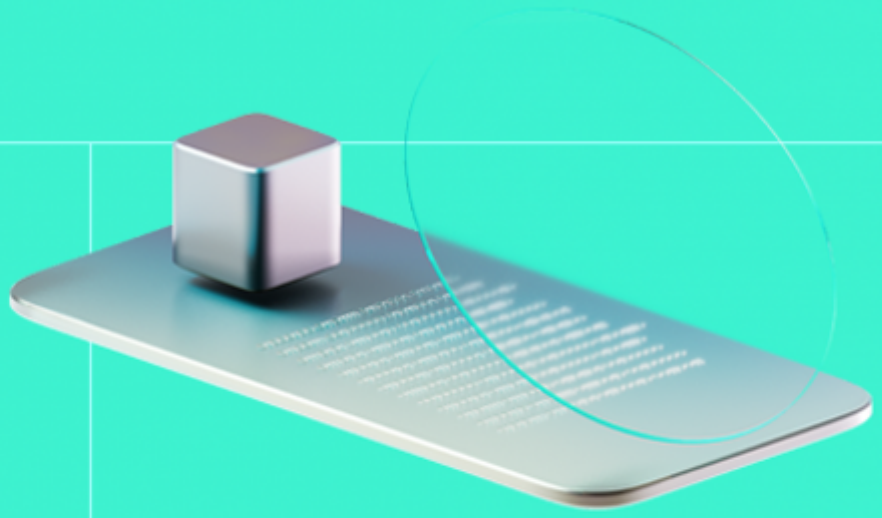
HACKEN

Smart Contract Code Review And Security Analysis Report



Customer: BlastUp

Date: 09/05/2024

We express our gratitude to the BlastUp team for the collaborative engagement that enabled the execution of this Smart Contract Security Assessment.

BlastUP is a launchpad and staking platform within the Blast blockchain ecosystem.

Platform: Blast(EVM)

Language: Solidity

Tags: Blast, Launchpad, IDO, Staking

Timeline: 24/04/2024 - 26/04/2024

Methodology: https://hackenio.cc/sc_methodology

Review Scope

Repository	https://github.com/blastupio/launchpad-contracts
Commit	cb6957d

Audit Summary

10/10	9/10	75%	10/10
Security Score	Code quality score	Test coverage	Documentation quality score

Total 8.8/10

The system users should acknowledge all the risks summed up in the risks section of the report

12	3	7	2
Total Findings	Resolved	Accepted	Mitigated

Findings by severity

Critical	0
High	0
Medium	2
Low	9

Vulnerability	Status
F-2024-1535 - Signature Replay In Tier Registration	Mitigated
F-2024-1540 - Centralization Risk For Privileged Actors	Mitigated
F-2024-1534 - Inconsistent Validation Checks Across Token Setting Functions	Accepted
F-2024-1543 - Lack of Boundary Checks	Accepted
F-2024-1545 - Missing Validation of Tier Hierarchy	Accepted
F-2024-1609 - Inconsistency in Token Allocation Post-Claim	Accepted
F-2024-1735 - Privileged Functions Susceptible to Front-Running	Accepted
F-2024-1738 - `decimals()` is not a part of the `ERC-20` standard	Accepted
F-2024-2043 - Fee-On-Transfer Token Handling Flaw	Accepted
F-2024-1515 - Chainlink's latestRoundData() Might Return Stale or Incorrect Results	Fixed
F-2024-2028 - Strategic Splitting of Tokens Increases Allocation Unfairly	Fixed
F-2024-2040 - Inappropriate Handling of Decimal Precision for Tier Minimum Amounts	Fixed

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

Document

Name	Smart Contract Code Review and Security Analysis Report for BlastUp
Audited By	Eren Gonen
Approved By	Ataberk Yavuzer, Kaan Caglan
Website	https://blastup.io/
Changelog	06/05/2024 - Preliminary Report
	09/05/2024 - Final Report

Table of Contents

System Overview

BlastUP is a launchpad and staking platform within the Blast blockchain ecosystem with the following contracts:

Launchpad — The Launchpad contract, built with upgradeable ownership, primarily facilitates the management of token sales on a blockchain platform. It supports different user tiers, each with specific minimum amounts and weights that influence their participation in token sales. The contract interacts with an oracle for pricing conversions and handles registrations, sales, and the claiming of tokens based on sale conditions and user eligibility.
LaunchpadV2 — The 2.0 version of Launchpad contract. Users able to register their tier and allocations according to the staked amount on BLPStaking.sol and Inherits **Launchpad.sol**.
YieldStaking — The contract manages staking through an elaborate system of indexed mappings and struct arrays that track individual and total stakes, including accrued rewards and withdrawal timelines. Functionally, the contract allows users to deposit tokens into specific pools (USDB and WETH), where these tokens are then "locked" with an associated timer that dictates when they can be withdrawn. Reward calculations are dynamically adjusted through the integration with rebasing mechanisms of the staked tokens, allowing the contract to update the staking index based on the tokens' new supply metrics after each rebase event.

Privileged roles

- The **owner** of the **Launchpad** contract can:
  - Set a new signer address.
  - Set a new operator address.
  - Place tokens for a new sale event.
- The **owner** and **operator** of the **Launchpad** contract can:
  - Set new amounts for user tiers.
  - Set new weights for user tiers.
  - Set new registration **start** and **end** times.
  - Set a new public sale **start** time.
  - Set new FCFS sale **start** and **end** times.
  - Set a new tge **start** time.
  - Set a new vesting **start** time.
  - Claim remainders after a sale ends.
- The **owner** of the **YieldStaking** contract can:
  - Set the minimum USDB value for stakes.
  - Set the minimum time required before withdrawals can be made .

Executive Summary

This report presents an in-depth analysis and scoring of the customer's smart contract project. Detailed scoring criteria can be referenced in the scoring methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**.

