RISK AUDIT

for

blockasset.

on

January 28, 2025

FIDESIUM

Executive Summary

Report

36
/100

TOTAL

Medium risk

January 28, 2025

Abstract

Fidesium's automated risk assessment service was requested to perform a risk posture audit on Block Asset **contracts**

Repository Link:
https://github.com/Blockasset/blockasset-labs

Initial Commit Hash:

`8186bf567554bb7cee7ee27dd2182820548c7c82`

Issue Summary

Critical	High	Medium	Low	Info
2 Issues	2 Issues	8 Issues	2 Issues	0 Issues

Caveats

Block Asset's codebase is well written, but does incur a handful of high value flaws.

Test Approach

Fidesium performed both Whitebox and Blackbox testing, as per the scope of the engagement, and relied on automated security testing.

Methodology

The assessment methodology covered a range of phases and employed various tools, including but not limited to the following:

- Mapping Content and Functionality of API
- Application Logic Flaws
- Access Handling
- Authentication/Authorization Flaws
- Brute Force Attempt
- Input Handling
- Source Code Review
- Fuzzing of all input parameter
- Dependency Analysis

Severity Definitions

Critical	The issue can cause large economic losses, large-scale data disorder or loss of control of authority management.
High	The issue puts users' sensitive information at risk or is likely to lead to catastrophic financial implications.
Medium	The issue puts a subset of users' sensitive information at risk, reputation damage or moderate financial impact.
Low	The risk is relatively small and could not be exploited on a recurring basis, or is low-impact to the client's business.
Informational	The issue does not pose an immediate risk but is relevant to security best practices or defence in Depth.

Risk Issues

Vunerability	Description	Risk	Probability	Status
Reliance on Blockhash as source of randomness	The `resolve_raffle.rs` program relies on blockhash as a source of randomness.	Critical	Low	Active
Unlimited Account Sizing	The `add_to_group.rs` allows for unbounded account size increases.	Critical	Medium	Active
Lack of pausability	The staking programs lack pausability logic, limiting the ability to respond in an emergency	High	Medium	Active
Authentication Bypass	`init_auction` allows authentication bypass under certain conditions.	High	Low	Active
Reliance on Validator Timestamp subject to clockdrift	The `claim_group_rewards.rs` and `claim_stake_rewards.rs` programs rely on validator clock time.	Medium	Medium	Active
PDA Index Reuse	The `stake.rs` program allows for PDA reuse.	Medium	Medium	Active
Reentrancy vulnerability	The `claim_group_rewards.rs` program updates state after transfer.	Medium	Medium	Active
Missing authority revocation	The `init_raffle.rs` program relies on authorities for access control, but does not provide a method for updating or revoking authority.	Medium	Medium	Active
Front Running: Slippage Protection	`redeem_tickets` does not implement slipapge protection on token operations.	Medium	Medium	Active
Time based Race condition	`bid_on_auction` has a race condition due to the `ACTIVE_DURATION_SECONDS` window.	Medium	Medium	Active
Missing token account validation	`close_auction` lacks token account validation.	Medium	Medium	Active
Metadata Validation Gap	`init_auction_programmable` lacks sufficient metadata validation.	Medium	Low	Active
Reliance on Clock time	Multiple contracts rely on Clock time `Clock::get()`.	Low	Low	Active
Token fungibility assumption	The `init_token_identifier.rs` program defaults to an assumption of token fungibility.	Low	Medium	Active

Risk Overview

Team Risk

Low risk: 1

No issues found in founding team

Doxxing Status	Team Experience	Risk Summary
Public	Highly relevant	Low

Liquidity

Risk summary: N/A

As this is a Github assessment, liquidity risks have not been assessed

Whale Concentration

Risk summary: N/A

As this is a Github assessment, whale risks have not been assessed

Smart Contract Risks

Risk summary: 46

The contracts are mostly well written, but have a handful of flaws that should to be carefuly managed.

