GA GUARDIAN

eGMX

Exit Liquidity System

Security Assessment

January 22nd, 2025

Summary

Audit Firm Guardian

Prepared By Owen Thurm, Vladimir Zotov, Roberto Reigada,

Wafflemakr, Kose Dogus, Kiki

Client Firm eGMX

Final Report Date January 22, 2025

Audit Summary

eGMX engaged Guardian to review the security of their escrowed GMX exit liquidity system. From the 18th of November to the 25th of November, a team of 6 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Issues Detected Throughout the engagement 12 High/Critical issues were uncovered and promptly remediated by the eGMX team.

Security Recommendation Given the number of High and Critical issues detected as well as additional code changes made after the main review, Guardian recommends that an independent security review of the protocol at a finalized frozen commit is conducted before deployment.

For a detailed understanding of risk severity, source code vulnerability, and potential attack vectors, refer to the complete audit report below.

🔗 Blockchain network: **Arbitrum**

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

📊 Code coverage & PoC test suite: https://github.com/UmamiDAO/eGMX

Table of Contents

Project Overview

Project Summary

Project Name	eGMX
Language	Solidity
Codebase	https://github.com/GuardianAudits/eGMX-fuzzing
Commit(s)	2ef89ac34d6e712989b978b000a3f3f8a3a63a4c

Audit Summary

Delivery Date	January 22, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	6	0	0	0	0	6
High	6	0	0	0	1	5
Medium	8	0	0	3	0	5
Low	31	0	0	13	0	18

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	🔴 Critical	🟠 High	🟡 Medium
Likelihood: *Medium*	🟠 High	🟡 Medium	🟢 Low
Likelihood: *Low*	🟡 Medium	🟢 Low	🟢 Low

Impact

High Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.

Medium A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.

Low Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.

Medium An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.

Low Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	User Info Not Updated During Matching	Logical Error	Critical	Resolved
C-02	Current Constants.GMX_REWARD_ROUTER Was Disabled By GMX Team	Configuration	Critical	Resolved
C-03	gmxUnlockDate And glpUnlockData Should Be Updated On Every Deposit	Logical Error	Critical	Resolved
C-04	ExitVault Overcollects GMX/GLP Tokens From Users	Logical Error	Critical	Resolved
C-05	Faulty ownerDeposit Mechanism	Logical Error	Critical	Resolved
C-06	Utilizing Vault Transfer To Steal Funds	Gaming	Critical	Resolved
H-01	Match Withdrawal Donation Underflow	DoS	High	Resolved
H-02	ExitVaultEntryPoint.transferFrom Can Be Abused By The Vault Owner To Prevent A User From Withdrawing	DoS	High	Resolved
H-03	Lack Of Incentives For Users To Match Withdrawals	Configuration	High	Resolved
H-04	Users Forfeit Their esGMX, bnGMX And GMX Rewards When Entering The Vault	Configuration	High	Partially Resolved
H-05	Users Tricked To Match Withdrawals	Logical Error	High	Resolved
H-06	Owner's Initial GMX Not Updated When Matching	Logical Error	High	Resolved
M-01	deployVault Calls Can Be Front-run	Griefing	Medium	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
M-02	Precision Loss May Not Allow All Users To Claim	Logical Error	Medium	Resolved
M-03	Users Should Ensure They Hold Only esGMX Before Deploying A Vault To Optimize Gains	Configuration	Medium	Resolved
M-04	ExitVaultEntryPoint Centralizes Governance Power Instead Of Delegating To ExitVault Owners	Centralization	Medium	Acknowledged
M-05	WETH Rewards Stolen From Owner	Logical Error	Medium	Resolved
M-06	Protocol Will Be DoS'd When Private Mode True	DoS	Medium	Acknowledged
M-07	Unfair Reward Distribution On Donation Update	Logical Error	Medium	Acknowledged
M-08	Centralization Issues	Centralization	Medium	Resolved
L-01	RewardRouter Configuration	Configuration	Low	Resolved
L-02	Unnecessary Approval	Optimization	Low	Resolved
L-03	matchWithdrawRequest Will Never Be Fully Matched If Request.donation Is Non-Zero	Logical Error	Low	Resolved
L-04	getMaxVestAmountForVault Function Can Be Simplified	Optimization	Low	Acknowledged
L-05	earlyOwnerExit Calls Can Be Backrun	DoS	Low	Resolved
L-06	Missing Require Check In matchWithdrawRequest Function	Logical Error	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-07	deployVault Calls Can Be Griefed	Logical Error	Low	Acknowledged
L-08	Unused Custom Errors	Optimization	Low	Resolved
L-09	Use Of Debugging Imports	Optimization	Low	Acknowledged
L-10	Unlicensed Smart Contracts	Configuration	Low	Acknowledged
L-11	Missing Pause Modifiers	Validation	Low	Resolved
L-12	Vault Initialization Burns bnGMX Rewards	Logical Error	Low	Acknowledged
L-13	Missing Storage Gaps	Upgradeability	Low	Resolved
L-14	Incorrect Treasury Address Initialization	Deployment	Low	Acknowledged
L-15	Invalid Withdrawal Request Created	Validation	Low	Resolved
L-16	Vault ERC721 Tokens Lost For Invalid Recipients	Logical Error	Low	Resolved
L-17	Typo In “checFullPauseEntryPoint" Function	Typo	Low	Resolved
L-18	claimRewards Can Update Uninitialized Accounts	Validation	Low	Resolved
L-19	Owner Can Not Deposit More GLP	Informative	Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-20	Vaults Tokens Allowed To Be Rescued	Validation	Low	Resolved
L-21	Protocol Fee Can Be 100%	Validation	Low	Acknowledged
L-22	Missing Check In invalidateWithdrawRequest	Validation	Low	Resolved
L-23	GMX Distribution Favors Vault Owner	Rounding	Low	Resolved
L-24	ExitVaultEntryPoint Left With No Admin	Validation	Low	Acknowledged
L-25	Insufficient GMX Approval To Exit Vault	Logical Error	Low	Acknowledged
L-26	Token Rewards Claimed Twice	Gas Optimization	Low	Acknowledged
L-27	Owner Leaves Without Claiming Rewards	Logical Error	Low	Resolved
L-28	Any firstResponder Can Block Other Responders	Access Control	Low	Acknowledged
L-29	Duplicate Address In Constants	Validation	Low	Acknowledged
L-30	Inconsistent Vesting Amounts Due To Fluctuating esGmxToVest Calculation	Logical Error	Low	Acknowledged
L-31	ownerDeposit Can Delete Owner Rewards	Logical Error	Low	Resolved

C-01 | User Info Not Updated During Matching

Category	Severity	Location	Status
Logical Error	● Critical	ExitVault.sol: 248	Resolved

Description

In the matchWithdrawRequest function, the matcher's shares are incremented without first invoking the _depositGMX or _depositGLP functions. This prevents the wethRewardDebt for the matcher from being updated. As a result, the s.userInfo[matcher].glpStream.lastClaim mapping remains unchanged for the matcher, even though his s.userInfo[matcher].glpStream.shares have increased.

In the GMXYieldStrategy contract, when updating the matcherInfo during position matching, the contract only updates the shares field in the gmxStream structure but fails to update the corresponding esGMX vesting information.

This results in several issues such as a permanent DoS or drained funds. Furthermore, The WETH reward calculation can underflow if the share amount decreased since the last claim.

Here we can see the reward calculation:
`uint256 pendingWeth = shares * s.accumulatedTokensPerShare / 1e18 - wethRewardDebt;`

Therefore an underflow DoS occurs if:
`shares * s.accumulatedTokensPerShare / 1e18 < wethRewardDebt`

The WETH reward debt is calculated in the same way:
`wethRewardDebt = shares * s.accumulatedTokensPerShare / 1e18;`

That means if a wethRewardDebt is stored, the shares amount of the user decreases and it tries to calculate the pendingWeth amount again resulting in an underflow DoS. A decrease in shares can happen regularly when users withdraw their tokens.

If a user withdraws half of their tokens a DoS of the whole system occurs for this user until the user's rewards are doubled which could never happen if the owner exits the system in the meantime.

Recommendation

Be sure to update all relevant values such as the GMX shares, esGMX vesting information as well as recalculating the wethRewardDebt when processing matcher info. Use the _depositGMX or _depositGLP function to claim the pending rewards of the matcher, updating his userInfo before increasing his shares.