- Functional requirements are partially missed.
  - Use case are provided.
  - System roles are documented.
- Technical description is not provided.
  - Descriptions of the development environment is provided.
  - NatSpec are sufficient.
  - The Launchpad, LaunchpadV2 description is provided.
  - The YieldStaking description is provided.

Code quality

The total Code Quality score is **9** out of **10**.

- The development environment is configured.
- The majority of functions rely on admin actions rather than implementing governance structures for decision-making.

Test coverage

Code coverage of the project is **75%**.

- Coverage tool couldn't be run due to errors.
- During the manual inspection, it was identified that unit and fuzz testing are implemented, but integration tests are absent, leading to the oversight of crucial scenarios. For example, in claim fuzz testing, the contract consistently returns zero for user rewards, and fuzz testing successfully claims this zero reward.

Security score

Upon auditing, the code was found to contain **0** critical, **0** high, **2** medium, and **9** low severity issues, leading to a security score of **10** out of **10**.

All identified issues are detailed in the “Findings” section of this report.

Summary

The comprehensive audit of the customer's smart contract yields an overall score of **8.8**. This score reflects the combined evaluation of documentation, code quality, test coverage, and security aspects of the project.

Risks

- **Verification Limitations of Backend KYC:** KYC verification done via the backend cannot be verified.
- **Challenges in Synchronized Reward Distribution:** The reward distribution is based on index updates; therefore, if two users deposit at different timestamps but the index is not updated before and after when the second user deposits, the system treats them as if they deposited simultaneously.
- **Absence of Time-lock Mechanisms for Critical Operations**: Without time-locks on critical operations, there is no buffer to review or revert potentially harmful actions, increasing the risk of rapid exploitation and irreversible changes.
- **Absence of Pausable Feature for Unexpected Events:** Without the pausable feature, the contract cannot be immediately paused in the event of unexpected occurrences.
- **Owner's Unrestricted State Modification**: The absence of restrictions on state variable modifications by the owner leads to arbitrary changes, affecting contract integrity and user trust, especially during critical operations like minting phases.
- **Dependency on Unaudited External Libraries**: The project utilizes libraries or contracts without security audits, potentially introducing vulnerabilities. This compromises the security of the audited system, making it susceptible to attacks exploiting these external weaknesses.
- **Solidity Version Compatibility:** The Solidity version `0.8.20` employs the recently introduced `PUSH0` opcode in the Shanghai EVM. This opcode might not be universally supported across all blockchain networks and Layer 2 solutions. Thus, as a result, it might be not possible to deploy solution with version `0.8.20 >=` on some blockchains.

Findings

Vulnerability Details

F-2024-1535 - Signature Replay In Tier Registration - Medium

Description:

In the **Launchpad** smart contract, the `register()` function allows users to register for different tiers (e.g., BRONZE, SILVER, DIAMOND) by providing a signature from an authorized signer.

```
function register(uint256 id, Types.UserTiers tier, uint256 amountOf
Tokens, bytes memory signature) external virtual {
require(!placedTokens[id].approved, "BlastUP: you need to use regist
er with approve function");
_validateUserBalanceSignature(amountOfTokens, signature);
_register(amountOfTokens, id, tier);
}

function _validateUserBalanceSignature(uint256 amountOfTokens, bytes
memory signature) internal view {
address signer_ = keccak256(abi.encodePacked(msg.sender, amountOfTok
ens, address(this), block.chainid))
.toEthSignedMessageHash().recover(signature);
require(signer_ == signer, "BlastUP: Invalid signature");
}
```

This function validates the signature to ensure it was indeed signed by the authorized signer. However, vulnerability arises due to the signature's composition—it lacks a nonce or unique identifier for each transaction. As a result, once a signature is generated for a particular set of parameters (user, amount, and contract address), it can be reused maliciously for registering the same user across different tokens without additional authorization.

This vulnerability allows an attacker to:

- Bypass per-token registration controls, gaining unfair access or financial benefits across multiple tokens.
- Potentially manipulate token distribution and tier allocations, affecting the integrity and fairness of the token sale process.

Assets:

- Launchpad.sol [https://github.com/blastupio/launchpad-contracts]

Status: Mitigated

Classification

Impact: 2/5

Likelihood: 5/5

Exploitability: Independent

Complexity: Medium

Likelihood [1-5]: 5

Impact [1-5]: 2

Exploitability [0-2]: 0

Complexity [0-2]: 1

Final Score: 3.3 (Medium)

Hacken Calculator Version: 0.6

Severity: Medium

Recommendations

Remediation: Incorporate a nonce mechanism in the signature generation and verification process. Each registration attempt should include a unique, single-use number that is checked against stored values to ensure each signature can only be used once. Additionally, consider including the token address in the signed data to ensure that each signature is explicitly linked to a specific token registration

Resolution: The **BlastUp** team mitigated the issue with the following statement;

> The amount passed within a signature denotes BLP balance which is not specific to a concrete placedToken so basically this signature proves that "User X has Y BLP tokens bought from presale" thus, single signature is correct for any token because it is used to validate that user has enough balance to claim a specific tier (Y >= minAmountForTier[tier]) So, if minimal amount to join TITANIUM tier is 20_000 and user has a singature proving that he has 20_000 BLP tokens bought earlier, he can claim TITANIUM tier for any sale.

Evidences

POC

Reproduce:

Setup and Token Placement:

- The test initializes by setting up registration periods and token sale parameters such as prices, volume distributions among tiers, and sale timings using a struct called **PlacedToken**.

- Two ERC20 tokens (assumed here as **USDT** and **DAI**) are minted and approved for the maximum possible amount to the **launchpad** contract which handles token placement and registration.

Signature Generation:

- A unique signature for a user is generated to authenticate registration requests. This involves creating a cryptographic digest from the user's address, the amount of tokens, the **launchpad** contract address, and the blockchain ID.
- This digest is then signed using the admin's private key, and the resulting signature is encoded and returned.

