ABDK CONSULTING

SMART CONTRACT
AUDIT

Vovo

VovoFinance

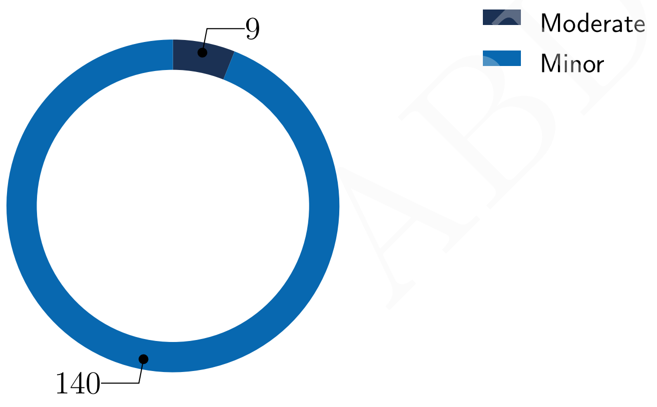
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
19th July 2022

We've been asked to review 2 files in a Github repository. We found 9 moderate, and a few less important issues.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Procedural	Fixed
CVF-2	Minor	Procedural	Info
CVF-3	Minor	Bad naming	Fixed
CVF-4	Minor	Suboptimal	Info
CVF-5	Minor	Bad datatype	Info
CVF-6	Minor	Bad datatype	Info
CVF-7	Minor	Bad datatype	Info
CVF-8	Minor	Documentation	Fixed
CVF-9	Minor	Suboptimal	Info
CVF-10	Minor	Bad datatype	Info
CVF-11	Minor	Bad datatype	Info
CVF-12	Minor	Bad datatype	Info
CVF-13	Minor	Procedural	Info
CVF-14	Minor	Bad naming	Info
CVF-15	Minor	Bad naming	Fixed
CVF-16	Minor	Bad datatype	Info
CVF-17	Minor	Bad datatype	Info
CVF-18	Minor	Bad datatype	Info
CVF-19	Minor	Unclear behavior	Fixed
CVF-20	Minor	Bad datatype	Info
CVF-21	Minor	Unclear behavior	Info
CVF-22	Minor	Unclear behavior	Fixed
CVF-23	Minor	Bad datatype	Info
CVF-24	Minor	Suboptimal	Fixed
CVF-25	Minor	Readability	Info
CVF-26	Minor	Bad datatype	Fixed
CVF-27	Minor	Readability	Fixed

ID	Severity	Category	Status
CVF-28	Minor	Overflow/Underflow	Fixed
CVF-29	Minor	Readability	Fixed
CVF-30	Minor	Suboptimal	Fixed
CVF-31	Minor	Suboptimal	Fixed
CVF-32	Minor	Readability	Fixed
CVF-33	Minor	Unclear behavior	Fixed
CVF-34	Minor	Unclear behavior	Fixed
CVF-35	Minor	Suboptimal	Fixed
CVF-36	Minor	Unclear behavior	Info
CVF-37	Minor	Readability	Fixed
CVF-38	Minor	Unclear behavior	Fixed
CVF-39	Minor	Unclear behavior	Fixed
CVF-40	Minor	Readability	Info
CVF-41	Moderate	Unclear behavior	Info
CVF-42	Minor	Bad datatype	Info
CVF-43	Minor	Suboptimal	Fixed
CVF-44	Minor	Readability	Fixed
CVF-45	Minor	Suboptimal	Fixed
CVF-46	Minor	Bad datatype	Info
CVF-47	Minor	Unclear behavior	Fixed
CVF-48	Minor	Readability	Fixed
CVF-49	Minor	Unclear behavior	Info
CVF-50	Minor	Flaw	Info
CVF-51	Minor	Unclear behavior	Info
CVF-52	Minor	Procedural	Info
CVF-53	Minor	Procedural	Fixed
CVF-54	Minor	Bad datatype	Info
CVF-55	Minor	Procedural	Fixed
CVF-56	Minor	Procedural	Info
CVF-57	Minor	Procedural	Fixed

ID	Severity	Category	Status
CVF-58	Minor	Procedural	Fixed
CVF-59	Minor	Bad naming	Info
CVF-60	Minor	Suboptimal	Info
CVF-61	Minor	Bad datatype	Info
CVF-62	Minor	Bad datatype	Info
CVF-63	Minor	Bad datatype	Info
CVF-64	Minor	Bad datatype	Info
CVF-65	Minor	Bad datatype	Info
CVF-66	Minor	Bad datatype	Info
CVF-67	Minor	Suboptimal	Info
CVF-68	Minor	Documentation	Fixed
CVF-69	Minor	Bad naming	Info
CVF-70	Minor	Procedural	Fixed
CVF-71	Minor	Bad naming	Info
CVF-72	Minor	Bad naming	Info
CVF-73	Minor	Documentation	Fixed
CVF-74	Minor	Unclear behavior	Info
CVF-75	Minor	Procedural	Info
CVF-76	Minor	Bad datatype	Info
CVF-77	Minor	Bad datatype	Info
CVF-78	Minor	Bad naming	Info
CVF-79	Minor	Procedural	Info
CVF-80	Minor	Bad datatype	Info
CVF-81	Minor	Bad datatype	Info
CVF-82	Minor	Flaw	Fixed
CVF-83	Minor	Suboptimal	Info
CVF-84	Minor	Suboptimal	Info
CVF-85	Minor	Bad datatype	Info
CVF-86	Minor	Bad datatype	Info
CVF-87	Minor	Unclear behavior	Fixed

ID	Severity	Category	Status
CVF-88	Moderate	Unclear behavior	Fixed
CVF-89	Moderate	Unclear behavior	Fixed
CVF-90	Minor	Suboptimal	Fixed
CVF-91	Minor	Unclear behavior	Fixed
CVF-92	Minor	Unclear behavior	Fixed
CVF-93	Minor	Procedural	Fixed
CVF-94	Minor	Overflow/Underflow	Fixed
CVF-95	Minor	Readability	Info
CVF-96	Moderate	Flaw	Info
CVF-97	Minor	Suboptimal	Info
CVF-98	Minor	Suboptimal	Fixed
CVF-99	Minor	Suboptimal	Info
CVF-100	Moderate	Flaw	Info
CVF-101	Minor	Suboptimal	Info
CVF-102	Moderate	Flaw	Info
CVF-103	Minor	Bad datatype	Info
CVF-104	Minor	Unclear behavior	Info
CVF-105	Moderate	Flaw	Fixed
CVF-106	Minor	Suboptimal	Fixed
CVF-107	Moderate	Flaw	Info
CVF-108	Minor	Readability	Info
CVF-109	Minor	Suboptimal	Fixed
CVF-110	Minor	Bad datatype	Fixed
CVF-111	Minor	Suboptimal	Fixed
CVF-112	Minor	Unclear behavior	Info
CVF-113	Minor	Suboptimal	Info
CVF-114	Minor	Unclear behavior	Info
CVF-115	Minor	Unclear behavior	Info
CVF-116	Minor	Unclear behavior	Info
CVF-117	Minor	Suboptimal	Fixed

ID	Severity	Category	Status
CVF-118	Minor	Bad datatype	Info
CVF-119	Minor	Flaw	Info
CVF-120	Minor	Unclear behavior	Info
CVF-121	Moderate	Flaw	Info
CVF-122	Minor	Procedural	Fixed
CVF-123	Minor	Suboptimal	Fixed
CVF-124	Minor	Suboptimal	Fixed
CVF-125	Minor	Unclear behavior	Info
CVF-126	Minor	Bad datatype	Info
CVF-127	Minor	Suboptimal	Info
CVF-128	Minor	Unclear behavior	Info
CVF-129	Minor	Procedural	Fixed
CVF-130	Minor	Suboptimal	Fixed
CVF-131	Minor	Flaw	Fixed
CVF-132	Minor	Suboptimal	Info
CVF-133	Minor	Bad naming	Fixed
CVF-134	Minor	Unclear behavior	Fixed
CVF-135	Minor	Procedural	Fixed
CVF-136	Minor	Procedural	Fixed
CVF-137	Minor	Bad datatype	Info
CVF-138	Minor	Bad datatype	Info
CVF-139	Minor	Procedural	Fixed
CVF-140	Minor	Bad datatype	Info
CVF-141	Minor	Suboptimal	Info
CVF-142	Minor	Bad datatype	Fixed
CVF-143	Minor	Bad datatype	Fixed
CVF-144	Minor	Unclear behavior	Info
CVF-145	Minor	Unclear behavior	Info
CVF-146	Minor	Unclear behavior	Fixed
CVF-147	Minor	Suboptimal	Fixed

ID	Severity	Category	Status
CVF-148	Minor	Suboptimal	Fixed
CVF-149	Minor	Documentation	Fixed

Contents

1 Document properties

Version

Version	Date	Author	Description
0.1	July 7, 2022	D. Khovratovich	Initial Draft
0.2	July 15, 2022	D. Khovratovich	Minor revision
1.0	July 19, 2022	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations.
We have reviewed the contracts in the 635cc430 commit:

- GlpVault.sol
- PrincipalProtectedVault.sol

The fixes were provided in the 8bced53 commit.

2.1 About ABDK

ABDK Consulting, established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function. The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

2.3 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places and that their visibility scopes and access levels are relevant. At this phase we understand overall system architecture and how different parts of the code are related to each other.
- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and is done properly. At this phase we understand user roles and permissions, as well as what assets the system ought to protect.

- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

3 Detailed Results

3.1 CVF-1

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Recommendation Should be "^0.7.0" according to a common best practice, unless there is something special about this particular version. Also relevant for the next files: GlpVault.sol, UniERC20.sol, Uni.sol, IGlpVault.sol, Univ3Swapper.sol, IVovoVault.sol.

Listing 1:

```
pragma solidity ^0.7.6;
```

3.2 CVF-2

- **Severity** Minor
- **Category** Procedural
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Recommendation We didn't review these files.
Client Comment Noted.

Listing 2:

```
import "../interfaces/curve/Gauge.sol";
import "../interfaces/curve/Curve.sol";

import "../interfaces/gmx/IRouter.sol";
import "../interfaces/gmx/IVault.sol";
```

3.3 CVF-3

- **Severity** Minor
- **Category** Bad naming
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Recommendation Use uppercase for constants.

