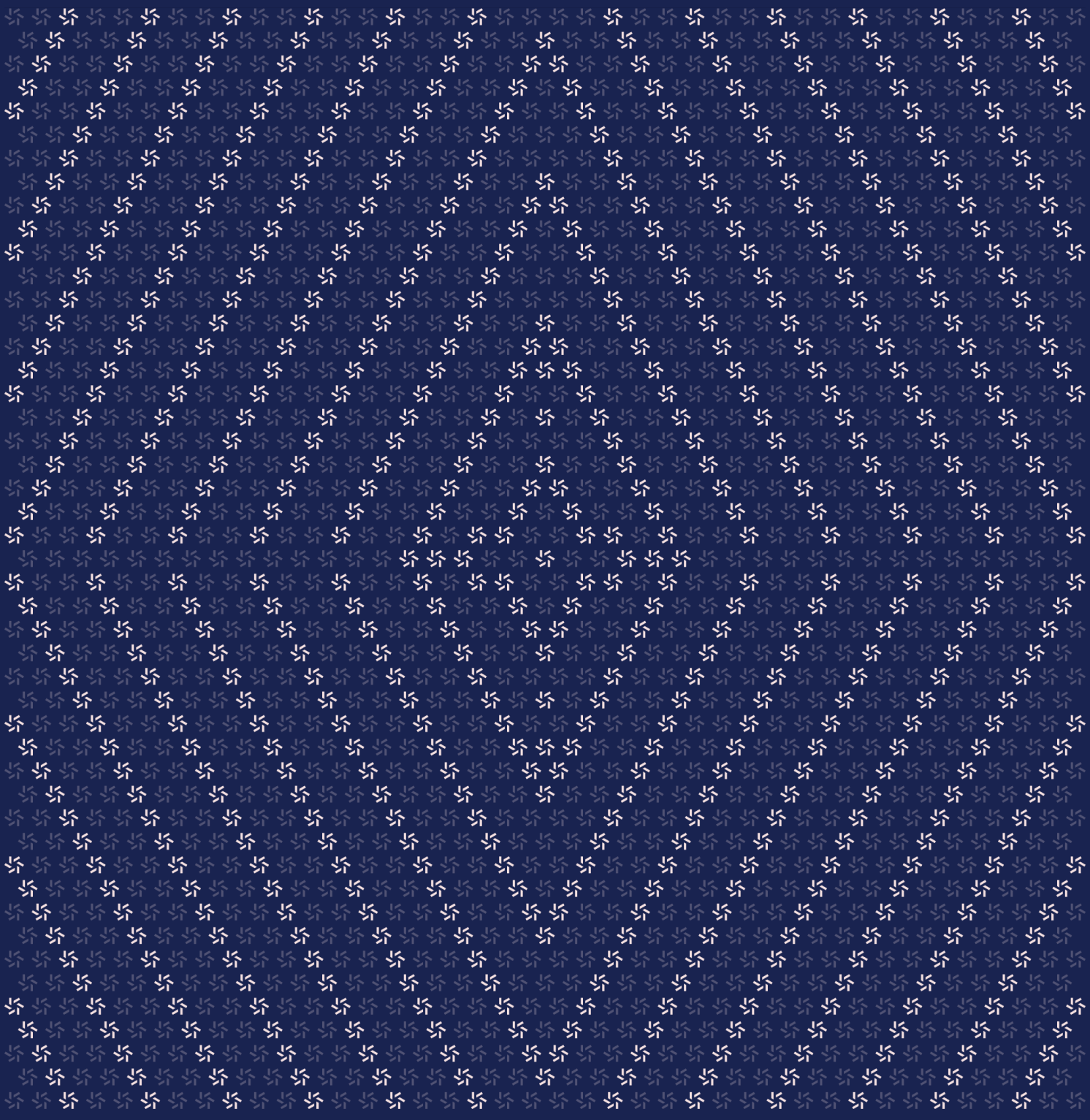
Zellic

Prepared for
Eugene Shen
Plume Network

Prepared by
Dimitri Kamenski
Kuilin Li
Zellic

November 12, 2024

Cultured
Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Plume Network from November 6th to November 7th, 2024. During this engagement, Zellic reviewed Cultured's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Is the removal of the governance/timelock dependency safe?
- Does the change allow for the index asset to be different from the collateral asset?
- Are there any leaks from the liquidity-pool funds?
- Can anyone game long/short positions?
- Do the simplification modifications to the reward contracts cause any unexpected behavior?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- The GMX V1 contracts as of the base commit
- Modifications to noncontract code, including scripts

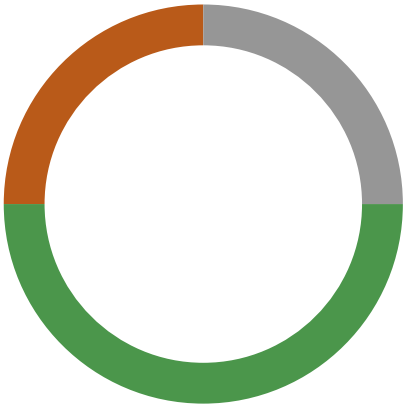
Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Cultured contracts, we discovered four findings. No critical issues were found. One finding was of high impact, two were of low impact, and the remaining finding was informational in nature.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	1
Medium	0
Low	2
Informational	1



2. Introduction

2.1. About Cultured

Plume Network contributed the following description of Cultured:

> Cultured is a framework that allows users to trade on arbitrary data feeds, some of which will correspond very directly to real-world data ("What's the current temperature in NYC?"), and some of which will correspond in a proxied way based on real-time AI analysis of input data from Twitter, Reddit, news, etc ("What's the sentiment on Donald Trump?"). Unlike prediction markets whose price changes purely based on orders on the platform, these indexes update on a minute-by-minute basis so that traders are always on their toes and forced to react.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

Basic coding mistakes. Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.

Business logic errors. Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.

Integration risks. Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.

Code maturity. We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational" finding higher than a "Low" finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients' threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3. Scope