Vulnerabilities Critical

Reliance on Blockhash as source of randomness

Vulnerability severity: **Critical**

Vulnerability probability: **Low**

The `resolve_raffle.rs` program relies on blockhash as a source of randomness.

```
let randomness = last_blockhash_accessor(&ctx.accounts.recent_blockhashes)?;
let winner_index = expand(randomness) % tickets.total;
```

Blockhashes can be manipulated or predicted by validators and are not a good source of randomness, additionally a malicious validator could simulate the transaction, and choose to include or exclude it based on whether they are happy with outcome

Recommendations:

Rely on Verifiable Random Functions through Switchboard:

```
use switchboard_v2::{VrfAccountData, VrfRequestRandomness};

#[derive(Accounts)]
pub struct ResolveWithVRF<'info> {
    #[account(mut)]
        pub raffle: Account<'info, Raffle>,
        pub vrf: AccountLoader<'info, VrfAccountData>,
        ...
        ...
        ...

pub fn resolve_raffle(ctx: Context) -> Result<()> {
    let vrf = ctx.accounts.vrf.load()?;

    if !vrf.has_result() {
        return Err(error!(ErrorCode::VrfNotReady));
    }

    let result = vrf.get_result()?;
    let random_value = u64::from_le_bytes(result[0..8].try_into().unwrap());
    let winner_index = random_value % ctx.accounts.raffle.total_tickets;
    ....
```

Additionally, we would recommend spreading randomness generation and consumption across two transactions, allowing for consistent VRF processing, as well as provable randomness. If VRF requests are too pricy, we advise applyin a commit reveal scheme to randomness, and combining multiple sources of randomness such as the blockhash, seed, clocktime, and aggregated oracle feeds. This will still be less secure than using VRF.

Vulnerabilities Critical

Unlimited Account Sizing

Vulnerability severity: **Critical**

Vulnerability probability: **Medium**

The `add_to_group.rs` allows for unbounded account size increases.

```
resize_account(
    &ctx.accounts.group_entry.to_account_info(),
    ctx.accounts.group_entry.try_to_vec()?.len() + 32,  // Unbounded size!
    &ctx.accounts.payer.to_account_info(),
    &ctx.accounts.system_program.to_account_info(),
)?;
```

This could allow an attacker to continuously add items, forcing an account resize with every addition, and increasing the rent costs. By creating many maximum-sized accounts an attacker could trigger a DoS leading to a total protocol failure.

Recommendations:

- Limit the group entry size
- Limit the maximum number of mints per group

Vulnerabilities High

Lack of pausability

Vulnerability severity: **High**

Vulnerability probability: **Medium**

The staking programs lack pausability logic, limiting the ability to respond in an emergency

Recommendations:

- Limit the group entry size
- Limit the maximum number of mints per group

Token fungibility assumption

```
fn validate_token_metadata(
    metadata: &Account<'_, Metadata>,
    is_fungible: bool,
) -> Result<()> {
    require!(
        metadata.data.name.len() > 0,
        ErrorCode::InvalidTokenStandard
    );

    if is_fungible {
        require!(
            metadata.data.symbol.len() > 0,
            ErrorCode::InvalidTokenStandard
        );
        require!(
            metadata.data.uri.len() > 0,
            ErrorCode::InvalidTokenStandard
        );
    } else {
        require!(
            metadata.data.uri.len() > 0 &&
            metadata.data.uri.starts_with("https://"),
            ErrorCode::InvalidTokenStandard
        );
    }

    Ok(())
}
```

Vulnerabilities High

Authentication Bypass

Vulnerability severity: **High**

Vulnerability probability: **Low**

`init_auction` allows authentication bypass under certain conditions.

```
if !project.public && !project.authorities.contains(&authority.key()) {
    return Err(error!(ErrorCode::InvalidProjectAuthority));
}
```

An attacker could monitor for changes to `public`.