Simulating Time Flow and Registration:

- The blockchain's virtual time (`block.timestamp`) is fast-forwarded to just after the registration period starts, allowing the registration function to be tested.
- A user (`address(1)`) is set up to register for a specific tier (in this case, **BRONZE**) by providing the amount of tokens needed for that tier, along with the generated signature.

Registration Function Execution:

- The user registers for the token sale using the **USDT** token by calling the **register** function with the previously obtained signature.
- The same user then attempts to use the same signature to register again, this time using the **DAI** token, demonstrating that the signature can be reused for different tokens within the same registration parameters.

Verification of Registration:

- After both registration calls, the test
  See more

Results:

```
Compiler run successful!

Ran 1 test for test/launchpad/SignatureReplay.t.sol:SignatureReplayTest
[PASS] test_signatureReplay() (gas: 1037766)
Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 2.64ms (1.05ms CPU
time)

Ran 1 test suite in 146.67ms (2.64ms CPU time): 1 tests passed, 0 failed, 0 skip
ped (1 total tests)
```

F-2024-2028 - Strategic Splitting of Tokens Increases Allocation Unfairly - Medium

Description:

The current allocation mechanism within the Launchpad contract allows users to potentially exploit the tier-based weight system by splitting their tokens across multiple accounts. This can lead to disproportionate allocations during token sales, undermining the fairness and integrity of the distribution process.

In the Launchpad system, users are assigned weights based on the amount of tokens they stake, categorized into tiers like Bronze, Silver, and Gold. The allocation for token sales is calculated based on these weights. However, if a user strategically splits their tokens into multiple accounts such that each qualifies for a lower tier but collectively holds more weight, they can achieve a higher total allocation than if they had staked all their tokens in a single account.

Assuming there are two users, Alice and Bob, with the following conditions:

- **Bob:** Stakes 10,000 tokens, qualifying for the Gold tier with a weight of 50.
- **Alice:** Splits her 6,000 tokens into three accounts of 2,000 each, each qualifying for the Bronze tier with a weight of 20.

Calculations:

Allocation Formula:

```
weight * placedTokens[id].initialVolumeForLowTiers / placedTokens[id
].lowTiersWeightsSum - boughtAmount;
```

Without Splitting:

- **Alice's Allocation:** 30*1E18/80 =375,000,000,000,000,000
- **Bob's Allocation:** 50*E18/80 =625,000,000,000,000,000

With Splitting:

- **One of Alice's Accounts Allocation:** 20*1E18/110=181,818,181,818,181,818
- **Total for Alice's Three Accounts:** 3*181,818,181,818,181,818=545,454,545,454,545,454
- **Bob's Allocation Now:** 50×1E18/110= 454,545,454,545,545,545

In this scenario, Alice's total allocation increases from 375,000,000,000,000,000 to 545,454,545,454,545,454 by splitting her tokens, and Bob's allocation decreases from 625,000,000,000,000,000 to 454,545,454,545,545,545 due to the increased number of total weights in the system. This issue leads to an unfair advantage for users who split

their tokens and can manipulate the system to gain higher allocations, thereby diluting the allocations for other users who might have staked more tokens in a single account.

Assets:

- LaunchpadV2.sol [https://github.com/blastupio/launchpad-contracts/]

Status: Fixed

Classification

Impact: 3/5

Likelihood: 4/5

Exploitability: Independent

Complexity: Complex

Likelihood [1-5]: 3

Impact [1-5]: 4

Exploitability [0-2]: 0

Complexity [0-2]: 2

Final Score: 3.1 (Medium)

Hacken Calculator Version: 0.6

Severity: Medium

Recommendations

Remediation: Consider restructuring the `minAmountForTier` and `weightForTier` values, or implementing a different mechanism for calculating tier weights.

Resolution: The BlastUP team fixed the issue in commit **be1e27d** by implementing new weights for the tiers.

Evidences

POC

Results:

```
[PASS] test_registerV2AllocationCheck() (gas: 1261618)
Logs:
  Alice Account1 Allocation:  181818181818181818
  Alice Account2 Allocation:  181818181818181818
  Alice Account3 Allocation:  181818181818181818
  Bob Total Allocation   :  454545454545454545
  Alice Total Allocation:  545454545454545454

Suite result: ok. 1 passed; 0 failed; 0 skipped; finished in 7.87ms (1.39ms CPU
time)
```

Files: LaunchpadUpgrades.t.sol

F-2024-1515 - Chainlink's latestRoundData() Might Return Stale or Incorrect Results - Low

Description:

The `_convertETHToUSDB()` function calls out to a Chainlink oracle receiving the `latestRoundData()`. If there is a problem with Chainlink starting a new round and finding consensus on the new value for the oracle (e.g. Chainlink nodes abandon the oracle, chain congestion, vulnerability/attacks on the chainlink system) consumers of this contract may continue using outdated stale or incorrect data (if oracles are unable to submit no new round is started).

```
function _convertETHToUSDB(uint256 volume) private view returns (uin
t256) {
// price * volume * real_usdb_decimals / (eth_decimals * oracle_deci
mals)
(, int256 ans,,,) = oracle.latestRoundData();
return uint256(ans) * volume * (10 ** decimalsUSDB) / (10 ** oracleD
ecimals) / (10 ** 18);
}
```

Assets:

- Launchpad.sol [https://github.com/blastupio/launchpad-contracts]
- YieldStaking.sol [https://github.com/blastupio/launchpad-contracts]

Status: Fixed

Classification

Impact: 3/5

Likelihood: 3/5

Exploitability: Independent

Complexity: Simple

Likelihood [1-5]: 3

Impact [1-5]: 3

Exploitability [1,2]: 1

Complexity [0-2]: 0

Final Score: 2.4082246852806923 [Low]

Severity: Low

Recommendations

Remediation: Add checks and timeout mechanisms to make sure the acquired result is the latest one.

