BLOCKAPEX

SMART CONTRACT SECURITY

v 1.0

Date: 18-06-2025

Prepared For: Amped Finance V2 (Staking)

About BlockApex

Founded in early 2021 BlockApex 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.

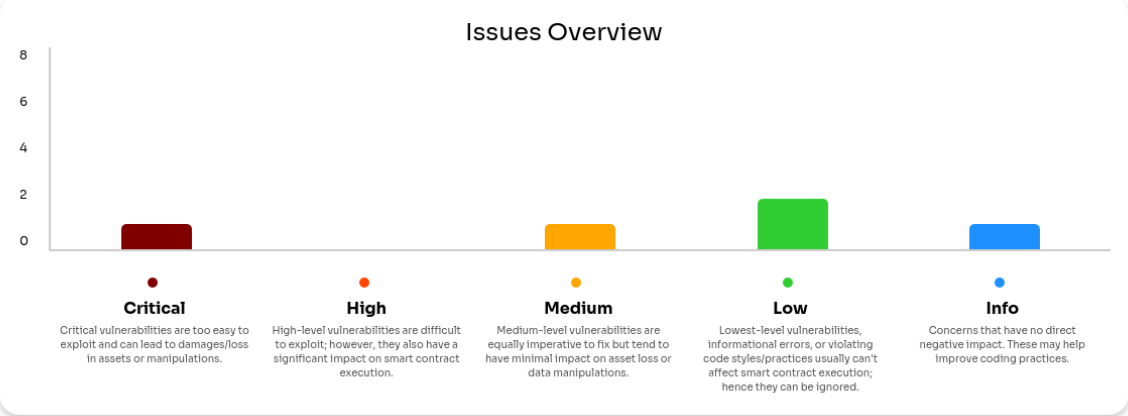
BlockApex has a focus on blockchain security, maintaining an expertise hub to navigate this dynamic field. We actively contribute to security research and openly share our findings with the community. Our work is available for review at our public repository, showcasing audit reports and insights into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

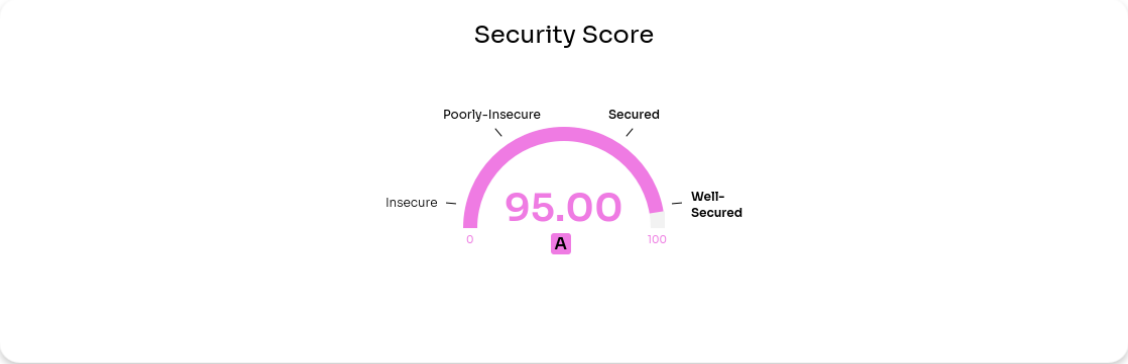
To stay informed about BlockApex's latest developments, breakthroughs, and services, we invite you to follow us on Twitter and explore our GitHub. For direct inquiries, partnership opportunities, or to learn more about how BlockApex can assist your organization in achieving its security objectives, please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

Contents

1 Executive Summary

BlockApex conducted a comprehensive white-box security audit of the YieldBearingALPVault smart contract in the Amped Finance ecosystem, a tokenized EIP-4626-style vault designed to optimize yield from fsALP tokens. The audit focused solely on this contract and excluded external dependencies like rewardRouter and ws. One auditor worked on this project and contract was reviewed line by line. The assessment identified five issues which include 1 critical, 1 medium, 2 low and 1 informational. All the issues were also reviewed after fixes.





1.1 Scope

1.1.1 In Scope

https://github.com/amped-finance/amped-smart-contracts/commit/5be7538429ca30c51806b105aa24ce85c5ee7b4e

- contracts/staking/YieldBearingALPVault.sol

1.1.2 Out of Scope

Anything not mentioned in scope

1.2 Methodology

The codebase was audited using a filtered audit technique. The auditors scanned the codebase in an iterative process for a time spanning 4 days. Starting with the recon phase, a basic understanding of the contract was developed, and the auditor worked on developing presumptions for the contract with the help relevant documentation/whitepaper. Furthermore, the audit moved on with the manual code reviews to find logical flaws in the codebase complemented with code optimizations,software, and security design patterns, code styles and best practices

1.3 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.