Listing 3:

```
address public constant usdc = address(0
  ↪ xFF970A61A04b1cA14834A43f5dE4533eBDDB5CC8);

address public constant crv = address(0
  ↪ x11cDb42B0EB46D95f990BeDD4695A6e3fA034978);
```

3.4 CVF-4

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Description Hardcoding mainnet addresses is a bad practice, as it makes it harder to test smart contracts.
Recommendation Consider passing the addresses as constructor arguments and storing in internal immutable variables.
Client Comment Noted with thanks. The reason we did not pass it into initalized function is the function parameters is already too many and it will lead to stack too deep error, though there might be other workaround. Our integration tests are running on mainnet fork, so there are not any testing issues.

Listing 4:

```
address public constant usdc = address(0xFF970A61A04b1cA14834A43f5dE4533eBDDB5CC8);

address public constant crv = address(0x11cDb42B0EB46D95f990BeDD4695A6e3fA034978);

    gmxPositionManager = address(0x87a4088Bd721F83b6c2E5102e2FA47022Cb1c831);
    gmxRouter = address(0xaBBc5F99639c9B6bCb58544ddf04EFA6802F4064);
    gmxVault = address(0x489ee077994B6658eAfA855C308275EAd8097C4A);
```

3.5 CVF-5

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Recommendation The type of this variable should be "IERC20".
Client Comment Noted. As does not make big difference, we prefer to keep it as it is to be compatible with currently deployed contracts.

Listing 5:

```
address public vaultToken; // deposited token of the vault
```

3.6 CVF-6

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Recommendation The type of this variable should be “ICurveFi”.
Client Comment Same as above.

Listing 6:

```
address public lpToken;
```

3.7 CVF-7

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Recommendation The type of this variable should be “Gauge”.
Client Comment Same as above.

Listing 7:

```
address public gauge;
```

3.8 CVF-8

- **Severity** Minor
- **Category** Documentation
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Description The number format for this variable is unclear.
Recommendation Consider documenting.

Listing 8:

```
uint256 public leverage;
```

3.9 CVF-9

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Description Fractional leverages are not supported, i.e. one may set leverage of of 2 or 3, but not 2.5.
Recommendation Consider adding support for fractional leverages.
Client Comment As there is no use case for fractional leverage for the vault, we prefer to keep it as it is.

Listing 9:

```
uint256 public leverage;
```

3.10 CVF-10

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Recommendation The type of this variable should be "Uni".
Client Comment Same comment as row 6.

Listing 10:

```
address public dex;
```

3.11 CVF-11

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Recommendation The type of this variable should be "IRouter".
Client Comment Same comment as row 6.

Listing 11:

```
address public gmxPositionManager;
address public gmxRouter;
```

3.12 CVF-12

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Recommendation The type of this variable should be “IVault”.
Client Comment Same comment as row 6.

Listing 12:

```
address public gmxVault;
```

3.13 CVF-13

- **Severity** Minor
- **Category** Procedural
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Recommendation All address parameters should be indexed.
Client Comment Makes sense, but since we already have contracts in production, prefer to keep the abi the same for future contracts so that they are compatible with the current subgraph. Would prefer to keep them as it is.

Listing 13:

```
event Deposit(address depositor, address account, uint256 amount
    ↪ , uint256 shares);

event OpenPosition(address underlying, uint256 underlyingPrice,
    ↪ uint256 vaultTokenPrice, uint256 sizeDelta, bool isLong,
    ↪ uint256 collateralAmountVaultToken);
event ClosePosition(address underlying, uint256 underlyingPrice,
    ↪  uint256 vaultTokenPrice,uint256 sizeDelta, bool isLong,
    ↪ uint256 collateralAmountVaultToken, uint256 fee);
event Withdraw(address account, uint256 amount, uint256 shares);
event WithdrawToVault(address owner, uint256 shares, address
    ↪ vault, uint256 receivedShares);
event GovernanceSet(address governor);
event AdminSet(address admin);
event GuardianSet(address guardian);

event RewardsSet(address rewards);
event GmxContractsSet(address gmxPositionManager, address
    ↪ gmxRouter, address gmxVault);

event KeeperAdded(address keeper);
event KeeperRemoved(address keeper);
event VaultRegistered(address fromVault, address toVault);
event VaultRevoked(address fromVault, address toVault);
```

3.14 CVF-14

- **Severity** Minor
- **Category** Bad naming
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Recommendation Events are usually named via nouns such as “Liquidity”, “GaugeDeposit”, “Poke” etc.
Client Comment Prefer to keep the event names so that it’s compatible with our current subgraph.

Listing 14:

```
event LiquidityAdded(uint256 tokenAmount, uint256 lpMinted);
event GaugeDeposited(uint256 lpDeposited);
event Poked(uint256 pricePerShare, uint256 feeShare);

event GovernanceSet(address governor);
event AdminSet(address admin);
event GuardianSet(address guardian);
event FeeSet(uint256 performanceFee, uint256 withdrawalFee);
event LeverageSet(uint256 leverage);
event isLongSet(bool isLong);
event RewardsSet(address rewards);
event GmxContractsSet(address gmxPositionManager, address
  ↪ gmxRouter, address gmxVault);
event MaxCollateralMultiplierSet(uint256 maxCollateralMultiplier
  ↪ );
event ParametersSet(bool isDepositEnabled, uint256 cap, uint256
  ↪ pokeInterval, bool isKeeperOnly);
event KeeperAdded(address keeper);
event KeeperRemoved(address keeper);
event VaultRegistered(address fromVault, address toVault);
event VaultRevoked(address fromVault, address toVault);
```

3.15 CVF-15

- **Severity** Minor
- **Category** Bad naming
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Recommendation The name should be "IsLongSet" with the first upper letter.

Listing 15:

```
event isLongSet(bool isLong);
```

3.16 CVF-16

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Recommendation The types of event parameters should be "IRouter", "IRouter", and "IVault" respectively.
Client Comment Same comment as row 6.

Listing 16:

```
event GmxContractsSet(address gmxPositionManager, address
    ↪ gmxRouter, address gmxVault);
```

3.17 CVF-17

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Recommendation The type of this argument should be "ICurveFi".
Client Comment Same comment as row 6.

Listing 17:

```
address _lpToken,
```

3.18 CVF-18

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Recommendation The type of this argument should be "Gauge".
Client Comment Same comment as row 6.

Listing 18:

```
address _gauge,
```

3.19 CVF-19

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Description There is not range check for this argument, while in the "setLeverage" function only leverages between 1 and 50 are allowed.

Listing 19:

```
103 uint256 _leverage ,
```

3.20 CVF-20

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Recommendation The Type of this argument should be "Uni".
Client Comment Same comment as row 6.

Listing 20:

```
108 address _dex
```

3.21 CVF-21

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Description Is it really required to be flexible with decimals and don't use default 18? It creates potential place for mistakes, some DeFi tools has hardcoded 18 decimals, there are also a lot of usages of hardcoded 18 in the code below, so it's better to decrease flexibility of the code in this case and be sure in advance that decimals is always 18.
Client Comment We are keeping the vaut decimal to be the same as the vault token decimal. For example, for USDC vault, we are using 6 decimal.

Listing 21:

```
111 _setupDecimals(_vaultDecimal);
```

3.22 CVF-22

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Recommendation The common practice is to explicitly check if the newValue != address(0).

Listing 22:

```
113 vaultToken = _vaultToken;
    underlying = _underlying;
    lpToken = _lpToken;
    gauge = _gauge;
    rewards = _rewards;

123 dex = _dex;
```

3.23 CVF-23

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Recommendation These constants should be named.
Client Comment Added the constant for 1e18. I prefer to keep others as raw numbers, as those constants are only used once, using the raw value has a better readability.

Listing 23:

```
135 managementFee = 200;
    performanceFee = 1000;
    slip = 30;

413 return balance(isMax).mul(1e18).div(totalSupply());

477 require(_performanceFee < 5000 && _managementFee < 500, "!too-
      ↪ much");

484 require(_leverage >= 1 && _leverage <= 50, "!leverage");

512 require(_maxCollateralMultiplier >= 1 &&
      ↪ _maxCollateralMultiplier <= 50, "!maxCollateralMultiplier
      ↪ ");
```

3.24 CVF-24

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Recommendation Decimals values should be named constants, and constants derived from those should be named as well.
Client Comment Added the constant for 1e18. And also represent 1e36 using the constant.

Listing 24:

```
151 uint256 lpValue = lpPrice.mul(lpAmount).mul(vaultTokenBase).div
      ↪ (1e36);

167    uint256 expectedLpAmount = tokenBalance.mul(1e18).div(
         ↪ vaultTokenBase).mul(1e18).div(ICurveFi(lpToken).
         ↪ get_virtual_price());

368    uint256 lpAmount = (withdrawAmount.sub(b)).mul(1e18).div(
         ↪ vaultTokenBase).mul(1e18).div(lpPrice);
```

3.25 CVF-25

- **Severity** Minor
- **Category** Readability
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Description This code is difficult to read, better to add a comment. Combining multiple operations into a single line makes the code difficult to read.
Recommendation Consider splitting into several lines and providing comments or a link to the math description elsewhere.
Client Comment The current comment for the function has explained this logic.
* if isMax is true: the value of lp in vaultToken + the amount of vaultToken in this contract + the value of open leveraged position + estimated pending rewards * if isMax is false: the value of lp in vaultToken + the amount of vaultToken in this contract.

Listing 25:

```
153 return lpValue.add(getActivePositionValue()).add(
      ↪ getEstimatedPendingRewardValue()).add(IERC20(vaultToken).
      ↪ balanceOf(address(this)));
```

3.26 CVF-26

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** PrincipalProtectedVault.sol

Recommendation All precisions should be named constants or values derived from named constants.

Listing 26:

```
uint256 lpValue = lpPrice.mul(lpAmount).mul(vaultTokenBase).div
  ↪ (1e36);

  uint256 expectedLpAmount = tokenBalance.mul(1e18).div(
    ↪ vaultTokenBase).mul(1e18).div(ICurveFi(lpToken).
    ↪ get_virtual_price());

  uint256 lpAmount = (withdrawAmount.sub(b)).mul(1e18).div(
    ↪ vaultTokenBase).mul(1e18).div(lpPrice);

uint256 expectedVaultTokenAmount = _amnt.mul(vaultTokenBase).mul
  ↪ (ICurveFi(lpToken).get_virtual_price()).div(1e36);

return balance(isMax).mul(1e18).div(totalSupply());
```

3.27 CVF-27

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** PrincipalProtectedVault.sol

Recommendation Should be ‘else return‘ for readability.