The engagement involved a review of the following targets:

Cultured Contracts

Type	Solidity
Platform	EVM-compatible
Target	cultured-contracts
Repository	https://github.com/cultured-rwa/cultured-contracts ↗
Version	`28e9f694655e92e509b84fa1fe5b3152e8fb7412`

Programs	`core/BasePositionManager.sol`
	`core/PositionUtils.sol`
	`core/Vault.sol`
	`gmx/GLP.sol`
	`staking/RewardRouter.sol`

2.4. Project Overview

Zellic was contracted to perform a security assessment for a total of two person-days. The assessment was conducted by two consultants over the course of one calendar day.

Contact Information

The following project managers were associated with the engagement:

Jacob Goreski
Engagement Manager
jacob@zellic.io ↗

Chad McDonald
Engagement Manager
chad@zellic.io ↗

The following consultants were engaged to conduct the assessment:

Dimitri Kamenski
Engineer
dimitri@zellic.io ↗

Kuilin Li
Engineer
kuilin@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

November 6, 2024	Kick-off call
November 6, 2024	Start of primary review period
November 7, 2024	End of primary review period

3. Detailed Findings

3.1. Long-position–reserved collateral cannot be rebased

Target	Vault.sol		
Category	Business Logic	**Severity**	High
Likelihood	High	**Impact**	High

Description

An important change between the base GMX V1 contracts and this fork is that for long positions, the collateral token no longer has to be the index token:

```
function _validateTokens(address _collateralToken, address _indexToken,
    bool _isLong) private view {
    if (_isLong) {
        // _validate(_collateralToken == _indexToken, 42);
        _validate(whitelistedTokens[_collateralToken], 43);
        _validate(!stableTokens[_collateralToken], 44);
        _validate(!stableTokens[_indexToken], 47);
        return;
    }

    _validate(whitelistedTokens[_collateralToken], 45);
    _validate(stableTokens[_collateralToken], 46);
    _validate(!stableTokens[_indexToken], 47);
    _validate(shortableTokens[_indexToken], 48);
}
```

This change implements the ability to trade price feeds that do not correspond to on-chain assets by having virtual index tokens that refer to price feeds.