Example:

```
function _convertETHToUSDB(uint256 volume) private view returns (uin
t256) {
(uint80 roundID, int256 price, , uint256 timestamp, uint80 answeredI
nRound) = oracle.latestRoundData();
require(answeredInRound >= roundID, "Stale price");
require(timestamp != 0,"Round not complete");
require(price > 0,"Chainlink price reporting 0");
}
```

Resolution: The BlastUP team fixed the issue in commit **f47890b** by adding the recommended checks.

F-2024-1534 - Inconsistent Validation Checks Across Token Setting Functions - Low

Description:

There is a inconsistency in the validation checks implemented across different functions responsible for setting token-related time stamps and volumes. While the `placeTokens()` function incorporates a comprehensive set of validations to ensure some parameters are within expected bounds and logical sequencing, similar stringent checks are noticeably absent in individual setting functions such as `setRegistrationStart()`, `setRegistrationEnd()`, etc.

```
function placeTokens(PlacedToken memory _placedToken, address token)
external onlyOwner {
…
require(
_placedToken.registrationStart > block.timestamp
&& _placedToken.registrationEnd > _placedToken.registrationStart
&& _placedToken.publicSaleStart > _placedToken.registrationEnd
&& _placedToken.fcfsSaleStart > _placedToken.publicSaleStart
&& _placedToken.saleEnd > _placedToken.fcfsSaleStart && _placedToken
.tgeStart > _placedToken.saleEnd
&& _placedToken.vestingStart > _placedToken.tgeStart,
"BlastUP: invalid timestamps"
);
…
}

function setRegistrationStart(address token, uint256 _registrationSt
art) external onlyOperatorOrOwner {
require(_registrationStart > block.timestamp, "BlastUP: invalid regi
startion start timestamp");
placedTokens[token].registrationStart = _registrationStart;
}

function setRegistrationEnd(address token, uint256 _registrationEnd)
external onlyOperatorOrOwner {
require(_registrationEnd > block.timestamp, "BlastUP: invalid regist
artion end timestamp");
placedTokens[token].registrationEnd = _registrationEnd;
}
```

Currently, the functions validate that the input timestamps are future-dated but do not verify the chronological order between them. For example, the registration end date could be set to occur before the registration start date if not checked properly. This oversight may lead to scenarios where token sale phases overlap or occur in an unintended order, leading to operational disruptions and participant confusion.

Assets:

- Launchpad.sol [https://github.com/blastupio/launchpad-contracts]

Status: Accepted

Classification

Impact: 3/5

Likelihood: 2/5

Exploitability: Dependent

Complexity: Simple

Likelihood [1-5]: 3

Impact [1-5]: 2

Exploitability [0-2]: 2

Complexity [0-2]: 0

Final Score: 1.8 (Low)

Hacken Calculator Version: 0.6

Severity: Low

Recommendations

Remediation: To address this issue, implement comprehensive chronological validation across related timestamp-setting functions to ensure logical sequencing.

Resolution: The BlastUp team acknowledged the issue with the following statement;

> Admins are assumed to be trusted and there will always be possibility that admins enter unexpected values even if we have more validation checks. Thus, we prefer to keep this as is without introducing additional complexity.

F-2024-1540 - Centralization Risk For Privileged Actors - Low

Description:

The Launchpad smart contract currently permits the owner or operator to modify tier settings, such as weights and minimum required amounts, through `setMinAmountsForTiers()` and `setWeightsForTiers()` functions without any user permission at any time.

```
function setMinAmountsForTiers(uint256[6] memory amounts) external o
nlyOperatorOrOwner {
minAmountForTier[UserTiers.BRONZE] = amounts[0];
minAmountForTier[UserTiers.SILVER] = amounts[1];
minAmountForTier[UserTiers.GOLD] = amounts[2];
minAmountForTier[UserTiers.TITANIUM] = amounts[3];
minAmountForTier[UserTiers.PLATINUM] = amounts[4];
minAmountForTier[UserTiers.DIAMOND] = amounts[5];
}

function setWeightsForTiers(uint256[6] memory weights) external only
OperatorOrOwner {
weightForTier[UserTiers.BRONZE] = weights[0];
weightForTier[UserTiers.SILVER] = weights[1];
weightForTier[UserTiers.GOLD] = weights[2];
weightForTier[UserTiers.TITANIUM] = weights[3];
weightForTier[UserTiers.PLATINUM] = weights[4];
weightForTier[UserTiers.DIAMOND] = weights[5];
}
```

This could allow these values to be set to zero or impractically low figures, potentially undermining the fairness and integrity of the token sale process.

Additionally, the contract design allows the owner or operator to effectively pause token claims post-vesting commencement by setting the `tgeStart` to a future date unexpectedly through the `setTgeStart()` function. This action renders the `getClaimableAmount()` function to calculate zero for all users, preventing any claims from being processed.

```
function getClaimableAmount(uint256 id, address user) public view re
turns (uint256) {
…
if (block.timestamp < placedTokens[id].tgeStart) return 0;
…
}
```

```
function claimTokens(uint256 id) external {
Types.PlacedToken storage placedToken = placedTokens[id];
Types.User storage user = users[id][msg.sender];
uint256 claimableAmount = getClaimableAmount(id, msg.sender);
require(claimableAmount > 0, "BlastUP: you have not enough claimable
tokens");
user.claimedAmount += claimableAmount;
IERC20(placedToken.token).safeTransfer(msg.sender, claimableAmount);
emit TokensClaimed(placedToken.token, id, msg.sender);
}
```

Assets:

- Launchpad.sol [https://github.com/blastupio/launchpad-contracts]

Status: Mitigated

Classification

Impact: 1/5

Likelihood: 2/5

Exploitability: Dependent

Complexity: Simple

Likelihood [1-5]: 2

Impact [1-5]: 3

Exploitability [0-2]: 2

Complexity [0-2]: 0

Final Score: 1.8 (Low)

Severity: Low

Recommendations

Remediation: To mitigate centralization risks associated with privileged functions, consider implementing a multi-signature or decentralized governance mechanism. Instead of relying solely on a single owner/administrator, distribute control and decision-making authority among multiple parties or stakeholders. This approach enhances security, reduces the risk of malicious actions by a single entity, and prevents single points of failure.