Listing 27:

```
return lpValue.add(IERC20(vaultToken).balanceOf(address(this)));
```

3.28 CVF-28

- **Severity** Minor
- **Category** Overflow/Underflow
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Description Phantom overflow is possible here i.e. a situation when the final calculation result would fit into the destination type while some intermediary calculation overflows.

Recommendation Consider using the “muldiv” function as described here: https://xn–2-umb.com/21/muldiv/ or using some other approach that prevents phantom overflow.

Listing 28:

```
uint256 lpValue = lpPrice.mul(lpAmount).mul(vaultTokenBase).div
  ↪ (1e36);

  uint256 expectedLpAmount = tokenBalance.mul(1e18).div(
    ↪ vaultTokenBase).mul(1e18).div(ICurveFi(lpToken).
    ↪ get_virtual_price());
  uint256 lpMinted = ICurveFi(lpToken).add_liquidity([
    ↪ tokenBalance, 0], expectedLpAmount.mul(DENOMINATOR.sub(
    ↪ slip)).div(DENOMINATOR));

  shares = (amount.mul(totalSupply())).div(_pool);

uint256 feeShare = totalSupply().mul(managementFee).mul(block.
  ↪ timestamp.sub(lastPokeTime)).div(86400*365).div(
  ↪ FEE_DENOMINATOR);

uint256 _sizeDelta = leverage.mul(amount).mul(_vaultTokenPrice).
  ↪ div(vaultTokenBase);

  _fee = _tradeProfit.mul(performanceFee).div(FEE_DENOMINATOR);

withdrawAmount = (balance(false).mul(shares)).div(totalSupply())
  ↪ ; // use minimum vault balance for withdraw

  uint256 lpAmount = (withdrawAmount.sub(b)).mul(1e18).div(
    ↪ vaultTokenBase).mul(1e18).div(lpPrice);

uint256 expectedVaultTokenAmount = _amnt.mul(vaultTokenBase).mul
  ↪ (ICurveFi(lpToken).get_virtual_price()).div(1e36);
ICurveFi(lpToken).remove_liquidity_one_coin(_amnt, 0,
  ↪ expectedVaultTokenAmount.mul(DENOMINATOR.sub(slip)).div(
  ↪ DENOMINATOR));

return balance(isMax).mul(1e18).div(totalSupply());

return currentTokenReward.mul(block.timestamp.sub(lastPokeTime))
  ↪ .div(currentPokeInterval);
```

3.29 CVF-29

- **Severity** Minor
- **Category** Readability
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Recommendation Should be “else return” for readability.

Listing 29:

```
return lpValue.add(IERC20(vaultToken).balanceOf(address(this)));
```

3.30 CVF-30

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Description The vault token balance is obtained and added in two places.
Recommendation Consider refactoring the code to avoid code duplication.

Listing 30:

```
  return lpValue.add(getActivePositionValue()).add(getEstimatedPendingRewardValue()).add(IERC20(vaultToken).balanceOf(address(this)));

return lpValue.add(IERC20(vaultToken).balanceOf(address(this)));
```

3.31 CVF-31

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Recommendation A single “approve“ or "safeIncreaseAllowance" call would be more efficient. It makes no sense to first set allowance to zero and then to the new value.

Listing 31:

```
165   IERC20(vaultToken).safeApprove(lpToken, 0);
      IERC20(vaultToken).safeApprove(lpToken, tokenBalance);

173   IERC20(lpToken).safeApprove(gauge, 0);
      IERC20(lpToken).safeApprove(gauge, lpBalance);

256   IERC20(crv).safeApprove(dex, 0);
      IERC20(crv).safeApprove(dex, _crv);

282 IERC20(vaultToken).safeApprove(gmxRouter, 0);
    IERC20(vaultToken).safeApprove(gmxRouter, amount);

404 IERC20(lpToken).safeApprove(lpToken, 0);
    IERC20(lpToken).safeApprove(lpToken, _amnt);
```

3.32 CVF-32

- **Severity** Minor
- **Category** Readability
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Description This code is difficult to read, it’s better to add a comment. Combining multiple operations into a single line makes the code difficult to read.
Recommendation Consider splitting into several lines and providing comments or a link to the math description elsewhere.
Client Comment Added the comment. Did not split the lines though, as the main cause of the long line is the decimal convertion, which makes it hard to break and reasonable to understand with comments.

Listing 32:

```
167 uint256 expectedLpAmount = tokenBalance.mul(1e18).div(
      ↪ vaultTokenBase).mul(1e18).div(ICurveFi(lpToken).
      ↪ get_virtual_price());
    uint256 lpMinted = ICurveFi(lpToken).add_liquidity([tokenBalance
      ↪ , 0], expectedLpAmount.mul(DENOMINATOR.sub(slip)).div(
      ↪ DENOMINATOR));
```

3.33 CVF-33

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** PrincipalProtectedVault.sol

Description Performs a multiplication on the result of a division.
Recommendation Consider doing multiplications first to avoid precision loss, and to use overflow-protection methods.

Listing 33:

```
uint256 expectedLpAmount = tokenBalance.mul(1e18).div(
    ↪ vaultTokenBase).mul(1e18).div(ICurveFi(lpToken).
    ↪ get_virtual_price());
```

3.34 CVF-34

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** PrincipalProtectedVault.sol

Recommendation There is no reason to use safeApprove in this case. It does not help to prevent attack on ERC-20 protocol. safeApprove method is deprecated, see https://github.com/OpenZeppelin/openzeppelin-contracts/issues/2219 Use a single approve with a success status check or call safeIncreaseAllowance from SafeERC20

Listing 34:

```
IERC20(lpToken).safeApprove(gauge, 0);
IERC20(lpToken).safeApprove(gauge, lpBalance);
```

3.35 CVF-35

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** PrincipalProtectedVault.sol

Recommendation The "isDepositEnaled" check should be done in the very beginning of the function.

Listing 35:

```
require(isDepositEnabled && _pool.add(amount) < cap, "!deposit")
    ↪ ;
```

3.36 CVF-36

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** PrincipalProtectedVault.sol

Description This check doesn't make much sense for a keeper, as a keeper may skip this check by calling the "colleclRewardByKeeper" and "closeTradeByKeeper" functions.
Recommendation Consider not performing this check for keeprs like this: require (keepers[msg.sender] || lastPokeTime.add(pokeInterval) < block.timestamp);
Client Comment Added "disablePokeInterval" flag. Only allow the keeper to call these two functions if that flag is true. We still want to disallow public to call this function unless "isKeeperOnly" is false.

Listing 36:

```
require(lastPokeTime.add(pokeInterval) < block.timestamp, "!poke
    ↪ time");
```

3.37 CVF-37

- **Severity** Minor
- **Status** Fixed
- **Category** Readability
- **Source** PrincipalProtectedVault.sol

Recommendation The value "86400*365" could be rendered as "365 days".

Listing 37:

```
uint256 feeShare = totalSupply().mul(managementFee).mul(block.
    ↪ timestamp.sub(lastPokeTime)).div(86400*365).div(
    ↪ FEE_DENOMINATOR);
```

3.38 CVF-38

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** PrincipalProtectedVault.sol

Recommendation There is no reason to use safeApprove in this case. It does not help to prevent attack on ERC-20 protocol. safeApprove method is deprecated, see https://github.com/OpenZeppelin/openzeppelin-contracts/issues/2219 Use a single approve with a success status check or call safeIncreaseAllowance from SafeERC20
Client Comment Fixed. Changed to safeIncreaseAllowance instead.

Listing 38:

```
IERC20(crv).safeApprove(dex, 0);
IERC20(crv).safeApprove(dex, _crv);
```

3.39 CVF-39

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Recommendation There is no reason to use safeApprove in this case. It does not help to prevent attack on ERC-20 protocol. safeApprove method is deprecated, see https://github.com/OpenZeppelin/openzeppelin-contracts/issues/2219 Use a single approve with a success status check or call safeIncreaseAllowance from SafeERC20
Client Comment Fixed. Changed to safeIncreaseAllowance instead.

Listing 39:

```
282 IERC20(vaultToken).safeApprove(gmxRouter, 0);
```

3.40 CVF-40

- **Severity** Minor
- **Category** Readability
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Description The code below looks like it is always executed, while actually it is executed only when “size != 0”.
Recommendation Consider refactoring like this: if (size != 0) { ... } emit ClosePosition(...);
Client Comment Think current way has better readability, as the tradeProfit and fee are explictly set to 0 in the ClosePosition event when the size == 0.

Listing 40:

```
308 }

424 }
```

3.41 CVF-41

- **Severity** Moderate
- **Category** Unclear behavior
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Description The parameter minOut is set to 0, so sandwich transaction attack is possible. Concretely, someone may manipulate the pool. In the first transaction the attacker executes the swap to shift the price in the pool, then in the 2nd transaction the execution of this code happens and then tin the 3rd transaction the attacker reverts the pool to the original state. When the user's (2nd) transaction is mined, the user swap is executed for a much worse price unfavourite to the user.
Client Comment Note that the GMX uses oracle price to settle the trade with 0 slippage, so it's not possible to execute the sanwitch attack, as the frontrunning won't impact the price.

Listing 41:

```
318 IRouter(gmxPositionManager).decreasePositionAndSwap(path,
    ↪ underlying, 0, size, isLong, address(this),
    ↪ _underlyingPrice, 0);
```

3.42 CVF-42

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Recommendation The type of the "vault" argument should be "IVovoVault".
Client Comment Same comment as row 6.

Listing 42:

```
344 function withdrawToVault(uint256 shares, address vault) external
    ↪ whenNotPaused nonReentrant {
```

3.43 CVF-43

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Recommendation This check is redundant as it is superseded by the next check.

Listing 43:

```
345 require(vault != address(0), "!vault");
```

3.44 CVF-44

- **Severity** Minor
- **Category** Readability
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Description This code is difficult to read, it's better to add a comment. Combining multiple operations into a single line makes the code difficult to read.
Recommendation Consider splitting into several lines and providing comments or a link to the math description elsewhere.