1.4 Summary of Findings Identified

S.NO	SEVERITY	FINDINGS	STATUS
#1	CRITICAL	Global Cooldown Enforcement Enables Denial of Service on Withdrawals	FIXED
#2	MEDIUM	Reward Tokens (ws, esAMP) Are Recoverable by Governance	FIXED
#3	LOW	Inaccurate Return Value in convertToAssets When Vault is Empty	FIXED
#4	LOW	Inaccurate Return Value in convertToShares When Vault is Empty	FIXED
#5	INFO	Redundant previewDeposit Function	FIXED

2 Findings and Risk Analysis

2.1 Global Cooldown Enforcement Enables Denial of Service on Withdrawals

Severity: Critical

Status: Fixed

Location:

contracts/staking/YieldBearingALPVault.sol

Description: The vault enforces a global cooldown period for withdrawals using a single lastDeposit timestamp shared across all users:

```
uint256 public lastDeposit;

function withdrawalsAvailableAt() public view returns (uint256) {
    if (lastDeposit == 0) return 0;
    uint256 cooldown = glpManager.cooldownDuration();
    return lastDeposit.add(cooldown);
}
```

This timestamp is updated on every deposit, regardless of which user performs the deposit:

```
lastDeposit = block.timestamp;
```

Withdrawals are blocked until the cooldown has passed:

```
require(block.timestamp &gt;= withdrawalsAvailableAt(), &quot;YieldBearingALP: cooldown
    active&quot;);
```

Recommendation: Refactor the cooldown logic to use per-user cooldown tracking:

```
mapping(address =&gt; uint256) public lastDeposit;
```

Update it only for the depositor:

```
lastDeposit[msg.sender] = block.timestamp;
```

Modify the withdrawal condition accordingly:

```
require(block.timestamp &gt;= lastDeposit[msg.sender] + cooldown, &quot;YieldBearingALP:
    cooldown active&quot;);
```

This ensures cooldowns are enforced fairly and independently, preventing users from interfering with one another's withdrawal eligibility.

2.2 Reward Tokens (ws, esAMP) Are Recoverable by Governance

Severity: Medium

Status: Fixed

Location:

contracts/staking/YieldBearingALPVault.sol

Description: The `recoverToken()` function allows the `gov` address to recover any ERC-20 token except `fsALP`, which is the core asset backing the vault:

```
function recoverToken(address _token, uint256 _amount, address _receiver) external onlyGov
     {
    require(_token != address(fsAlp), &quot;YieldBearingALP: cannot recover fsALP&quot;);
    IERC20(_token).safeTransfer(_receiver, _amount);
}
```

However, the contract also manages:

- `ws` (Wrapped Sonic) — a reward token actively used in auto-compounding
- `esAMP` — another reward token claimed by the vault

These tokens are not protected in the `recoverToken()` logic, meaning the gov can freely transfer them to an external address.

Recommendation: Add explicit checks to protect core reward tokens from being recovered unintentionally:

```
require(
    _token != address(fsAlp) &amp;&amp;
    _token != address(ws) &amp;&amp;
    _token != address(esAmp),
    &quot;YieldBearingALP: cannot recover core vault tokens&quot;
);
```

2.3 Inaccurate Return Value in convertToAssets When Vault is Empty

Severity: Low

Status: Fixed

Location:

contracts/staking/YieldBearingALPVault.sol

Description: The `convertToAssets` function is intended to calculate the amount of `fsALP` assets represented by a given number of `yALP` shares. However, when `totalSupply == 0`, the function currently returns the same number of assets as shares:

```
function convertToAssets(uint256 shares) public view returns (uint256) {
    uint256 supply = totalSupply;
    return supply == 0 ? shares : shares.mul(totalAssets()).div(supply);
}
```

This behavior is incorrect. If the vault has no supply and no assets, then shares should not correspond to any real asset value. Returning `shares` creates a misleading 1:1 mapping and may be misinterpreted by other contracts or UIs as meaningful, when in fact the vault is empty.

Recommendation: Update the return condition when `totalSupply == 0` to return `0` instead of `shares`:

```
return supply == 0 ? 0 : shares.mul(totalAssets()).div(supply);
```

This change ensures that share-to-asset conversion always reflects actual vault state and avoids incorrect valuation when the vault is uninitialized.

2.4 Inaccurate Return Value in convertToShares When Vault is Empty

Severity: Low

Status: Fixed

Location:

contracts/staking/YieldBearingALPVault.sol

Description: The `convertToShares` function is responsible for converting a given amount of `fsALP` assets into the equivalent number of `yALP` shares. However, the current implementation returns `assets` directly when `totalSupply == 0`:

```
function convertToShares(uint256 assets) public view returns (uint256) {
    uint256 supply = totalSupply;
    return supply == 0 ? assets : assets.mul(supply).div(totalAssets());
}
```

While this may appear convenient for the initial deposit, it is logically inconsistent. When the vault has no `fsALP` and no `yALP`, there is no valid exchange rate between assets and shares. Returning the same value (`assets`) falsely implies a 1:1 share issuance rate, which may not align with the vault's real state or intended minting logic.

Recommendation: Return `0` when `totalSupply == 0` to clearly indicate that no shares can be issued due to the absence of existing vault liquidity:

```
return supply == 0 ? 0 : assets.mul(supply).div(totalAssets());
```

Alternatively, enforce correct share estimation logic during the actual deposit to avoid encoding assumptions in view functions.

2.5 Redundant previewDeposit Function

Severity: Info

Status: Fixed

Location:

contracts/staking/YieldBearingALPVault.sol

Description: The contract defines a `previewDeposit` function as follows:

```
function previewDeposit(uint256 assets) public view returns (uint256) {
    return convertToShares(assets);
}
```

However, `convertToShares` is already a `public` function that provides the same output. The `previewDeposit` function acts as a direct pass-through without adding any logic, validation, or abstraction. Since `convertToShares` already fulfills the exact same purpose and is publicly callable, this additional wrapper appears redundant.

Recommendation: Remove the `previewDeposit` function entirely unless there is a specific forward-compatibility or interface compliance reason to retain it. Consumers can directly use `convertToShares(assets)` to obtain the same output with no functional difference.

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