BLOCKAPEX

SMART CONTRACT SECURITY

V 1.0

DATE: 29th MAY 2024

PREPARED FOR: AMPED FINANCE

SECURITY REPORT
BLOCKAPEX VERIFIED

About BlockApex

Founded in early 2021, is a security-first blockchain consulting firm. We offer services in a wide range of areas including Audits for Smart Contracts, Blockchain Protocols, Tokenomics along with Invariant development (i.e., test-suite) and Decentralized Application Penetration Testing. With a dedicated team of over 40+ experts dispersed globally, BlockApex has contributed to enhancing the security of essential software components utilized by many users worldwide, including vital systems and technologies.

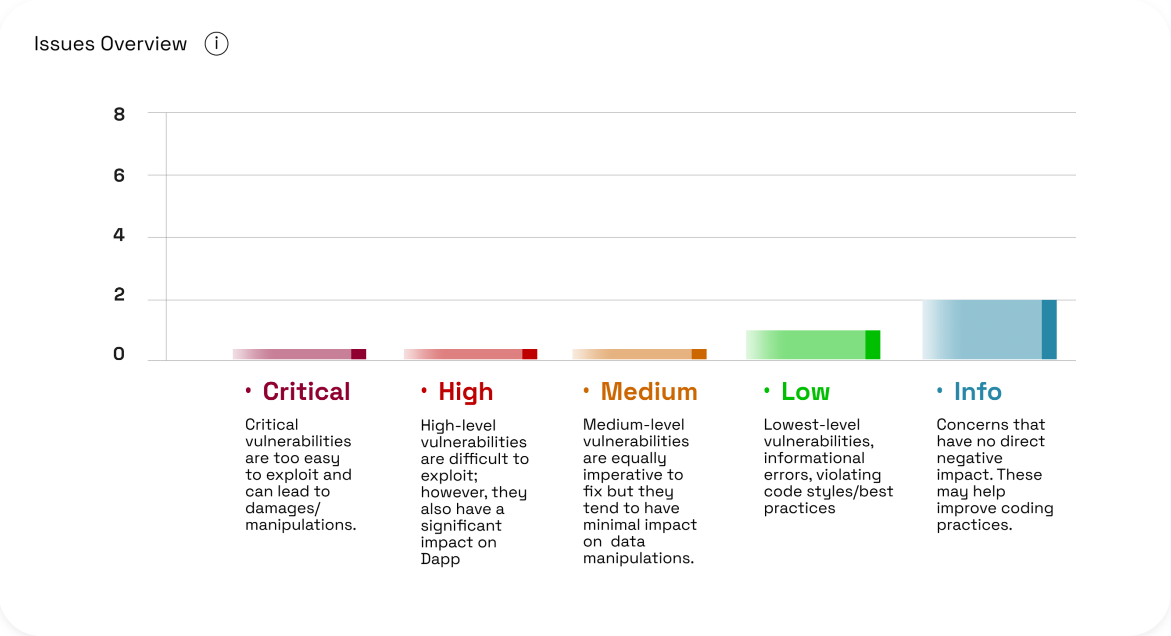
BlockApex has a focus on blockchain security, maintaining an expertise hub to navigate this dynamic field. We actively contribute to security research and openly share our findings with the community. Our work is available for review at our public repository, showcasing audit reports and insights into our innovative practices.

To stay informed about BlockApex's latest developments, breakthroughs, and services, we invite you to follow us on Twitter and explore our GitHub. For direct inquiries, partnership opportunities, or to learn more about how BlockApex can assist your organization in achieving its security objectives, please visit our Contact page at our website , or reach out to us via email at hello@blockapex.io.

Contents

1 Executive Summary

Our team conducted a Filtered Audit, engaging Two auditors to independently examine the Amped Finance Contracts. This rigorous approach included a meticulous line-by-line code review to identify potential vulnerabilities. Following the initial manual inspection, all flagged issues were thoroughly re-examined and re-tested to confirm their validity and ensure accuracy in our final assessment.



Critical vulnerabilities are too easy to exploit and can lead to damages/ manipulations.

High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on Dapp

Medium-level vulnerabilities are equally imperative to fix but they tend to have minimal impact on data manipulations.

Lowest-level vulnerabilities, informational errors, violating code styles/best practices

Concerns that have no direct negative impact. These may help improve coding practices.

1.1 Scope

1.1.1 In Scope

Amped protocol is a fork of GMX protocol (Commit Hash: 6d393fa8ee26b374856f6b603cb96241ca77b3cb). Amped Finance is a decentralised perpetual exchange on LightLink that allows users to trade leveraged positions with low fees and zero price impact. Amped supports both spot and perpetuals trading. User can use spot features to carryout swapping of assets and leverage perpetuals platform to carryout futures trading with a leverage of upto 50x. The audit will focus on both the contract's logic and structure, as well as the deployment scripts to ensure a comprehensive security review.