Listing 44:

```
368 uint256 lpAmount = (withdrawAmount.sub(b)).mul(1e18).div(
    ↪ vaultTokenBase).mul(1e18).div(lpPrice);
```

3.45 CVF-45

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Recommendation The expresion "b.add(_diff)" is equivalent to "_after".

Listing 45:

```
373 withdrawAmount = b.add(_diff);
```

3.46 CVF-46

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Recommendation The type of the "_asset" argument should be "IERC20".
Client Comment Same comment as row 6.

Listing 46:

```
383 function withdrawAsset(address _asset) external onlyGovernor {
```

3.47 CVF-47

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Recommendation There is no reason to use safeApprove in this case. It does not help to prevent attack on ERC-20 protocol. safeApprove method is deprecated, see https://github.com/OpenZeppelin/openzeppelin-contracts/issues/2219. Use a single approve with a success status check or call safeIncreaseAllowance from SafeERC20.

Listing 47:

```
IERC20(lpToken).safeApprove(lpToken, 0);
IERC20(lpToken).safeApprove(lpToken, _amnt);
```

3.48 CVF-48

- **Severity** Minor
- **Category** Readability
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Description This code is difficult to read, it's better to add a comment. Combining multiple operations into a single line makes the code difficult to read.
Recommendation Consider splitting into several lines and providing comments or a link to the math description elsewhere.
Client Comment Added the comment. Did not split the lines though, as the main cause of the long line is the decimal convertion, which makes it hard to break and reasonable to understand with comments.

Listing 48:

```
uint256 expectedVaultTokenAmount = _amnt.mul(vaultTokenBase).mul
    ↪ (ICurveFi(lpToken).get_virtual_price()).div(1e36);
ICurveFi(lpToken).remove_liquidity_one_coin(_amnt, 0,
    ↪ expectedVaultTokenAmount.mul(DENOMINATOR.sub(slip)).div(
    ↪ DENOMINATOR));
```

3.49 CVF-49

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Description Performs a multiplication on the result of a division.
Recommendation Consider doing multiplications first to avoid precision loss, and to use overflow-protection methods.
Client Comment This line is already doing multplication first.

Listing 49:

```
406 uint256 expectedVaultTokenAmount = _amnt.mul(vaultTokenBase).mul
    ↪ (ICurveFi(lpToken).get_virtual_price()).div(1e36);
```

3.50 CVF-50

- **Severity** Minor
- **Category** Flaw
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Recommendation The statement ignores return value.
Client Comment Prefer to keep it as it is, as the return value is not useful here.

Listing 50:

```
407 ICurveFi(lpToken).remove_liquidity_one_coin(_amnt, 0,
    ↪ expectedVaultTokenAmount.mul(DENOMINATOR.sub(slip)).div(
    ↪ DENOMINATOR));
```

3.51 CVF-51

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Recommendation The common practice is to explicitly check if the newValue != address(0).
Client Comment Added the check for governor and admin, as the contract execeedes the size limit if add the check for all. As long as the governer and admin are not address(0), the rest can be modified.

Listing 51:

```
governor = _governor;

admin = _admin;

guardian = _guardian;

dex = _dex;

rewards = _rewards;

gmxRouter = _gmxRouter;
gmxVault = _gmxVault;
```

3.52 CVF-52

- **Severity** Minor
- **Category** Procedural
- **Status** Info
- **Source** PrincipalProtectedVault.sol

Description These events are emitted even if nothing actually changed.
Client Comment Prefer to emit events even if nothing changed, as they are notifications to inform that the functions have been called.

Listing 52:

```
457 emit GovernanceSet(governor);

463 emit AdminSet(admin);

468 emit GuardianSet(guardian);

486 emit LeverageSet(leverage);

492 emit isLongSet(isLong);

497 emit RewardsSet(rewards);

514 emit MaxCollateralMultiplierSet(maxCollateralMultiplier);

529 emit KeeperAdded(_keeper);

534 emit KeeperRemoved(_keeper);

539 emit VaultRegistered(fromVault, toVault);

544 emit VaultRevoked(fromVault, toVault);
```

3.53 CVF-53

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** PrincipalProtectedVault.sol

Recommendation These functions should emit an event.

Listing 53:

```
472 dex = _dex;

508 slip = _slip;
```

3.54 CVF-54

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** PrincipalProtectedVault.sol

Recommendation Limits should be named constants.
Client Comment Prefer to keep them as raw numbers, as those constants are only used once, using the raw value has a better readability.

Listing 54:

```
require(_performanceFee < 5000 && _managementFee < 500, "!too-much");

require(_leverage >= 1 && _leverage <= 50, "!leverage");

require(_maxCollateralMultiplier >= 1 && _maxCollateralMultiplier <= 50, "!maxCollateralMultiplier");
```

3.55 CVF-55

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** GlpVault.sol

Description This import is not used.
Recommendation Consider removing it.

Listing 55:

```
import "../interfaces/IVovoVault.sol";
```

3.56 CVF-56

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** GlpVault.sol

Description We didn't review these files.
Client Comment Noted.

Listing 56:

```
import "../interfaces/gmx/IRewardTracker.sol";
import "../interfaces/gmx/IRewardRouter.sol";
import "../interfaces/gmx/IGlpManager.sol";
import "../interfaces/gmx/IStakedGlp.sol";
import "../interfaces/gmx/IRouter.sol";
import "../interfaces/gmx/IVault.sol";
import "../interfaces/gmx/IRewardTracker.sol";
```

3.57 CVF-57

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** GlpVault.sol

Recommendation Consider using Upgradeable verions of all contracts https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/token/ERC20/utils/SafeERC20Upgradeable.sol
Client Comment Fixed. Also do the same for PrincipalProtectedVault.sol and UniERC20.sol.

Listing 57:

```
using SafeERC20 for IERC20;
```

3.58 CVF-58

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** GlpVault.sol

Recommendation Consider using Upgradeable verions of all contracts https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/utils/AddressUpgradeable.sol
Client Comment Fixed. Also do the same for PrincipalProtectedVault.sol and UniERC20.sol.

Listing 58:

```
using Address for address;
```

3.59 CVF-59

- **Severity** Minor
- **Category** Bad naming
- **Status** Info
- **Source** GlpVault.sol

Recommendation Constant name should be uppercase, you can use = "0x..." with no types conversion.
Client Comment Makes sense, but prefer to keep it the same as our already deployed contract to avoid any potential issues if we want to do an upgrade.

Listing 59:

```
address public constant usdc = address(0xFF970A61A04b1cA14834A43f5dE4533eBDDB5CC8);

address public constant weth = address(0x82aF49447D8a07e3bd95BD0d56f35241523fBab1);

address public constant glp = address(0x4277f8F2c384827B5273592FF7CeBd9f2C1ac258);

address public constant glpManager = address(0x321F653eED006AD1C29D174e17d96351BDe22649);

address public constant fsGLP = address(0x1aDDD80E6039594eE970E5872D247bf0414C8903);

address public constant stakedGlp = address(0x2F546AD4eDD93B956C8999Be404cdCAFde3E89AE);

address public constant rewardRouter = address(0xA906F338CB21815cBc4Bc87ace9e68c87eF8d8F1);

address public constant feeGlpTracker = address(0x4e971a87900b931fF39d1Aad67697F49835400b6);

address public constant feeGmxTracker = address(0xd2D1162512F927a7e282Ef43a362659E4F2a728F);
```

3.60 CVF-60

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** GlpVault.sol

Description Hardcoding mainnet addresses is a bad practice as it makes it harder to test smart contracts.
Recommendation Consider passing the addresses as constructor arguments and storing in immutable variables.
Client Comment Noted. The reason we did not pass it into initalized function is the function parameters is already too many and it will lead to stack too deep error, though there might be other workaround. Our integration tests are running on mainnet fork, so there are not any testing issues.

Listing 60:

```
address public constant usdc = address(0xFF970A61A04b1cA14834A43f5dE4533eBDDB5CC8);

address public constant weth = address(0x82aF49447D8a07e3bd95BD0d56f35241523fBab1);

address public constant glp = address(0x4277f8F2c384827B5273592FF7CeBd9f2C1ac258);

address public constant glpManager = address(0x321F653eED006AD1C29D174e17d96351BDe22649);

address public constant fsGLP = address(0x1aDDD80E6039594eE970E5872D247bf0414C8903);

address public constant stakedGlp = address(0x2F546AD4eDD93B956C8999Be404cdCAFde3E89AE);

address public constant rewardRouter = address(0xA906F338CB21815cBc4Bc87ace9e68c87eF8d8F1);

address public constant feeGlpTracker = address(0x4e971a87900b931fF39d1Aad67697F49835400b6);

address public constant feeGmxTracker = address(0xd2D1162512F927a7e282Ef43a362659E4F2a728F);

    gmxPositionManager = address(0x87a4088Bd721F83b6c2E5102e2FA47022Cb1c831);
    gmxRouter = address(0xaBBc5F99639c9B6bCb58544ddf04EFA6802F4064);
    gmxVault = address(0x489ee077994B6658eAfA855C308275EAd8097C4A);
```

3.61 CVF-61

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** GlpVault.sol

Recommendation The type of this constant should be “IERC20” or “IWETH9”.
Client Comment Same comment as row 6.

Listing 61:

```
address public constant weth = address(0
  ↪ x82aF49447D8a07e3bd95BD0d56f35241523fBab1);
```

3.62 CVF-62

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** GlpVault.sol

Recommendation The type of this variable should be “IERC20”.
Client Comment Same comment as row 6.

Listing 62:

```
address public constant glp = address(0
  ↪ x4277f8F2c384827B5273592FF7CeBd9f2C1ac258);
```

3.63 CVF-63

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** GlpVault.sol

Recommendation The type of this constant should be “IGlpManager“.
Client Comment Same comment as row 6.

Listing 63:

```
address public constant glpManager = address(0
  ↪ x321F653eED006AD1C29D174e17d96351BDe22649);
```

3.64 CVF-64

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** GlpVault.sol

Recommendation The type of this constant should be “IStakedGlp”.
Client Comment Same comment as row 6.