Resolution: The BlastUP team mitigated the issue with the following statement;

> All roles will be set to a multisignature contract, thus decreasing risk on exploit.

F-2024-1543 - Lack of Boundary Checks - Low

Description:

The **Launchpad** contract lacks a boundary check on the `tgePercent` variable when it is set. This variable represents the percentage of tokens available for immediate claim at the Token Generation Event (TGE). The absence of a validation check to ensure `tgePercent` does not exceed 100% can lead to logical errors in the `getClaimableAmount()` function, potentially causing incorrect calculations.

The `getClaimableAmount()` function calculates the number of tokens a user can claim based on their initial allocation and the percentages set for immediate availability at TGE and subsequent vesting. The critical section of code is:

```
function getClaimableAmount(address token, address user) public view
returns (uint256) {
uint256 tgeAmount = users[token][user].boughtAmount * placedTokens[t
oken].tgePercent / 100;
…
}
```

Here, `tgePercent` should logically represent a percentage (i.e., a value between 0 and 100 inclusive), but the contract does not enforce this range when `tgePercent` is set.

Additionally, the **YieldStaking** contract allows the owner to set minTimeToWithdraw, which defines the minimum time a user must wait before withdrawing their staked assets. Currently, there is no upper boundary check on this value, which might enable the owner to set an unreasonably high withdrawal time, effectively locking users' assets indefinitely.

```
function setMinTimeToWithdraw(uint256 _minTimeToWithdraw) external o
nlyOwner {
minTimeToWithdraw = _minTimeToWithdraw;
}
```

Assets:

- Launchpad.sol [https://github.com/blastupio/launchpad-contracts]
- YieldStaking.sol [https://github.com/blastupio/launchpad-contracts]

Status: Accepted

Classification

Impact: 2/5

Likelihood: 2/5

Exploitability: Independent

Complexity: Simple

Likelihood [1-5]: 2

Impact [1-5]: 2

Exploitability [0-2]: 1

Complexity [0-2]: 0

Final Score: 1.7 (Low)

Hacken Calculator Version: 0.6

Severity: Low

Recommendations

Remediation:

1. Implement a validation check within the function or process that sets `tgePercent` to ensure it does not exceed 100%
2. Implement a sensible upper limit for `minTimeToWithdraw`.

Resolution: The BlastUP team acknowledged the issue with the following statement;

> Admins are assumed to be trusted and there will always be possibility that admins enter unexpected values even if we have more validation checks. Thus, we prefer to keep this as is without introducing additional complexity.

F-2024-1545 - Missing Validation of Tier Hierarchy - Low

Description:

The **Launchpad** smart contract lacks validations in the functions `setMinAmountsForTiers()` and `setWeightsForTiers()`, which allow setting the tier weights and minimum amounts. Specifically, the contract does not enforce that higher tiers (e.g., SILVER, GOLD) must have higher weight and minimum amount requirements compared to lower tiers (e.g., BRONZE). This absence of hierarchical validation could lead to a tier configuration where a higher tier has less influence or requirements than a lower tier, contradicting the intended progression and incentives of a tiered system.

Affected Code:

```
function setMinAmountsForTiers(uint256[6] memory amounts) external o
nlyOperatorOrOwner {
minAmountForTier[UserTiers.BRONZE] = amounts[0];
minAmountForTier[UserTiers.SILVER] = amounts[1];
minAmountForTier[UserTiers.GOLD] = amounts[2];
minAmountForTier[UserTiers.TITANIUM] = amounts[3];
minAmountForTier[UserTiers.PLATINUM] = amounts[4];
minAmountForTier[UserTiers.DIAMOND] = amounts[5];
}

function setWeightsForTiers(uint256[6] memory weights) external only
OperatorOrOwner {
weightForTier[UserTiers.BRONZE] = weights[0];
weightForTier[UserTiers.SILVER] = weights[1];
weightForTier[UserTiers.GOLD] = weights[2];
weightForTier[UserTiers.TITANIUM] = weights[3];
weightForTier[UserTiers.PLATINUM] = weights[4];
weightForTier[UserTiers.DIAMOND] = weights[5];
}
```

Assets:

- Launchpad.sol [https://github.com/blastupio/launchpad-contracts]

Status: Accepted

Classification

Impact: 2/5

Likelihood: 3/5

Exploitability: Dependent

Complexity: Simple

Likelihood [1-5]: 2

Impact [1-5]: 3

Exploitability [0-2]: 2

Complexity [0-2]: 0

Final Score: 1.8 (Low)

Hacken Calculator Version: 0.6

Severity: Low

Recommendations

Remediation: Implement validation logic in `setMinAmountsForTiers` and `setWeightsForTiers` to ensure that each tier's requirements and weights do not decrease as tiers progress. The validation should enforce that each tier's minimum amount and weight are greater than or equal to the previous tier

Resolution: The BlastUP team acknowledged the issue with the following statement;

> Admins are assumed to be trusted and there will always be possibility that admins enter unexpected values even if we have more validation checks. Thus, we prefer to keep this as is without introducing additional complexity..

F-2024-1735 - Privileged Functions Susceptible to Front-Running - Low

Description: The **Launchpad** and **YieldStaking** smart contracts contain functions (`setMinAmountsForTiers()`, `setWeightsForTiers()`, `setMinTimeToWithdraw()`, `setMinUSDBStakeValue()`, `setRegistrationStart()`, `setPublicSaleStart()`, `setFCFSSaleStart()`, `setTgeStart()`, `setVestingStart()`) that are vulnerable to front-running. This occurs because these functions modify important contract state variables that affect the outcome of other contract functions. If these setter functions are front-run by other transactions, users can take advantage of older values before the new settings take effect, leading to potential abuses.