Additionally, if `public` were to change in another transaction, an attacker could identify old state and construct a transaction based on that state

Recommendations:

- Implement granular access controls
- Implement expiration time on authorities
- Implement time based authority validation to prevent stale state attacks
- Provide explicit error codes for granular authentication failures
- Revalidate state before any action

Vulnerabilities Medium

Reliance on Validator Timestamp subject to clockdrift

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

The `claim_group_rewards.rs` and `claim_stake_rewards.rs` programs rely on validator clock time.

```
let reward_seconds = end_time_stamp - start_time_stamp;
if reward_seconds <= 0 {
    return Ok(());
}
```

Validators can include timestamps which are slightly (25 seconds) out of sync with real time. An attacker could monitor validator timestamps, and frontrun these transactions

Recommendations:

- Rely on external time oracle. Ensure you validate against oracle poisoning by enforcing a weighted consensus, requiring multiple validators, and validating deviations against a median
- Implement Moving Time averages for all time sensitive computations
- Ensure a minimum time between time sensitive operations
- Introduce a maximum acceptable time deviation `require!( (now - expected_time).abs() <= MAX_TIME_DEVIATION, ErrorCode::SuspiciousTimeDeviation );`

PDA Index Reuse

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

The `stake.rs` program allows for PDA reuse.

This could lead to a repeated stake/unstake loop, and could potentially lead to economic manipulation, and bypass of cooldown periods

Recommendations:

- Track staking indices
- Introduce an index blacklist to prevent abuse
- Add index analytics for better monitoring and detection

Vulnerabilities Medium

Reentrancy vulnerability

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

The `claim_group_rewards.rs` program updates state after transfer.

```
transfer(
    CpiContext::new(
        ctx.accounts.token_program.to_account_info(),
        Transfer {
            from: ctx.accounts.staking_pool_reward_token_account.to_account_info(),
            to: ctx.accounts.staker_reward_token_account.to_account_info(),
            authority: ctx.accounts.staking_pool.to_account_info(),
        },
    )
    .with_signer(staking_pool_signer),
    claim_amount,
)?;
```

occurs before

```
ctx.accounts.grouping_vault.total_reward_paid = ctx
    .accounts
    .grouping_vault
    .total_reward_paid
    .checked_add(claim_amount)
    .ok_or(ErrorCode::NumericalOverflow)?;
```

An attacker could use a malicious contract that stakes tokens, and reenters claim_rewards with stale state

Recommendations:

- Update state before transfer, following the Check-Effects-Interaction pattern
- Implement a reentrancy guard `require!(!ctx.accounts.group_entry.is_claiming, ErrorCode::ClaimInProgress);`

Vulnerabilities Medium

Missing authority revocation

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

The `init_raffle.rs` program relies on authorities for access control, but does not provide a method for updating or revoking authority.

```
if !project.public && !project.authorities.contains(&authority.key()) {
    return Err(error!(ErrorCode::InvalidProjectAuthority));
}
```

If an authority private key is compromised, or an authority key is lost, this could lead to business continuity risk and/or total protocol failure.

Recommendations:

- Implement a robust authority management system, including authority add, update, and remove
- Implement multisig requirements

Vulnerabilities Medium

Front Running: Slippage Protection

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

`redeem_tickets` does not implement slipapge protection on token operations.

```
let cpi_accounts_transfer = token::Transfer {
    from: user_token_account.to_account_info(),
    to: raffle_token_account.to_account_info(),
    authority: user.to_account_info(),
};
let cpi_program_transfer = ctx.accounts.token_program.to_account_info();
let cpi_context_transfer = CpiContext::new(cpi_program_transfer, cpi_accounts_transfer);
token::transfer(cpi_context_transfer, total_ticket_fee)?;
```

An attacker could detect the transfer in the mempool, and then front/backrun this transaction.

Recommendations:

Implement slippage protection, using oracle feeds and locked in amounts.

Time based Race condition

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

`bid_on_auction` has a race condition due to the `ACTIVE_DURATION_SECONDS` window.

```
if (auction.end <= now) && (auction.updated_at + ACTIVE_DURATION_SECONDS <= now) {
    return Err(error!(ErrorCode::AuctionEnded));
}
```

An attacker could monitor an auction in the runup to close, and congest the network, frontrun winning bids, or use multiple accounts to drive up price.

