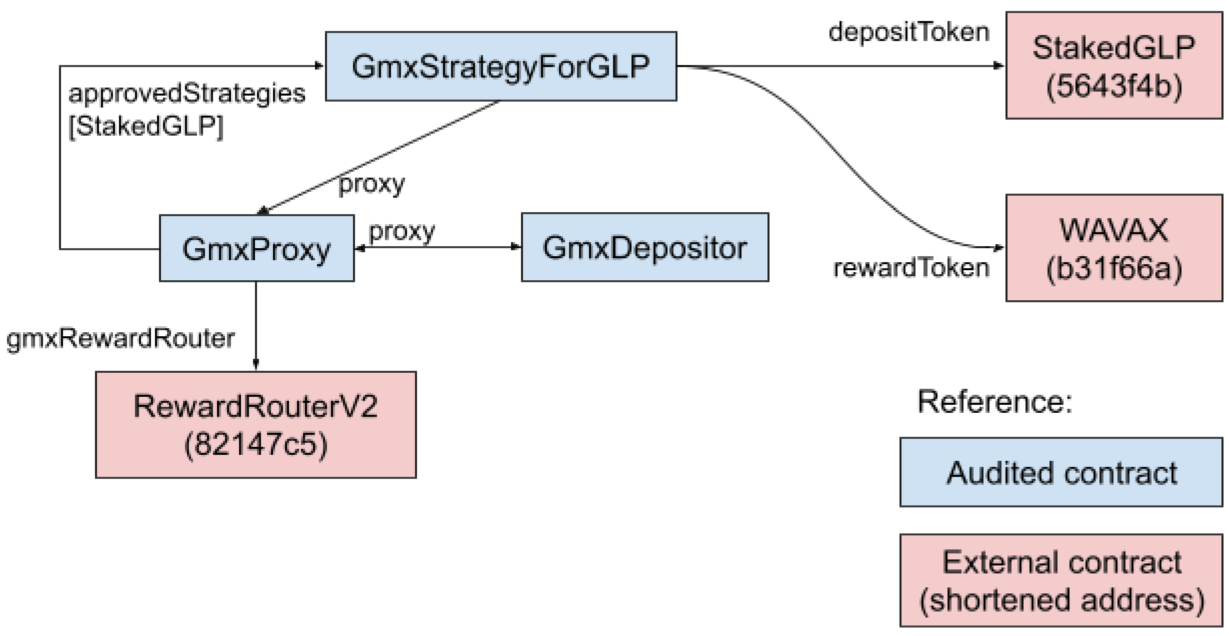

Figure: References Between Audited and External Contracts

The audited contracts interact with several contracts in the Avalanche network. The external contracts directly referenced by the audited contracts are:

a. StackedGLP (`0x5643f4b25e36478ee1e90418d5343cb6591bcb9d`).
b. WAVAX (`0xB31f66AA3C1e785363F0875A1B74E27b85FD66c7`).
c. RewardRouterV2 (`0x82147C5A7E850eA4E28155DF107F2590fD4ba327`).

StakedGlp as an ERC20 Token

The `StakedGlp` contract, deployed in the avalanche blockchain on address `0x5643F4b25E36478eE1E90418d5343cb6591BcB9d`, is not an standard ERC20 but it has `approve()`, `transfer()` and `transferFrom()` functions compatible enough with the ERC20 definition that they can be used by the `GmxStrategyForGLP` contract as if it were an ERC20 token.

This allows the following functions in the `GmxStrategyForGLP` contract to be used:

- `deposit()`.
- `depositFor()`.
- `withdraw()`.
- `revokeAllowance()` (defined in `YakStrategyV2`).
- `recoverERC20()` (defined in `YakStrategyV2`).

Tests

The development team informed us that they have a private git repository with tests covering the functionality of the `GmxStrategyForGLP` and other strategies outside the scope of this audit.

Changelog

- 2022-11-11 – Draft made after the first week of auditing based on the `a818f34b553763c0c25d63fbb9c2c2ada7280def` commit.
- 2022-11-23 – Report with all the findings for commit `a818f34b553763c0c25d63fbb9c2c2ada7280def`.
- 2022-11-28 – Check fixes on commit `93645e8732daf92472ac19c867c9cce55806b1bc`.

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the YieldYak project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.