Assets:

- Launchpad.sol [https://github.com/blastupio/launchpad-contracts]
- YieldStaking.sol [https://github.com/blastupio/launchpad-contracts]

Status: Accepted

Classification

Impact: 3/5

Likelihood: 3/5

Exploitability: Independent

Complexity: Simple

Likelihood [1-5]: 3
Impact [1-5]: 3
Exploitability [1,2]: 1
Complexity [0-2]: 0
Final Score: 2.4082246852806923 [Low]
Hacken Calculator Version: 0.6

Severity: Low

Recommendations

Remediation: Implement measures to ensure that the contract is pausable and only execute these functions when the contract is in a paused state.

Resolution: The BlastUP team acknowledged the issue with the following statement;

Requiring two-step pause process for all admin functionality execution will add more complexity without much benefit/decrease of risk.

F-2024-1738 - `decimals()` is not a part of the `ERC-20` standard - Low

Description:

The `decimals()` function is not a part of the ERC-20 standard, and was added later as an optional extension. As such, some valid ERC20 tokens do not support this interface, so it is unsafe to blindly cast all tokens to this interface, and then call this function.

Affected code:
./contracts/Launchpad.sol

```
require(_placedToken.tokenDecimals == IERC20Metadata(token).decimals
(), "BlastUP: invalid decimals");
```

Assets:

- Launchpad.sol [https://github.com/blastupio/launchpad-contracts]
- YieldStaking.sol [https://github.com/blastupio/launchpad-contracts]

Status: Accepted

Classification

Impact: 4/5

Likelihood: 2/5

Exploitability: Dependent

Complexity: Simple

Likelihood [1-5]: 2
Impact [1-5]: 4
Exploitability [0-2]: 2
Complexity [0-2]: 0
Final Score: 2.1 (Low)
Hacken Calculator Version: 0.6

Severity: Low

Recommendations

Remediation:

When working with ERC-20 tokens in your Solidity code, be aware that the `decimals()` function is not a part of the ERC-20 standard and is considered an optional extension. Not all valid ERC-20 tokens implement this interface, so avoid blindly casting all tokens to this interface and calling the `decimals()` function. Instead, check the token's

documentation or contract to determine whether it supports this extension before using it.

Resolution: The BlastUP team acknowledged the issue with the following statement;

> There is no plans for supporting tokens which do not have decimals() method. Every external token will be reviewed by admins and can only be added by them through placeTokens() function

F-2024-2040 - Inappropriate Handling of Decimal Precision for Tier Minimum Amounts - Low

Description:

The smart contract specifies hardcoded minimum amounts for different user tiers without considering the token's decimal precision. This leads to a significant issue where the token used in **BLPStaking**, which has 18 decimals, does not correspond correctly with the tier minimum requirements set in the **Launchpad** contract. As per the current setup, anyone staking merely 1 token (1 * 10^18 in terms of smallest units) could potentially qualify for the highest tier (Diamond), due to a misunderstanding of token decimals in tier minimum definitions.

Affected Code:

```
function initialize(address _owner, address _signer, address _operat
or, address _points, address _pointsOperator) public initializer {
...
minAmountForTier[Types.UserTiers.BRONZE] = 2_000;
minAmountForTier[Types.UserTiers.SILVER] = 5_000;
minAmountForTier[Types.UserTiers.GOLD] = 10_000;
minAmountForTier[Types.UserTiers.TITANIUM] = 20_000;
minAmountForTier[Types.UserTiers.PLATINUM] = 50_000;
minAmountForTier[Types.UserTiers.DIAMOND] = 150_000;
...
}
```

Assets:

- Launchpad.sol [https://github.com/blastupio/launchpad-contracts]
- LaunchpadV2.sol [https://github.com/blastupio/launchpad-contracts/]

Status: Fixed

Classification

Impact: 3/5

Likelihood: 2/5

Exploitability: Independent

Complexity: Simple

Likelihood [1-5]: 3

Impact [1-5]: 2

Exploitability [0-2]: 0

Complexity [0-2]: 0

Final Score: 2.5 (Low)

Hacken Calculator Version: 0.6

Severity: Low

Recommendations

Remediation: The tier minimum amounts should be adjusted according to the token's decimals.

Resolution: The BlastUP team fixed the issue in commit **2e05419** by implementing **10^18** precision for minimum tier amounts.

F-2024-2043 - Fee-On-Transfer Token Handling Flaw - Low

Description:

The Launchpad smart contract interacts with ERC-20 tokens in a manner that assumes the transfer methods `safeTransferFrom()` move exactly the amount specified by the caller. This assumption can lead to issues when interacting with fee-on-transfer tokens, where a portion of the tokens transferred is deducted as a fee. The function `claimRemainders()` in the contract attempts to transfer the total remaining tokens (`volume`) back to the contract owner after a sale. If the token being handled deducts a fee on transfer, the actual amount sent will be less than volume, potentially causing the transfer to revert if the contract's balance is less than the specified volume.

One way to address this problem is to measure the balance before and after the transfer, and use the difference as the amount, rather than the stated amount.

Affected code:
./contracts/LaunchpadV2.sol

```
246: IERC20(token).safeTransferFrom(msg.sender, address(this), sumVo
lume);
318: IERC20(paymentContract).safeTransferFrom(msg.sender, placedToke
n.addressForCollected, volume);
```

Assets:

- Launchpad.sol [https://github.com/blastupio/launchpad-contracts]

Status: Accepted

Classification

Impact: 4/5

Likelihood: 2/5

Exploitability: Semi-Dependent

Complexity: Simple

Likelihood [1-5]: 2
Impact [1-5]: 4
Exploitability [0-2]: 1
Complexity [0-2]: 0
Final Score: 2.4(Low)
Hacken Calculator Version: 0.6