Listing 64:

```
address public constant stakedGlp = address(0x2F546AD4eDD93B956C8999Be404cdCAFde3E89AE);
```

3.65 CVF-65

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** GlpVault.sol

Recommendation The type of these constants should be “IRewardTracker”.
Client Comment Same comment as row 6.

Listing 65:

```
address public constant feeGlpTracker = address(0x4e971a87900b931fF39d1Aad67697F49835400b6);

address public constant feeGmxTracker = address(0xd2D1162512F927a7e282Ef43a362659E4F2a728F);
```

3.66 CVF-66

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** GlpVault.sol

Recommendation The type of this variable should be "IERC20".
Client Comment Same comment as row 6.

Listing 66:

```
address public underlying; // underlying token of the leverage position
```

3.67 CVF-67

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** GlpVault.sol

Recommendation It's possible to do a storage packing optimisations https://docs.soliditylang.org/en/v0.8.14/internals/layout_in_storage.html You can save some gas on storing the variables in the storage if you list them in a way allowing the compiler to pack several variables in a row to the one bytes32 storage.
Client Comment Noted with thanks. Since these variables are rarely updated, the gas saving benefit is not big. Would prefer to keep it as it is.

Listing 67:

```
address public constant usdc = address(0
    ↪ xFF970A61A04b1cA14834A43f5dE4533eBDDB5CC8);
// weth token address
address public constant weth = address(0
    ↪ x82aF49447D8a07e3bd95BD0d56f35241523fBab1);
// glp token address
address public constant glp = address(0
    ↪ x4277f8F2c384827B5273592FF7CeBd9f2C1ac258);
// glpManager address
address public constant glpManager = address(0
    ↪ x321F653eED006AD1C29D174e17d96351BDe22649);
// fsGLP token address
address public constant fsGLP = address(0
    ↪ x1aDDD80E6039594eE970E5872D247bf0414C8903);
// staked Glp address
address public constant stakedGlp = address(0
    ↪ x2F546AD4eDD93B956C8999Be404cdCAFde3E89AE);
// glp reward router address
address public constant rewardRouter = address(0
    ↪ xA906F338CB21815cBc4Bc87ace9e68c87eF8d8F1);
// glp fee reward tracker address
address public constant feeGlpTracker = address(0
    ↪ x4e971a87900b931fF39d1Aad67697F49835400b6);
// gmx fee reward tracker address
address public constant feeGmxTracker = address(0
    ↪ xd2D1162512F927a7e282Ef43a362659E4F2a728F);

uint256 public constant FEE_DENOMINATOR = 10000;

address public underlying; // underlying token of the leverage
    ↪ position
(...)
```

3.68 CVF-68

- **Severity** Minor
- **Category** Documentation
- **Status** Fixed
- **Source** GlpVault.sol

Description The number format of this variable is unclear.
Recommendation Consider documenting.

Listing 68:

```
uint256 public maxCollateralMultiplier;

uint256 public leverage;
```

3.69 CVF-69

- **Severity** Minor
- **Category** Bad naming
- **Status** Info
- **Source** GlpVault.sol

Description The semantics of this variable is unclear.
Recommendation Consider renaming to “maxDeposit”.
Client Comment Since a meaning of "cap" is "upper limit", it means the upper limit of the vault here. Think it's quite clear and hope to keep it as it is, to be aligned with Princpal Protected Vault and our frontend.

Listing 69:

```
uint256 public cap;
```

3.70 CVF-70

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** GlpVault.sol

Description This variable is never read.
Recommendation Consider removing it.

Listing 70:

```
uint256 public underlyingBase;
```

3.71 CVF-71

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** GlpVault.sol

Description The value of this variable stores the minimum poke interval rather than the exact interval.
Recommendation Consider renaming to "minPokeInterval".
Client Comment Makes sense, but prefer to keep the same name as Princpipal Protected Vault, also aligned with the current deployed contracts.

Listing 71:

```
uint256 public pokeInterval;
```

3.72 CVF-72

- **Severity** Minor
- **Status** Info
- **Category** Bad naming
- **Source** GlpVault.sol

Description This variable not only affects withdrawals but also deposits.
Recommendation Consider renaming it.
Client Comment This variable does only mean withdraw interval, since the deposit is allowed after withdraw interval has finished.

Listing 72:

```
uint256 public withdrawInterval;
```

3.73 CVF-73

- **Severity** Minor
- **Status** Fixed
- **Category** Documentation
- **Source** GlpVault.sol

Description The semantics of this variale is unclear.
Recommendation Consider documenting.

Listing 73:

```
uint256 public currentTokenReward;
```

3.74 CVF-74

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** GlpVault.sol

Description Fractional leverages are not supported, i.e. one may set leverage of of 2 or 3, but not 2.5.
Recommendation Consider adding support for fractional leverages.
Client Comment As there is no use case for fractional leverage for the vault, we prefer to keep it as it is.

Listing 74:

```
uint256 public leverage;
```

3.75 CVF-75

- **Severity** Minor
- **Status** Info
- **Category** Procedural
- **Source** GlpVault.sol

Recommendation It make sense to split side-logic of the contract (like keeping admin, guardian, etc) to parent contract and keep only core-logic here. Also you will be able to reuse it in another contract.
Client Comment Makes sense, but prefer to keep it simple and avoid handling the upgradeability logic from another contract.

Listing 75:

```
address public governor;
```

3.76 CVF-76

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** GlpVault.sol

Recommendation The type of these variables should be "IRouter".
Client Comment Same comment as row 6.

Listing 76:

```
address public gmxPositionManager;
address public gmxRouter;
```

3.77 CVF-77

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** GlpVault.sol

Recommendation The type of this variable should be "IVault".
Client Comment Same comment as row 6.

Listing 77:

```
address public gmxVault;
```

3.78 CVF-78

- **Severity** Minor
- **Category** Bad naming
- **Status** Info
- **Source** GlpVault.sol

Recommendation Events are usually named via nouns, such as "Deposit", "GlpDeposit" etc.
Client Comment Prefer to keep the event names so that it's compatible with our current subgraph.

Listing 78:

```
event Deposited(address depositor, address account, uint256
    ↪ shares, uint256 glpAmount, address tokenIn, uint256
    ↪ tokenInAmount);
event DepositedGlp(address depositor, address account, uint256
    ↪ shares, uint256 glpAmount);
event Poked(uint256 tokenReward, uint256 glpAmount, uint256
    ↪ pricePerShare, uint256 fee);

event GovernanceSet(address governor);
event AdminSet(address admin);
event GuardianSet(address guardian);
event FeeSet(uint256 performanceFee, uint256 withdrawalFee);
event LeverageSet(uint256 leverage);
event isLongSet(bool isLong);
event GmxContractsSet(address gmxPositionManager, address
    ↪ gmxRouter, address gmxVault);
event ParametersSet(bool isDepositEnabled, uint256
    ↪ _maxCollateralMultiplier, uint256 cap, uint256
    ↪ pokeInterval, uint256 withdrawInterval, bool isKeeperOnly,
    ↪  bool isFreeWithdraw, address rewards);
event KeeperSet(address keeper, bool isActive);
event VaultSet(address fromVault, address toVault, bool isActive
    ↪ );
```

3.79 CVF-79

- **Severity** Minor
- **Category** Procedural
- **Status** Info
- **Source** GlpVault.sol

Recommendation All address parameters should be indexed.
Client Comment Makes sense, but since we already have contracts in production, prefer to keep the abi the same for future contracts so that they are compatible with the current subgraph. Would prefer to keep them as it is. Also, adding indexed causes the contract to exceed its size limit.

Listing 79:

```
event Deposited(address depositor, address account, uint256
    ↪ shares, uint256 glpAmount, address tokenIn, uint256
    ↪ tokenInAmount);
event DepositedGlp(address depositor, address account, uint256
    ↪ shares, uint256 glpAmount);

event CollectedRewardByKeeper(address keeper, uint256
    ↪ tokenReward, uint256 glpAmount);
event ClosedTradeByKeeper(address keeper, uint256 earning,
    ↪ uint256 glpAmount);
event OpenPosition(address underlying, uint256 underlyingPrice,
    ↪ uint256 wethPrice, uint256 sizeDelta, bool isLong, uint256
    ↪  collateralAmount);
event ClosePosition(address underlying, uint256 underlyingPrice,
    ↪  uint256 vaultTokenPrice,uint256 sizeDelta, bool isLong,
    ↪ uint256 collateralAmount, uint256 fee);
event Withdraw(address account, uint256 shares, uint256
    ↪ glpAmount, address tokenOut, uint256 tokenOutAmount);
event WithdrawGlp(address account, uint256 shares, uint256
    ↪ glpAmount);
event WithdrawToVault(address owner, uint256 shares, uint256
    ↪ glpAmount, address vault, uint256 receivedShares);
event GovernanceSet(address governor);
event AdminSet(address admin);
event GuardianSet(address guardian);

event GmxContractsSet(address gmxPositionManager, address
    ↪ gmxRouter, address gmxVault);
event ParametersSet(bool isDepositEnabled, uint256
    ↪ _maxCollateralMultiplier, uint256 cap, uint256
    ↪ pokeInterval, uint256 withdrawInterval, bool isKeeperOnly,
    ↪  bool isFreeWithdraw, address rewards);
event KeeperSet(address keeper, bool isActive);
event VaultSet(address fromVault, address toVault, bool isActive
    ↪ );
```

3.80 CVF-80

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** GlpVault.sol

Recommendation The type of the token parameters should be "IERC20".
Client Comment Same comment as row 6.

Listing 80:

```
87 event Deposited(address depositor, address account, uint256
    ↪ shares, uint256 glpAmount, address tokenIn, uint256
    ↪ tokenInAmount);

94 event Withdraw(address account, uint256 shares, uint256
    ↪ glpAmount, address tokenOut, uint256 tokenOutAmount);
```

3.81 CVF-81

- **Severity** Minor
- **Status** Info
- **Category** Bad datatype
- **Source** GlpVault.sol

Recommendation The type of this argument should be "IERC20".
Client Comment Same comment as row 6.

Listing 81:

```
112 address _underlying,
```

3.82 CVF-82

- **Severity** Minor
- **Status** Fixed
- **Category** Flaw
- **Source** GlpVault.sol

Description There is no range check for this argument, while in "setLeverage" the leverage is required to be between 1 and 50.
Recommendation Consider adding an appropriate check.