Recommendations:

- Implement dynamic extension windows, based on auction activity
- Implement price velocity throttling
- Implement bid size restrictions

Vulnerabilities

Missing token account validation

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

`close_auction` lacks token account validation.

```
if bidder_token_account.mint != auction.treasury_mint
    || bidder_token_account.owner != auction.highest_bid_payer.unwrap()
{
    return Err(error!(ErrorCode::InvalidTreasury));
}
```

An attacker could spoof the token account, potentially leading to panic or loss of funds.

Recommendations:

- Validate project authority and owner
- Validate token account SPL type
- Validate token balances before transfer

Metadata Validation Gap

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

`init_auction_programmable` lacks sufficient metadata validation.

An attacker could spoof metadata, pass a non collection NFT, or define malicious or unexpected creator share percentages

Recommendations:

- Validate collection data
- Validate creator share percentages
- Validate creator/order priority
- Validate metadata PDA derivation
- Validate creators, and their position requirements
- Validate total shares
- Validate token matches programmable NFT standard
- Validate token mutability

Vulnerabilities Low

Reliance on Clock time

Vulnerability severity: **Low**

Vulnerability probability: **Low**

Multiple contracts rely on Clock time `Clock::get()`.

Clock time could be manipulated within a block, potentially leading to unexpected transaction orderings or other race conditions.

Recommendations:

- Use slot numbers in addition to clocktime to enforce ordering
- Implement buffer periods to avoid last second manipulations

Vulnerabilities Low

Token fungibility assumption

Vulnerability severity: **Low**

Vulnerability probability: **Medium**

The `init_token_identifier.rs` program defaults to an assumption of token fungibility.

```
let is_fungible = match token_standard {
    Some(TokenStandard::Fungible) => true,
    Some(TokenStandard::NonFungible) | Some(TokenStandard::ProgrammableNonFungible) => false,
    _ => true,
};
```

A malicious , malformed, or unexpected token standard could drive the program down an undesired path, potentially leading to unexpected results and market manipulation

Recommendations:

Implement a custom error and default to throwing it, as well as providing additional validations against token surface

```
#[error_code]
    pub enum ErrorCode {
    #[msg("Invalid or unknown token standard")]
    InvalidTokenStandard,
}...
...
...
let is_fungible = match token_standard {
    TokenStandard::Fungible | TokenStandard::FungibleAsset => true,
    TokenStandard::NonFungible | TokenStandard::ProgrammableNonFungible => false,
    _ => return Err(error!(ErrorCode::InvalidTokenStandard))
};
...
...
if is_fungible {
    require!(
        ctx.accounts.mint_metadata.supply.is_some(),
        ErrorCode::InvalidTokenStandard
    );
}
...
...
validate_token_metadata(
    &ctx.accounts.mint_metadata,
    is_fungible
)?;
...
...
```

Disclaimer

Disclaimer

This report is governed by the Fidesium terms and conditions.

This report does not constitute an endorsement or disapproval of any project or team, nor does it reflect the economic value or potential of any related product or asset. It is not investment advice and should not be used as the basis for investment decisions. Instead, this report provides an assessment intended to improve code quality and mitigate risks inherent in cryptographic tokens and blockchain technology.

Fidesium does not guarantee the absence of bugs or vulnerabilities in the technology assessed, nor does it comment on the business practices, models, or regulatory compliance of its creators. All services, reports, and materials are provided "as is" and "as available," without warranties of any kind, including but not limited to merchantability, fitness for a particular purpose, or non-infringement.

Cryptographic assets and blockchain technologies are novel and carry inherent technical risks, uncertainties, and the possibility of unpredictable outcomes. Assessment results may contain inaccuracies or depend on third-party systems, and reliance on them is solely at the Customer’s risk.

Fidesium assumes no liability for content inaccuracies, personal injuries, property damages, or losses related to the use of its services, reports, or materials. Third-party components are provided "as is," and any warranties are strictly between the Customer and the third-party provider.

These services and materials are intended solely for the Customer's use and benefit. No third party or their representatives may claim rights to or rely on these services, reports, or materials under any circumstances.