Severity: Low

Recommendations

Remediation: To mitigate potential vulnerabilities and ensure accurate accounting with fee-on-transfer tokens, modify your contract's token transfer logic to measure the recipient's balance before and after the transfer. Use this observed difference as the actual transferred amount for any further logic or calculations. For example:

```
function transferTokenAndPerformAction(address token, address from,
address to, uint256 amount) public {
uint256 balanceBefore = IERC20(token).balanceOf(to);

// Perform the token transfer
IERC20(token).transferFrom(from, to, amount);

uint256 balanceAfter = IERC20(token).balanceOf(to);
uint256 actualReceived = balanceAfter - balanceBefore;

// Proceed with logic using actualReceived instead of the initial am
ount
require(actualReceived >= minimumRequiredAmount, "Received amount is
less than required");

// Further logic here
}
```

Resolution: The BlastUP team acknowledged the issue with the following statement;

> There are no plans to support tokens that have a fee on transfer. Every external token will be reviewed by admins and can only be added by them through the placeTokens() function

F-2024-1609 - Inconsistency in Token Allocation Post-Claim - Info

Description:

The `claimRemainders()` function within the **Launchpad** contract is designed to transfer any unclaimed tokens to the DAO's address post-sale. However, this function sets the `placedTokens[token].volume` to zero without appropriately resetting the `placedTokens[token].initialVolumeForHighTiers` or `placedTokens[token].initialVolumeForLowTiers`.

```
function claimRemainders(uint256 id) external onlyOperatorOrOwner {
...
uint256 volume = placedToken.volume;
placedToken.volume = 0;
placedToken.volumeForYieldStakers = 0;
...
}
```

This oversight leads to inconsistent behavior in the `userAllowedAllocation()` function, particularly affecting users in higher tiers (e.g., Titanium or above), who will see their potential token allocations drop to zero, whereas users in lower tiers may continue to receive allocations based on unchanged initial volumes.

```
function userAllowedAllocation(uint256 id, address user) public view
returns (uint256) {
if (!users[id][user].registered) return 0;
if (getStatus(id) == Types.SaleStatus.PUBLIC_SALE) {
Types.UserTiers tier = users[id][user].tier;
uint256 weight = weightForTier[tier];
uint256 boughtAmount = users[id][user].boughtPublicSale;
if (users[id][user].tier < Types.UserTiers.TITANIUM) {
return weight * placedTokens[id].initialVolumeForLowTiers / placedTo
kens[id].lowTiersWeightsSum
- boughtAmount;
} else {
return weight * placedTokens[id].initialVolumeForHighTiers / placedT
okens[id].highTiersWeightsSum
- boughtAmount;
}
} else if (users[id][user].tier >= Types.UserTiers.TITANIUM) {
return placedTokens[id].volume;
} else {
return 0;
}
}
```

Assets:

- Launchpad.sol [https://github.com/blastupio/launchpad-contracts]

Status: Accepted

Classification

Impact: 3/5

Likelihood: 2/5

Exploitability: Semi-Dependent

Complexity: Simple

Likelihood [1-5]: 2

Impact [1-5]: 2

Exploitability [0-2]: 1

Complexity [0-2]: 1

Final Score: 1.6 (Informational)

Hacken Calculator Version: 0.6

Severity: Info

Recommendations

Remediation: To resolve this inconsistency, ensure that all related volume parameters are reset or appropriately adjusted when `claimRemainders()` is executed.

Observation Details

F-2024-1732 - Missing Events - Info

Description:

Events for critical state changes should be emitted for tracking actions off-chain.

It was observed that events are missing in the following functions:

```
setSigner()
setOperator()
setRegistrationStart()
setRegistrationEnd()
setPublicSaleStart()
setFCFSSaleStart()
setSaleEnd()
setTgeStart()
setVestingStart()
setMinTimeToWithdraw()
setMinUSDBStakeValue()
```

Events are crucial for tracking changes on the blockchain, especially for actions that alter significant contract states or permissions. The absence of events in these functions means that external entities, such as user interfaces or off-chain monitoring systems, cannot effectively track these important changes.

Assets:

- Launchpad.sol [https://github.com/blastupio/launchpad-contracts]
- YieldStaking.sol [https://github.com/blastupio/launchpad-contracts]

Status: Accepted

Recommendations

Remediation: Consider implementing and emitting events for the necessary functions.

F-2024-1733 - Use `Ownable2Step` rather than `Ownable - Info

Description:

Ownable2Step and Ownable2StepUpgradeable prevent the contract ownership from mistakenly being transferred to an address that cannot handle it (e.g. due to a typo in the address), by requiring that the recipient of the owner permissions actively accept via a contract call of its own.

```
contract YieldStaking is OwnableUpgradeable{...}
```

```
contract Launchpad is OwnableUpgradeable, ILaunchpad {...}
```

Assets:

- Launchpad.sol [https://github.com/blastupio/launchpad-contracts]
- YieldStaking.sol [https://github.com/blastupio/launchpad-contracts]

Status: Accepted

Recommendations

Remediation:

Consider using `Ownable2Step` or `Ownable2StepUpgradeable` from OpenZeppelin Contracts to enhance the security of your contract ownership management. These contracts prevent the accidental transfer of ownership to an address that cannot handle it, such as due to a typo, by requiring the recipient of owner permissions to actively accept ownership via a contract call. This two-step ownership transfer process adds an additional layer of security to your contract's ownership management.

F-2024-1734 - Missing Zero Address Validation - Info

Description:

In Solidity, the Ethereum address `0x0000000000000000000000000000000000000000` is known as the "**zero address**". This address has significance because it is the default value for uninitialized address variables and is often used to represent an invalid or non-existent address.

The "**Missing zero address control**" issue arises when a Solidity smart contract does not properly check or prevent interactions with the zero address, leading to unintended behavior.

For instance, a contract might allow tokens to be sent to the zero address without any checks, which essentially burns those tokens as they become irretrievable. While sometimes this is intentional, without proper control or checks, accidental transfers could occur.