Listing 82:

```
114 uint256 _leverage,
```

3.83 CVF-83

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** GlpVault.sol

Recommendation The common practice is to explicitly check if the newValue != address(0)
Client Comment Prefer to skip the check to reduce contract size, and we will take caution when deploying contracts.

Listing 83:

```
122 underlying = _underlying;
    rewards = _rewards;

472 governor = _governor;

478 admin = _admin;

483 guardian = _guardian;

508 gmxPositionManager = _gmxPositionManager;
    gmxRouter = _gmxRouter;
510 gmxVault = _gmxVault;
```

3.84 CVF-84

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** GlpVault.sol

Recommendation The common practice is to explicitly check if the newValue != address(0)
Client Comment Are you suggesting to name the constants? Prefer to keep the raw value here for better readability as the value is only used once.

Listing 84:

```
129 pokeInterval = 7 days;
130 withdrawInterval = 1 days;

141 managementFee = 200;
    performanceFee = 1000;
```

3.85 CVF-85

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** GlpVault.sol

Recommendation These values should be named constants.
Client Comment The recommendation does not seem to be relevant.

Listing 85:

```
keepers[msg.sender] = true;
```

3.86 CVF-86

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** GlpVault.sol

Recommendation The type of the token argument should be “IERC20”.
Client Comment Same comment as row 6.

Listing 86:

```
function deposit(address tokenIn, uint256 tokenInAmount, uint256 minGlp) external payable returns(uint256) {

function depositFor(address tokenIn, uint256 tokenInAmount, uint256 minGlp, address account) public whenNotPaused payable nonReentrant returns(uint256) {

function mintAndStakeGlp(address tokenIn, uint256 tokenInAmount, uint256 minGlp) private returns(uint256 glpAmount) {

function withdraw(address tokenOut, uint256 shares, uint256 minOut) external whenNotPaused returns(uint256 tokenOutAmount) {
```

3.87 CVF-87

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Fixed
- **Source** GlpVault.sol

Recommendation Here ">=" would be more logical.

Listing 87:

```
180 require(block.timestamp > lastPokeTime.add(withdrawInterval), "!
      ↪ deposit-time");

221 require(block.timestamp > lastPokeTime.add(withdrawInterval), "!
      ↪ deposit-time");
```

3.88 CVF-88

- **Severity** Moderate
- **Category** Unclear behavior
- **Status** Fixed
- **Source** GlpVault.sol

Description In case "pokeInterval" is less than "withdrawInterval", it is possible to prevent deposits by calling "poke" often enough.
Recommendation Consider explicitly requiring "pokeInterval" to be significantly larger than "withdrawInterval".
Client Comment Fixed. Added "require _pokeInterval > _withdrawInterval.mul(2)" for setParameters function

Listing 88:

```
180 require(block.timestamp > lastPokeTime.add(withdrawInterval), "!
      ↪ deposit-time");

221 require(block.timestamp > lastPokeTime.add(withdrawInterval), "!
      ↪ deposit-time");
```

3.89 CVF-89

- **Severity** Moderate
- **Category** Unclear behavior
- **Status** Fixed
- **Source** GlpVault.sol

Description In case "tokenIn" is zero, i.e. plain ether is being deposited, "_after" may never e larger than "_before", thus this line will either set "tokenInAmount" to zero or revert.
Recommendation Consider leaving "tokenInAmount" unchanged here in case "tokenIn" is zero.

Listing 89:

```
185 tokenInAmount = _after.sub(_before);
```

3.90 CVF-90

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** GlpVault.sol

Description This assignment is redundant, as the value assigned here will be overwritten before being read.
Recommendation Consider removing this assignment.
Client Comment Fixed. Also fixed for "shares = 0".

Listing 90:

```
uint256 glpAmount = 0;
```

3.91 CVF-91

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Fixed
- **Source** GlpVault.sol

Description Here “msg.value“ shouldn’t e used, as the “uniTransferFromSenderToThis” function called earlier could send part of “msg.value” back to the caller.
Recommendation Consider using instead .the original “tokenInAmount” as passed to the function.

Listing 91:

```
glpAmount = IRewardRouter(rewardRouter).mintAndStakeGlpETH{value
  ↪ : msg.value}(0, minGlp);
```

3.92 CVF-92

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Fixed
- **Source** GlpVault.sol

Recommendation Using of "<=" would be more intuitive.

Listing 92:

```
require(isDepositEnabled && _pool.add(glpAmount) < cap, "!
  ↪ deposit");
```

3.93 CVF-93

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** GlpVault.sol

Recommendation The "isDepositEnabled" flag should be checked at the very beginning of the function.

Listing 93:

```
require(isDepositEnabled && _pool.add(glpAmount) < cap, "!
    ↪ deposit");

require(isDepositEnabled && _pool.add(glpAmount) < cap, "!
    ↪ deposit");
```

3.94 CVF-94

- **Severity** Minor
- **Category** Overflow/Underflow
- **Status** Fixed
- **Source** GlpVault.sol

Description Phantom overflow is possible here, i.e. situation when the final calculation result would fit into the destination type, while some intermediary calculation overflows.
Recommendation Consider using the "muldiv" function as described here: https://xn–2-umb.com/21/muldiv/ or some other approach that prevents phantom overflow.

Listing 94:

```
  shares = (glpAmount.mul(totalSupply())).div(_pool);

  shares = (glpAmount.mul(totalSupply())).div(_pool);

uint256 fee = balance(false).mul(managementFee).mul(block.timestamp.sub(lastPokeTime)).div(86400*365).div(FEE_DENOMINATOR);

uint256 _sizeDelta = leverage.mul(amount).mul(_wethPrice).div(1e18);

  _fee = _tradeProfit.mul(performanceFee).div(FEE_DENOMINATOR);

uint256 glpAmount = balance(false).mul(shares).div(totalSupply()); // use min vault balance for withdraw

uint256 glpAmount = (balance(false).mul(shares)).div(totalSupply()); // use min vault balance for withdraw

glpAmount = balance(false).mul(shares).div(totalSupply()); // use min vault balance for withdraw

 return balance(isMax).mul(1e18).div(totalSupply());

  positionValueUsd = newPositionValue.mul(positionValueUsd).div(positionValue);

return (glpWethReward.add(gmxWethReward)).mul(_wethPrice).div(getGlpPrice()).div(1e12);

return IGlpManager(glpManager).getAum(true).mul(1e6).div(IERC20(glp).totalSupply());
```

3.95 CVF-95

- **Severity** Minor
- **Status** Info
- **Category** Readability
- **Source** GlpVault.sol

Recommendation Use named args notation for clarity and to reduce the chance to misuse the arguments values. When you explicitly set every argument value by an argument name it decreases the chance to misplace the value.
Client Comment Are you suggesting to use the low level call? Prefer not to use that as think it is generally not a common practice.
https://kushgoyal.com/ethereum-solidity-how-use-call-delegatecall/
Also, we have checked the arguments are passed correctly and it has been working in production smoothly for a while.

3.96 CVF-96

- **Severity** Moderate
- **Status** Info
- **Category** Flaw
- **Source** GlpVault.sol

Description Return value is ignored.
Recommendation Check it or use safeTransferFrom.
Client Comment The stakedGlp does not have the implementation for safeTransferFrom. If you check the "tranfer" and "transferFrom" function, there's no case it will return false, as it either returns true or revert, so do not think the checking is necessary. Also prefer not to add checking because of contract size limit.
https://arbiscan.io/address/0x2F546AD4eDD93B956C8999Be404cdCAFde3E89AE#code

Listing 95:

```
224 IStakedGlp(stakedGlp).transferFrom(msg.sender, address(this),
    ↪ glpAmount);
```

3.97 CVF-97

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** GlpVault.sol

Recommendation This function should return the minted "glpAmount'" and probably some other useful information similar to the information logged in an event.
Client Comment As the returned glpAmount is not useful, prefer to keep it as it is so as not to increase the contract size.

Listing 96:

```
242 function poke() external whenNotPaused nonReentrant {
```

3.98 CVF-98

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** GlpVault.sol

Recommendation Using of “<=” would be more logical.

Listing 97:

```
244 require(lastPokeTime + pokeInterval < block.timestamp, "!poke
    ↪ time");
```

3.99 CVF-99

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** GlpVault.sol

Description This is difficult to read.
Recommendation Consider splitting into a few lines.
Client Comment Added the comment, did not split because of contract size limit.

Listing 98:

```
246 uint256 fee = balance(false).mul(managementFee).mul(block.
    ↪ timestamp.sub(lastPokeTime)).div(86400*365).div(
    ↪ FEE_DENOMINATOR);
```

3.100 CVF-100

- **Severity** Moderate
- **Category** Flaw
- **Status** Info
- **Source** GlpVault.sol

Description Return value is ignored.
Recommendation Check it or use safeTransfer.
Client Comment "The stakedGlp does not have the implementation for safeTransferFrom. If you check the ""tranfer"" and ""transferFrom"" function, there's no case it will return false, as it either returns true or revert, so do not think the checking is necessary. Also prefer not to add checking because contract will exceeds the size limit with that change. https://arbiscan.io/address/0x2F546AD4eDD93B956C8999Be404cdCAFde3E89AE#code"

Listing 99:

```
247 IStakedGlp(stakedGlp).transfer(rewards, fee);
```

3.101 CVF-101

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** GlpVault.sol

Description Checking this conditions for a keeper doesn't make much sense as a keeper may skip this check by calling functions like "collectRewardByKeeper" and "closeTradeByKeeper".
Recommendation Consider skipping this check for a keeper like this: require (keepers[msg.sender] || lastPokeTime + pokeInterval < block.timestamp);
Client Comment Added "disablePokeInterval" flag. Only allow the keeper to call these two functions if that flag is true. We still want to disallow public to call this function unless "isKeeperOnly" is false.

Listing 100:

```
require(lastPokeTime + pokeInterval < block.timestamp, "!poke
    ↪ time");
```

3.102 CVF-102

- **Severity** Moderate
- **Category** Flaw
- **Status** Info
- **Source** GlpVault.sol

Description The returned value is ignored.
Recommendation Consider using safe transfer here.
Client Comment "The stakedGlp does not have the implementation for safeTransferFrom. If you check the ""tranfer"" and ""transferFrom"" function, there's no case it will return false, as it either returns true or revert, so do not think the checking is necessary. Also prefer not to add checking because contract will exceeds the size limit with that change.
https://arbiscan.io/address/0x2F546AD4eDD93B956C8999Be404cdCAFde3E89AE#code"

Listing 101:

```
IStakedGlp(stakedGlp).transfer(rewards, fee);
```

3.103 CVF-103

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** GlpVault.sol

Recommendation The value “86400*365” could be rendered as “365 days”. This value should be a named constant.
Client Comment Changed to 365 days. Prefer to keep the raw value here for better readability as the value is only used once.