Additionally, save the accumulated amount instead of the debt in the user struct and then calculate the users share of the stake based on the difference between the current accumulated amount and the saved one.

Resolution

eGMX Team: The issue was resolved in PR#13.

C-02 | Current Constants.GMX_REWARD_ROUTER Was Disabled By GMX Team

Category	Severity	Location	Status
Configuration	● Critical	Constants.sol: 11	Resolved

Description

The current reward router `GMX_REWARD_ROUTER` = 0x159854e14A862Df9E39E1D128b8e5F70B4A3cE9B; present in the `Constants.sol` file has been deprecated by GMX in a recent transaction.

A new reward routernew reward router is now being used. Consequently, any operation on the old reward router will revert, affecting the functionality of contracts that rely on it. The `ExitVault` contract uses the deprecated reward router address for various operations, such as staking and unstaking GMX tokens.

This reliance on the outdated address will cause these operations to fail. Do also note that the [new reward router](https://arbiscan.io/address/0x5E4766F932ce00aA4a1A82d3Da85adf15C5694A1) interacts with a new `RewardTracker` (`ExtendedGmxTracker`) which includes GMX tokens as reward.

Recommendation

Consider updating the `Constants.GMX_REWARD_ROUTER` to the new reward router address. Ensure that the protocol is ready to also support the GMX rewards now given by the `ExtendedGmxTracker`.

Resolution

eGMX Team: Resolved.

C-03 | gmxUnlockDate And glpUnlockData Should Be Updated On Every Deposit

Category	Severity	Location	Status
Logical Error	● Critical	ExitVault.sol: 119, ExitVault.sol: 126	Resolved

Description

In the ExitVault contract, the variables gmxUnlockDate and glpUnlockDate are initialized only during the first deposit to mark the end of the one-year vesting period for the esGMX tokens. Specifically, they are set in the deposit function when these variables are zero:

```
if (s.gmxUnlockDate = 0) {s.gmxUnlockDate = block.timestamp + SECONDS_IN_YEAR;

       emit GMXVestingStarted(msg.sender, s.gmxUnlockDate);} if (s.glpUnlockDate = 0) {s.glpUnlockDate = block.timestamp +
SECONDS_IN_YEAR; emit GLPVestingStarted(msg.sender, s.glpUnlockDate);}
```

However, these unlock dates are not updated on subsequent deposits. This means that if additional deposits are made after the initial deposit, the gmxUnlockDate and glpUnlockDate remain unchanged. As a result, the vault owner can call earlyOwnerExit once the initial unlock dates have passed, even if not all the esGMX tokens from the later deposits have been fully vested into GMX. This allows the owner to exit earlier than expected before the vesting period for the new deposits is complete.

When the owner calls earlyOwnerExit, the vault withdraws all vested tokens and signals the transfer of the vault's account to a specified receiver. However, during this process, the state variables gmxSupply and glpSupply are not updated to reflect the withdrawal. These variables continue to represent the total supply as if the tokens were still in the vault.

As a consequence, the accumulated reward per share variables s.accumulatedGmxWethPerShare and s.accumulatedGlpWethPerShare become inaccurate because they rely on gmxSupply and glpSupply for their calculations. When users attempt to claim rewards or interact with the vault, the contract may attempt to distribute more WETH rewards than what it actually possesses reverting due to insufficient balance.

This inconsistency in the vault's accounting can cause a Denial of Service for users, preventing them from withdrawing their funds or claiming rewards. Finally, it should also be taken into consideration that the vault owner can use earlyOwnerExit to signal an account transfer and move out all the GMX-related funds.

This is only possible if one year has passed since the first user deposit. When the acceptTransfer is executed, it will unstake all the user's funds from the vester contracts and transfer them to the receiver, receiving funds that belong to the depositors.

Recommendation

To prevent the vault owner from exiting earlier than expected and to ensure that all users receive their intended GMX rewards, the gmxUnlockDate and glpUnlockDate should be updated with each new deposit. On the other hand, in the earlyOwnerExit function, update the gmxSupply and glpSupply variables to account for the tokens withdrawn during the exit.

Consider also designing a withdrawal system where only the owner's GMX tokens are transferred to receiver and the remaining can be claimed by stakers, including rewards. Consider creating an escrow contract where the GMX and GLP from users are sent during earlyOwnerExit to avoid being removed during the accountTransfer call by recipient.

Resolution

eGMX Team: The issue was resolved in PR#12.

C-04 | ExitVault Overcollects GMX/GLP Tokens From Users

Category	Severity	Location	Status
Logical Error	● Critical	ExitVault.sol: 364	Resolved

Description

In the ExitVault contract, users are required to deposit GMX tokens to assist in vesting esGMX tokens. The intention is that users contribute the precise amount of GMX needed to vest a corresponding amount of esGMX, based on the vesting mechanics defined by the GMX protocol.

However, the current implementation in the ExitVault miscalculates the required amount of GMX, and therefore, forces the users to deposit more GMX tokens than necessary. The core of the issue lies in the _depositWithGmx function within the ExitVault contract, which determines the amount of esGMX to vest (esGmxToVest) based on the user's GMX deposit. The calculation is as follows:

```
(uint256 maxVestWithGMX, uint256 maxGMXCapacity, uint256 maxGLPCapacity) =
getMaxVestAmountForVault(address(this)); uint256 esGmxToVest = (_amount * maxVestWithGMX) /
maxGMXCapacity;
```

This formula attempts to proportionally assign esGMX to vest based on the amount of GMX deposited. However, it does not accurately reflect the vesting requirements defined in the GMX Vester contract, specifically the getPairAmount function, which calculates the exact amount of GMX required to vest a given amount of esGMX.

In the GMX Vester contract, the getPairAmount function ensures that the amount of GMX needed is proportional to the user's combined average staked amount and the maximum vestable amount of esGMX, following this formula:

```
pairAmount = esAmount * combinedAverageStakedAmount / maxVestableAmount;
```

By not aligning with this formula, the ExitVault overestimates the amount of GMX needed for vesting. This results in users depositing more GMX tokens than required. Users are locking up excess GMX without receiving additional vesting benefits, which is inefficient and can discourage participation.

The overcollection not only misaligns user expectations but also contradicts the protocol's goal of optimizing resource utilization. Users contribute more capital than necessary and the excess GMX remains idle within the vault, providing no additional advantage in terms of vesting esGMX. This same concept also affects the GLP/GLPVester flow.

Recommendation

To resolve this issue, the ExitVault contract should be modified to accurately calculate the required amount of GMX needed to vest the available esGMX, directly reflecting the logic used in the GMX Vester contract's getPairAmount function.

The _depositWithGmx function should be updated to use the correct formula for determining esGmxToVest. This involves calculating the amount of esGMX that can be vested based on the amount of GMX deposited, the combined average staked amount, and the maximum vestable amount, as per the Vester's logic.

Resolution

eGMX Team: Acknowledged.

C-05 | Faulty ownerDeposit Mechanism

Category	Severity	Location	Status
Logical Error	● Critical	ExitVault.sol: 135	Resolved

Description PoC

The function ownerDeposit and its implementation brings many problems for the system. It creates several issues for the both sides of the vault. Firstly and most critical issue is : "Double Counting ownerInitialGMX and ownerInitialGLP in ownerDeposit". During vault initialization, ownerInitialGMX and ownerInitialGLP variables are initialized with the comment: “Store the total balance so that we know how much to transfer back to the owner after a year exits.”

Which means owner will be able to withdraw this amount after the end of vesting. However in the function ownerDeposit these values are incremented regardless if owner makes a new deposit or not. Hence the initial amounts that are transferred via full account transfer will be double counted when owner make the deposit for them. Which means owner can withdraw more than what he actually has and this withdrawal will come with a cost to other users, as the cost will be taken from their share.

Secondly: "Diminished Yield When Owner Deposits". When an owner deposits their shares are reduced in anticipation of them being increased later on in the respective internal deposit function. However, before they are increased the account will get yield based on shares. Since it was reduced early the yield will also be reduced. Leading to a loss of yield for the owner whenever they deposit. And finally, another issue is: "Initial Stakes Of Owner Won't be Counted in Reward Calculation".