However, in the unchanged `increasePosition` code, there is an important reserve requirement:

```
function increasePosition(address _account, address _collateralToken,
    address _indexToken, uint256 _sizeDelta, bool _isLong)
    external override nonReentrant {

    // [...]

    _validateTokens(_collateralToken, _indexToken, _isLong);

    // [...]
```

```
// reserve tokens to pay profits on the position
uint256 reserveDelta = usdToTokenMax(_collateralToken, _sizeDelta);
position.reserveAmount = position.reserveAmount.add(reserveDelta);
_increaseReservedAmount(_collateralToken, reserveDelta);
```

In the above code, `_sizeDelta` is the leveraged size of the position in USD. The function `usdToTokenMax` converts this, rounding up for slippage, into a quantity of the collateral token, which is then reserved. This guarantees that, as long as the reserved assets always equal or exceed the actual assets held by the protocol, the protocol will always be solvent. This is because, for both long and short positions, this reserved amount is the maximum profit that could be sent to the trader at position close.

On the other hand, if the collateral token is not necessarily equal to the index token for long positions, then this reserve amount may be exceeded. Since there is no mechanism to rebase the collateral reserved by an open position or force the closure of positions that are too profitable, this may allow individual positions to permanently expose the contract to an unlimited amount of risk.

Impact

In the base GMX V1 contracts, the maximum profit of a position is reserved when it is opened. For example, if the trader opens a 5x leveraged long ETH position with 1 ETH as collateral, when the position is opened, 5 ETH will be reserved. Then, later, no matter what the price of ETH is, the max payout to the trader is the entire 5 ETH — as the price approaches infinity, the payout asymptotically approaches 5 ETH.

However, since the change to `_validateTokens` decouples the collateral and index assets, now the reserved quantity is different. In the same example, if the trader opens a 5x leveraged long ABC position with 1 ETH as collateral, at open, 5 ETH will still be reserved. But, if ABC doubles in price while ETH remains flat, the value of the position has increased, whereas the value of the collateral remains the same.

Whenever the economic risk of a position is probabilistic, instead of guaranteed, the risk parameters should be able to be rebased on newer market values, and liquidation should be possible if the position falls out of the range of risk that the protocol is willing to take on. However, since GMX V1 fully guarantees the payout for longs, it does not expect payouts to be larger than the reserved assets, and so there is no mechanism to rebase a position for future prices, or liquidate a long position due to it becoming too profitable. So, long positions can easily become dangerously toxic over time, and, even if the protocol as a whole becomes at risk of being insolvent due to this, there would still be no mechanism to close these positions.

Recommendations

One remediation strategy is to only allow short positions on the underlying contract. Long positions can be implemented at the user interface level, shown with a defined user-facing maximum price-feed value and be implemented as a short of an index whose value is the maximum minus the

indexed quantity. This is not user-friendly, though, because a lower maximum is not user-friendly, and a higher maximum would lock up more collateral per unit of position size.

Another remediation strategy is to fork a different base platform. For example, GMX V2 explicitly supports synthetic markets with decoupled position and collateral assets, with an auto-deleveraging feature that probabilistically mitigates this risk.

Remediation

This issue has been acknowledged by Plume Network, as stated by the client, an Automatic Deleveraging system fix will be implemented off-chain with the following scope:

- Track total reserves of the system
- Track total long position exposure
- Check if exposure is within a threshold of the reserves (e.g. exposure is 90% of reserves)
- Reduce profitable positions sizes if the threshold has been reached

This new component falls out of the scope of this audit and has not been reviewed by Zellic.