Listing 102:

```
uint256 fee = balance(false).mul(managementFee).mul(block.
    ↪ timestamp.sub(lastPokeTime)).div(86400*365).div(
    ↪ FEE_DENOMINATOR);
```

3.104 CVF-104

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Info
- **Source** GlpVault.sol

Recommendation The trade profits should be returned from the “closeTrade” function rather than derived from the WETH balance of the contract.
Client Comment Both ways should be the same unless someone accidentally send WETH to this contract, and it might even be safer in this way to avoid any unkown cases of miscalculating trade profit.

Listing 103:

```
uint256 wethBalance = IERC20(weth).balanceOf(address(this));
```

3.105 CVF-105

- **Severity** Moderate
- **Category** Flaw
- **Status** Fixed
- **Source** GlpVault.sol

Description The returned value is ignored here.
Recommendation Should be assigned to the “glpAmount” variable.

Listing 104:

```
mintAndStakeGlp(weth, wethBalance, 0);
```

3.106 CVF-106

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** GlpVault.sol

Recommendation Safer to place nonReentrant here, for example if keeper is a contract with a public method which calls collectRewardByKeeper inside, then someone can do a reentry attack.
Client Comment Fixed, also added it for withdrawGlp() function.

Listing 105:

```
function collectRewardByKeeper() external returns(uint256 glpAmount) {

function closeTradeByKeeper() external returns(uint256 glpAmount) {

function withdraw(address tokenOut, uint256 shares, uint256 minOut) external whenNotPaused returns(uint256 tokenOutAmount) {
```

3.107 CVF-107

- **Severity** Moderate
- **Category** Flaw
- **Status** Info
- **Source** GlpVault.sol

Description The parameter minOut is set to 0, so sandwich transaction attack is possible. Concretely, someone may manipulate the pool. In the first transaction the attacker executes the swap to shift the price in the pool, then in the 2nd transaction the execution of this code happens and then tin the 3rd transaction the attacker reverts the pool to the original state. When the user's (2nd) transaction is mined, the user swap is executed for a much worse price unfavourite to the user.
Client Comment Note that the GMX uses oracle price to settle the trade with 0 slippage, so it's not possible to execute the sanwitch attack, as the frontrunning won't impact the price.

Listing 106:

```
glpAmount = mintAndStakeGlp(weth, tokenReward, 0);
```

3.108 CVF-108

- **Severity** Minor
- **Category** Readability
- **Status** Info
- **Source** GlpVault.sol

Recommendation It's better to pass the arguments in a named way like call({arg1name: arg1value, ...}); then it's easier to read and decreases the chance to misuse the argument.
Client Comment Are you suggesting to use the low level call? Prefer not to use that as think it is generally not a common practice.
https://kushgoyal.com/ethereum-solidity-how-use-call-delegatecall/
Also, we have checked the arguments are passed correctly and it has been working in production smoothly for a while.

Listing 107:

```
  IRouter(gmxPositionManager).decreasePosition(underlying,
    ↪ underlying, 0, size, isLong, address(this),
    ↪ _underlyingPrice);

uint256 _after = IERC20(weth).balanceOf(address(this));

emit ClosePosition(underlying, _underlyingPrice, _wethPrice,
  ↪ size, isLong, _tradeProfit, _fee);
```

3.109 CVF-109

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** GlpVault.sol

Recommendation There is no reason to use safeApprove in this case. It does not help to prevent attack on ERC-20 protocol. safeApprove method is deprecated, see https://github.com/OpenZeppelin/openzeppelin-contracts/issues/2219 Use a single approve with a success status check or call safeIncreaseAllowance from SafeERC20

Listing 108:

```
IERC20(weth).safeApprove(gmxRouter, 0);
IERC20(weth).safeApprove(gmxRouter, amount);
```

3.110 CVF-110

- **Severity** Minor
- **Category** Bad datatype
- **Status** Fixed
- **Source** GlpVault.sol

Recommendation The precision values such as “1e18”, “1e6”, or “1e12” should be named constants or should be derived from named constants.

Listing 109:

```
304 uint256 _sizeDelta = leverage.mul(amount).mul(_wethPrice).div(1e18);

427  return balance(isMax).mul(1e18).div(totalSupply());

455 return positionValueUsd.mul(1e6).div(getGlpPrice());

462 return (glpWethReward.add(gmxWethReward)).mul(_wethPrice).div(getGlpPrice()).div(1e12);

466 return IGlpManager(glpManager).getAum(true).mul(1e6).div(IERC20(glp).totalSupply());
```

3.111 CVF-111

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** GlpVault.sol

Recommendation Just a single call to “approve‘“ would be more efficient here.

Listing 110:

```
305 IERC20(weth).safeApprove(gmxRouter, 0);
    IERC20(weth).safeApprove(gmxRouter, amount);

361 IERC20(tokenIn).safeApprove(glpManager, 0);
    IERC20(tokenIn).safeApprove(glpManager, tokenInAmount);
```

3.112 CVF-112

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** GlpVault.sol

Description The parameter minOut is set to 0, so sandwich transaction attack is possible. Concretely, someone may manipulate the pool. In the first transaction the attacker executes the swap to shift the price in the pool, then in the 2nd transaction the execution of this code happens and then tin the 3rd transaction the attacker reverts the pool to the original state. When the user's (2nd) transaction is mined, the user swap is executed for a much worse price unfavourite to the user.
Client Comment Note that the GMX uses oracle price to settle the trade with 0 slippage, so it's not possible to execute the sanwitch attack, as the frontrunning won't impact the price.

Listing 111:

```
308 IRouter(gmxPositionManager).increasePosition(_path, underlying,
    ↪ amount, 0, _sizeDelta, isLong, _underlyingPrice);

322   glpAmount = mintAndStakeGlp(weth, wethBalance, 0);
```

3.113 CVF-113

- **Severity** Minor
- **Status** Info
- **Category** Suboptimal
- **Source** GlpVault.sol

Description The code below looks like it is always executed, while actually it is executed only when "size != 0".
Recommendation Consider refactoring like this: if (size != 0) { ... } emit ClosePosition(...);
Client Comment Think current way has better readability, as the tradeProfit and fee are explictly set to in the ClosePosition event when the size == 0.

Listing 112:

```
338 }

438 }
```

3.114 CVF-114

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Info
- **Source** GlpVault.sol

Description The parameter minOut is set to 0, so sandwich transaction attack is possible. Concretely, someone may manipulate the pool. In the first transaction the attacker executes the swap to shift the price in the pool, then in the 2nd transaction the execution of this code happens and then tin the 3rd transaction the attacker reverts the pool to the original state. When the user's (2nd) transaction is mined, the user swap is executed for a much worse price unfavourite to the user.
Client Comment Note that the GMX uses oracle price to settle the trade with 0 slippage, so it's not possible to execute the sanwitch attack, as the frontrunning won't impact the price.

Listing 113:

```
IRouter(gmxPositionManager).decreasePositionAndSwap(path,
    ↪ underlying, 0, size, isLong, address(this),
    ↪ _underlyingPrice, 0);
```

3.115 CVF-115

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Info
- **Source** GlpVault.sol

Description The parameter minOut is set to 0, so sandwich transaction attack is possible. Concretely, someone may manipulate the pool. In the first transaction the attacker executes the swap to shift the price in the pool, then in the 2nd transaction the execution of this code happens and then tin the 3rd transaction the attacker reverts the pool to the original state. When the user's (2nd) transaction is mined, the user swap is executed for a much worse price unfavourite to the user.
Client Comment Note that the GMX uses oracle price to settle the mint with 0 slippage, so it's not possible to execute the sanwitch attack, as the frontrunning won't impact the mint price price.

Listing 114:

```
glpAmount = IRewardRouter(rewardRouter).mintAndStakeGlp(tokenIn,
    ↪ tokenInAmount, 0, minGlp);
```

3.116 CVF-116

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Info
- **Source** GlpVault.sol

Recommendation This function should accept a "minShares" argument to allow user specifying the minimum amount of vault shared to receive.
Client Comment Agreed, if you do not see any major issues, would prefer to keep it as it is to avoid any confusion with the existing deployed contracts in production.

Listing 115:

```
function withdrawToVault(uint256 shares, address vault) external
    ↪ whenNotPaused nonReentrant {
```

3.117 CVF-117

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** GlpVault.sol

Recommendation This check is redundant as it is superseded by the next check.

Listing 116:

```
require(vault != address(0), "!vault");
```

3.118 CVF-118

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** GlpVault.sol

Recommendation The type of the "vault" argument should be "IERC20".
Client Comment Same comment as row 6.

Listing 117:

```
function withdrawToVault(uint256 shares, address vault) external
    ↪ whenNotPaused nonReentrant {
```

3.119 CVF-119

- **Severity** Minor
- **Category** Flaw
- **Status** Info
- **Source** GlpVault.sol

Recommendation You should check the returned success status of the "approve" call. Or use safeIncreaseAllowance from SafeERC20.
Client Comment "The stakedGlp does not have the implementation for safeIncreaseAllowance. If you check the ""approve"" function below, there's no case it will return false, as it either returns true or revert, so do not think the checking is necessary. Also prefer not to add checking because contract will exceeds the size limit with that change.
https://arbiscan.io/address/0x2F546AD4eDD93B956C8999Be404cdCAFde3E89AE#code"

Listing 118:

```
377 IERC20(stakedGlp).approve(vault, glpAmount);
```

3.120 CVF-120

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Info
- **Source** GlpVault.sol

Recommendation This function should accept a “minOut” argument to allow user specifying the minimum Glp amount to receive.
Client Comment Agreed, but since user can easily calculate the exact glpAmount to withdraw before calling the function, prefer to keep it as it is to reduce contract size if you do not see any major issues.