During initialization, Owners GMX's and GLP's will be staked automatically but these won't be counted in accumulatedGmxWethPerShare and accumulatedGlpWethPerShare until owner does a ownerDeposit. Hence the yield accrued in between will be given to other depositors. Moreover it won't be possible for owner to deposit more than their share amounts in one step because amount will be withdrawn from share.

Furthermore, it is not even possible to deposit new GLP tokens to the vault by the owner because there is no transfer capability in ownerDeposit function for GLP.

Recommendation

While it is possible to try to implement a fix for every issue individually, the optimal resolution that will fix all the issues would be the to:
• Remove the ownerDeposit function altogether and vest the initial amounts for the owner during initialization.
• Add an owner check to deposit function to update ownerInitialGMX and ownerInitialGLP variables.

These changes will basically remove the difference between owner's deposit for the already staked tokens and owner's new deposits with transferring new tokens and will prevent all issues mentioned above from occurring.

Resolution

eGMX Team: The issue was resolved in PR#15.

C-06 | Utilizing Vault Transfer To Steal Funds

Category	Severity	Location	Status
Gaming	● Critical	ExitVaultEntryPoint.sol: 112	Resolved

Description PoC

Owner can transfer the ownership of the entire vault to any other address via `transferFrom` function in `ExitVaultEntryPoint` contract. However this transfer only changes the owner of the NFT and `owner` value in `GmxAStorage` struct, but does not update any other user related values (`GMXStream` and `GLPStream` values).

This creates an important attack vector and also some unexpected scenarios because of the ambiguity of the action. Firstly, `ownerInitialGMX` and `ownerInitialGLP` are variables that will be transferred back to owner after a year exits. And owner can use ownership transfer mechanism to increment these values and steal funds from other users.

Here is the attack path:
1. Transfer ownership to your second address.
2. `createWithdrawRequest`, so that when it is matched, your address won't be the owner anymore and `ownerInitialGmx` won't be updated.
3. Transfer ownership back to main address.
4. You sold your shares but still holding `ownerInitialGmx` and `ownerInitialGlp` which you can get those amounts back when vesting ends.

Apart from this issue it is also not clear what is this ownership transfer tries to achieve. For example, when token distribution happens, Vested GMX distribution will go to the new owner because the transfer is done directly to the `s.owner`, hence the part that is not donated will go to new owner, but reward accumulation still happens to old owner because owners' reward related variables are not updated.

Another thing is while the shares are not transferred, `ownerInitialGMX/GLP` will belong to new owner now. So while it is not clear what new owner will receive from this ownership transfer, it will also create an attack vector to steal funds from other users.

Recommendation

During ownership transfer, update everything related to both old owner and new owner. This includes claiming rewards for both parties before transfer, updating `GMXStream` and `GLPStream` alongside with `s.owner` change.

Resolution

eGMX Team: Resolved.

H-01 | Match Withdrawal Donation Underflow

Category	Severity	Location	Status
DoS	High	ExitVault.sol: 215	Resolved

Description PoC

In the matchWithdrawRequest the donation amount is not deducted from the withdrawal request when the transaction is a partial withdrawal fill.

In this case the donation amount can be larger than the remaining amount for the withdrawal request and lead to an underflow panic revert on the subsequent withdrawal match. This prevents the user's withdrawal request from being filled after this case has been reached.

Recommendation

Consider refactoring the withdrawal donation computation by using a ratio of the request.donation to the original entire withdrawal request amount to compute the donationPart.

Resolution

eGMX Team: The issue was resolved in PR#13.

H-02 | ExitVaultEntryPoint.transferFrom Can Be Abused By The Vault Owner To Prevent A User From Withdrawing

Category	Severity	Location	Status
DoS	High	ExitVault.sol: 249, ExitVault.sol: 254	Resolved

Description

The transferFrom function in the ExitVaultEntryPoint contract allows the vault owner to transfer ownership of the vault to another user. However, this can be abused to prevent a user from completing their withdrawals. The issue happens when the initial owner fully completes their withdrawal and then transfers their ownership/NFT.

As the initial vault owner had withdrawn all his funds at this point, s.ownerInitialGMX and s.ownerInitialGLP will be zero, leading to an underflow when the new owner attempts to complete the withdrawal of his shares in the matchWithdrawRequest function:

```
function matchWithdrawRequest (address _staker, address _token, uint256 _fillAmount, uint256 _minDonation)

      external checkFullPausedVault {Update the new staker UserInfo. if (_token = TOKEN_GMX) {_depositGMX(0, _staker); Claims rewards stakerInfo.gmxStream.shares = totalShares; matcherInfo.gmxStream.shares = totalShares; if (_staker = s.owner) s.ownerInitialGMX = totalShares; <--------} else {_depositGLP(0, _staker); Claims rewards stakerInfo.glpStream.shares = totalShares; matcherInfo.glpStream.shares = totalShares; if (_staker = s.owner) s.ownerInitialGLP = totalShares; <--------}
```

Recommendation

Under the current implementation the s.ownerInitialGMX and s.ownerInitialGLP state variables do not add any functionality to the contracts except this restriction that can be abused this way. Consider removing them.

Resolution

eGMX Team: The issue was resolved in PR#21.

H-03 | Lack Of Incentives For Users To Match Withdrawals

Category	Severity	Location	Status
Configuration	High	ExitVault.sol: 215	Resolved

Description

In the `ExitVault` contract, withdrawals are fulfilled by other users who match these requests, taking over their staking positions and receiving a portion of their shares as `donation` for helping them exit. This `donation` is used to incentivize others to fulfill their withdrawal requests.

However, once all the esGMX tokens have been fully vested (i.e., the vault reaches `maxVestableAmountGmx` and `maxVestableAmountGlp`), the vault cannot vest additional esGMX tokens.

At this point, new users have no incentive to match withdrawal requests because they can no longer benefit from the vesting rewards as no more esGMX will be converted into GMX.

This creates a scenario where existing users who wish to withdraw are unable to do so unless they find someone willing to match their withdrawal request without the prospect of earning vesting rewards.

As matching a withdrawal request would not yield any benefits to the new participant, it's unlikely anyone would agree to fulfill such requests, even with a `donation`. This situation resembles the old "King of the Ether" game, where users are effectively locked into the contract with no viable means of exiting their positions unless someone else takes their place.

The only option left is to offer increasingly higher donations to entice someone to take over, leading to an impractical and potentially infinite loop without resolution.

Recommendation

Introduce a mechanism that allows users to withdraw their tokens directly when the vault has reached its maximum vesting capacity.

Resolution

eGMX Team: Resolved.

H-04 | Users Forfeit Their esGMX, bnGMX And GMX Rewards When Entering The Vault

Category	Severity	Location	Status
Configuration	High	ExitVault.sol	Partially Resolved

Description

In the ExitVault contract, users can deposit GMX and GLP tokens to participate in the vesting of esGMX tokens and get a portion of them as rewards. While the vault is active, for at least a year, the esGMX, bnGMX, and additional GMX rewards generated from the staked tokens in the different reward trackers are neither claimed nor distributed to the participating users.

Instead, these accumulated rewards are only claimed by the vault owner upon his exit, when the earlyOwnerExit function is executed and followed by accepting the account transfer.

This design results in two significant issues:
• Firstly, users effectively forfeit access to their esGMX, bnGMX and GMX rewards for at least the duration of the vault's operation, despite their GMX/GLP stakes contributing to the generation of these rewards. They do not receive any of these rewards during the active period of the vault.
• Secondly, the accumulated rewards are eventually claimed solely by the vault owner upon exit, rather than being distributed to the users who actually generate them. This creates a scenario where users' contributions lead to benefits that they do not receive, raising concerns about fairness and discouraging users from participating in the vault due to the deprivation of their rewards.

Moreover, this issue fundamentally undermines the benefits that the protocol aims to provide to users. The esGMX, bnGMX and GMX rewards that users miss out on during the vault's operation are likely worth more than the portion of incentives they receive from the esGMX vesting process.

This means that users may actually be worse off by participating in the vault, as they forfeit substantial rewards over the course of a full year, rewards that would likely exceed the benefits gained from the esGMX vesting. Consequently, the vault's current design may inadvertently disadvantage users instead of providing the intended incentives.

Recommendation

Update the vault's reward distribution mechanism to ensure that users receive their fair share of GMX rewards during the vault's active period. Implement a system where the vault regularly claims the accumulated rewards and distributes the GMX among the users according to their staking contributions and the predefined donation and protocol fee percentages. On the other hand, make use of the esGMX rewards accrued to increase the maxVestableAmount in the vault.