Missing checks were observed in the following functions of the `Launchpad`, `LaunchpadV2` and `YieldStaking` contracts:

- `./Launchpad.sol: constructor()`
- `./Launchpad.sol: initialize()`
- `./Launchpad.sol: setSigner()`
- `./Launchpad.sol: setOperator()`
- `./Launchpad.sol: setRegistrationStart()`
- `./Launchpad.sol: setRegistrationEnd()`
- `./Launchpad.sol: setPublicSaleStart()`
- `./Launchpad.sol: setFCFSSaleStart()`
- `./Launchpad.sol: setSaleEnd()`
- `./Launchpad.sol: setTgeStart()`
- `./Launchpad.sol: setVestingStart()`
- `./Launchpad.sol: placeTokens()`
- `./LaunchpadV2.sol: constructor()`
- `./LaunchpadV2.sol: initialize()`
- `./YieldStaking.sol: constructor()`
- `./YieldStaking.sol: initialize()`

Assets:

- Launchpad.sol [https://github.com/blastupio/launchpad-contracts]
- YieldStaking.sol [https://github.com/blastupio/launchpad-contracts]
- LaunchpadV2.sol [https://github.com/blastupio/launchpad-contracts/]

Status: Accepted

Recommendations

Remediation: Implement zero address validation for the given parameters. This can be achieved by adding require statements that ensure address parameters are not the zero address.

F-2024-1737 - Floating pragma - Info

Description: The project uses floating pragma `^0.8.25`.

This may result in the contracts being deployed using the wrong pragma version, which is different from the one they were tested with. For example, they might be deployed using an outdated pragma version which may include bugs that affect the system negatively.

Assets:

- Launchpad.sol [https://github.com/blastupio/launchpad-contracts]
- YieldStaking.sol [https://github.com/blastupio/launchpad-contracts]
- LaunchpadV2.sol [https://github.com/blastupio/launchpad-contracts/]

Status: Accepted

Recommendations

Remediation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment. Consider known bugs for the compiler version that is chosen.

F-2024-1739 - Redundant Error Declaration - Info

Description:

The `Launchpad` contract defines the error `InvalidSaleStatus`, but it is not utilized in its respective implementations. This suggests a possible oversight or inconsistency in the contract designs.

The redundancy in error declarations can lead to unnecessary gas consumption during deployment and may impact the overall code quality.

Assets:

- Launchpad.sol [https://github.com/blastupio/launchpad-contracts]

Status: Fixed

Recommendations

Remediation: Remove the redundant error.

Resolution: The BlastUP team fixed the issue in commit **fb4f1a4** by removing the redundant error.

F-2024-1742 - Lack of Zero Amount Check - Info

Description: In the `stake()` , `claimReward()` , `withdraw()` functions of a **YieldStaking** contract, there is a notable absence of checks to ensure that the amount of token being staked is greater than zero. This oversight can lead to unnecessary execution of these functions when the amount is zero, potentially causing redundant Gas expenditure and affecting the contract's efficiency.

Assets:

- YieldStaking.sol [https://github.com/blastupio/launchpad-contracts]

Status: Accepted

Recommendations

Remediation: It is recommended to add a check for zero amounts in mentioned functions. This will prevent the functions from executing when the staking amount is zero.

F-2024-2027 - Unrestricted Tier Downgrade in Registration Functions - Info

Description: The registration functions `register()`, `registerWithApprove()`, `registerV2()`, `registerV2WithApprove()` in the **LaunchpadV2** and **Launchpad** contracts allows users to register themselves by specifying their tier, which is provided directly by the user during the function call. Currently, there is no validation mechanism to ensure that the tier value submitted by the user corresponds appropriately to their actual eligibility or previous tier level. This oversight can lead to situations where a user with a higher tier might accidentally register themselves under a lower tier, with no possibility of correcting this error since the contract does not support tier modification or unregistering once registered.

Assets:

- Launchpad.sol [https://github.com/blastupio/launchpad-contracts]
- LaunchpadV2.sol [https://github.com/blastupio/launchpad-contracts/]

Status: Accepted

Recommendations

Remediation: Implement tier validation within the registration functions to verify that the tier parameter provided by the user does not conflict with their actual tier.

F-2024-2029 - Redundant import - Info

Description: The use of unnecessary imports will increase the Gas consumption of the code. Thus they should be removed.

Assets:

- LaunchpadV2.sol [https://github.com/blastupio/launchpad-contracts/]

Status: Fixed

Recommendations

Remediation: Remove the *"./interfaces/IChainlinkOracle.sol"* import.

Resolution: The BlastUP team fixed the issue in commit **b3020b0** by removing the redundant import.

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed based on best industry practices at the time of the writing of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements.

While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the Consultant cannot guarantee the explicit security of the audited smart contracts.

Appendix 1. Severity Definitions

When auditing smart contracts, Hacken is using a risk-based approach that considers **Likelihood**, **Impact**, **Exploitability** and **Complexity** metrics to evaluate findings and score severities.

Reference on how risk scoring is done is available through the repository in our Github organization:

hknio/severity-formula

Severity	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to the loss of user funds or contract state manipulation.
High	High vulnerabilities are usually harder to exploit, requiring specific conditions, or have a more limited scope, but can still lead to the loss of user funds or contract state manipulation.
Medium	Medium vulnerabilities are usually limited to state manipulations and, in most cases, cannot lead to asset loss. Contradictions and requirements violations. Major deviations from best practices are also in this category.
Low	Major deviations from best practices or major Gas inefficiency. These issues will not have a significant impact on code execution, do not affect security score but can affect code quality score.

Appendix 2. Scope

The scope of the project includes the following smart contracts from the provided repository:

Scope Details

Repository	https://github.com/blastupio/launchpad-contracts/tree/master
Commit	cb6957dde5944e6cfd885ceed8539140dce51a98
Whitepaper	N/A
Requirements	https://docs.blastup.io/blastup-docs
Technical Requirements	Confidential

Contracts in Scope

./contracts/Launchpad.sol

./contracts/YieldStaking.sol

./contracts/LaunchpadV2.sol