3.2. VaultPriceFeed may be misconfigured, causing unexpected pricing calculations

Target	VaultPriceFeed.sol		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Low	**Impact**	Low

Description

The VaultPriceFeed contract is used to determine the pricing of the `indexToken` in a specified vault. The VaultPriceFeed in GMX is supposed to track specific on-chain assets, so its behavior may differ when taken outside of that functionality. Plume intends to use arbitrary (potentially off-chain) assets as suitable index tokens.

There are several potentially dangerous defaults set on the VaultPriceFeed that may not apply to Plume's use cases.

The `Vault.getMaxPrice()` function that is used on `getRedemptionAmount()`, `usdToTokenMin()`, `getGlobalShortDelta()`, and `getDelta()` passes an `_includeAmmPrice == true`. When combined with default parameters for the `VaultPriceFeed.isAmmEnabled = true` and `isSecondaryPriceEnabled = true`, the `VaultPriceFeed.getPriceV1()` may default to alternative pricing mechanisms that may not apply to off-chain index tokens.

The `getPriceV1()` function increments the returned price as follows:

```
uint256 price = getPrimaryPrice(_token, _maximise);

if (_includeAmmPrice && isAmmEnabled) {
    uint256 ammPrice = getAmmPrice(_token);
    if (ammPrice > 0) {
        if (_maximise && ammPrice > price) {
            price = ammPrice;
        }
        if (!_maximise && ammPrice < price) {
            price = ammPrice;
        }
    }
}

if (isSecondaryPriceEnabled) {
    price = getSecondaryPrice(_token, price, _maximise);
}
```

Provided the token address matches the stored BNB, ETH, or BTC values, the `getAmmPrice()` will attempt to fetch the price from a necessary price from the `PancakePair` AMM reserve ratios. However, the default values for BNB, ETH, and USD are all `address(0)`. This suggests a token index of `address(0)` may incorrectly get mapped to default values of actual on-chain assets.

Impact

Through a minor misconfiguration during deployment, it is possible that pricing mechanisms reflect on-chain assets as opposed to desired off-chain tickers. This could result in gaming certain `indexToken` positions.

Recommendations

We would recommend disabling certain features by default, such as `isAmmEnabled` and potentially `isSecondaryPriceEnabled`, if the contracts will always be used with custom price feeds. Furthermore, we recommend preventing privileged users from enabling `priceFeeds[token]` where `token` matches the address of any BNB, ETH, or BTC values.

Remediation

This issue has been acknowledged without a confirmed fix implemented by Plume Network. Plume Network has stated that `isAmmEnabled` and `isSecondaryPriceEnabled` will be disabled as recommended.

3.3. Price feed may be gamed if insufficient rounds are captured

Target	VaultPriceFeed.sol		
Category	Coding Mistakes	**Severity**	Low
Likelihood	Medium	**Impact**	Low

Description

The VaultPriceFeed ensures that the `indexToken` is priced accurately depending on necessary pricing mechanisms. In the case that we use the standard Chainlink-style price-feed aggregator, it is possible to game positions with no round data registered.

The primary pricing mechanism uses Chainlink-style pricing aggregators as follows:

```
uint80 roundId = priceFeed.latestRound();

for (uint80 i = 0; i < priceSampleSpace; i++) {
    if (roundId <= i) { break; }
    uint256 p;

    if (i == 0) {
        int256 _p = priceFeed.latestAnswer();
        require(_p > 0, "VaultPriceFeed: invalid price");
        p = uint256(_p);
    } else {
        (, int256 _p, , ,) = priceFeed.getRoundData(roundId - i);
        require(_p > 0, "VaultPriceFeed: invalid price");
        p = uint256(_p);
    }

    if (price == 0) {
        price = p;
        continue;
    }

    if (_maximise && p > price) {
        price = p;
        continue;
    }

    if (!_maximise && p < price) {
        price = p;
```

```
    }
}
```

In the event that the latest `roundId == 0`, the price will return as 0. Thus, if an `indexToken` is added before the price feed has time to accrue data, the position price will start at 0. A user may be able to long a position knowing the price will only ever increase.