Resolution

eGMX Team: Partially Resolved.

H-05 | Users Tricked To Match Withdrawals

Category	Severity	Location	Status
Logical Error	High	ExitVault.sol: 223	Resolved

Description PoC

Stakers can opt to exit the vault by creating a withdrawal request that can me matched by other users, incentivizing them with a donation amount. Malicious users can trick matchers to accept their orders by creating a high donation amount compared to the request amount.

By front-running the `ExitVault.matchWithdrawRequest`, they can invalidate their request and create a new request with a higher amount causing the `donationPart` calculation to drastically drop.

Consider this scenario:
• userA creates a withdrawal request: `amount = 10 donation = 5`
• userB sends a tx to match the request, `_fillAmount = 5 _minDonation=5` = (should receive 5 in donation)
• userA frontruns the tx, invalidates request and creates a new one: `amount = 100 donation=5`
• userB now receives `5 * 5 / (100-5) = 0.26`

Recommendation

Validate the `_minDonation` amount against the `donationPart` instead of the `request.donation`.

Resolution

eGMX Team: The issue was resolved in PR#13.

H-06 | Owner's Initial GMX Not Updated When Matching

Category	Severity	Location	Status
Logical Error	High	ExitVault.sol: 249	Resolved

Description PoC

In the `matchWithdrawRequest` function of the `ExitVault` contract, there is an accounting error when the matcher is the owner. While the owner's `gmxStream.shares` are increased when matching a withdrawal request, their `ownerInitialGMX` value is not updated accordingly.

This creates a mismatch between the owner's actual shares and their recorded initial GMX amount.
The issue occurs because:
1. When matching a withdrawal request, the owner receives additional shares through `matcherInfo.gmxStream.shares = totalShares`
2. However, `ownerInitialGMX` is only decremented when the owner is the staker (`_staker = s.owner`).
There is no corresponding increment when the owner is the matcher
This mismatch has leads to the following issues:
• The owner can only request withdrawals up to their `ownerInitialGMX` amount
• Additional shares obtained through matching become effectively locked
• The owner's withdrawal capacity doesn't reflect their true position
• The discrepancy grows with each matched withdrawal request

The same can be said for the owner's initial GLP.

Recommendation

Add a check in the `matchWithdrawRequest` function to update `ownerInitialGMX` when the matcher is the owner. After updating matcher shares, add: `if (msg.sender = s.owner) s.ownerInitialGMX = totalShares;` Repeat this pattern for GLP.

Resolution

eGMX Team: The issue was resolved in PR#21.

M-01 | deployVault Calls Can Be Front-run

Category	Severity	Location	Status
Griefing	Medium	ExitVaultEntryPoint.sol: 89	Resolved

Description

The `ExitVaultEntryPoint.deployVault` function is designed to deploy a new vault contract using the `CREATE2` opcode, mint an ERC721 token representing ownership of the vault and initialize the newly deployed vault with specific parameters.

However, the function accepts a `_counter` parameter provided by the caller, which must match the current `proxiesCounter` stored in the contract.

When a user attempts to deploy a vault by calling `deployVault` with a specific `_counter`, an attacker or even a legitimate user could submit their own `deployVault` transaction with the same `_counter` before the original one is mined. If this transaction is processed first, it increments the `proxiesCounter` and successfully deploys a vault.

Consequently, when the original user's transaction is executed, the `_counter` no longer matches the updated `proxiesCounter`, causing the transaction to revert with an `InvalidCounter` error. This results in wasted approvals, as users must perform multiple approvals before calling `deployVault`.

Recommendation

Remove the `proxiesCounter = counter` restriction and instead let the user provide any `_counter` . Instead of then storing the `_counter` in the `mapping(uint256 id = address) public vaults;` store the salt: `keccak256(abi.encodePacked(_counter, msg.sender))`

This will require an update in the `vaults` mapping as: `mapping(uint256 id = bytes32) public vaults` Finally, in order to compute the `tokenId` that must be minted to the `owner`, convert the `bytes32` of the salt to an `uint256` .

Resolution

eGMX Team: Resolved.

M-02 | Precision Loss May Not Allow All Users To Claim

Category	Severity	Location	Status
Logical Error	Medium	TokenVester.sol: 116	Resolved

Description

In the ExitVault contract, when users attempt to claim their rewards, the contract may revert with an error indicating that a transfer amount exceeds the contract's GMX token balance by a minimal amount (e.g., 1 wei). This issue is caused by the precision loss in the calculations of reward distributions within the ExitVault contract.

The critical point of failure is in the reward calculation and distribution functions, where the contract updates users' reward debts and calculates pending rewards based on accumulated per share values. For example, when calculating pending rewards:

```
uint256 pendingGmx = esGmxToVest * (block.timestamp - lastClaim) / 365 days;
pendingGmx = claimedGmx + pendingGmx > esGmxToVest * esGmxToVest - claimedGmx :
pendingGmx;
```

And when updating reward debts:

```
s.userInfo[_recipient].gmxStream.wethRewardDebt = shares * s.accumulatedGmxWethPerShare
/ 1e18;
```

These calculations can introduce rounding errors due to integer division. As a result, when the contract attempts to distribute rewards, it may calculate that it needs to transfer slightly more tokens than it actually holds, leading to a revert when calling the transfer function of the GMX token contract.

Recommendation

Before performing a transfer, check the contract's actual token balance and adjust the transfer amount if necessary to avoid attempting to transfer more than the available balance:

```
uint256 contractBalance = IERC20(TOKEN_GMX).balanceOf(address(this));
uint256 transferAmount = reward > contractBalance * contractBalance : reward;
```

Resolution

eGMX Team: Resolved.

M-03 | Users Should Ensure They Hold Only esGMX Before Deploying A Vault To Optimize Gains

Category	Severity	Location	Status
Configuration	Medium	ExitVault.sol	Resolved

Description

To deploy an ExitVault contract the user must perform a full account transfer. This process transfers all tokens and stakes associated with the user's GMX protocol account into the vault, including any staked GMX, esGMX, and other tokens. For instance, if a user has staked 1000 GMX tokens and has earned 100 esGMX tokens that they wish to convert through the vault, initiating the vault deployment and performing the account transfer moves all these assets into the vault.

Once the vault holds the staked GMX and the esGMX tokens, it can immediately use the staked GMX to unlock the esGMX without needing additional participants. This renders the collaborative aspect of the vault pointless, as the vault no longer requires contributions from other users to maximize its vesting capacity.

Moreover, the user's staked GMX and other tokens are now locked within the vault and the user loses direct operational control over them. They cannot claim rewards, adjust their staking positions, or interact with their tokens outside the vault. To regain access and control, the user would need to exit the vault by performing another full account transfer, which can only be done after a year due to the vesting period.

This situation may lead vault creators to inadvertently lock up their staked tokens and lose flexibility in managing their assets, contrary to their intentions. It also negates the primary purpose of the vault system, which is to pool resources from multiple users to collectively unlock esGMX tokens, maximizing gains through collaboration. By having sufficient GMX within the vault to unlock the esGMX independently, the need for other users to participate is eliminated.

Recommendation

Ensure that this behavior is documented and known by the users before deploying a vault. To optimize gains and maintain control over their assets, users should ensure they hold only esGMX tokens before deploying a vault. Prior to initiating the vault deployment and account transfer, users should unstake their GMX tokens and withdraw any other staked assets, leaving only the esGMX tokens in their account.

By doing so, when they perform the account transfer to the vault, only the esGMX tokens are moved, and the vault will not have sufficient GMX to unlock the esGMX on its own. This preserves the need for collaborative participation, allowing multiple users to contribute GMX to the vault to maximize vesting capacity collectively. On the other hand, consider adding the following require statement in the ExitVault.initialize function to prevent the described scenario:

```
uint256 totalGMXGLP = IStakedGmx(TOKEN_STAKED_GMX).depositBalances(address(this), TOKEN_GMX) +
IERC20(TOKEN_STAKED_GLP).balanceOf(address(this)); require(totalGMXGLP < stakedEsGmxBalance);
```

Resolution

eGMX Team: Resolved.