Targets:

- Contracts in Scope: amped-smart-contracts/*
- Initial Commit Hash: c7c4285be52e45fb054b5dfcde3fd1dd12b4eb16
- Final Commit Hash: b1fd198a26d0a3f9f105354881528cd3a794b76d
- Platform: Ethereum
- Language: Solidity

1.1.2 Out of Scope

Features or functionalities not explicitly listed within the "In Scope" section, such as backend operations unrelated to the direct functioning of the smart contracts or external system integrations, are considered outside the scope of this audit.

1.2 Methodology

The codebase was audited using a filtered audit technique by a team of Two(2) auditors over a span of 2 weeks. The process began with a reconnaissance phase where the auditors developed a foundational understanding of the codebase. This initial phase helped form presumptions for the developed codebase. As the audit progressed into the manual code review phase, auditors made the Proofs of Concept (POCs) to verify their findings. This phase was designed to identify logical flaws, complemented by code optimizations, software and security design patterns, code styles, and best practices

1.3 Project Goals

The engagement was scoped to provide a security assessment of the Amped contract as compared to the original GMX contract. Specifically, we sought to answer the following non-exhaustive list of questions:

1. How comprehensively does the diff audit identify all the changes between the GMX and Amped contracts?
2. Do the changes introduced in the Amped contract inadvertently create new vulnerabilities?
3. Are there any common security issues present in the modified code, such as reentrancy attacks, integer overflows/underflows, or improper input validation?
4. Are the roles and permissions correctly implemented and enforced according to the changes made in the Amped contract?
5. Does the modified contract ensure that only authorized entities can perform sensitive operations?

1.4 Status Descriptions

Acknowledged: The issue has been recognized and is under review. It indicates that the relevant team is aware of the problem and is actively considering the next steps or solutions.

Fixed: The issue has been addressed and resolved. Necessary actions or corrections have been implemented to eliminate the vulnerability or problem.

Closed: This status signifies that the issue has been thoroughly evaluated and acknowledged by the development team. While no immediate action is being taken.

1.5 Summary of Findings Identified

S.No	Severity	Findings	Status
#1	LOW	Missing Check for Existing Referral Code in **_setTraderReferralCode** Function	ACKNOWLEDGED
#2	INFO	Logic changed in **setBuffer** Function	ACKNOWLEDGED
#3	INFO	Redundant require statements in **getPrimaryPrice** Function	FIXED

2 Findings and Risk Analysis

2.1 Missing Check for Existing Referral Code in _setTraderReferralCode Function

Severity: Low

Status: Acknowledged

Location :

1. `Contracts/core/PositionRouter.sol`

Description The `_setTraderReferralCode` function lacks an important check to determine if the user already has a referral code set. This omission can lead to unintended overwriting of existing referral codes, which might not be the desired behavior. The original implementation includes this check to ensure that once a referral code is set, it cannot be easily changed.

Code Affected

```
function _setTraderReferralCode(bytes32 _referralCode) internal {
        if (_referralCode != bytes32(0) &amp;&amp; referralStorage != address(0)) {
            IReferralStorage(referralStorage).setTraderReferralCode(msg.sender, _referralCode);
        }
}
```

Impact Users could exploit this to change their referral codes to gain undue benefits.

Proof of Concept

```
// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.0;

import &quot;hardhat/console.sol&quot;;

contract MockTest {

    address public  referralStorage;

    constructor(address _referralStorage) {
        referralStorage = _referralStorage;
    }

    function setTraderReferralCode(bytes32 _referralCode) public {
        _setTraderReferralCode(_referralCode);
    }

    function _setTraderReferralCode(bytes32 _referralCode) internal {
        if (_referralCode != bytes32(0) &amp;&amp; referralStorage != address(0)) {
            IReferralStorage(referralStorage).setTraderReferralCode(msg.sender, _referralCode);
        }
    }
}

```

```
interface IReferralStorage {
    function setTraderReferralCode(address _account, bytes32 _code) external;
    function traderReferralCodes(address _account) external view returns (bytes32);
}

contract ReferralStorage is IReferralStorage{

    mapping (address =&gt; bytes32) public override traderReferralCodes;
    event SetTraderReferralCode(address account, bytes32 code);

    function setTraderReferralCode(address _account, bytes32 _code) external override  {
        _setTraderReferralCode(_account, _code);
    }

     function _setTraderReferralCode(address _account, bytes32 _code) private {
        traderReferralCodes[_account] = _code;
        emit SetTraderReferralCode(_account, _code);
    }
}
```

Recommendation Add the missing check to ensure that the referral code is not overwritten if one already exists.