Additionally, the default for `priceSampleSpace == 3` means that `roundId` needs to be at least 3 in order for the maximizing or minimizing logic to succeed.

Finally, it is worth mentioning that chain reorganizations may happen between when the feed is created to when the `indexToken` is approved. Thus, it is essential that the index token be approved for new positions no less than the minimum block time for finality on the target chain.

Impact

If pricing mechanisms fail, the price will default to 0, which may lead to users gaming long positions.

Recommendations

Ensure that price feeds have sufficient rounds to allow for price minimums or maximums to be calculated and that these minimum rounds fall inside finalized blocks on the target chain.

Remediation

This issue has been acknowledged by Plume Network. Plume Network has stated that price feeds will have lots of rounds before trading begins.

3.4. Extra calls into timelock EOA will fail

Target	BasePositionManager.sol, PositionManager.sol		
Category	Coding Mistakes	**Severity**	High
Likelihood	N/A	**Impact**	Informational

Description

Plume Network intends to replace the Timelock contract with a governance-owned EOA. In order to implement this change, these calls into the timelock were removed in PositionUtils:

```
ITimelock(timelock).enableLeverage(_vault);
// ITimelock(timelock).enableLeverage(_vault);
 IRouter(_router).pluginIncreasePosition(_account, _collateralToken,
    _indexToken, _sizeDelta, _isLong);
ITimelock(timelock).disableLeverage(_vault);
// ITimelock(timelock).disableLeverage(_vault);
```

However, not all calls into the timelock were removed. In BasePositionManager, the timelock is invoked in order to read the margin-fee basis points to emit referral events:

```
emit IncreasePositionReferral(
    _account,
    _sizeDelta,
    ITimelock(timelock).marginFeeBasisPoints(),
    referralCode,
    referrer
);
```

```
emit DecreasePositionReferral(
    _account,
    _sizeDelta,
    ITimelock(timelock).marginFeeBasisPoints(),
    referralCode,
    referrer
);
```

Additionally, in PositionManager, the timelock is called before and after position increases, decreases, and liquidations.

```
ITimelock(timelock).enableLeverage(_vault);
IVault(_vault).liquidatePosition(_account, _collateralToken, _indexToken,
    _isLong, _feeReceiver);
ITimelock(timelock).disableLeverage(_vault);
```

```
ITimelock(timelock).enableLeverage(_vault);
IOrderBook(orderBook).executeIncreaseOrder(_account, _orderIndex,
    _feeReceiver);
ITimelock(timelock).disableLeverage(_vault);
```

```
ITimelock(timelock).enableLeverage(_vault);
IOrderBook(orderBook).executeDecreaseOrder(_account, _orderIndex,
    _feeReceiver);
ITimelock(timelock).disableLeverage(_vault);
```

Impact

Since the `timelock` address will be an EOA, all of these calls will revert, either on the `EXTCODESIZE` check emitted by the Solidity compiler or on the return-value type check. So, these features on the PositionManager and BasePositionManager will not be usable.

Recommendations

We recommend removing these calls.

Alternatively, we recommend deploying the Timelock contract as the base GMX V1 expects. Even though the forked protocol does not require a timelock, because of the tight integration between the governance timelock and the rest of the code, directly removing the timelock will deploy the project in a changed, and therefore un-battle-tested, state. On the other hand, if the Timelock contract itself is deployed as is, the buffer can be set to zero and never increased, which functionally bypasses the timelock feature of the Timelock. Although cumbersome, this would mean that no changes are necessary to implement this change where the timelock is removed.

Remediation

This issue has been acknowledged by Plume Network, and a fix was implemented in commit `af802121` ↗.

4. System Design

This provides a description of the high-level components of the system and how they interact, including details like a function's externally controllable inputs and how an attacker could leverage each input to cause harm or which invariants or constraints of the system are critical and must always be upheld.

Not all components in the audit scope may have been modeled. The absence of a component in this section does not necessarily suggest that it is safe.