M-04 | ExitVaultEntryPoint Centralizes Governance Power Instead Of Delegating To ExitVault Owners

Category	Severity	Location	Status
Centralization	Medium	ExitVault.sol: 91	Acknowledged

Description

In the current design of the `ExitVaultEntryPoint` and `ExitVault` contracts, the governance power associated with the staked GMX tokens is centralized within the `ExitVaultEntryPoint` contract's treasury rather than being held by the individual `ExitVault` owners.

When users deposit their GMX tokens into an `ExitVault`, the tokens are staked under the contract's address, and the resulting voting power accumulates to the `ExitVaultEntryPoint`'s treasury.

This setup means that all the governance rights derived from these staked tokens are controlled by the `ExitVaultEntryPoint`'s treasury rather than the actual owners of the tokens. Consequently, the vault owners are deprived of their ability to participate in governance decisions proportionally to their stake.

Recommendation

Consider updating the `ExitVault` contract so it delegates the governance power to the individual `ExitVault` owners.

Resolution

eGMX Team: Acknowledged.

M-05 | WETH Rewards Stolen From Owner

Category	Severity	Location	Status
Logical Error	Medium	TokenVester.sol: 142	Resolved

Description PoC

The `ExitVault` will distribute `GMX` and `WETH` rewards to stakers based on the `sbfGMX` and `fGLP` provided. If owner also transfers `sbfGMX` and `fGLP` tokens to vault during deployment, these will also earn `WETH` rewards.

The issue relies on how the the protocol calculates the accumulated rewards per share, as it does not include the `sbfGMX` and `fsGLP` tokens in the vault using `gmxSide` and `glpSide`

There are two major impacts here:
• External users can steal owner's WETH rewards generated by `sbfGMX` and `fGLP` tokens in vault
• Owner is DoS'ed from depositing or claiming, as the `pendingWeth` calculation leads to more WETH than the contract's balance.

Recommendation

Consider using all `sbfGMX` and `fGLP` in the vault for the `esGMX` vesting, by depositing during initialization.

Resolution

eGMX Team: Resolved.

M-06 | Protocol Will Be DoS'd When Private Mode True

Category	Severity	Location	Status
DoS	Medium	Global	Acknowledged

Description

GMX's reward tracker has the ability to set `inPrivateClaimingMode` to true. When this happens claiming will revert as the action is no longer enabled. If this happens any function that interacts with the `_updateVester` will revert.

Recommendation

Document the risk that funds can be DoS'd for periods of time when this variable is set to true.

Resolution

eGMX Team: Acknowledged.

M-07 | Unfair Reward Distribution On Donation Update

Category	Severity	Location	Status
Logical Error	Medium	ExitVault.sol: 333-347	Acknowledged

Description

The `donationPart` can be updated with the `increaseDonation` function. This update will probably lead to an unfair distribution of rewards:
• Current `donationPart` is 10%
• Alice & Bob are the only stakers and deposit the same amt of funds
• `y` amt of rewards are accumulated
• Bob claims his part of the rewards and receives `y/2*0.1%` of the accumulated rewards
• One block later the donation amt is increased to 20%
• Alice claims her part of the rewards and receives `y/2*0.2%` of the accumulated rewards

Now one staker received double the rewards for depositing the same amount of funds over the same period of time. Both should receive 10% of the accumulated rewards up to the point of changing the `donationPart` and 20% after that.

Recommendation

Use an accumulator calculation to distribute the rewards to the users and remove the protocol and owner share directly in the `_updateVester` function.

Resolution

eGMX Team: Acknowledged.

M-08 | Centralization Issues

Category	Severity	Location	Status
Centralization	Medium	ExitVaultEntryPoint.sol	Resolved

Description

The ExitVaultEntryPoint contract grants the admin extensive control, posing significant centralization risks. Specifically, the rescueFunds function allows the admin to withdraw arbitrary tokens from any vault under the contract's management, including user-deposited assets and accrued rewards such as GMX and WETH. This enables the admin to transfer user funds without consent.

Additionally, the admin has the authority to upgrade the implementation of the ExitVault contract via the UpgradeableBeacon. While intended for enhancements and bug fixes, this upgradability feature allows the admin to deploy malicious implementations that could manipulate user balances or drain assets across all existing vaults.

Together, these privileges place excessive trust in a single admin, increasing the risk of unauthorized fund withdrawals and malicious activities that could compromise the entire protocol and its users.

Recommendation

To mitigate centralization risks, it is essential to implement stricter access controls and limitations on the admin's privileges. For the rescueFunds function, restrict withdrawals to specific tokens that are not associated with user deposits or rewards and incorporate additional checks to prevent unauthorized access to user assets.

Regarding contract upgrades, adopt a decentralized governance mechanism, such as a multi-signature wallet or a DAO governance model, to require consensus among multiple trusted parties before any upgrade can be executed.

Resolution

eGMX Team: The issue was resolved in PR#17.

L-01 | RewardRouter Configuration

Category	Severity	Location	Status
Configuration	Low	Global	Resolved

Description

Currently there is no way to update the reward router implementation that is used to signal and accept account transfers in GMX. However in the past 6 months the `RewardRouterV2` contract has been updated several times by the GMX team.

Therefore there should be a seamless way to update the address of the `RewardRouterV2` instance that the protocol interacts with in case this contract is updated by GMX.

Recommendation

Consider adding a function to update the `RewardRouterV2` contract in both the `ExitVaultEntryPoint` and the `ExitVault` Beacon proxies.

Resolution

eGMX Team: Resolved.

L-02 | Unnecessary Approval

Category	Severity	Location	Status
Optimization	Low	ExitVault.sol: 114	Resolved

Description

In the `deposit` function there is an approval made on line 114 no matter whether it is a GMX deposit or not. Furthermore, inside the `_depositWithGmx` function the same approval is made to the `stakedGmx` token, therefore the approval made directly in the `deposit` function can be removed.

Recommendation

Remove the approval invocation in the `deposit` function.

Resolution

eGMX Team: Resolved.

L-03 | matchWithdrawRequest Will Never Be Fully Matched If Request.donation Is Non-Zero

Category	Severity	Location	Status
Logical Error	Low	ExitVault.sol: 235	Resolved

Description

In the ExitVault contract, users can create withdrawal requests specifying the amount they wish to withdraw and an optional donation to incentivize others to fulfill their request. The matchWithdrawRequest function allows another user to fulfill this withdrawal request by substituting their own tokens and taking over the staker's position.

However, there is an issue in the calculation within the matchWithdrawRequest function that prevents users from fully withdrawing their requested amount in a single transaction unless the request.donation is set to zero.

The problem is that due to the way donationPart and totalShares are calculated, there will be a residual amount ("dust") that cannot be withdrawn. This residual amount remains even after attempting to match the full remaining withdrawal request, forcing users to perform additional matches for negligible amounts, which is impractical.

The problematic code snippet is as follows:

```
uint256 donationPart = _fillAmount * request.donation / (request.remaining -
request.donation); uint256 totalShares = _fillAmount + donationPart; if (totalShares >
request.remaining) revert InvalidRatio();
```

When a user tries to fulfill the entire remaining amount (_fillAmount equals request.remaining), the calculation of donationPart becomes flawed because the denominator (request.remaining - request.donation) becomes zero or negative when request.donation is equal to request.remaining. This can cause a division by zero or revert the transaction due to the InvalidRatio check.

Even when the donation is less than request.remaining, integer division and rounding errors can prevent totalShares from exactly matching request.remaining, leaving a non withdrawable residual amount. This issue means that users cannot fully withdraw their tokens in a single matchWithdrawRequest if a donation is specified, as there will always be some small amount left due to the calculation error.

Recommendation

Consider refactoring the withdrawal donation computation by using a ratio of the request.donation to the original entire withdrawal request amount to compute the donationPart.

Resolution

eGMX Team: Resolved.

L-04 | getMaxVestAmountForVault Function Can Be Simplified

Category	Severity	Location	Status
Optimization	Low	ExitVaultStorage.sol: 142	Acknowledged

Description

The `getMaxVestAmountForVault` function calculates the maximum amount of esGMX tokens that can be vested using GMX and GLP within the vault. The current implementation includes conditional logic that compares the `maxVestableAmountGmx` and `maxVestableAmountGlp` with the vault's `esGMXBalance`, selecting the lesser of the two for both GMX and GLP vesting capacities:

```
maxVestWithGMX = maxVestableAmountGmx > esGMXBalance * esGMXBalance : maxVestableAmountGmx;
maxVestWithGLP = maxVestableAmountGlp > esGMXBalance * esGMXBalance : maxVestableAmountGlp;
```

However, the `ExitVault` contract is not claiming the esGMX and bnGMX tokens from their respective reward trackers that would increase the `maxVestableAmountGmx` or `maxVestableAmountGlp` beyond the current `esGMXBalance`.