Developer Response We are allowing traders to change referral code when they need to, as many influencers ask for it as when users see stats of a influencer they refrain trading using their link.

2.2 Logic changed in setBuffer()

Severity: Info

Status: Acknowledged

Location :

1. `Contracts/peripherals/Timelock.sol`

Description During the differential audit of the GMX codebase and the Amped codebase, a notable thing was identified in the `Timelock.sol` contract regarding the `setBuffer()` function. This function allows the admin to set a buffer time. However, the key difference lies in the constraints placed on modifying the buffer time in both codebases. In the GMX codebase, the `setBuffer()` function includes a restriction that prevents the buffer time from being decreased. This ensures that once the buffer time is set, it can only be increased, providing a safeguard against potential admin actions that might reduce the waiting period for time-sensitive operations. In contrast, the corresponding `setBuffer()` function in the Amped codebase has had this restriction commented out. This alteration allows the admin the flexibility to both increase and decrease the buffer time.

Code Affected

```
function setBuffer(uint256 _buffer) external onlyAdmin {
        require(_buffer &lt;= MAX_BUFFER, &quot;Timelock: invalid _buffer&quot;);
        // require(_buffer &gt; buffer, &quot;Timelock: buffer cannot be decreased&quot;);
        buffer = _buffer;
    }
```

Recommendation The amped team has taken the above approach as a design choice and it's the intended as per the amped protocol team. To ensure better transparency and security within the Amped protocol, it is recommended that time-sensitive functions should not allow the buffer time to be decreased. This practice helps to maintain a consistent security margin and prevents potential misuse or accidental reduction of the buffer period.

Developer Response We acknowledge that, the admin can decrease buffer, we want to keep it like that.

2.3 Redundant require statments in getPrimaryPrice function

Severity: Info

Status: Fixed

Location :

1. `Contracts/core/VaultPriceFeed.sol`

Description In the `getPrimaryPrice` function, there is a redundant require check that ensures the fetched price is greater than zero. This redundancy might have been an oversight by the developer during the modification of the function logic.

Code Affected

```
function getPrimaryPrice(address _token, bool _maximise) public override view returns (uint256) {
        address priceFeedAddress = priceFeeds[_token];
        require(priceFeedAddress != address(0), &quot;VaultPriceFeed: invalid price feed&quot;);

        if (chainlinkFlags != address(0)) {
            bool isRaised = IChainlinkFlags(chainlinkFlags).getFlag(FLAG_ARBITRUM_SEQ_OFFLINE);
            if (isRaised) {
                    // If flag is raised we shouldn&#x27;t perform any critical operations
                revert(&quot;Chainlink feeds are not being updated&quot;);
            }
        }

        IPriceFeed priceFeed = IPriceFeed(priceFeedAddress);

        uint256 price = 0;
        // uint80 roundId = priceFeed.latestRound();

        price = priceFeed.latestAnswer();
        require(price &gt; 0, &quot;VaultPriceFeed: invalid price&quot;);

        // Commented Code

        require(price &gt; 0, &quot;VaultPriceFeed: could not fetch price&quot;);
        // normalise price precision
        uint256 _priceDecimals = priceDecimals[_token];
        return price.mul(PRICE_PRECISION).div(10 ** _priceDecimals);
    }
```

Recommendation Remove the redundant require checks.

Disclaimer:

The smart contracts provided by the client with the purpose of security review have been thoroughly analyzed in compliance with the industrial best practices till date w.r.t. Smart Contract Weakness Classification (SWC) and Cybersecurity Vulnerabilities in smart contract code, the details of which are enclosed in this report.

This report is not an endorsement or indictment of the project or team, and they do not in any way guarantee the security of the particular object in context. This report is not considered, and should not be interpreted as an influence, on the potential economics of the token (if any), its sale, or any other aspect of the project that contributes to the protocol's public marketing.

Crypto assets/ tokens are the results of the emerging blockchain technology in the domain of decentralized finance and they carry with them high levels of technical risk and uncertainty. No report provides any warranty or representation to any third-party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the reports in any way, including to make any decisions to buy or sell any token, product, service, or asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and is not a guarantee as to the absolute security of the project.

Smart contracts are deployed and executed on a blockchain. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. The scope of our review is limited to a review of the programmable code and only the programmable code, we note, as being within the scope of our review within this report. The smart contract programming language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer or any other areas beyond the programming language's compiler scope that could present security risks.

This security review cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While BlockApex has done their best in conducting the analysis and producing this report, it is important to note that one should not rely on this report only - we recommend proceeding with several independent code security reviews and a public bug bounty program to ensure the security of smart contracts.