Description

This project is a fork of GMX V1, forked at commit `58c4ac801a95236db24c0857f820bdf4935d5814`. The fork changes a few things.

- In BasePositionManager and PositionUtils, a few calls into the Timelock contract were commented out, in order to remove the timelock governance functionality from the system:

```
address timelock = IVault(_vault).gov();
// address timelock = IVault(_vault).gov();

 // should be called strictly before position is updated in Vault
 IShortsTracker(shortsTracker).updateGlobalShortData(_account,
    _collateralToken, _indexToken, _isLong, _sizeDelta, markPrice, false);

ITimelock(timelock).enableLeverage(_vault);
// ITimelock(timelock).enableLeverage(_vault);
 uint256 amountOut = IRouter(router).pluginDecreasePosition(_account,
    _collateralToken, _indexToken, _collateralDelta, _sizeDelta, _isLong,
    _receiver);
ITimelock(timelock).disableLeverage(_vault);
// ITimelock(timelock).disableLeverage(_vault);
```

- In Vault, for long positions, the check that the collateral token and index token are the same asset was removed:

```
function _validateTokens(address _collateralToken, address _indexToken,
   bool _isLong) private view {
    if (_isLong) {
       _validate(_collateralToken == _indexToken, 42);
       // _validate(_collateralToken == _indexToken, 42);
         _validate(whitelistedTokens[_collateralToken], 43);
         _validate(!stableTokens[_collateralToken], 44);
       _validate(!stableTokens[_indexToken], 47);
         return;
```

```
    }

    _validate(whitelistedTokens[_collateralToken], 45);
    _validate(stableTokens[_collateralToken], 46);
    _validate(!stableTokens[_indexToken], 47);
    _validate(shortableTokens[_indexToken], 48);
}
```

- In the GLP token contract, some strings were changed to rename the token to "Cultured LP".
- Code was removed from the RewardRouter contract, including calls to `stakedGlpTracker`:

```
function mintAndStakeGlpETH(
    uint256 _minUsdg,
    uint256 _minGlp
) external payable nonReentrant returns (uint256) {
    // [...]
    IRewardTracker(stakedGlpTracker).stakeForAccount(account, account,
        feeGlpTracker, glpAmount);
```

```
function unstakeAndRedeemGlp(
    address _tokenOut,
    uint256 _glpAmount,
    uint256 _minOut,
    address _receiver
) external nonReentrant returns (uint256) {
    // [...]
    IRewardTracker(stakedGlpTracker).unstakeForAccount(account, feeGlpTracker,
        _glpAmount, account);
```

```
    function claim() external nonReentrant {
        address account = msg.sender;

        IRewardTracker(feeGmxTracker).claimForAccount(account, account);
        IRewardTracker(feeGlpTracker).claimForAccount(account, account);

        IRewardTracker(stakedGmxTracker).claimForAccount(account, account);
        IRewardTracker(stakedGlpTracker).claimForAccount(account, account);
    }
```

```
function claimFees() external nonReentrant {
    address account = msg.sender;

    IRewardTracker(feeGmxTracker).claimForAccount(account, account);
    IRewardTracker(feeGlpTracker).claimForAccount(account, account);
}
```

Invariants

Here is a list of the invariants that were altered by the in-scope changes.

- The collateral token can now be a different token than the index token, for long positions. For an analysis of this, see Finding 3.1. ↗.
- The index token now no longer needs to be a whitelisted token. As long as the address (or dummy address) is not in the `stableTokens` list, it is a valid index and can be accepted as an index by `_validateTokens`.

Test coverage

No tests were added.

Attack surface

The attack surface we considered for this audit is identical to the attack surface of GMX V1, where the trader is untrusted and may open and close exploitative positions and where the governance and price feeds are assumed to be trusted.

5. Assessment Results

At the time of our assessment, the reviewed code was deployed to the Plume Testnet.

During our assessment on the scoped Cultured contracts, we discovered four findings. No critical issues were found. One finding was of high impact, two were of low impact, and the remaining finding was informational in nature.

5.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.