As there are no mechanisms within the vault that allow for accruing extra esGMX or bnGMX to influence these maximum vesting amounts the comparison between `maxVestableAmount` and `esGMXBalance` is unnecessary because `maxVestableAmountGmx` and `maxVestableAmountGlp` will never exceed `esGMXBalance`.

Simplifying the function by directly assigning the `esGMXBalance` to both `maxVestWithGMX` and `maxVestWithGLP` would make the code clearer and more efficient.

Recommendation

Simplify the `getMaxVestAmountForVault` function by removing the unnecessary conditional checks and directly assigning the `esGMXBalance` to `maxVestWithGMX` and `maxVestWithGLP` :

```
function getMaxVestAmountForVault()

      public view returns (uint256 maxVestWithGMX, uint256 maxVestWithGLP, uint256 gmxForMaxVest, uint256
glpForMaxVest){GmxAStorage storage = _getStorage(); uint256 esGMXBalance =
IERC20(TOKEN_ESGMX).balanceOf(address(this)); uint256 stakedEsGMXBalance =
IStakedGmx(TOKEN_STAKED_GMX).depositBalances(address(this), TOKEN_ESGMX); uint256 totalEsGMXBalance =
esGMXBalance + stakedEsGMXBalance; maxVestWithGMX = totalEsGMXBalance; maxVestWithGLP = totalEsGMXBalance;
gmxForMaxVest = IesTokenVester(s.gmxVester).getPairAmount(address(this), totalEsGMXBalance); glpForMaxVest =
IesTokenVester(s.glpVester).getPairAmount(address(this), totalEsGMXBalance);}
```

Do notice that the `address _vault` parameter was also removed as it was not used.

Resolution

eGMX Team: Acknowledged.

L-05 | earlyOwnerExit Calls Can Be Backrun

Category	Severity	Location	Status
DoS	Low	ExitVault.sol: 304	Resolved

Description

In the ExitVault contract, the vault owner can initiate an exit by calling the earlyOwnerExit function, which withdraws all vested tokens and signals the transfer of the vault's account to a specified receiver.

To complete the exit, the owner (or the new receiver) must call the acceptTransfer function provided by the GMX RewardRouterV2 contract. However, the acceptTransfer function includes the following requirements:

```
require(IERC20(gmxVester).balanceOf(_sender) = 0, "sender has vested tokens");
require(IERC20(glpVester).balanceOf(_sender) = 0, "sender has vested tokens");
```

This means the sender (the vault) must have no active vested tokens so the receiver can successfully accept the transfer. However, after the owner calls earlyOwnerExit, but before the receiver can call RewardRouterV2.acceptTransfer, a malicious actor can backrun this process by calling ExitVault.deposit to deposit GMX or GLP tokens into the vault.

This deposit reinitiates vesting positions within the vault, causing the RewardRouterV2.acceptTransfer call to fail due to the newly vested tokens, as the vault no longer satisfies the zero balance requirement. As a result, the vault owner is forced to call earlyOwnerExit again to withdraw the new vested tokens and re-signal the account transfer.

Recommendation

Implement safeguards to prevent unauthorized deposits during the owner's exit process. One approach is to introduce a state variable, such as isExiting, which is set to true when earlyOwnerExit is called. Update the deposit() function to include a check that prohibits any deposits when isExiting is true:

```
function deposit(address _token, uint256 _amount) external checkFullPausedVault
{require(isExiting, "Deposits are disabled during owner exit"); existing deposit
logic}
```

Resolution

eGMX Team: Resolved.

L-06 | Missing Require Check In matchWithdrawRequest Function

Category	Severity	Location	Status
Logical Error	Low	ExitVault.sol: 215	Resolved

Description

In the `ExitVault` contract, the `matchWithdrawRequest` function enables a user (the matcher) to fulfill a withdrawal request created by another user (the staker), effectively substituting a portion of the staker's position.

However, this function does not check that the `matcher` is not the same as the `staker`.

Recommendation

Add the following `require` check in the `matchWithdrawRequest` function:

```
require(_staker = msg.sender, "Cannot match your own withdrawal request");
```

Resolution

eGMX Team: Resolved.

L-07 | deployVault Calls Can Be Griefed

Category	Severity	Location	Status
Logical Error	Low	ExitVaultEntryPoint.sol: 89	Acknowledged

Description

In the ExitVaultEntryPoint contract, the deployVault function utilizes the CREATE2 opcode to deploy new ExitVault instances at deterministic addresses. This means that the vault's address can be precomputed and is publicly known before the actual deployment. An attacker could front-run the deployVault call and transfer tokens to the precomputed vault address before it is deployed.

When the vault is eventually deployed and attempts to accept a GMX full account transfer via the acceptAccountTransfer function, it interacts with the GMX RewardRouterV2 contract, which includes a _validateReceiver function. This function contains several require statements that check whether the receiver (the newly deployed vault) has zero balances and zero cumulative rewards in various GMX reward trackers and vesting contracts:

```
function _validateReceiver(address _receiver)

       private view {require(IRewardTracker(stakedGmxTracker).averageStakedAmounts(_receiver) = 0,
"stakedGmxTracker.averageStakedAmounts > 0"); require(IRewardTracker(stakedGmxTracker).cumulativeRewards(_receiver) = 0,
"stakedGmxTracker.cumulativeRewards > 0"); require(IRewardTracker(bonusGmxTracker).averageStakedAmounts(_receiver) = 0,
"bonusGmxTracker.averageStakedAmounts > 0"); require(IRewardTracker(bonusGmxTracker).cumulativeRewards(_receiver) = 0,
"bonusGmxTracker.cumulativeRewards > 0"); require(IRewardTracker(feeGmxTracker).averageStakedAmounts(_receiver) = 0,
"feeGmxTracker.averageStakedAmounts > 0"); require(IRewardTracker(feeGmxTracker).cumulativeRewards(_receiver) = 0,
"feeGmxTracker.cumulativeRewards > 0"); require(IVester(gmxVester).transferredAverageStakedAmounts(_receiver) = 0,
"gmxVester.transferredAverageStakedAmounts > 0"); require(IVester(gmxVester).transferredCumulativeRewards(_receiver) = 0,
"gmxVester.transferredCumulativeRewards > 0"); require(IRewardTracker(stakedGlpTracker).averageStakedAmounts(_receiver) = 0,
"stakedGlpTracker.averageStakedAmounts > 0"); require(IRewardTracker(stakedGlpTracker).cumulativeRewards(_receiver) = 0,
"stakedGlpTracker.cumulativeRewards > 0"); require(IRewardTracker(feeGlpTracker).averageStakedAmounts(_receiver) = 0,
"feeGlpTracker.averageStakedAmounts > 0"); require(IRewardTracker(feeGlpTracker).cumulativeRewards(_receiver) = 0,
"feeGlpTracker.cumulativeRewards > 0"); require(IVester(glpVester).transferredAverageStakedAmounts(_receiver) = 0,
"gmxVester.transferredAverageStakedAmounts > 0"); require(IVester(glpVester).transferredCumulativeRewards(_receiver) = 0,
"gmxVester.transferredCumulativeRewards > 0"); require(IERC20(gmxVester).balanceOf(_receiver) = 0, "gmxVester.balance > 0");
require(IERC20(glpVester).balanceOf(_receiver) = 0, "glpVester.balance > 0");}
```

If an attacker has sent any amount of tokens or initiated any staking activities to the precomputed vault address before deployment, these require statements will fail because the vault address will now have non-zero balances or reward amounts. Consequently, the acceptTransfer call will revert, causing the vault deployment process to fail. This vulnerability allows a malicious user to perform a Denial of Service attack on any vault deployment.

Recommendation

This issue is primarily informative as there is no fully effective mitigation against this attack vector due to the deterministic nature of CREATE2 addresses and due to the restrictions given by GMX to accept a full account transfer. However, considering that the contract operates on Arbitrum, where front-running is limited by the network's sequencing and design, the practical risk of such an attack is minimal.

Resolution

eGMX Team: Acknowledged.