Listing 119:

```
413 function withdrawGlp(uint256 shares) external whenNotPaused
    ↪ returns(uint256 glpAmount) {
```

3.121 CVF-121

- **Severity** Moderate
- **Category** Flaw
- **Status** Info
- **Source** GlpVault.sol

Description Return value is ignored.
Recommendation Check the returned success status or use safeTransfer.
Client Comment The stakedGlp does not have the implementation for safeTransferFrom. If you check the ""tranfer"" and ""transferFrom"" function, there's no case it will return false, as it either returns true or revert, so do not think the checking is necessary? Also prefer not to add checking because contract will exceeds the size limit with that change. https://arbiscan.io/address/0x2F546AD4eDD93B956C8999Be404cdCAFde3E89AE#code

Listing 120:

```
418 IStakedGlp(stakedGlp).transfer(msg.sender, glpAmount);
```

3.122 CVF-122

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** GlpVault.sol

Recommendation It is a good practice to put a comment into an empty block to explain why the block is empty.

Listing 121:

```
422 receive() external payable {}
```

3.123 CVF-123

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** GlpVault.sol

Description The expression "collateralAmount.add(delta)" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 122:

```
445 positionValueUsd = collateralAmount.add(delta) > feeUsd ?
      ↪ collateralAmount.add(delta).sub(feeUsd) : 0;
```

3.124 CVF-124

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** GlpVault.sol

Description The expression “delta.add(feeUsd)” is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 123:

```
447 positionValueUsd = collateralAmount > delta.add(feeUsd) ?
    ↪ collateralAmount.sub(delta).sub(feeUsd) : 0;
```

3.125 CVF-125

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** GlpVault.sol

Description These events are emitted even if nothing actually changed.
Client Comment Prefer to emit events even if nothing changed, as they are notifications to inform that the functions have been called.

Listing 124:

```
473 emit GovernanceSet(governor);

479 emit AdminSet(admin);

484 emit GuardianSet(guardian);

498 emit LeverageSet(leverage);

504 emit isLongSet(isLong);

541 emit KeeperSet(_keeper, _isActive);
```

3.126 CVF-126

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** GlpVault.sol

Recommendation The values used here should be named constants.
Client Comment Prefer to keep others as raw numbers, as those constants are only used once, using the raw value has a better readability.

Listing 125:

```
489 require(_performanceFee < 5000 && _managementFee < 500, "!too-
    ↪ much");

496 require(_leverage >= 1 && _leverage <= 50, "!leverage");

524 require(_maxCollateralMultiplier >= 1 &&
    ↪ _maxCollateralMultiplier <= 50, "!maxCollateralMultiplier
    ↪ ");
```

3.127 CVF-127

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** GlpVault.sol

Recommendation The trade doesn't have to be closed in case isLong == _isLong.
Client Comment Agreed, but the governor would not do that, and even if it's done by mistake, the impact is minimal. Will leave it for now to avoid increasing contract size.

Listing 126:

```
502 closeTrade();
```

3.128 CVF-128

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Info
- **Source** UniERC20.sol

Recommendation Safer to use 0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE, because address(0) is a default value e.g. in mapping and could be misused
Client Comment Agreed, would prefer to keep it as it is to avoid any confusion with the existing deployed contracts in production.

Listing 127:

```
15 function isETH(IERC20 token) internal pure returns (bool) {
```

3.129 CVF-129

- **Severity** Minor
- **Status** Fixed
- **Category** Procedural
- **Source** UniERC20.sol

Recommendation Redundant brackets.

Listing 128:

```
return (address(token) == address(0));
```

3.130 CVF-130

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** UniERC20.sol

Recommendation This could be simplified as: return token == IERC20(0);

Listing 129:

```
return (address(token) == address(0));
```

3.131 CVF-131

- **Severity** Minor
- **Status** Fixed
- **Category** Flaw
- **Source** UniERC20.sol

Recommendation After the usage of this method, using of msg.value inside the code is not correct in a caller method, it should be noted in the comment.

Listing 130:

```
function uniTransferFromSenderToThis(IERC20 token, uint256
    ↪ amount) internal {
```

3.132 CVF-132

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** Uni.sol

Description No minAmountOut argument - sandwitch transaction attack danger. if minAmountOut is set to 0, so someone may manipulate the pool to increase the price, let the swap happen for not-market price and then revert the pool back with the post-transaction, so swap will happen with a bad price.
Client Comment If we want to have the slippage control, the only way is to calculate minExpectedReturn offchain and pass the value to onchain. However, since we want the poke() function to not take any argument and let anyone to be able to trigger the function, we cannot pass the value into the function.
In the short term, since the contract is deployed on Arbitrum with single sequencer without any MEV concerns, it should be fine.
In the long term, we plan to use the flashbot directly to trigger the function, in the same way as what yearn does currently. (line 1542: https://etherscan.io/address/0x9d7c11D1268C8FD831f1b92A304aCcb2aBEbfDe1#code)

Listing 131:

```
function swap(address tokenIn, address tokenOut, uint256
    ↪ amountIn) external;
```

3.133 CVF-133

- **Severity** Minor
- **Category** Bad naming
- **Status** Fixed
- **Source** Uni.sol

Description This interface doesn't represent a Uniswap API nor it has any Uniswap specific.
Recommendation Consider renaming to "ISwapper".

Listing 132:

```
interface Uni {
```

3.134 CVF-134

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Fixed
- **Source** Uni.sol

Recommendation This function should return the actual output amount.

Listing 133:

```
function swap(address tokenIn, address tokenOut, uint256
    ↪ amountIn) external;
```

3.135 CVF-135

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** Univ3Swapper.sol

Recommendation Should inherit IUni interface to exactly show that it implements it.

Listing 134:

```
contract Univ3Swapper {
```

3.136 CVF-136

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** Univ3Swapper.sol

Recommendation Can be immutable to use less gas on reading.

3.137 CVF-137

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** Univ3Swapper.sol

Recommendation The type of this variable should be “ISwapRouter”.
Client Comment Same comment as row 6.

Listing 135:

```
address public swapRouter;
```

3.138 CVF-138

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** Univ3Swapper.sol

Recommendation The type of this variable should be “IWETH9”.
Client Comment Same comment as row 6.

Listing 136:

```
address public weth;
```

3.139 CVF-139

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** Univ3Swapper.sol

Recommendation These variables should be declared as immutable.

Listing 137:

```
address public swapRouter;
address public weth;
```

3.140 CVF-140

- **Severity** Minor
- **Category** Bad datatype
- **Status** Info
- **Source** Univ3Swapper.sol

Recommendation The type of the "_swapRouter" argument should be "ISwapRouter".
Client Comment Same comment as row 6.

Listing 138:

```
constructor(address _swapRouter, address _weth, uint24 _poolFee1, uint24 _poolFee2) {
```

3.141 CVF-141

- **Severity** Minor
- **Category** Suboptimal
- **Status** Info
- **Source** Univ3Swapper.sol

Recommendation The common practice is to explicitly check if the newValue != address(0).
Client Comment Will take caution to not set it to 0 when deploying, prefer to skip this check to save contract storage.

Listing 139:

```
swapRouter = _swapRouter;
weth = _weth;
```

3.142 CVF-142

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Univ3Swapper.sol

Recommendation The type of the "_weth" argument should be "IWETH".
Client Comment Same comment as row 6.

Listing 140:

```
constructor(address _swapRouter, address _weth, uint24 _poolFee1
    ↪ , uint24 _poolFee2) {
```

3.143 CVF-143

- **Severity** Minor
- **Status** Fixed
- **Category** Bad datatype
- **Source** Univ3Swapper.sol

Recommendation The type o the token arguments should be "IERC20".
Client Comment Same comment as row 6.

Listing 141:

```
function swap(address tokenIn, address tokenOut, uint256
    ↪ amountIn) external {
```

3.144 CVF-144

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** Univ3Swapper.sol

Description What if direct pool [tokenIn, poolFee, tokenOut] exists? Can we use it and avoid double conversion?
Client Comment Since this swapping is specially used to convert CRV to USDC on arbitrum, which has to go throught CRV-WETH-USDC. If things change in future, will write another swapping contract for that.

Listing 142:

```
path: abi.encodePacked(tokenIn, poolFee1, weth, poolFee2,
    ↪ tokenOut),
```

3.145 CVF-145

- **Severity** Minor
- **Status** Info
- **Category** Unclear behavior
- **Source** Univ3Swapper.sol

Description The parameter minOut is set to 0, so sandwich transaction attack is possible. Concretely, someone may manipulate the pool. In the first transaction the attacker executes the swap to shift the price in the pool, then in the 2nd transaction the execution of this code happens and then tin the 3rd transaction the attacker reverts the pool to the original state. When the user's (2nd) transaction is mined, the user swap is executed for a much worse price unfavourite to the user.
Client Comment If we want to have the slippage control, the only way is to calculate minExpectedReturn offchain and pass the value to onchain. However, since we want the poke() function to not take any argument and let anyone to be able to trigger the function, we cannot pass the value into the function.
In the short term, since the contract is deployed on Arbitrum with single sequencer without any MEV concerns, it should be fine.
In the long term, we plan to use the flashbot directly to trigger the function, in the same way as what yearn does currently. (line 1542: https://etherscan.io/address/0x9d7c11D1268C8FD831f1b92A304aCcb2aBEbfDe1#code)

Listing 143:

```
36  amountOutMinimum: 0
```

3.146 CVF-146

- **Severity** Minor
- **Status** Fixed
- **Category** Unclear behavior
- **Source** Univ3Swapper.sol

Description The returned value is ignored.
Recommendation Consider returning it from the function.

Listing 144:

```
38  ISwapRouter(swapRouter).exactInput(params);
```

3.147 CVF-147

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** Univ3Swapper.sol

Recommendation These functions should emit an event

Listing 145:

```
poolFee2 = _poolFee2;

governor = _governor;
```

3.148 CVF-148

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** Univ3Swapper.sol

Recommendation The common practice is to explicitly check if the newValue != address(0)

Listing 146:

```
governor = _governor;
```

3.149 CVF-149

- **Severity** Minor
- **Category** Documentation
- **Status** Fixed
- **Source** IVovoVault.sol

Description It is unclear what kind of asset is deposited here.
Recommendation Consider explaining in a documentation comment and/or renaming the function or its argument.

Listing 147:

```
function deposit(uint256 amount) external;
```