L-08 | Unused Custom Errors

Category	Severity	Location	Status
Optimization	Low	ExitVaultEntryPoint.sol, TokenVester.sol	Resolved

Description

In the `ExitVaultEntryPoint` and `TokenVester` contracts, several custom errors are declared but not utilized within the contract logic. Specifically in the`ExitVaultEntryPoint` contract:
• `error AddressZero();`
• `error IllegalUpgrade();`

In the `TokenVester` contract:
• `error WithdrawalExceedsBalance();`

Recommendation

Consider removing the mentioned custom errors.

Resolution

eGMX Team: Resolved.

L-09 | Use Of Debugging Imports

Category	Severity	Location	Status
Optimization	Low	Global	Acknowledged

Description

Some contracts contain import statements for the `console.sol` library from the Forge Standard Library, specifically:

```
import {console} from "forge-std/console.sol";
```

Including `console.sol` is intended for debugging and logging during the development and testing phases. However, retaining these imports in production contracts is unnecessary.

Recommendation

Consider removing all import statements of the `console.sol` library from the contracts before deploying them to a production environment.

Resolution

eGMX Team: Acknowledged.

L-10 | Unlicensed Smart Contracts

Category	Severity	Location	Status
Configuration	Low	Global	Acknowledged

Description

The `ExitVault`, `ExitVaultEntryPoint`, `ExitVaultStorage`, `TokenVester` and `Pause` contracts are currently marked as unlicensed, as indicated by the SPDX license identifier at the top of the file:

```
SPDX-License-Identifier: UNLICENSED
```

Using unlicensed contracts can lead to legal uncertainties and conflicts regarding the usage, modification and distribution rights of the code.

Recommendation

It is recommended to choose and apply an appropriate open-source license to the smart contract. Some options are:
1. MIT License: A permissive license that allows for reuse with minimal restrictions.
2. GNU General Public License (GPL): A copyleft license that ensures derivative works are also open-source.
3. Apache License 2.0: A permissive license that provides an express grant of patent rights from contributors to users.

Resolution

eGMX Team: Acknowledged.

L-11 | Missing Pause Modifiers

Category	Severity	Location	Status
Validation	Low	ExitVault.sol: 135, ExitVaultEntryPoint.sol: 112	Resolved

Description

All functions that are callable by stakers or the owner in the `ExitVault` contract have a `checkFullPausedVault` modifier except the `ownerDeposit` function. The same holds for all external non-admin functions in the `ExitVaultEntryPoint` contract except for the `transferFrom` function.

Recommendation

Consider adding pause modifiers to all functions or following a consistent pause approach like pausing inflows but allowing outflows of the system.

Resolution

eGMX Team: Resolved.

L-12 | Vault Initialization Burns bnGMX Rewards

Category	Severity	Location	Status
Logical Error	Low	ExitVault.sol: 82	Acknowledged

Description

During `ExitVault.initialize`, if owner has any `bnGMX` tokens, these are transferred to the vault during `accceptTransfer`. After full account transfer has been made to vault, the `ExitVault.initialize` function will also unstake any deposited `esGMX` using the router.

The issue relies on `RewardRouterV2._unstakeGmx` as it will burn all `bnGMX` tokens from the vault as a penalty. The owner loses funds that could have been used to get more `sbfGMX` for `esGMX` vesting.

Recommendation

Merely informative issue. While the `RewardRouterV2._unstakeGmx` call would prevent the issue as the bnGMX tokens would not be burnt, it would require some major refactoring to properly account for the vault owner tokens that were left staked.

Resolution

eGMX Team: Acknowledged.

L-13 | Missing Storage Gaps

Category	Severity	Location	Status
Upgradeability	Low	Pause.sol	Resolved

Description

A general best practice is to set a storage gap of 50, and decrement it by the number of variables that are present in the contract. This was not done in the Pause.sol contract that is inherited from `ExitVaultEntryPoint.sol`.

Recommendation

Consider adding storage gaps at the end of the `Pause.sol` contract to ensure future upgrades are safe.

Resolution

eGMX Team: Resolved.

L-14 | Incorrect Treasury Address Initialization

Category	Severity	Location	Status
Deployment	Low	ExitVaultEntryPointScript.s.sol: 62	Acknowledged

Description

The protocol's deployment script sets the treasury address as the foundry script contract itself. There is no admin function in the `ExitVaultEntryPoint.sol` contract to update this treasury address. This is a critical address that will receive all protocol's fees from the deployed vaults, as well as the delegation votes for GMX DAO.

Recommendation

Consider assigning the correct treasury wallet during deployment.

Resolution

eGMX Team: Acknowledged.

L-15 | Invalid Withdrawal Request Created

Category	Severity	Location	Status
Validation	Low	ExitVault.sol: 178	Resolved

Description

Stakers can create withdrawal request to exit the vault early, incentivizing users with a donation. However, the `ExitVault.createWithdrawRequest` does not correctly validate the user's shares against the request amount and donation.

Therefore, some invalid request may occur, making request matching fail:
• `donation = amount`: division by 0 when request is matched
• `donation > amount`: underflow when request is matched
• `amount + donation > shares`: request can't be fully matched

Recommendation

Validate the above conditions so the request amount and donation correctly match the user's shares.

Resolution

eGMX Team: Resolved.

L-16 | Vault ERC721 Tokens Lost For Invalid Recipients

Category	Severity	Location	Status
Logical Error	Low	ExitVaultEntryPoint.sol	Resolved

Description

The `ExitVaultEntryPoint` mints an `ERC721` token to owner when a vault is deployed. Additionally, owner can transfer the vault's ownership using `transferFrom`.

None of these actions contain the safety check `ERC721Utils.checkOnERC721Received` to verify that the recipient can hold ERC721 tokens.

Recommendation

Consider using the `_safeMint` and `safeTransferFrom` instead. Make sure these calls are performed at the end of the function execution to avoid reentrancy issues. If this is an expected behavior, consider documenting it for user to avoid stuck `ERC721` tokens

Resolution

eGMX Team: Resolved.

L-17 | Typo In “checFullPauseEntryPoint" Function

Category	Severity	Location	Status
Typo	Low	Pause.sol: 83	Resolved

Description

There is a typo in the `checFullPauseEntryPoint` function.

Recommendation

Write `check` instead of `chec`.

Resolution

eGMX Team: Resolved.

L-18 | claimRewards Can Update Uninitialized Accounts

Category	Severity	Location	Status
Validation	Low	ExitVault.sol: 161-165	Resolved

Description

The `claimRewards` function can be used to update the own `lastClaim` variable to block.timestamp without owning any shares. This could lead to front-end bugs.

Recommendation

Revert if the caller's account is empty.

Resolution

eGMX Team: Resolved.

L-19 | Owner Can Not Deposit More GLP

Category	Severity	Location	Status
Informative	Low	ExitVault.sol: 107, ExitVault.sol: 135-153	Resolved

Description

The `deposit` function reverts with `UseOwnerDepositFunction` error if it is called by the owner. The `ownerDeposit` can deposit new GMX tokens into the system with the `_stakeGmx` boolean, but it does not allow the owner to add new GLP tokens.

Therefore the owner can't deposit new GLP tokens into the system.

Recommendation

If you allow the owner to deposit new GMX tokens into the system consider also allowing the owner to deposit new GLP tokens.

Resolution

eGMX Team: Resolved.

L-20 | Vaults Tokens Allowed To Be Rescued

Category	Severity	Location	Status
Validation	Low	ExitVaultEntryPoint.sol: 124	Resolved

Description

The `ExitVaultEntryPoint` admin can rescue funds from vaults without any validation. If any user sends tokens to the vault by mistake, then this call will not harm the protocol. In case the token is `GMX`, it can break the rewards accounting in the vault.

Recommendation

Consider validating the token being rescued, it it's GMX token, only allow admin to perform this action after users leave the vault.

Resolution

eGMX Team: Resolved.

L-21 | Protocol Fee Can Be 100%

Category	Severity	Location	Status
Validation	Low	ExitVaultEntryPoint.sol: 117	Acknowledged

Description

Admin is able to change the protocol fee using `ExitVaultEntryPoint.setProtocolFee`. If the fee is set to 100%, then vaults deployed will not be able to distribute GMX rewards to owner or stakers.

Recommendation

Consider lowering the max value of the protocol fee.

Resolution

eGMX Team: Acknowledged.

L-22 | Missing Check In invalidateWithdrawRequest

Category	Severity	Location	Status
Validation	Low	ExitVault.sol: 274-280	Resolved

Description

The `invalidateWithdrawRequest` does not check if the calling user has an active withdraw request, it can be called anytime and emits an event. This could lead to frontend bugs.

Recommendation

Revert if the user does not have an active withdraw request.

Resolution

eGMX Team: Resolved.

L-23 | GMX Distribution Favors Vault Owner

Category	Severity	Location	Status
Rounding	Low	TokenVester.sol: 108	Resolved

Description

Vault's `GMX` rewards are distributed between the protocol, staker and vault' owner. However, the current calculations will round in favor of the owner and not the protocol.

Recommendation

Refactor the calculations so protocol receives the most GMX rewards possible.

Resolution

eGMX Team: Resolved.

L-24 | ExitVaultEntryPoint Left With No Admin

Category	Severity	Location	Status
Validation	Low	Pause.sol: 75	Acknowledged

Description

The `ExitVaultEntryPoint.setFirstResponder` allows an existing `firstResponder` to set the status of a "new" responder. However, if there is only one responder (admin), and his status as `false`, then the contract will be left with no `firstResponder` and no way to add them back again.

Recommendation

Validate that `_firstResponder = msg.sender` or `_firstResponder = admin` to avoid clearing the admin from it's role.

Resolution

eGMX Team: Acknowledged.

L-25 | Insufficient GMX Approval To Exit Vault

Category	Severity	Location	Status
Logical Error	Low	ExitVault.sol: 313	Acknowledged

Description

During `ExitVault.earlyOwnerExit`, the function performs two approvals:

•`ERC20(TOKEN_sbfGMX).approve(_receiver, ERC20(TOKEN_sbfGMX).balanceOf(address(this)));`

•`ERC20(TOKEN_GMX).approve(TOKEN_STAKED_GMX,IStakedGmx(TOKEN_STAKED_GMX).depositBalances(address(this), TOKEN_GMX));`

The first approval is needed for the `RewardsRouter.signalTransfer` call, and the second one is used during the `RewardsRouter.acceptTransfer` call. However, the new `RewardsRouterV2.acceptTransfer` claims `GMX` rewards from the `extendedGmxTracker` and stakes them in the `stakedGmxTracker`.

Then, the router tries to unstake and stake all the `GMX` in the `stakedGmxTracker` but the allowance needed has increased, reverting the `acceptTransfer` call.

Recommendation

During the `ExitVault.earlyOwnerExit`, verify if there are any GMX rewards to claim from the `extendedTracker` and add the value to the `GMX` token approval.

Resolution

eGMX Team: Acknowledged.

L-26 | Token Rewards Claimed Twice

Category	Severity	Location	Status
Gas Optimization	Low	ExitVault.sol: 161	Acknowledged

Description

User can claim rewards from the vault using `ExitVault.claimRewards`. It will claim from the `gxmStream` and/or `glpStream` according to the user's shares.

In case user has deposited in both stream, this function will try to claim rewards twice in `_updateVester`, although the second call won't claim new tokens. Additionally, if the user does not have any deposit in the streams, the function won't do any changes but the transaction succeeds.

Recommendation

Consider refactoring the logic to avoid unnecessary zero reward claims.

Resolution

eGMX Team: Acknowledged.

L-27 | Owner Leaves Without Claiming Rewards

Category	Severity	Location	Status
Logical Error	Low	ExitVault.sol: 304	Resolved

Description

Owners can exit the vault early if there are no user deposits that can lock the vault, or else they will need to wait 1 full year.

In either case, the `ExitVault.earlyOwnerExit` does not claim rewards before leaving, just as the `ExitVault.matchWithdrawRequest` does for stakers. This can lead to issues if the owner later tries to claim.

Recommendation

Execute `_depositGMX(0, msg.sender)` and `_depositGLP(0, msg.sender)` before exiting the vault.

Resolution

eGMX Team: Resolved.

L-28 | Any firstResponder Can Block Other Responders

Category	Severity	Location	Status
Access Control	Low	Pause.sol: 75	Acknowledged

Description

In the `Pause` contract, the `setFirstResponder` function allows any address with the first responder role to grant or revoke the first responder status of any other address. This creates a flat privilege structure where all first responders have equal power to modify the access control system.

This is problematic because:
1. Any first responder can remove other first responders.
2. A compromised first responder account could add malicious addresses as first responders.
3. There's no hierarchical control over who can manage first responder roles.

First responders have significant power in the protocol, including the ability to:
• Pause/unpause the entire protocol.
• Pause/unpause the entry point.
• Pause/unpause specific vaults.

Having no hierarchy in role management creates unnecessary security risks.

Recommendation

Implement an owner or admin role that has exclusive permission to manage first responders.

Resolution

eGMX Team: Acknowledged.

L-29 | Duplicate Address In Constants

Category	Severity	Location	Status
Validation	Low	Constants.sol: 19	Acknowledged

Description

Currently there are two addresses in the Constant contract that have the same address. `GMX_GMX_REWARDS_TRACKER`, `TOKEN_sbfGMX`.

Recommendation

Consider modifying these constants so that the same address does not belong to different constants.

Resolution

eGMX Team: Acknowledged.

L-30 | Inconsistent Vesting Amounts Due To Fluctuating esGmxToVest Calculation

Category	Severity	Location	Status
Logical Error	Low	ExitVault.sol: 364, ExitVault.sol: 395	Acknowledged

Description

In the ExitVault contract, specifically within the _depositWithGmx function, the amount of esGMX that a user is allowed to vest is calculated using the formula:

```
uint256 esGmxToVest = (_amount * maxVestWithGMX) / maxGMXCapacity;
```

Here, _amount represents the user's GMX deposit, while maxVestWithGMX and maxGMXCapacity are dynamic values that fluctuate over time due to changes in the vault's esGMX balance, vesting activities, and reward accruals.

As these variables change, the ratio of maxVestWithGMX to maxGMXCapacity varies, leading to inconsistent outcomes where users depositing the same amount of GMX at different times receive different amounts of esGMX to vest.

This results in an unfair advantage for early users who might receive more esGMX compared to later users for identical deposits.
The same issue is also present in the _depositWithGlp function.

Recommendation

The calculation of esGmxToVest should be adjusted to provide consistent vesting amounts regardless of when users deposit their GMX tokens.

This calculation can be based on the user's proportional share of the total GMX deposited in the vault, aligning esGmxToVest with the user's stake relative to the vault's total deposits.

This would ensure that each user's vesting amount is equitable and independent of fluctuations in maxVestWithGMX and maxGMXCapacity.

Resolution

eGMX Team: Acknowledged.

L-31 | ownerDeposit Can Delete Owner Rewards

Category	Severity	Location	Status
Logical Error	Low	ExitVault.sol: 144-146, 149-151	Resolved

Description

The ownerDeposit function reduces the shares of the owner before calling _depositWithGmx or _depositWithGlp which will increase the shares again. This is done to vest the user's tokens without changing the share amount.

These functions will also calculate the account rewards based on the current amount of shares, send them to the user, and update the last accumulator and claim timestamp afterward.

When the owner deploys the contract and by doing so deposits funds into the contract which accrue WETH rewards, the owner will probably lose rewards when he calls the ownerDeposit function later:
• Owner deploys the contract and deposits funds
• The owner funds accrue WETH rewards
• After one month the owner calls ownerDeposit to vest all of his tokens
• The ownerDeposit function reduces his shares by the full amount and enters the deposit flow. The deposit flow will not accrue any rewards as the owner's shares are currently zero. It also overwrites the owner's lastClaim and wethRewardDebt variables and therefore the system acts as if the owner just deposited the funds and deletes the rewards the owner accrued in the past month.

Recommendation

Call claimRewards at the beginning of the ownerDeposit function.

Resolution

eGMX Team: Resolved.

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Guardian to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Guardian’s position is that each company and individual are responsible for their own due diligence and continuous security. Guardian’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by Guardian is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

Notice that smart contracts deployed on the blockchain are not resistant from internal/external exploit. Notice that active smart contract owner privileges constitute an elevated impact to any smart contract’s safety and security. Therefore, Guardian does not guarantee the explicit security of the audited smart contract, regardless of the verdict.

About Guardian Audits

Founded in 2022 by DeFi experts, Guardian Audits is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian Audits upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits