Advanced Blockchain

Security Assessment

February 11, 2022

Prepared For:
Michael Geike | *Advanced Blockchain AG/Incredulous Labs Ltd.*
geike@advancedblockchain.com

O | *Advanced Blockchain AG/Incredulous Labs Ltd.*
o@advancedblockchain.com

Prepared By:
Natalie Chin | *Trail of Bits*
natalie.chin@trailofbits.com

Jim Miller | *Trail of Bits*
james.miller@trailofbits.com

Sam Moelius | *Trail of Bits*
sam.moelius@trailofbits.com

Changelog:

December 10, 2021:	Initial report draft
February 11, 2022:	Fix Log (Appendix G)

Executive Summary

From November 1 to November 26, 2021, Advanced Blockchain engaged Trail of Bits to review the security of the Picasso Parachain and Bribe Protocol smart contracts. Trail of Bits conducted this assessment over six person-weeks, with three engineers working from the commits and repositories listed in the Project Dashboard.

During the first week of the engagement, we focused on gaining a high-level understanding of the Picasso Parachain, familiarizing ourselves with the protocol, and manually reviewing the Picasso runtime and Vault pallet. During the second week, we reviewed the Vault, its interactions with strategies, and its integration with other pallets. During the third week, we reviewed the Oracle, Call Filter, and Adapter pallets and the Bribe Protocol. We used Slither, our static analysis tool, to provide broad coverage of the projects and to detect common issues. During the fourth week, we completed our review of the Vault, Call Filter, and Adapter pallets.

The scope of our review was significantly limited by our inability to run the parachain locally. The Composable Finance team was also unable to address the errors affecting our local setup.

Trail of Bits identified 26 issues ranging from high to informational severity. These issues stemmed from a failure to adhere to safe code standards. The code lacked comprehensive data validation, the use of `SafeMath` for critical operations, and proper handling of errors returned by important functions; improper error handling in the Oracle pallet, for example, could prevent oracles from reclaiming their stakes and receiving payouts (TOB-ADVD-018).

Additionally, at Advanced Blockchain's request, we reviewed the Bribe Protocol Ethereum smart contracts for one day. The issues that we identified were primarily related to unsafe token integrations and a risk of reward balance manipulation through flash loans; the unsafe token integrations include that with `bidAsset`, which, if set to an asset with a callback mechanism, could lead to an exploitable reentrancy.

The projects are works in progress; most of the components have minimal test coverage and limited documentation, which hindered our verification of the code's correctness. Some of the issues we identified could have been prevented by proper testing; for instance, the incorrect equality check detailed in TOB-ADVD-007 could have been detected through unit testing. Other issues, such as those pertaining to insufficient error handling and a lack of `SafeMath`, are indicative of an immature codebase.

In addition to implementing the recommendations outlined in this report, Advanced Blockchain should take the following steps:

- Build a robust local parachain deployment script that can be run across any platform.
- Clearly outline the functionalities of the platform's core components.
- Identify the system invariants, along with the pre- and postconditions of all functions.
- Implement proper unit testing that achieves 100% coverage and follow testing best practices, such as checking all unexpected code paths and using fuzzing or symbolic execution to check code invariants.

Update: On February 11, 2022, Trail of Bits reviewed the fixes implemented for the issues presented in this report. A detailed review of the current status of each issue is provided in Appendix G.

Project Dashboard

Application Summary

Name	Advanced Blockchain
Versions	composable -Runtime (Composable Finance) 6893bc64a888b923c84a4a0f7a6728a86c4a1ae4 2fe73ea382cb4da9bfb2159f9eaa083c2b21119f 1356339090509e25c9442b686d1873f79f6a8df8 composable - Pallets (Composable Finance) 57aaee2e71214aafbfa50d01796a7cca86530cd6 bribe-protocol (Bribe Protocol) c93f3a083310ab8d8fb07a938a2eaa0c4d69bd75 c2aff1c7be8189efee7eb5a1f6a7fa6419eb16b3
Types	Rust, Solidity
Platforms	Substrate, Ethereum

Engagement Summary

Dates	November 1–November 26, 2021
Method	Full knowledge
Consultants Engaged	3
Level of Effort	6 person-weeks

Vulnerability Summary

Total High-Severity Issues	9	■ ■ ■ ■ ■ ■ ■ ■ ■
Total Medium-Severity Issues	2	■ ■
Total Low-Severity Issues	5	■ ■ ■ ■ ■
Total Informational-Severity Issues	6	■ ■ ■ ■ ■ ■
Total Undetermined-Severity Issues	4	■ ■ ■ ■
Total	26	

Category Breakdown

Access Controls	1	■
Auditing and Logging	1	■
Configuration	4	■ ■ ■ ■
Data Validation	9	■ ■ ■ ■ ■ ■ ■ ■ ■
Denial of Service	1	■
Documentation	1	■
Error Reporting	1	■
Patching	4	■ ■ ■ ■
Timing	1	■
Undefined Behavior	3	■ ■ ■
Total	26	

Code Maturity Evaluation

Category Name	Description
Access Controls	**Moderate**. The most critical functions are callable only by privileged users. However, there is a lack of clear documentation on the abilities of certain actors. Moreover, access control testing is limited, and we identified one improper permissions check.
Arithmetic	**Weak**. `SafeMath` is used throughout most of the codebase, though we identified a few issues stemming from portions in which it is not used. Moreover, there is no documentation on the critical arithmetic operations; nor are they tested through advanced testing techniques like fuzzing.
Assembly Use/Low-Level Calls	**Not applicable.**
Decentralization	**Further investigation required.** The system places trust in key actors. However, because of the lack of documentation, it is unclear whether these actors can exercise additional unintended privileges.
Code Stability	**Weak.** The code was undergoing significant changes during the audit and will likely continue to evolve before reaching its final version.
Upgradeability	**Not applicable.**
Function Composition	**Strong**. The functions have clear, narrow purposes, and critical functions can be easily extracted for testing.
Front-Running	**Not applicable.**
Monitoring	**Moderate**. Most critical functions emit descriptive events. However, the system's insufficient error handling may lead to silent failures that emit incorrect events (TOB-ADVD-018). We also identified a few functions that do not emit events (TOB-ADVD-019).
Specification	**Weak.** We identified several gaps in the high-level documentation on the system (TOB-ADVD-008). Moreover, the build system failed when we ran it according to the provided instructions.
Testing & Verification	**Weak.** The system's tests are run through a continuous integration pipeline. However, the system would benefit from additional testing, as it has function and line coverage of only ~57%. The integration of

	automated verification techniques such as fuzzing would also be beneficial.

Engagement Goals

The engagement was scoped to provide a security assessment of Advanced Blockchain components including the Picasso Parachain and Ethereum Bribe Protocol contracts.

We sought to answer the following questions about the Picasso Parachain:

- Does the parachain use the Substrate framework safely?
- Could malicious users manipulate asset prices or steal funds?
- Are the arithmetic operations executed appropriately and prevented from overflowing?
- Is it possible for the blockchain to enter a state in which a runtime operation takes so long to process that the node exceeds the Picasso Parachain's execution time limit?

We sought to answer the following questions about the Ethereum Bribe Protocol contracts:

- Is it possible for participants to steal or lose tokens?
- Could flash loans be used to exploit the protocol?
- Are rewards properly calculated and distributed to users?

Coverage

Picasso runtime. We performed a detailed manual review of the Picasso runtime, focusing on its use of Substrate features, the integration of pallets, and the configuration of critical values (TOB-ADVD-017).

Apollo Oracle pallet. We conducted a detailed manual review of the Apollo Oracle. We checked for any known oracle implementation issues and reviewed its access controls and use of `SafeMath`.

Call Filter pallet. We performed a manual review of the Call Filter to ensure that its functions can be properly disabled and that the Call Filter cannot disable itself.

Adapter pallet. We manually reviewed the Adapter pallet to ensure that all transfer functions have sufficient access controls and error handling.

Vault pallet. We performed a manual review of the Vault pallet to ensure that it implements proper access controls, uses `SafeMath` appropriately, and was implemented and configured safely.

Bribe Protocol. We ran Slither on the Bribe Protocol codebase and manually reviewed the results to check for reentrancies, risks associated with arbitrary tokens, and flash loan risks. Due to time constraints, Bribe was only reviewed for one day.

Picasso Parachain Recommendations Summary

This section aggregates all the Picasso Parachain recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short Term

❑ **Either remove the `MockOrderbook` data type or change it such that the mock orderbook is compiled only in a test environment.** (The `#[cfg(test)]` Rust annotation can be used to make that change.) TOB-ADVD-001

❑ **Update the vulnerable and unmaintained dependencies in the Composable Finance codebase.** This will prevent the exploitation of known vulnerabilities against the project. TOB-ADVD-002

❑ **Add checks to verify that all builds on development and main branches are successful.** These checks will ensure that code will not be merged into either branch unless all builds are successful. TOB-ADVD-003

❑ **Refer to the `polkadot-primitives` dependency by branch instead of tag in the `common` package.** This will ensure consistency between this package and the majority of the other Composable Finance packages, which specify Polkadot dependencies by branch. TOB-ADVD-004

❑ **Add thorough unit and integration tests to the codebase.** A comprehensive set of unit tests will help expose errors and prevent regression, in addition to providing a form of documentation for users. TOB-ADVD-005

❑ **Change the publicly callable functions' weights such that they depend on benchmarks or input data length.** This will help prevent denial-of-service attacks caused by the exhaustion of resources. TOB-ADVD-006

❑ **In the `do_liquidate` function, change the `!=` to `==` to ensure that only vault managers can update strategies and allocations and that attackers cannot manipulate yield availability.** Additionally, document the expectations on vault managers, including those regarding the timing of strategy liquidations. TOB-ADVD-007

❑ **Review and properly document the abovementioned aspects of the codebase.** TOB-ADVD-008

❑ **Have the `claim` function use safe multiplication and division when calculating the `to_payout` value.** TOB-ADVD-009

❑ **Establish a lower limit for token prices to prevent the Apollo Oracle pallet from accepting a price of zero from an oracle.** TOB-ADVD-010

❑ **Implement on-chain monitoring of the exchange and oracle contracts to detect any suspicious activity.** TOB-ADVD-011

❑ **Add checks to prevent an oracle from submitting prices if its stake no longer meets the minimum requirement.** TOB-ADVD-012

❑ **Either implement a weight for `forceUnreserve` or do not allow users to interact with it.** TOB-ADVD-013

❑ **Document the expected values of incoming arguments to make users aware of those arguments' implications.** TOB-ADVD-014

❑ **Consider requiring that users specify the asset and/or account associated with a vault, along with its ID, when depositing funds into that vault.** TOB-ADVD-015

❑ **Either adjust the computation of `delta_time` to use `SafeMath`, or add a check to `off_chain_run_auctions` to remove any order with a `started` value reflecting a time in the future.** TOB-ADVD-016

❑ **Increase the slash amount such that it is a significant portion of the minimum stake amount.** This will ensure that nodes are properly incentivized to behave honestly. TOB-ADVD-017

❑ **Adjust the `reclaim_stake` and `handle_payout` functions so that transfer and deposit operations will not silently fail.** TOB-ADVD-018

❑ **Add events for all critical operations that result in state changes.** Events aid in contract monitoring and the detection of suspicious behavior. TOB-ADVD-019

Long Term

❑ **Ensure that the staging and production environments do not use any code that should live only in tests.** TOB-ADVD-001

❑ **Use the `cargo-audit` tool to ensure that the dependency versions used by the Composable Finance project are free of known vulnerabilities, and integrate the tool into the continuous integration pipeline.** TOB-ADVD-002

❑ **Always add sufficient build and runtime checks to continuous integration pipelines on all branches.** TOB-ADVD-003

❑ **Regularly run `cargo-vendor` to detect inconsistent treatment of dependencies.** (Note, though, that we are not suggesting that you vendor your dependencies.) TOB-ADVD-004

❑ **Regularly run a test coverage tool such as `cargo-llvm-cov`, `cargo-tarpaulin`, or `grcov` over the codebase and analyze the results.** Regularly reviewing the test coverage will help identify code that is not adequately tested. TOB-ADVD-005

❑ **Avoid setting static weights for the system's extrinsics.** TOB-ADVD-006

❑ **Add unit tests, including for corner cases, to ensure that the code behaves as expected.** TOB-ADVD-007

❑ **Consider writing a formal specification of the protocol.** TOB-ADVD-008

❑ **Review all arithmetic functions that perform critical operations to ensure that they use `SafeMath`.** TOB-ADVD-009, TOB-ADVD-016

❑ **Analyze all user-provided price arguments to ensure that the values on the sorted price list adhere to the limits set for them.** TOB-ADVD-010

❑ **Assume that an attacker may be able to compromise some of the oracles.** To mitigate a partial compromise, ensure that the price computations are robust. TOB-ADVD-011

❑ **Ensure that all exposed interfaces have been properly implemented.** TOB-ADVD-013

❑ **Analyze and document all expected system parameters, identify their safe bounds, and make that information available to users so that they can evaluate an oracle's accuracy.** TOB-ADVD-014

❑ **Revise the documentation to warn users about the risks associated with specifying only a vault ID when making a deposit.** TOB-ADVD-015

❑ **Review the arbitrary and template constants defined in the Composable Finance runtime and adjust them to better suit this runtime.** TOB-ADVD-017

❑ **Consider using static analysis to detect all unhandled errors in critical functions.** TOB-ADVD-018

Bribe Protocol Recommendations Summary

This section aggregates all the Bribe Protocol recommendations made during the engagement. Short-term recommendations address the immediate causes of issues. Long-term recommendations pertain to the development process and long-term design goals.

Short Term

❑ **Either ensure that none of the tokens have callback features, or follow the checks-effects-interactions pattern.** TOB-ADVD-020

❑ **Disallow `receiptToken` transfers or change the reward calculation to ensure that attackers cannot arbitrarily increase their balances by taking out flash loans.** TOB-ADVD-021

❑ **Ensure dependencies are up to date.** Several node modules have been documented as malicious because they execute malicious code when installing dependencies to projects. Keep modules current and verify their integrity after installation. TOB-ADVD-022

❑ **Analyze the effects of a deviation from the actual number of blocks mined per second, and document the related risks so that users will be aware of them prior to using the system.** TOB-ADVD-023

❑ **Measure the gas savings from optimizations and carefully weigh them against the possibility of an optimization-related bug.** TOB-ADVD-024

❑ **Develop user documentation on edge cases in which the signature-forwarding process can be front-run or an attacker can steal a user's tokens via phishing.** TOB-ADVD-025

❑ **Use neither `ABIEncoderV2` nor any experimental Solidity feature.** Refactor the code such that structs do not need to be passed to or returned from functions. TOB-ADVD-026

Long Term

❑ **Integrate static analysis tools like Slither into the continuous integration pipeline to detect unsafe pragmas and reentrancies.** TOB-ADVD-020, TOB-ADVD-026

❑ **Analyze all token interactions in the contract to ensure that they do not introduce unexpected behavior into the system.** TOB-ADVD-021

❏ **Consider integrating automated dependency auditing into the development workflow.** If a dependency cannot be updated when a vulnerability is disclosed, ensure that the Bribe Protocol codebase does not use and is not affected by the vulnerable functionality of the dependency. TOB-ADVD-022

❏ **Identify all variables that are affected by external factors, and document the risks associated with deviations from their true values.** TOB-ADVD-023

❏ **Monitor the development and adoption of Solidity compiler optimizations to assess their maturity.** TOB-ADVD-024

❏ **Document best practices for Bribe Protocol users.** In addition to taking other precautions, users must be extremely careful when signing a message, avoid signing messages from suspicious sources, and always require hashing schemes to be public. TOB-ADVD-025

Picasso Parachain Findings Summary

#	Title	Type	Severity
7	Incorrect equality check in do_liquidate	Access Controls	High
10	Risk of median price manipulation	Data Validation	High
11	Compromise of a single oracle enables limited price manipulation	Data Validation	High
12	Failure to enforce minimum oracle stake requirement	Data Validation	High
13	Lack of force_unreserve weight causes runtime panics	Data Validation	High
18	Insufficient error handling prevents oracles from reclaiming stakes and receiving payouts	Data Validation	High
17	Slash amount is significantly lower than the minimum stake amount	Configuration	High
3	Continuous integration builds are not enabled on all branches	Undefined Behavior	Medium
14	First-in price selection method enables oracle collusion	Timing	Medium
9	Lack of SafeMath could lead to incorrect crowd loan bonuses	Data Validation	Low
15	Use of vault ID to specify deposit recipient is error-prone	Configuration	Low
16	Lack of SafeMath in the dutch-auction pallet could result in incorrect prices	Data Validation	Low
1	Runtime contains a mock orderbook	Undefined Behavior	Low
5	Insufficient test coverage	Error Reporting	Informational
8	Lack of codebase documentation	Documentation	Informational
19	Lack of events for critical operations in Vault	Auditing and Logging	Informational

6	Use of static-weight extrinsics with dynamic arguments can cause a denial of service	Denial of Service	Undetermined
4	Inconsistent use of Polkadot dependencies	Patching	Undetermined
2	Vulnerable Rust dependencies	Patching	Undetermined

7. Incorrect equality check in do_liquidate

Severity: High
Type: Access Controls
Target: `composable/frame/vault/src/lib.rs`

Difficulty: High
Finding ID: TOB-ADVD-007

Description

The `do_liquidate` function, which executes the privileged operation of liquidating a strategy, should be callable only by vault managers. However, because of a typo in the function, it checks that the caller is *not* a vault manager.

The incorrect equality check could enable an attacker to change the amount of funds allocated to a vault strategy by moving funds out of the strategy. While this would not result in a loss of funds, it would result in significantly less yield on the strategy.

```
impl<T: Config> Pallet<T> {
	/// liquidates strategy allocation
	pub fn do_liquidate(
		origin: OriginFor<T>,
		vault_id: &VaultIndex,
		strategy_account_id: &T::AccountId,
	) -> DispatchResult {
		let from = ensure_signed(origin)?;
		let vault =
			Vaults::<T>::try_get(&vault_id).map_err(|_|
Error::<T>::VaultDoesNotExist)?;
		ensure!(from != vault.manager, Error::<T>::OnlyManagerCanDoThisOperation);
		Allocations::<T>::remove(vault_id, strategy_account_id);
		Ok(())
	}
```

Figure 7.1: `frame/vault/src/lib.rs#L493-L506`

Additionally, if implemented correctly, the function would allow a vault manager to liquidate any strategy that he or she wanted to. As such, a rogue manager could pull funds out of a vault's strategies, causing depositors to receive less yield than expected. This ability would introduce a management-centralization risk and should be more clearly documented at the very least.

Exploit Scenario

Eve, an attacker, calls a function that uses `do_liquidate` as a helper function. This enables her to liquidate a strategy in a vault, significantly reducing the yield available to vault liquidity providers.

Recommendations

Short term, in the `do_liquidate` function, change the `!=` to `==` to ensure that only vault managers can update strategies and allocations and that attackers cannot manipulate yield availability. Additionally, document the expectations on vault managers, including those regarding the timing of strategy liquidations.

Long term, add unit tests, including for corner cases, to ensure that the code behaves as expected.

10. Risk of median price manipulation

Severity: High
Type: Data Validation
Target: `frame/oracle/src/lib.rs`

Difficulty: High
Finding ID: TOB-ADVD-010

Description

The `submit_price` function, which enables oracles to add new values to the sorted price list, fails to perform proper data validation. As a result, a node can submit a batch of prices that will cause the price of an asset to be set to, and reported as, zero.

```
/// Call to submit a price, gas is returned if all logic gates passed
/// Should be called from offchain worker but can be called manually too
/// Operational transaction
///
/// - `price`: price to submit
/// - `asset_id`: Id for the asset
///
/// Emits `PriceSubmitted` event when successful.
#[pallet::weight((T::WeightInfo::submit_price(T::MaxAnswerBound::get()), Operational))]
pub fn submit_price(
       origin: OriginFor<T>,
       price: T::PriceValue,
       asset_id: T::AssetId,
) -> DispatchResultWithPostInfo {
       let who = ensure_signed(origin)?;
       let author_stake = OracleStake::<T>::get(&who).unwrap_or_else(Zero::zero);
       ensure!(Requested::<T>::get(asset_id), Error::<T>::PriceNotRequested);
       ensure!(author_stake >= T::MinStake::get(), Error::<T>::NotEnoughStake);

       let set_price = PrePrice {
              price,
              block: frame_system::Pallet::<T>::block_number(),
              who: who.clone(),
       };
       PrePrices::<T>::try_mutate(asset_id, |current_prices| -> Result<(), DispatchError> {
              // There can convert current_prices.len() to u32 safely
              // because current_prices.len() limited by u32
              // (type of AssetsInfo::<T>::get(asset_id).max_answers).
              if current_prices.len() as u32 >= AssetsInfo::<T>::get(asset_id).max_answers
{
                     return Err(Error::<T>::MaxPrices.into())
              }
              if current_prices.iter().any(|candidate| candidate.who == who) {
                     return Err(Error::<T>::AlreadySubmitted.into())
              }
              current_prices.push(set_price);
              Ok(())
       })?;
       Self::deposit_event(Event::PriceSubmitted(who, asset_id, price));
       Ok(Pays::No.into())
}
```

Figure 10.1: `frame/oracle/src/lib.rs#L473-L512`

To determine the nodes that will receive payouts for price submissions, the Apollo Oracle pallet sorts the node-submitted prices to identify the median. To perform this operation, it calls the `get_median_price` function. This function can return a price of zero, which could have consequences for both consumers of the oracle result and the Angular pallet.

```
pub fn get_median_price(
	prices: &[PrePrice<T::PriceValue, T::BlockNumber, T::AccountId>],
) -> Option<T::PriceValue> {
	if prices.is_empty() {
		return None
	}

	let mut numbers: Vec<T::PriceValue> =
		prices.iter().map(|current_prices| current_prices.price).collect();

	numbers.sort_unstable();

	let mid = numbers.len() / 2;
	if numbers.len() % 2 == 0 {
		Some(numbers[mid - 1].saturating_add(numbers[mid]) / 2u32.into())
	} else {
		Some(numbers[mid])
	}
}
```

Figure 10.2: `frame/oracle/src/lib.rs#L692-L710`

Exploit Scenario

The Vault pallet contains 50,000 PICA tokens worth USD 5 million. Eve, a malicious user with control of an oracle, changes the median price to zero. She can then take advantage of the incorrect price when executing transactions in consumer pallets.

Recommendations

Short term, establish a lower limit for token prices to prevent the Apollo Oracle pallet from accepting a price of zero from an oracle.

Long term, analyze all user-provided price arguments to ensure that the values on the sorted price list adhere to the limits set for them.

11. Compromise of a single oracle enables limited price manipulation

Severity: High	Difficulty: High
Type: Data Validation	Finding ID: TOB-ADVD-011
Target: `frame/oracle/src/lib.rs`	

Description
By compromising only one oracle, an attacker could gain control of the median price and set it to a value within a certain range.

The Apollo Oracle computes the median of the first set of values provided by the price oracles. If the number of prices is odd (i.e., the median is the value in the center of the ordered list of prices), an attacker could skew the median price, setting it to a value between the lowest and highest prices submitted by the oracles.

Exploit Scenario
There are three available oracles: O_0, with a price of 603; O_1, with a price of 598; and O_2, which has been compromised by Eve. Eve is able to set the median price to any value in the range [`598`, `603`]. Eve can then turn a profit by adjusting the rate when buying and selling assets.

Recommendations
Short term, be mindful of the fact that there is no simple fix for this issue; regardless, we recommend implementing on-chain monitoring of the exchange and oracle contracts to detect any suspicious activity.

Long term, assume that an attacker may be able to compromise some of the oracles. To mitigate a partial compromise, ensure that the price computations are robust.

12. Failure to enforce minimum oracle stake requirement

Severity: High
Type: Data Validation
Target: `frame/oracle/src/lib.rs`

Difficulty: Medium
Finding ID: TOB-ADVD-012

Description

When a user requests the price of an asset, oracles submit prices by calling the `submit_price` function. When an oracle submits a price, this function checks whether that oracle has staked the minimum amount necessary to submit a price. However, it appears that this check is performed only by the `submit_price` function.

When enough prices have been submitted, the function calls the `update_price` function, which determines the price and calls the `handle_payout` function. The `handle_payout` function determines whether each oracle should be paid or slashed, based on whether the price it submitted was accurate, and proceeds accordingly. However, it does not check whether an oracle's stake meets the minimum requirement. If it attempts to slash the stake of an oracle that lacks the minimum stake, the attempt will silently fail.

```
pub fn handle_payout(
        pre_prices: &[PrePrice<T::PriceValue, T::BlockNumber, T::AccountId>],
        price: T::PriceValue,
        asset_id: T::AssetId,
) {
        for answer in pre_prices {
                let accuracy: Percent;
                if answer.price < price {
                        accuracy = PerThing::from_rational(answer.price, price);
                } else {
                        let adjusted_number = price.saturating_sub(answer.price - price);
                        accuracy = PerThing::from_rational(adjusted_number, price);
                }
                let min_accuracy = AssetsInfo::<T>::get(asset_id).threshold;
                if accuracy < min_accuracy {
                        let slash_amount = T::SlashAmount::get();
                        let try_slash = T::Currency::can_slash(&answer.who, slash_amount);
                        if !try_slash {
                                log::warn!("Failed to slash {:?}", answer.who);
                        }
                        T::Currency::slash(&answer.who, slash_amount);
                        Self::deposit_event(Event::UserSlashed(
                                answer.who.clone(),
                                asset_id,
                                slash_amount,
                        ));
```

Figure 12.1: `frame/oracle/src/lib.rs#L547-L572`

The `submit_price` function allows an oracle to submit prices for multiple assets (as long as it has received requests for those prices). This means that an oracle could stake the minimum amount and then submit bogus prices for multiple assets. After the function detected the first inaccurate price, the oracle would no longer have sufficient funds to submit prices; however, the function would still consider all of the bogus prices submitted by the oracle.

Exploit Scenario
Eve, an attacker with control of an oracle, stakes the minimum amount, submits bogus prices for several assets, and is effectively slashed only once.

Recommendations
Short term, add checks to prevent an oracle from submitting prices if its stake no longer meets the minimum requirement.

13. Lack of force_unreserve weight causes runtime panics

Severity: High
Difficulty: Medium
Type: Data Validation
Finding ID: TOB-ADVD-013
Target: `runtime/picasso/src/weights/balances.rs`, `runtime/picasso/src/weights/democracy.rs`

Description

The runtime weight for `force_unreserve` has not been implemented.

```
fn transfer_all() -> Weight {
               (100_000_000 as Weight)
                      .saturating_add(T::DbWeight::get().reads(1 as Weight))
                      .saturating_add(T::DbWeight::get().writes(1 as Weight))
       }

       fn force_unreserve() -> Weight {
               todo!()
       }
```

Figure 13.1: `runtime/picasso/src/weights/balances.rs#L54-L62`

However, when the runtime is deployed, the pallet interactor still allows users to interact with `forceUnreserve`. Since the weight has not been implemented, a call to `forceUnreserve` causes the runtime to panic.

```
Version: 3.0.0-e9cf3ab-x86_64-linux-gnu

   0: sp_panic_handler::set::{{closure}}
   1: std::panicking::rust_panic_with_hook
             at
/rustc/59eed8a2aac0230a8b53e89d4e99d55912ba6b35/library/std/src/panicking.rs:628:17
   2: std::panicking::begin_panic_handler::{{closure}}
             at
/rustc/59eed8a2aac0230a8b53e89d4e99d55912ba6b35/library/std/src/panicking.rs:519:13
   3: std::sys_common::backtrace::__rust_end_short_backtrace
             at
/rustc/59eed8a2aac0230a8b53e89d4e99d55912ba6b35/library/std/src/sys_common/backtrace.rs:141:
18
   4: rust_begin_unwind
             at
/rustc/59eed8a2aac0230a8b53e89d4e99d55912ba6b35/library/std/src/panicking.rs:517:5
   5: core::panicking::panic_fmt
             at
/rustc/59eed8a2aac0230a8b53e89d4e99d55912ba6b35/library/core/src/panicking.rs:101:14
   6: core::panicking::panic
             at
/rustc/59eed8a2aac0230a8b53e89d4e99d55912ba6b35/library/core/src/panicking.rs:50:5
   7: <picasso_runtime::weights::balances::WeightInfo<T> as
pallet_balances::weights::WeightInfo>::force_unreserve
   8: <pallet_balances::pallet::Call<T,I> as
frame_support::weights::GetDispatchInfo>::get_dispatch_info
```

```
   9:
frame_executive::Executive<System,Block,Context,UnsignedValidator,AllPallets,COnRuntimeUpgra
de>::validate_transaction
  10: <picasso_runtime::Runtime as
sp_transaction_pool::runtime_api::runtime_decl_for_TaggedTransactionQueue::TaggedTransaction
Queue<sp_runtime::generic::block::Block<sp_runtime::generic::header::Header<u32,sp_runtime::
traits::BlakeTwo256>,sp_runtime::generic::unchecked_extrinsic::UncheckedExtrinsic<sp_runtime
::multiaddress::MultiAddress<<<sp_runtime::MultiSignature as
sp_runtime::traits::Verify>::Signer as
sp_runtime::traits::IdentifyAccount>::AccountId,u32>,picasso_runtime::Call,sp_runtime::Multi
Signature,(frame_system::extensions::check_spec_version::CheckSpecVersion<picasso_runtime::R
untime>,frame_system::extensions::check_tx_version::CheckTxVersion<picasso_runtime::Runtime>
,frame_system::extensions::check_genesis::CheckGenesis<picasso_runtime::Runtime>,frame_syste
m::extensions::check_mortality::CheckMortality<picasso_runtime::Runtime>,frame_system::exten
sions::check_nonce::CheckNonce<picasso_runtime::Runtime>,frame_system::extensions::check_wei
ght::CheckWeight<picasso_runtime::Runtime>,pallet_transaction_payment::ChargeTransactionPaym
ent<picasso_runtime::Runtime>)>>>>::validate_transaction
  11:
sp_transaction_pool::runtime_api::runtime_decl_for_TaggedTransactionQueue::validate_transact
ion_native_call_generator::{{closure}}
  12: std::panicking::try
  13: std::thread::local::LocalKey<T>::with
  14: sc_executor::native_executor::WasmExecutor::with_instance::{{closure}}
  15: sc_executor::wasm_runtime::RuntimeCache::with_instance
  16: <sc_executor::native_executor::NativeElseWasmExecutor<D> as
sp_core::traits::CodeExecutor>::call
  17: sp_state_machine::execution::StateMachine<B,H,N,Exec>::execute_aux
  18:
sp_state_machine::execution::StateMachine<B,H,N,Exec>::execute_using_consensus_failure_handl
er
  19: <sc_service::client::call_executor::LocalCallExecutor<Block,B,E> as
sc_client_api::call_executor::CallExecutor<Block>>::contextual_call
  20: <sc_service::client::client::Client<B,E,Block,RA> as
sp_api::CallApiAt<Block>>::call_api_at
  21:
sp_transaction_pool::runtime_api::runtime_decl_for_TaggedTransactionQueue::validate_transact
ion_call_api_at
  22: <picasso_runtime::RuntimeApiImpl<__SR_API_BLOCK__,RuntimeApiImplCall> as
sp_transaction_pool::runtime_api::TaggedTransactionQueue<__SR_API_BLOCK__>>::TaggedTransacti
onQueue_validate_transaction_runtime_api_impl
  23: sp_transaction_pool::runtime_api::TaggedTransactionQueue::validate_transaction
  24: tracing::span::Span::in_scope
  25: sc_transaction_pool::api::validate_transaction_blocking
  26: <core::future::from_generator::GenFuture<T> as core::future::future::Future>::poll
  27: <core::future::from_generator::GenFuture<T> as core::future::future::Future>::poll
  28: <futures_util::future::future::map::Map<Fut,F> as core::future::future::Future>::poll
  29: <sc_service::task_manager::prometheus_future::PrometheusFuture<T> as
core::future::future::Future>::poll
  30: <futures_util::future::select::Select<A,B> as core::future::future::Future>::poll
  31: <tracing_futures::Instrumented<T> as core::future::future::Future>::poll
  32: tokio::park::thread::CachedParkThread::block_on
  33: tokio::runtime::handle::Handle::block_on
  34: tokio::loom::std::unsafe_cell::UnsafeCell<T>::with_mut
  35: tokio::runtime::task::harness::Harness<T,S>::poll
  36: tokio::runtime::blocking::pool::Inner::run
  37: std::sys_common::backtrace::__rust_begin_short_backtrace
  38: core::ops::function::FnOnce::call_once{{vtable.shim}}
  39: <alloc::boxed::Box<F,A> as core::ops::function::FnOnce<Args>>::call_once
             at
/rustc/59eed8a2aac0230a8b53e89d4e99d55912ba6b35/library/alloc/src/boxed.rs:1636:9
```

```
      <alloc::boxed::Box<F,A> as core::ops::function::FnOnce<Args>>::call_once
             at
/rustc/59eed8a2aac0230a8b53e89d4e99d55912ba6b35/library/alloc/src/boxed.rs:1636:9
      std::sys::unix::thread::Thread::new::thread_start
             at
/rustc/59eed8a2aac0230a8b53e89d4e99d55912ba6b35/library/std/src/sys/unix/thread.rs:106:17
  40: start_thread
             at /build/glibc-eX1tMB/glibc-2.31/nptl/pthread_create.c:477:8
  41: clone
             at
/build/glibc-eX1tMB/glibc-2.31/misc/../sysdeps/unix/sysv/linux/x86_64/clone.S:95


Thread 'tokio-runtime-worker' panicked at 'not yet implemented',
runtime/picasso/src/weights/balances.rs:61

This is a bug. Please report it at:

       https://github.com/ComposableFi/composable/issues/new
```

Figure 13.2: The output generated upon a runtime panic

Exploit Scenario

An attacker, Eve, notices that `forceUnreserve` is callable even though its weight has not been implemented and calls the function. The call triggers a runtime panic, crashing the parachain and preventing it from syncing with the relay chain.

Recommendations

Short term, either implement a weight for `forceUnreserve` or do not allow users to interact with it.

Long term, ensure that all exposed interfaces have been properly implemented.

References

- How to handle errors in Substrate

18. Insufficient error handling prevents oracles from reclaiming stakes and receiving payouts

Severity: High	Difficulty: High
Type: Data Validation	Finding ID: TOB-ADVD-018
Target: `frame/oracle/src/lib.rs`	

Description

Oracles are required to stake a minimum amount of funds to submit prices for assets. After a certain amount of time has passed since the initial staking operation, an oracle can reclaim its stake by calling the `reclaim_stake` function. This function releases ("unreserves") funds from the signer and transfers them to the controller. However, if an error occurs in the `transfer` function, the `reclaim_stake` function will ignore that error, allowing the transfer to fail silently. The controller and signer will then be removed as oracles, making it impossible for the controller to reclaim the stake by calling the function again.

```
pub fn reclaim_stake(origin: OriginFor<T>) -> DispatchResultWithPostInfo {
        let who = ensure_signed(origin)?;
        let signer = ControllerToSigner::<T>::get(&who).ok_or(Error::<T>::UnsetSigner)?;
        let block = frame_system::Pallet::<T>::block_number();
        let withdrawal = DeclaredWithdraws::<T>::get(&signer).ok_or(Error::<T>::Unknown)?;
        ensure!(block > withdrawal.unlock_block, Error::<T>::StakeLocked);
        DeclaredWithdraws::<T>::remove(&signer);
        T::Currency::unreserve(&signer, withdrawal.stake);
        let _ = T::Currency::transfer(&signer, &who, withdrawal.stake, AllowDeath);

        ControllerToSigner::<T>::remove(&who);
        SignerToController::<T>::remove(&signer);

        Self::deposit_event(Event::StakeReclaimed(signer, withdrawal.stake));
        Ok(().into())
}
```

Figure 18.1: `frame/oracle/src/lib.rs#L457-L472`

Errors that occur in the `handle_payout` function, which rewards oracles for submitting accurate prices, are similarly ignored. This function calls the `deposit_into_existing` function to pay out rewards. However, if that function experiences an error, `handle_payout` will ignore the error, allowing the deposit to fail silently and preventing the oracle from claiming its reward.

```
pub fn handle_payout(
        pre_prices: &[PrePrice<T::PriceValue, T::BlockNumber, T::AccountId>],
        price: T::PriceValue,
        asset_id: T::AssetId,
) {
    for answer in pre_prices {
            ...
            if accuracy < min_accuracy {
                  ...
           } else {
                  let reward_amount = T::RewardAmount::get();
                  let controller = SignerToController::<T>::get(&answer.who)
                         .unwrap_or_else(|| answer.who.clone());

                  let _ = T::Currency::deposit_into_existing(&controller, reward_amount);
                  Self::deposit_event(Event::UserRewarded(
                         answer.who.clone(),
                         asset_id,
                         reward_amount,
                  ));
           }
    }
}
```

Figure 18.2: `frame/oracle/src/lib.rs#L547-L586`

Exploit Scenario

A malicious signer, Eve, recognizes the `reclaim_stake` function's lack of error handling. She intentionally causes a call to `transfer` to fail so that a controller will be unable to reclaim its funds.

Recommendations

Short term, adjust the `reclaim_stake` and `handle_payout` functions so that transfer and deposit operations will not silently fail.

Long term, consider using static analysis to detect all unhandled errors in critical functions.

17. Slash amount is significantly lower than the minimum stake amount

Severity: High	Difficulty: Low
Type: Configuration	Finding ID: TOB-ADVD-017
Target: `runtime/picasso/src/lib.rs`	

Description

Parachain nodes are not properly incentivized to behave honestly, as the `SlashAmount` is significantly lower than the `MinStake` amount.

The value of PICA is defined in the runtime library as follows:

```
        pub const PICA: Balance = 1_000_000_000_000;
```

Figure 17.1: `runtime/common/src/lib.rs#L73`

The minimum stake required for a node to stake in the network is 1,000 PICA tokens, but the slash amount is only 5.

```
//TODO set
parameter_types! {
      pub const StakeLock: BlockNumber = 50;
      pub const StalePrice: BlockNumber = 5;

      /// TODO: discuss with omar/cosmin
      pub const MinStake: Balance = 1000 * PICA;
      pub const RequestCost: Balance = PICA;
      pub const RewardAmount: Balance = 5 * PICA;
      // Shouldn't this be a ratio based on locked amount?
      pub const SlashAmount: Balance = 5;
      pub const MaxAnswerBound: u32 = 25;
      pub const MaxAssetsCount: u32 = 100_000;
```

Figure 17.2: `runtime/picasso/src/lib.rs#L369-L383`

Exploit Scenario

Eve's node has staked the minimum amount. Eve behaves maliciously, and her stake is slashed, but only by five PICA tokens. She therefore retains a significant portion of her stake.

Recommendations

Short term, increase the slash amount such that it is a significant portion of the minimum stake amount. This will ensure that nodes are properly incentivized to behave honestly.

Long term, review the arbitrary and template constants defined in the Composable Finance runtime and adjust them to better suit this runtime.

References

- Staking and Slashing on the Polkadot Network
- Slashing incentives on Proof of Stake Blockchains

3. Continuous integration builds are not enabled on all branches

Severity: Medium
Difficulty: Low
Type: Undefined Behavior
Finding ID: TOB-ADVD-003
Target: `runtime/picasso/src/lib.rs`

Description
The Picasso GitHub repository lacks proper continuous integration checks. As a result, code that will break a build can be merged into protected branches.

When we initially ran `cargo-build`, the build failed because of a WASM target error:

```
  error: the wasm32-unknown-unknown target is not supported by default, you may need to
enable the "js" feature. For more information see:
https://docs.rs/getrandom/#webassembly-support
     --> ~/.cargo/registry/src/github.com-1ecc6299db9ec823/getrandom-0.2.3/src/lib.rs:219:9
      |
  219 | /         compile_error!("the wasm32-unknown-unknown target is not supported by \
  220 | |                         default, you may need to enable the \"js\" feature. \
  221 | |                         For more information see: \
  222 | |                         https://docs.rs/getrandom/#webassembly-support");
      | |________________________________________________________________________^

  error[E0433]: failed to resolve: use of undeclared crate or module `imp`
     --> ~/.cargo/registry/src/github.com-1ecc6299db9ec823/getrandom-0.2.3/src/lib.rs:246:5
      |
  246 |     imp::getrandom_inner(dest)
      |     ^^^ use of undeclared crate or module `imp`
```

Figure 3.1: The initial output of `cargo-build`

Exploit Scenario
Alice, a Composable Finance developer, adds new features to the Picasso runtime. The changes break the build. However, because continuous integration checks are not run when code is merged, the build failure is not detected.

Recommendations
Short term, add checks to verify that all builds on development and main branches are successful. These checks will ensure that code will not be merged into either branch unless all builds are successful.

Long term, always add sufficient build and runtime checks to continuous integration pipelines on all branches.

14. First-in price selection method enables oracle collusion

Severity: Medium
Difficulty: Low
Type: Timing
Finding ID: TOB-ADVD-014
Target: `frame/oracle/src/lib.rs`

Description
The Apollo Oracle allows any node with sufficient funds to submit asset prices. However, it truncates the array of prices, filtering out prices submitted after a certain number of answers (`n`, the `max_answers` value) has been submitted. In this way, the system incentivizes nodes to submit incorrect prices early on.

As all submitted prices are public, a node that submits a price early will be rewarded if that price falls within a certain range of the median (of the first `n` answers submitted). However, this first-in-first-out price selection method means that the oracle price may be skewed by early price submissions.

As a result, the `max_answers` value for an asset must be chosen carefully, and oracle price consumers must understand the risks of using an oracle whose `max_answers` value is too low.

Exploit Scenario
Eve sets up a network of five oracles, each with the PICA stake required to submit asset prices. All five oracles submit incorrect prices. Alice, an honest user with control of an oracle, then submits the correct price. Because the Apollo Oracle considers only the first five submissions, consumers of the oracle price are provided an incorrect price despite the submission of other more accurate prices, such as the one provided by Alice's oracle.

Recommendations
Short term, document the expected values of incoming arguments to make users aware of those arguments' implications.

Long term, analyze and document all expected system parameters, identify their safe bounds, and make that information available to users so that they can evaluate an oracle's accuracy.

9. Lack of SafeMath could lead to incorrect crowd loan bonuses

Severity: Low	Difficulty: High
Type: Data Validation	Finding ID: TOB-ADVD-009
Target: `frame/crowdloan-bonus/src/lib.rs`	

Description

The Crowd Loan Bonus pallet defines a `claim` function through which users can attempt to claim a crowd loan bonus from the crowd loan pot. This function takes an `amount` value and computes the amount of the payout (the `to_payout` value) using the total balance of the pot and the token supply value. Specifically, the function calculates the payout as (`amount * pot_balance_value`) / `token_supply_value` but does so without using `SafeMath`.

```
pub fn claim(origin: OriginFor<T>, amount: u128) -> DispatchResult {
        if amount.is_zero() {
                return Ok(())
        }
        let who = ensure_signed(origin)?;
        ensure!(Self::is_claimable().unwrap_or(false), Error::<T>::NotClaimable);

        let token_id = T::CurrencyId::get();
        let token_supply = T::Currency::total_issuance(token_id);
        let pot_balance = T::NativeCurrency::free_balance(&Self::account_id());
        let token_supply_value: u128 = token_supply.saturated_into();
        let pot_balance_value: u128 = pot_balance.saturated_into();

        ensure!(pot_balance_value > 0 && token_supply_value > 0, Error::<T>::EmptyPot);

        let to_payout = pot_balance_value.mul(amount).div(token_supply_value);
        let amount_value: T::Balance = amount.saturated_into();
        let converted_payout: NativeBalanceOf<T> = to_payout.saturated_into();

        T::Currency::burn_from(token_id, &who, amount_value)
                .map_err(|_| Error::<T>::InsufficientTokens)?;

        T::NativeCurrency::transfer(&Self::account_id(), &who, converted_payout,
AllowDeath)?;
        Self::deposit_event(Event::Claimed(who, amount));
        Ok(())
}
```

Figure 9.1: `frame/crowdloan-bonus/src/lib.rs#L130-L155`

An overflow in the multiplication would result in a payout of less than the correct value.

Exploit Scenario

An attacker realizes that the `claim` function does not use `SafeMath`. When another user attempts to claim a bonus, the attacker induces that user to provide an `amount` that will cause an overflow, resulting in a lower payout.

Recommendations

Short term, have the `claim` function use safe multiplication and division when calculating the `to_payout` value.

Long term, review all arithmetic functions that perform critical operations to ensure that they use `SafeMath`.

15. Use of vault ID to specify deposit recipient is error-prone

Severity: Low
Type: Configuration
Target: `frame/vault/src/lib.rs`

Difficulty: High
Finding ID: TOB-ADVD-015

Description

When a user creates a new vault, a unique vault ID is generated for it. To deposit funds into a vault, a user calls the `deposit` function, using only the vault's ID to specify the intended recipient.

```
  fn deposit(
                vault_id: &Self::VaultId,
                from: &Self::AccountId,
                asset_amount: Self::Balance,
        ) -> Result<Self::Balance, DispatchError> {
                ensure!(
                        asset_amount > T::MinimumDeposit::get(),
                        Error::<T>::AmountMustGteMinimumDeposit
                );
                Pallet::<T>::do_deposit(vault_id, from, asset_amount)
        }
```

Figure 15.1: `frame/vault/src/lib.rs#L764-L774`

```
fn do_deposit(
        vault_id: &VaultIndex,
        from: &T::AccountId,
        amount: T::Balance,
) -> Result<T::Balance, DispatchError> {
        let vault =
                Vaults::<T>::try_get(&vault_id).map_err(|_| Error::<T>::VaultDoesNotExist)?;

        ensure!(vault.capabilities.deposits_allowed(), Error::<T>::DepositsHalted);

        ensure!(
        T::Currency::can_withdraw(vault.asset_id, from, amount).into_result().is_ok(),
                        Error::<T>::TransferFromFailed
                );

        let to = Self::account_id(vault_id);

        let vault_aum = Self::assets_under_management(vault_id)?;
        if vault_aum.is_zero() {
                ensure!(
                        T::Currency::can_deposit(vault.lp_token_id, from, amount) ==
                                DepositConsequence::Success,
                                Error::<T>::MintFailed
                );

        // No assets in the vault means we should have no outstanding LP tokens, we can thus
        // freely mint new tokens without performing the calculation.
        T::Currency::transfer(vault.asset_id, from, &to, amount, true)
                .map_err(|_| Error::<T>::TransferFromFailed)?;
        T::Currency::mint_into(vault.lp_token_id, from, amount)
                .map_err(|_| Error::<T>::MintFailed)?;
        Ok(amount)
```

Figure 15.2: `frame/vault/src/lib.rs#L626-L657`

As shown in figure 15.1 and figure 15.2, there are no additional checks to ensure that the user is depositing funds into the correct vault. Requiring the user to also specify the vault’s asset, for example, would prevent the user from accidentally depositing funds into the wrong vault by misentering a vault ID (assuming the vaults had different assets). Alternatively, requiring the user to specify the account associated with the vault would also lead to fewer errors.

Exploit Scenario
An attacker, Eve, recognizes that the use of a vault ID to specify the recipient vault is error-prone. She then creates several vaults, with the goal of having users mistakenly deposit funds into them (either by making typos or falling victim to her phishing attempts).

Recommendations
Short term, consider requiring that users specify the asset and/or account associated with a vault, along with its ID, when depositing funds into that vault.

Long term, revise the documentation to warn users about the risks associated with specifying only a vault ID when making a deposit.

16. Lack of SafeMath in the dutch-auction pallet could result in incorrect prices

Severity: Low
Type: Data Validation
Target: `frame/dutch-auction/src/lib.rs`

Difficulty: High
Finding ID: TOB-ADVD-016

Description

The `off_chain_run_auctions` function in the `dutch-auction` pallet iterates through auction orders and computes an adjusted price for each order before submitting it to the `Orderbook`. This adjusted price is calculated by an order function and is based on the amount of time since the order's submission.

```
fn off_chain_run_auctions(now: Timestamp) -> DispatchResult {
    // avoid removing during iteration as unsafe
    let mut removed = Vec::new();
    for (order_id, order) in Orders::<T>::iter() {
           match order.state {
                  AuctionState::AuctionStarted => {
                         if now > order.started + ONE_HOUR {
                                removed.push(order_id);
                         } else {
              // for final protocol may be will need to transfer currency onto auction
              // pallet sub account and send dex order with idempotency tracking id final
protocol seems should include multistage lock/unlock
https://github.com/paritytech/xcm-format or something
                                 let delta_time = now - order.started;
                                 let total_initial_price: LiftedFixedBalance
                               order.source_initial_price.total_initial_price.into();
                                 let total_price =
total_initial_price.safe_mul(&order.source_total_amount.into())?;
                                 let price = match order.function {
                                    AuctionStepFunction::LinearDecrease(parameters) =>
                                           parameters.price(total_price, delta_time),
                          AuctionStepFunction::StairstepExponentialDecrease(parameters) =>
                                           parameters.price(total_price, delta_time),
                                           }?
                                           .checked_mul_int(1u64)
                                           .ok_or(ArithmeticError::Overflow)?;
```

Figure 16.1: `frame/dutch-auction/src/lib.rs#L217-240`

As shown in figure 16.1, the function assumes that the `started` value (the order's submission time) corresponds to a time in the past, so it does not use `SafeMath` when computing the `delta_time` value. However, if an attacker were able to submit an order with a value corresponding to a time in the future, it could cause an underflow in the computation of `delta_time`, resulting in an incorrect price.

Exploit Scenario

An attacker, Eve, discovers an exploit that allows her to submit an auction order with a `started` value corresponding to a time in the future. When she submits the order, `off_chain_run_auctions` computes an incorrect price for it.

Recommendations
Short term, either adjust the computation of `delta_time` to use `SafeMath`, or add a check to `off_chain_run_auctions` to remove any order with a `started` value reflecting a time in the future.

Long term, review all arithmetic functions that perform critical operations to ensure that they use `SafeMath`.

1. Runtime contains a mock orderbook

Severity: Low
Type: Undefined Behavior
Target: `runtime/picasso/src/lib.rs`

Difficulty: High
Finding ID: TOB-ADVD-001

Description

The Picasso runtime contains a mock orderbook. Mock code should be included only in tests; if it is left in a production build, it may enable unexpected behavior when the code changes in the future.

```
pub struct MockOrderbook;
impl Orderbook for MockOrderbook {
        type AssetId = CurrencyId;
        type Balance = Balance;
        type AccountId = AccountId;
        type OrderId = u128;
        fn post(
                _account_from: &Self::AccountId,
                _asset: Self::AssetId,
                _want: Self::AssetId,
                _source_amount: Self::Balance,
                _source_price: Self::Balance,
                _amm_slippage: Permill,
        ) -> Result<Self::OrderId, DispatchError> {
                Ok(0)
        }
        fn market_sell(
                _account: &Self::AccountId,
                _asset: Self::AssetId,
                _want: Self::AssetId,
                _amount: Self::Balance,
                _amm_slippage: Permill,
        ) -> Result<Self::OrderId, DispatchError> {
                Ok(0)
        }
        fn take(
                _account: &Self::AccountId,
                _orders: impl Iterator<Item = Self::OrderId>,
                _up_to: Self::Balance,
        ) -> Result<TakeResult<Self::Balance>, DispatchError> {
                Ok(TakeResult { amount: 0, total_price: 0 })
        }

        fn is_order_executed(_order_id: &Self::OrderId) -> bool {
                false
        }
}
```

Figure 1.1: `runtime/picasso/src/lib.rs#L872-L908`

Exploit Scenario

A network upgrade results in code changes that make it possible to use the mock orderbook in the vault. An attacker uses this orderbook to his benefit, which would not be possible if the mock orderbook had not been compiled in a production build.

Recommendations

Short term, either remove the `MockOrderbook` data type or change it such that the mock orderbook is compiled only in a test environment. (The [`#[cfg(test)]` Rust annotation]() can be used to make that change.)

Long term, ensure that the staging and production environments do not use any code that should live only in tests.

5. Insufficient test coverage

Severity: Informational	Difficulty: Not applicable
Type: Error Reporting	Finding ID: TOB-ADVD-005
Target: Throughout the codebase	

Description

The codebase lacks sufficient testing and has function and line coverage of ~57%, respectively (see figures 5.1 and 5.2).

Robust unit and integration tests are critical to the detection of bugs and logic errors early in the development process. Projects should include thorough coverage against both bad inputs and “happy path” scenarios. Thorough coverage helps increase users’ and developers’ confidence in the functionality of code.

Exploit Scenario

Eve discovers a flaw in a Composable Finance crate function—one that likely would have been detected by unit or integration tests—and uses it in an exploit against the crate.

Recommendations

Short term, add thorough unit and integration tests to the codebase. A comprehensive set of unit tests will help expose errors and prevent regression, in addition to providing a form of documentation for users.

Long term, regularly run a test coverage tool such as `cargo-llvm-cov`, `cargo-tarpaulin`, or `grcov` over the codebase and analyze the results. Regularly reviewing the test coverage will help identify code that is not adequately tested.

8. Lack of codebase documentation

Severity: Informational	Difficulty: Low
Type: Documentation	Finding ID: TOB-ADVD-008
Target: Throughout the codebase	

Description

The codebase lacks code documentation, high-level descriptions, and examples, making the contracts difficult to review and increasing the likelihood of user mistakes.

The Picasso Parachain documentation would benefit from details on the following:

- The methods of determining stake amounts and constants
- The running of nodes
- The types of actors in the system and their expected interactions
- The PICA token's integration into the system
- The ways in which users interact with vaults
- The process through which users retrieve asset IDs
- The behavior of off-chain workers and the retrieval of prices from the oracle
- The expected values of the constants across the codebase
- The slashing and reward mechanisms (meant to ensure that users behave as expected)
- The intended use of the Sudo pallet in the runtime
- The flow of messages sent through cross-chain message passing (XCMP) or in the cross-consensus message (XCM) format

The Bribe Protocol would benefit from detailed documentation that includes the following:

- The logic behind the use of `assetIndexes` in a two-token index pool
- The mathematical formulas used to calculate rewards
- A diagram showing the flow of funds through the protocol and the interactions between different tokens

The documentation should include all expected properties and assumptions relevant to the abovementioned aspects of the codebase.

Recommendations

Short term, review and properly document the abovementioned aspects of the codebase.

Long term, consider writing a formal specification of the protocol.

19. Lack of events for critical operations in Vault

Severity: Informational	Difficulty: Low
Type: Auditing and Logging	Finding ID: TOB-ADVD-019
Target: `frame/vault/src/lib.rs`	

Description
Several critical operations do not trigger events. As a result, it will be difficult to review the correct behavior of the contracts once they have been deployed.

For instance, when the `claim_surcharge` function is called to withdraw rent from a vault and reward the caller, it does not emit an event confirming the operation's success. The `add_surcharge` and `delete_tombstoned` functions also perform critical operations without emitting events.

Without events, users cannot easily detect suspicious behavior.

Recommendations
Short term, add events for all critical operations that result in state changes. Events aid in contract monitoring and the detection of suspicious behavior.

6. Use of static-weight extrinsics with dynamic arguments can cause a denial of service

Severity: Undetermined
Type: Denial of Service
Target: `composable/frame/vault/src/lib.rs`

Difficulty: Low
Finding ID: TOB-ADVD-006

Description

The publicly callable functions in the Vault pallet have a static weight of `10_000` and accept a variety of arguments. If the fees for executing those extrinsics are relatively low, an attacker may be able to execute them many times with long payloads to cause a denial of service.

```
/// Deposit funds in the vault and receive LP tokens in return.
/// # Emits
///  - Event::Deposited
///
/// # Errors
///  - When the origin is not signed.
///  - When `deposit < MinimumDeposit`.
#[pallet::weight(10_000)]
pub fn deposit(
       origin: OriginFor<T>,
       vault: VaultIndex,
       asset_amount: T::Balance,
) -> DispatchResultWithPostInfo {
       let from = ensure_signed(origin)?;
       let lp_amount = <Self as Vault>::deposit(&vault, &from, asset_amount)?;
       Self::deposit_event(Event::Deposited { account: from, asset_amount, lp_amount });
       Ok(().into())
}
```

Figure 6.1: `composable/frame/vault/src/lib.rs#L418-L435`

Recommendations

Short term, change the publicly callable functions’ weights such that they depend on benchmarks or input data length. This will help prevent denial-of-service attacks caused by the exhaustion of resources.

Long term, avoid setting static weights for the system’s extrinsics.

4. Inconsistent use of Polkadot dependencies

Severity: Undetermined
Difficulty: Undetermined
Type: Patching
Finding ID: TOB-ADVD-004
Target: `runtime/common/Cargo.toml`, `node/Cargo.toml`

Description
The Composable Finance code specifies Polkadot dependencies by both tag and branch. As a result, it uses multiple copies of the dependencies.

For example, the `polkadot-primitives` dependency is specified by tag in the `common` package (figure 4.1) and by branch in the `composable` package (figure 4.2).

```
polkadot-primitives = { git = "https://github.com/paritytech/polkadot", tag = "v0.9.12",
default-features = false }
```

Figure 4.1: `runtime/common/Cargo.toml#L20`

```
polkadot-primitives = { git = "https://github.com/paritytech/polkadot", branch =
"release-v0.9.12"}
```

Figure 4.2: `node/Cargo.toml#L79`

Currently, the tag is three commits behind the branch (see figures 4.3 and 4.4). Thus, the tag and branch refer to discrete bodies of code.

```
commit ec34cf7e059b91609e5b3ac4ae0f604b34ce01d9 (tag: v0.9.12-rc2, tag: v0.9.12)
Author: Martin Pugh <martin@parity.io>
Date:   Thu Oct 21 14:20:27 2021 +0200
```

Figure 4.3: `v0.9.12` commit

```
commit 5feed981cf0fe671447eabaa9c0d235a8dd0003e (tag: v0.9.12-rc5, tag: v0.9.12-1,
origin/release-v0.9.12, release-v0.9.12)
Author: joe petrowski <25483142+joepetrowski@users.noreply.github.com>
Date:   Wed Oct 27 13:34:20 2021 +0200
```

Figure 4.4: `release-0.9.12` commit

Exploit Scenario
A bug fix is pushed to the `release-v0.9.12` branch. The bug fix is incorporated into the `composable` package but not into the `common` package. Eve, an attacker, discovers a path to code containing the bug and exploits it.

Recommendations
Short term, refer to the `polkadot-primitives` dependency by branch instead of tag in the `common` package. This will ensure consistency between this package and the majority of the other Composable Finance packages, which specify Polkadot dependencies by branch.

Long term, regularly run `cargo-vendor` to detect inconsistent treatment of dependencies. (Note, though, that we are not suggesting that you vendor your dependencies.)

2. Vulnerable Rust dependencies

Severity: Undetermined
Type: Patching
Target: Composable Finance dependencies

Difficulty: Undetermined
Finding ID: TOB-ADVD-002

Description

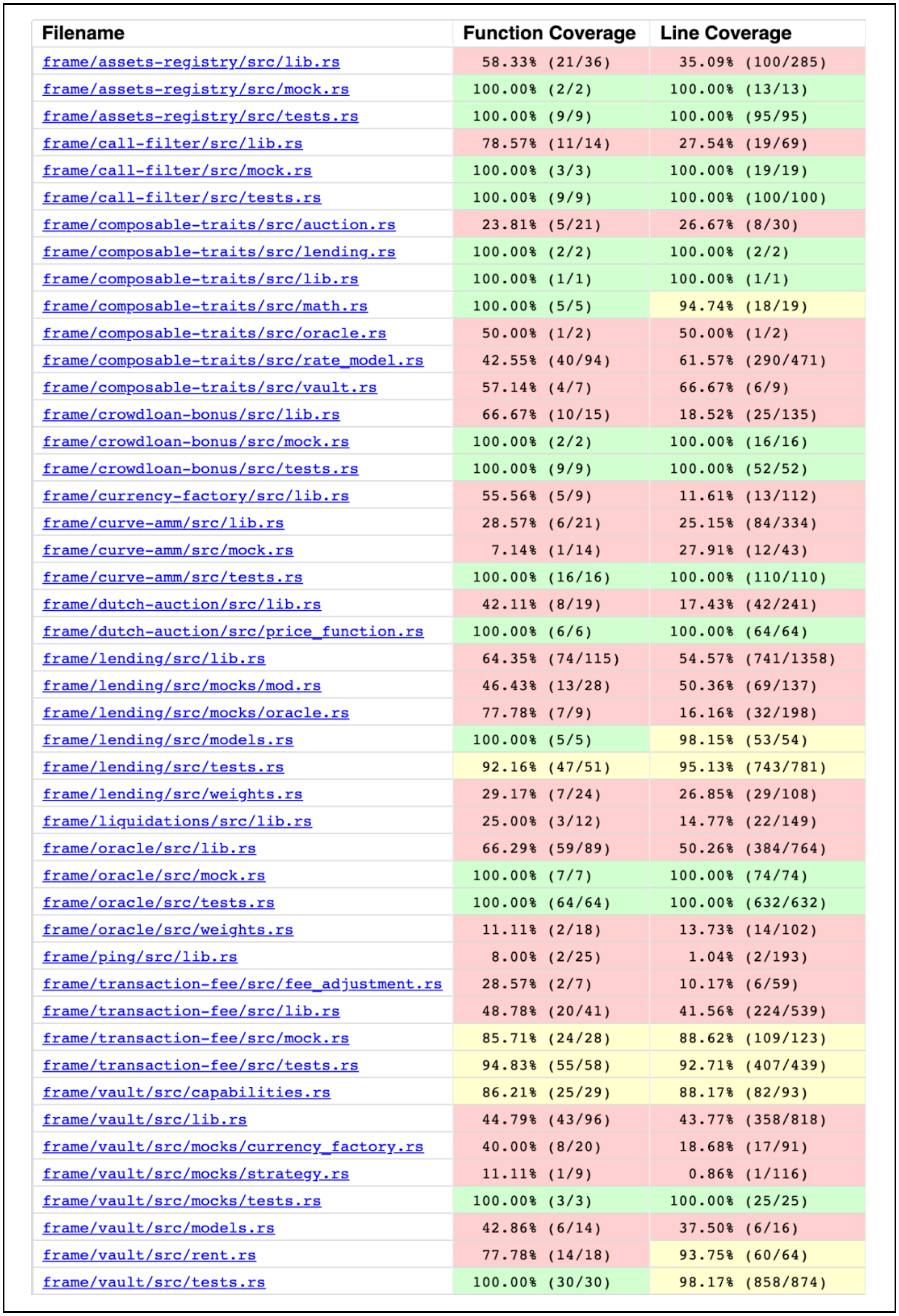
The Composable Finance project depends on eight outdated package versions that contain known vulnerabilities. These vulnerabilities, which we identified using the `cargo-audit` tool, are listed below with their RUSTSEC advisory numbers.

Dependency	Advisory Number(s)	Description(s)
`chrono`	RUSTSEC-2020-0159	Potential segfault in `localtime_r` invocations
`hyper`	RUSTSEC-2021-0079 RUSTSEC-2021-0078	Integer overflow in hyper's parsing of the `Transfer-Encoding` header leads to data loss Lenient `hyper` header parsing of `Content-Length` could allow request smuggling
`time`	RUSTSEC-2020-0071	Potential segfault in the time crate
`wasmtime`	RUSTSEC-2021-0110	Multiple Vulnerabilities in Wasmtime
`aes-soft` (unmaintained crate)	RUSTSEC-2021-0059	`aesni` has been merged into the `aes` crate
`aesni` (unmaintained crate)	RUSTSEC-2021-0059	`aesni` has been merged into the `aes` crate
`cpuid-bool` (unmaintained crate)	RUSTSEC-2021-0064	`cpuid-bool` has been renamed to `cpufeatures`
`net2` (unmaintained crate)	RUSTSEC-2020-0016	`net2` crate has been deprecated; use `socket2` instead

Recommendations

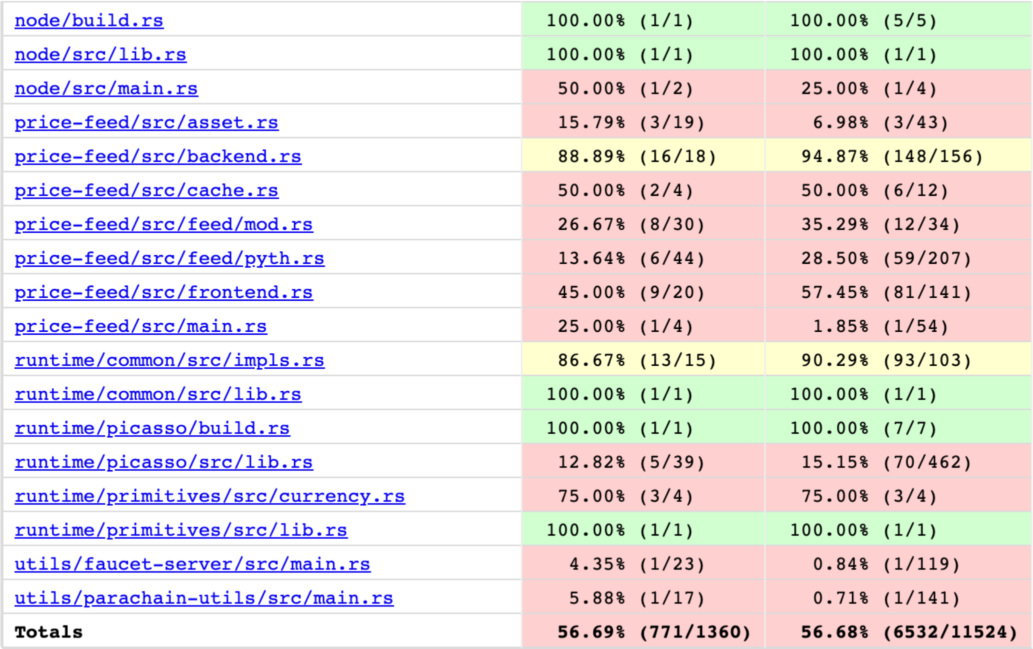
Short term, update the vulnerable and unmaintained dependencies in the Composable Finance codebase. This will prevent the exploitation of a known vulnerability against the project.

Long term, use the `cargo-audit` tool to ensure that the dependency versions used by the Composable Finance project are free of known vulnerabilities, and integrate the tool into the continuous integration pipeline.



Filename	Function Coverage	Line Coverage
frame/assets-registry/src/lib.rs	58.33% (21/36)	35.09% (100/285)
frame/assets-registry/src/mock.rs	100.00% (2/2)	100.00% (13/13)
frame/assets-registry/src/tests.rs	100.00% (9/9)	100.00% (95/95)
frame/call-filter/src/lib.rs	78.57% (11/14)	27.54% (19/69)
frame/call-filter/src/mock.rs	100.00% (3/3)	100.00% (19/19)
frame/call-filter/src/tests.rs	100.00% (9/9)	100.00% (100/100)
frame/composable-traits/src/auction.rs	23.81% (5/21)	26.67% (8/30)
frame/composable-traits/src/lending.rs	100.00% (2/2)	100.00% (2/2)
frame/composable-traits/src/lib.rs	100.00% (1/1)	100.00% (1/1)
frame/composable-traits/src/math.rs	100.00% (5/5)	94.74% (18/19)
frame/composable-traits/src/oracle.rs	50.00% (1/2)	50.00% (1/2)
frame/composable-traits/src/rate_model.rs	42.55% (40/94)	61.57% (290/471)
frame/composable-traits/src/vault.rs	57.14% (4/7)	66.67% (6/9)
frame/crowdloan-bonus/src/lib.rs	66.67% (10/15)	18.52% (25/135)
frame/crowdloan-bonus/src/mock.rs	100.00% (2/2)	100.00% (16/16)
frame/crowdloan-bonus/src/tests.rs	100.00% (9/9)	100.00% (52/52)
frame/currency-factory/src/lib.rs	55.56% (5/9)	11.61% (13/112)
frame/curve-amm/src/lib.rs	28.57% (6/21)	25.15% (84/334)
frame/curve-amm/src/mock.rs	7.14% (1/14)	27.91% (12/43)
frame/curve-amm/src/tests.rs	100.00% (16/16)	100.00% (110/110)
frame/dutch-auction/src/lib.rs	42.11% (8/19)	17.43% (42/241)
frame/dutch-auction/src/price_function.rs	100.00% (6/6)	100.00% (64/64)
frame/lending/src/lib.rs	64.35% (74/115)	54.57% (741/1358)
frame/lending/src/mocks/mod.rs	46.43% (13/28)	50.36% (69/137)
frame/lending/src/mocks/oracle.rs	77.78% (7/9)	16.16% (32/198)
frame/lending/src/models.rs	100.00% (5/5)	98.15% (53/54)
frame/lending/src/tests.rs	92.16% (47/51)	95.13% (743/781)
frame/lending/src/weights.rs	29.17% (7/24)	26.85% (29/108)
frame/liquidations/src/lib.rs	25.00% (3/12)	14.77% (22/149)
frame/oracle/src/lib.rs	66.29% (59/89)	50.26% (384/764)
frame/oracle/src/mock.rs	100.00% (7/7)	100.00% (74/74)
frame/oracle/src/tests.rs	100.00% (64/64)	100.00% (632/632)
frame/oracle/src/weights.rs	11.11% (2/18)	13.73% (14/102)
frame/ping/src/lib.rs	8.00% (2/25)	1.04% (2/193)
frame/transaction-fee/src/fee_adjustment.rs	28.57% (2/7)	10.17% (6/59)
frame/transaction-fee/src/lib.rs	48.78% (20/41)	41.56% (224/539)
frame/transaction-fee/src/mock.rs	85.71% (24/28)	88.62% (109/123)
frame/transaction-fee/src/tests.rs	94.83% (55/58)	92.71% (407/439)
frame/vault/src/capabilities.rs	86.21% (25/29)	88.17% (82/93)
frame/vault/src/lib.rs	44.79% (43/96)	43.77% (358/818)
frame/vault/src/mocks/currency_factory.rs	40.00% (8/20)	18.68% (17/91)
frame/vault/src/mocks/strategy.rs	11.11% (1/9)	0.86% (1/116)
frame/vault/src/mocks/tests.rs	100.00% (3/3)	100.00% (25/25)
frame/vault/src/models.rs	42.86% (6/14)	37.50% (6/16)
frame/vault/src/rent.rs	77.78% (14/18)	93.75% (60/64)
frame/vault/src/tests.rs	100.00% (30/30)	98.17% (858/874)

Figure 5.1: Pallet test coverage



node/build.rs	100.00% (1/1)	100.00% (5/5)
node/src/lib.rs	100.00% (1/1)	100.00% (1/1)
node/src/main.rs	50.00% (1/2)	25.00% (1/4)
price-feed/src/asset.rs	15.79% (3/19)	6.98% (3/43)
price-feed/src/backend.rs	88.89% (16/18)	94.87% (148/156)
price-feed/src/cache.rs	50.00% (2/4)	50.00% (6/12)
price-feed/src/feed/mod.rs	26.67% (8/30)	35.29% (12/34)
price-feed/src/feed/pyth.rs	13.64% (6/44)	28.50% (59/207)
price-feed/src/frontend.rs	45.00% (9/20)	57.45% (81/141)
price-feed/src/main.rs	25.00% (1/4)	1.85% (1/54)
runtime/common/src/impls.rs	86.67% (13/15)	90.29% (93/103)
runtime/common/src/lib.rs	100.00% (1/1)	100.00% (1/1)
runtime/picasso/build.rs	100.00% (1/1)	100.00% (7/7)
runtime/picasso/src/lib.rs	12.82% (5/39)	15.15% (70/462)
runtime/primitives/src/currency.rs	75.00% (3/4)	75.00% (3/4)
runtime/primitives/src/lib.rs	100.00% (1/1)	100.00% (1/1)
utils/faucet-server/src/main.rs	4.35% (1/23)	0.84% (1/119)
utils/parachain-utils/src/main.rs	5.88% (1/17)	0.71% (1/141)
Totals	**56.69% (771/1360)**	**56.68% (6532/11524)**

Figure 5.2: Node and runtime test coverage

Bribe Protocol Findings Summary

#	Title	Type	Severity
20	Reentrancy risks in interactions with bidAsset	Data Validation	High
21	receiptToken balance-tracking system could enable manipulation of reward calculations	Data Validation	High
22	Project dependencies contain vulnerabilities	Patching	Low
23	Calculation of auction expiration time depends on an approximation of seconds per block	Configuration	Informational
24	Solidity compiler optimizations can be problematic	Undefined Behavior	Informational
25	Risks associated with EIP-2612	Configuration	Informational
26	Risks associated with use of ABIEncoderV2	Patching	Undetermined

20. Reentrancy risks in interactions with bidAsset

Severity: High	Difficulty: High
Type: Data Validation	Finding ID: TOB-ADVD-020
Target: `contracts/BribeMultiAssetPoolBase.sol`	

Description

The Bribe Protocol has many reentrancy patterns that may be exploitable if the `bidAsset` ERC20 token is set to an asset with a callback mechanism.

Many interactions with the `bidAsset` token do not follow the checks-effects-interactions pattern. If `bidAsset` is set to an asset with a callback mechanism, the failure to use this pattern can lead to exploitable reentrancies.

For example, the `_bid` function's use of the reentrant `safeTransferFrom` function allows the caller to execute a double transfer:

```
/// @dev place a bid to proposal specified by `proposalId` with `amount` of bid asset
/// @param proposalId proposal id
/// @param amount amount of bid asset to bid
function _bid(
    uint256 proposalId,
    uint128 amount,
    bool support
) internal virtual {
    require(blockedProposals[proposalId] == false, "PROPOSAL_BLOCKED");

    Bid storage currentBid = bids[proposalId];
    address prevHighestBidder = currentBid.highestBidder;
    uint128 currentHighestBid = currentBid.highestBid;
    uint128 newHighestBid;

    if (prevHighestBidder == address(0)) {
        uint64 endTime = uint64(getAuctionExpiration(proposalId));
        currentBid.endTime = endTime;
    }

    require(currentBid.endTime > block.timestamp, "BID_ENDED");

    // if msg.sender == currentHighestBidder increase the bid amount
    if (prevHighestBidder == msg.sender) {
        bidAsset.safeTransferFrom(msg.sender, address(this), amount);

        newHighestBid = currentHighestBid + amount;
    } else {
        require(amount > currentHighestBid, "LOW_BID");

        bidAsset.safeTransferFrom(msg.sender, address(this), amount);

        // refund to previous highest bidder
        if (prevHighestBidder != address(0)) {
            pendingRewardToBeDistributed -= currentHighestBid;
            bidAsset.safeTransfer(prevHighestBidder, currentHighestBid);
        }
```

```
            newHighestBid = amount;
        }

        // write the new bid info to storage
        pendingRewardToBeDistributed += amount;
        currentBid.highestBid = newHighestBid;
        currentBid.support = support;
        currentBid.highestBidder = msg.sender;

        emit HighestBidIncreased(proposalId, prevHighestBidder, msg.sender, newHighestBid);
    }
```

Figure 20.1: `contracts/BribeMultiAssetPoolBase.sol#L148-L196`

Exploit Scenario

The `bidAsset` token is set to an asset with a callback mechanism and deployed. Eve exploits the reentrancy in the `_bid` function to double her profits.

Recommendations

Short term, either ensure that none of the tokens have callback features, or follow the checks-effects-interactions pattern.

Long term, use Slither to detect reentrancies in the codebase and follow the token integration guidelines provided in Appendix C.

References

- A series of tweets on the imBTC reentrancy attack

21. receiptToken balance-tracking system could enable manipulation of reward calculations

Severity: High
Type: Data Validation
Target: `contracts/AavePool.sol`

Difficulty: Medium
Finding ID: TOB-ADVD-021

Description
The `AavePool` contract uses the `receiptToken` balance to determine the number of reward tokens to mint to a user. By taking out a flash loan, a user could arbitrarily increase that number.

The reward flow assumes that a user's `receiptToken` balance is mapped directly to the number of tokens staked by the user. If a user took out a flash loan of `receiptToken`, the user's `receiptToken` balance would increase significantly, allowing the user to claim an inflated amount of rewards:

```
/// @dev get the user stkAave aave reward share
/// @param user user address
function _userPendingstkAaveRewards(
    AssetIndex storage assetIndex,
    IWrapperToken receiptToken,
    address user
) internal returns (uint256) {
    _updateStkAaveStakeRewardIndex(assetIndex);

    return
        ((assetIndex.rewardIndex - users[user].stkAaveIndex.rewardIndex) *
            receiptToken.balanceOf(user)) / receiptToken.totalSupply();
}
```

Figure 21.1: `contracts/AavePool.sol#L337-L349`

Users invoke the `_userPendingstkAaveRewards` function by calling `claimReward`:

```
/// @dev claimReward for msg.sender
function claimReward() external {
    // accrue rewards for both stkAave and Aave token balances
    _accrueRewards(msg.sender);
    [...]
    emit RewardClaim(msg.sender, pendingBid, pendingStkAaveReward, block.timestamp);
}

/// @dev _accrueRewards accrue rewards for an address
/// @param user address to accrue rewards for
function _accrueRewards(address user) internal override {
    require(user != address(0), "INVALID_ADDRESS");

    UserInfo storage _user = users[user];

    uint256 pendingBidReward;
    uint256 pendingstkAaveReward;
```

```
        uint256 userAaveBalance = wrapperAaveToken.balanceOf(user);
        uint256 userStkAaveBalance = wrapperStkAaveToken.balanceOf(user);

        if (userStkAaveBalance > 0) {
            uint128 currentAssetBidIndex =
assetIndexes[getAssetIndex(stkAaveToken)].bidIndex;
            // calculate pendingBidRewards
            pendingBidReward += _userPendingBidRewards(
                currentAssetBidIndex,
                wrapperStkAaveToken,
                _user.stkAaveIndex,
                userStkAaveBalance
            );
            // distribute stkAaveTokenRewards to the user too
            pendingstkAaveReward = _userPendingstkAaveRewards(
                assetIndexes[getAssetIndex(stkAaveToken)],
                wrapperStkAaveToken,
                user
            );
        }

        if (userAaveBalance > 0) {
            uint128 currentAssetBidIndex = assetIndexes[getAssetIndex(aaveToken)].bidIndex;
            // calculate pendingBidRewards
            pendingBidReward += _userPendingBidRewards(
                currentAssetBidIndex,
                wrapperAaveToken,
                _user.aaveIndex,
                userAaveBalance
            );
        }

        // write to storage
        _user.totalPendingBidReward += uint128(pendingBidReward);
        _user.totalPendingStkAaveReward += uint128(pendingstkAaveReward);
        _user.stkAaveIndex = assetIndexes[getAssetIndex(stkAaveToken)];
        _user.aaveIndex = assetIndexes[getAssetIndex(aaveToken)];

        emit RewardAccrue(user, pendingBidReward, pendingstkAaveReward, block.timestamp);
    }
```

Figure 21.2: `contracts/AavePool.sol#L337-L349`

When a user makes a deposit to `BribeMultiAssetPoolBase`, the `AavePool` contract mints `receiptTokens` to the user, mapping the number of tokens directly to the number of tokens deposited by the user:

```
///  @dev deposit governance token
/// @param amount amount to deposit
/// @notice emit {Deposit} event
function _deposit(
    IERC20 asset,
    IWrapperToken receiptToken,
    address recipient,
    uint128 amount
) internal {
    require(amount > 0, "INVALID_AMOUNT");
    // claim pending bid rewards only
    _accrueRewards(recipient);

    asset.safeTransferFrom(msg.sender, address(this), amount);

    // performs check that recipient != address(0)
    receiptToken.mint(recipient, amount);

    emit Deposit(asset, recipient, amount, block.timestamp);
}
```

Figure 21.3: `contracts/BribeMultiAssetPoolBase.sol#L98-L118`

Exploit Scenario

Eve, an attacker, takes out a flash loan of `receiptTokens` on another platform. She calls `claimReward()` in the same transaction, temporarily inflating her `receiptToken` balance. Because the calculation of rewards uses that inflated `receiptToken` balance, she receives more rewards than she should.

Recommendations

Short term, disallow `receiptToken` transfers or change the reward calculation to ensure that attackers cannot arbitrarily increase their balances by taking out flash loans.

Long term, analyze all token interactions in the contract to ensure that they do not introduce unexpected behavior into the system.

22. Project dependencies contain vulnerabilities

Severity: Low	Difficulty: Low
Type: Patching	Finding ID: TOB-ADVD-022
Target: `package.json`	

Description

Although dependency scans did not yield a direct threat to the Bribe Protocol codebase, `npm audit` identified dependencies with known vulnerabilities. Due to the sensitivity of the deployment code and its environment, it is important to ensure dependencies are not malicious. Problems with dependencies in the JavaScript community could have a significant effect on the Bribe Protocol system as a whole. The following output details these issues.

Dependency	Description(s)
`@openzeppelin/contracts`	UUPSUpgradeable vulnerability TimelockController vulnerability
`ansi-regex`	Inefficient Regular Expression Complexity
`glob-parent`	Regular expression denial of service
`normalize-url`	ReDos
`tar`	Arbitrary File Creation/Overwrite due to insufficient absolute path sanitization
`underscore`	Arbitrary Code Execution in underscore
`web3`	Insecure Credential Storage in web3
`y18n`	Prototype Pollution

Table 22.1: NPM advisories affecting the Bribe Protocol's dependencies

Exploit Scenario

Alice installs the dependencies of the Bribe Protocol on a clean machine. Unbeknownst to Alice, a dependency of the project has become malicious or exploitable. Alice subsequently uses the dependency, disclosing sensitive information to an unknown actor.

Recommendations

Short term, ensure dependencies are up to date. Several node modules have been documented as malicious because they execute malicious code when installing dependencies to projects. Keep modules current and verify their integrity after installation.

Long term, consider integrating automated dependency auditing into the development workflow. If a dependency cannot be updated when a vulnerability is disclosed, ensure that the Bribe Protocol codebase does not use and is not affected by the vulnerable functionality of the dependency.

23. Calculation of auction expiration time depends on an approximation of seconds per block

Severity: Informational	Difficulty: Medium
Type: Configuration	Finding ID: TOB-ADVD-023
Target: `contracts/InterestRateModel`	

Description

The auction expiration formula uses an approximation of the number of blocks mined per day on the Ethereum blockchain. This number can change across different blockchains and from day to day. The current value assumes that a new block is mined every 13 seconds; however, there is no guarantee that the Ethereum mainnet will always adhere to that same mining rate.

To calculate an auction's expiration time, the system multiplies the number of the `proposal.endBlock` by the number of seconds per block (`secondPerBlock`); it uses the resultant value and the current timestamp to determine when the auction will expire:

```
/// @dev get auction expiration of `proposalId`
/// @param proposalId proposal id
function getAuctionExpiration(uint256 proposalId) internal view override returns (uint256) {
    IAaveGovernanceV2.ProposalWithoutVotes memory proposal =
IAaveGovernanceV2(aaveGovernance)
        .getProposalById(proposalId);
    return block.timestamp + (proposal.endBlock - block.number) * secondPerBlock - 1 hours;
}
```

Figure 23.1: `contracts/AavePool.sol#L351-L357`

However, `secondPerBlock` is an estimated value and may change depending on transaction throughput. Additionally, different blockchains may have different block-settling times, which could also alter this number.

Exploit Scenario

The time of an auction's expiration is calculated using an incorrect number of blocks per second, resulting in a deviation from the actual expiration time.

Recommendations

Short term, analyze the effects of a deviation from the actual number of blocks mined per second, and document the related risks so that users will be aware of them prior to using the system.

Long term, identify all variables that are affected by external factors, and document the risks associated with deviations from their true values.

References

- Ethereum Average Block Time Chart

24. Solidity compiler optimizations can be problematic

Severity: Informational	Difficulty: Low
Type: Undefined Behavior	Finding ID: TOB-ADVD-024
Target: `truffle-config.js`	

Description

The Bribe Protocol has enabled optional compiler optimizations in Solidity.

There have been several optimization bugs with security implications. Moreover, optimizations are actively being developed. Solidity compiler optimizations are disabled by default, and it is unclear how many contracts in the wild actually use them. Therefore, it is unclear how well they are being tested and exercised.

High-severity security issues due to optimization bugs have occurred in the past. A high-severity bug in the `emscripten`-generated `solc-js` compiler used by Truffle and Remix persisted until late 2018. The fix for this bug was not reported in the Solidity CHANGELOG. Another high-severity optimization bug resulting in incorrect bit shift results was patched in Solidity 0.5.6. More recently, another bug due to the incorrect caching of `keccak256` was reported.

A compiler audit of Solidity from November 2018 concluded that the optional optimizations may not be safe.

It is likely that there are latent bugs related to optimization and that new bugs will be introduced due to future optimizations.

Exploit Scenario

A latent or future bug in Solidity compiler optimizations—or in the Emscripten transpilation to `solc-js`—causes a security vulnerability in the contracts.

Recommendations

Short term, measure the gas savings from optimizations and carefully weigh them against the possibility of an optimization-related bug.

Long term, monitor the development and adoption of Solidity compiler optimizations to assess their maturity.

25. Risks associated with EIP-2612

Severity: Informational	Difficulty: High
Type: Configuration	Finding ID: TOB-ADVD-025
Target: Bribe Protocol	

Description

The use of EIP-2612 increases the risk of `permit` function front-running as well as phishing attacks.

EIP-2612 uses signatures as an alternative to the traditional `approve` and `transferFrom` flow. These signatures allow a third party to transfer tokens on behalf of a user, with verification of a signed message.

The use of EIP-2612 makes it possible for an external party to front-run the `permit` function by submitting the signature first. Then, since the signature has already been used and the funds have been transferred, the actual caller's transaction will fail. This could also affect external contracts that rely on a successful `permit()` call for execution.

EIP-2612 also makes it easier for an attacker to steal a user's tokens through phishing by asking for signatures in a context unrelated to the Advanced Blockchain contracts. The hash message may look benign and random to the user.

Exploit Scenario

Bob has 1,000 iTokens. Eve creates an ERC20 token with a malicious airdrop called `ProofOfSignature`. To claim the tokens, users must sign a hash. Eve generates a hash to transfer Bob's tokens to her account. Eve asks Bob to sign the hash, claiming he will receive free tokens. Bob signs the hash, and Eve uses it to steal Bob's tokens.

Recommendations

Short term, develop user documentation on edge cases in which the signature-forwarding process can be front-run or an attacker can steal a user's tokens via phishing.

Long term, document best practices for Bribe Protocol users. In addition to taking other precautions, users must do the following:

- Be extremely careful when signing a message
- Avoid signing messages from suspicious sources
- Always require hashing schemes to be public

References

EIP-2612 Security Considerations

26. Risks associated with use of ABIEncoderV2

Severity: Undetermined
Type: Patching
Target: Throughout the codebase

Difficulty: Low
Finding ID: TOB-ADVD-026

Description
The contracts use Solidity's `ABIEncoderV2`. This encoder has caused numerous issues in the past, and its use may still pose risks.

More than 3% of all GitHub issues for the Solidity compiler are related to current or former experimental features, primarily `ABIEncoderV2`, which was long considered experimental. Several issues and bug reports are still open and unresolved. `ABIEncoderV2` has been associated with more than 20 high-severity bugs, some of which are so recent that they have not yet been included in a Solidity release.

For example, in March 2019 a severe bug introduced in Solidity 0.5.5 was found in the encoder.

Exploit Scenario
The Bribe Protocol contracts are deployed. After the deployment, a bug is found in the encoder, which means that the contracts are broken and can all be exploited in the same way.

Recommendations
Short term, use neither `ABIEncoderV2` nor any experimental Solidity feature. Refactor the code such that structs do not need to be passed to or returned from functions.

Long term, integrate static analysis tools like Slither into the continuous integration pipeline to detect unsafe pragmas.

A. Vulnerability Classifications

Vulnerability Classes	
Class	**Description**
Access Controls	Related to authorization of users and assessment of rights
Auditing and Logging	Related to auditing of actions or logging of problems
Authentication	Related to the identification of users
Configuration	Related to security configurations of servers, devices, or software
Cryptography	Related to protecting the privacy or integrity of data
Data Exposure	Related to unintended exposure of sensitive information
Data Validation	Related to improper reliance on the structure or values of data
Denial of Service	Related to causing a system failure
Error Reporting	Related to the reporting of error conditions in a secure fashion
Patching	Related to keeping software up to date
Session Management	Related to the identification of authenticated users
Testing	Related to test methodology or test coverage
Timing	Related to race conditions, locking, or the order of operations
Undefined Behavior	Related to undefined behavior triggered by the program

Severity Categories	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices or Defense in Depth.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is relatively small or is not a risk the customer has indicated is important.
Medium	Individual users' information is at risk; exploitation could pose

	reputational, legal, or moderate financial risks to the client.
High	The issue could affect numerous users and have serious reputational, legal, or financial implications for the client.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is commonly exploited; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of a complex system.
High	An attacker must have privileged insider access to the system, may need to know extremely complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Classifications

Code Maturity Classes	
Category Name	**Description**
Access Controls	Related to the authentication and authorization of components
Arithmetic	Related to the proper use of mathematical operations and semantics
Assembly Use	Related to the use of inline assembly
Centralization	Related to the existence of a single point of failure
Upgradeability	Related to contract upgradeability
Function Composition	Related to separation of the logic into functions with clear purposes
Front-Running	Related to resilience against front-running
Key Management	Related to the existence of proper procedures for key generation, distribution, and access
Monitoring	Related to the use of events and monitoring procedures
Specification	Related to the expected codebase documentation
Testing & Verification	Related to the use of testing techniques (unit tests, fuzzing, symbolic execution, etc.)

Rating Criteria	
Rating	**Description**
Strong	The component was reviewed, and no concerns were found.
Satisfactory	The component had only minor issues.
Moderate	The component had some issues.
Weak	The component led to multiple issues; more issues might be present.
Missing	The component was missing.

Not Applicable	The component is not applicable.
Not Considered	The component was not reviewed.
Further Investigation Required	The component requires further investigation.

C. Token Integration Checklist

The following checklist provides recommendations for interactions with arbitrary tokens. Every unchecked item should be justified, and its associated risks, understood. An up-to-date version of the checklist can be found in `crytic/building-secure-contracts`.

For convenience, all Slither utilities can be run directly on a token address, such as the following:

```
slither-check-erc 0xdac17f958d2ee523a2206206994597c13d831ec7 TetherToken
```

To follow this checklist, use the below output from Slither for the token:

```
- slither-check-erc [target] [contractName] [optional: --erc ERC_NUMBER]
- slither [target] --print human-summary
- slither [target] --print contract-summary
- slither-prop . --contract ContractName # requires configuration, and use of Echidna and
Manticore
```

General Security Considerations

- ❏ **The contract has a security review.** Avoid interacting with contracts that lack a security review. Check the length of the assessment (i.e., the level of effort), the reputation of the security firm, and the number and severity of the findings.
- ❏ **You have contacted the developers.** You may need to alert their team to an incident. Look for appropriate contacts on `blockchain-security-contacts`.
- ❏ **They have a security mailing list for critical announcements.** Their team should advise users (like you!) when critical issues are found or when upgrades occur.

ERC Conformity

Slither includes a utility, `slither-check-erc`, that reviews the conformance of a token to many related ERC standards. Use `slither-check-erc` to review the following:

- ❏ **`Transfer` and `transferFrom` return a boolean.** Several tokens do not return a boolean on these functions. As a result, their calls in the contract might fail.
- ❏ **`The name`, `decimals`, and `symbol` functions are present if used.** These functions are optional in the ERC20 standard and may not be present.
- ❏ **`Decimals` returns a `uint8`.** Several tokens incorrectly return a `uint256`. In such cases, ensure that the value returned is below 255.
- ❏ **The token mitigates the known ERC20 race condition.** The ERC20 standard has a known ERC20 race condition that must be mitigated to prevent attackers from stealing tokens.

- ❏ **The token is not an ERC777 token and has no external function call in `transfer` or `transferFrom`.** External calls in the transfer functions can lead to reentrancies.

Slither includes a utility, `slither-prop`, that generates unit tests and security properties that can discover many common ERC flaws. Use `slither-prop` to review the following:

- ❏ **The contract passes all unit tests and security properties from `slither-prop`.** Run the generated unit tests and then check the properties with Echidna and Manticore.

Finally, there are certain characteristics that are difficult to identify automatically. Conduct a manual review of the following conditions:

- ❏ **`Transfer` and `transferFrom` should not take a fee.** Deflationary tokens can lead to unexpected behavior.
- ❏ **Potential interest earned from the token is taken into account.** Some tokens distribute interest to token holders. This interest may be trapped in the contract if not taken into account.

Contract Composition

- ❏ **The contract avoids unnecessary complexity.** The token should be a simple contract; a token with complex code requires a higher standard of review. Use Slither's `human-summary` printer to identify complex code.
- ❏ **The contract uses `SafeMath`.** Contracts that do not use `SafeMath` require a higher standard of review. Inspect the contract by hand for `SafeMath` usage.
- ❏ **The contract has only a few non-token-related functions.** Non-token-related functions increase the likelihood of an issue in the contract. Use Slither's `contract-summary` printer to broadly review the code used in the contract.
- ❏ **The token has only one address.** Tokens with multiple entry points for balance updates can break internal bookkeeping based on the address (e.g., `balances[token_address][msg.sender]` may not reflect the actual balance).

Owner Privileges

- ❏ **The token is not upgradeable.** Upgradeable contracts may change their rules over time. Use Slither's `human-summary` printer to determine if the contract is upgradeable.
- ❏ **The owner has limited minting capabilities.** Malicious or compromised owners can abuse minting capabilities. Use Slither's `human-summary` printer to review minting capabilities, and consider manually reviewing the code.
- ❏ **The token is not pausable.** Malicious or compromised owners can trap contracts relying on pausable tokens. Identify pausable code by hand.

- ❑ **The owner cannot blacklist the contract.** Malicious or compromised owners can trap contracts relying on tokens with a blacklist. Identify blacklisting features by hand.
- ❑ **The team behind the token is known and can be held responsible for abuse.** Contracts with anonymous development teams or teams that reside in legal shelters require a higher standard of review.

Token Scarcity

Reviews of token scarcity issues must be executed manually. Check for the following conditions:

- ❑ **The supply is owned by more than a few users.** If a few users own most of the tokens, they can influence operations based on the tokens' repartition.
- ❑ **The total supply is sufficient.** Tokens with a low total supply can be easily manipulated.
- ❑ **The tokens are located in more than a few exchanges.** If all the tokens are in one exchange, a compromise of the exchange could compromise the contract relying on the token.
- ❑ **Users understand the risks associated with a large amount of funds or flash loans.** Contracts relying on the token balance must account for attackers with a large amount of funds or attacks executed through flash loans.
- ❑ **The token does not allow flash minting.** Flash minting can lead to substantial swings in the balance and the total supply, which necessitate strict and comprehensive overflow checks in the operation of the token.

D. Code Quality Recommendations

Picasso Substrate Recommendations

- **Do not silence Rust compiler warnings.** Silencing these warnings, particularly those related to unused variables and functions, can introduce blockchain bloat into the Substrate code.
- **Ensure that the documentation on each function matches the function's implementation.** This will enhance code clarity and readability.
  - `frame/vault/src/lib.rs#L437-L444`
- **Rename all instances of `currencyId` to `assetId`.** This will ensure that the system's behavior is consistent.
- **Make better use of the type system.** The `rate_model.rs` file (in the `composable-traits` pallet) uses the `Percent`, `Rate`, and `Ratio` types somewhat interchangeably. The `Rate` and `Ratio` types correspond to fixed-point integers, and the `Percent` type represents values in the range `[0,1]`. In certain parts of the code (e.g., the `get_supply_rate` function), a variable of the `Rate` or `Ratio` type is used for a value that should always be in the range `[0,1]` (and would therefore be better suited to the `Percent` type).

Bribe Protocol Solidity Recommendations

- **Initialize all amounts before using them in calculations.** This explicitness will help ensure that default values are easily identifiable.
  - `contracts/AavePool.sol#L254-L255`
- **Simplify the boolean equality checks to enhance the code's readability.**
  - `contracts/BribeMultiAssetPoolBase.sol#L65`
  - `contracts/BribeMultiAssetPoolBase.sol#L91`
  - `contracts/BribeMultiAssetPoolBase.sol#L156`

E. Detecting Functions Missing nonReentrant Modifiers

The Bribe Protocol codebase has many functions that are meant to be protected by a `nonReentrant` modifier to prevent reentrancy attacks. We used Slither to identify and review functions that are not protected by this modifier; additional details are provided in TOB-ADVD-020.

The script follows an approach in which every reachable function must be either protected by a `nonReentrant` modifier or included on the `ACCEPTED_LIST`.

The output of the script identifies two functions that are missing the `nonReentrant` modifier:

```
### Check AavePool calls nonReentrant
        - AavePool.refund(uint256) should have a nonReentrant modifier
        - AavePool.withdrawRemainingReward() should have a nonReentrant modifier
```

Figure E.1: `check-nonreentrant.py` output

The script is included in the figure below.

```
from slither import Slither
from slither.core.declarations import Contract
from typing import List

slither = Slither(".", ignore_compile=True)

def find_contract_in_compilation_units(contract_name: str) -> Contract:
    contracts = slither.get_contract_from_name(contract_name)
    return (contracts[0]) if len(contracts)>0 else print("Contract not found")

def _check_access_controls(
    contract: Contract, modifiers_access_controls: List[str], ACCEPTED_LIST: List[str]
):
    print(f"### Check {contract} calls {modifiers_access_controls[0]}")
    no_bug_found = True
    for function in contract.functions_entry_points:
        if function.is_constructor:
            continue
        if function.view:
            continue

        if not function.modifiers or (
            not any((str(x) in modifiers_access_controls) for x in function.modifiers)
        ):
            if not function.name in ACCEPTED_LIST:
                print(f"\t- {function.canonical_name} should have a
{modifiers_access_controls[0]} modifier")
                no_bug_found = False
    if no_bug_found:
        print("\t- No bug found")
```

```
accepted = ["renounceOwnership", "transferOwnership", "vote",
    "distributeRewards", "setStartTimestamp", "setEndTimestamp",
    "setRewardPerSecond", "multicall", "blockProposalId",
    "unblockProposalId", "pause", "unpause",
    "setDelayPeriod", "permitToken"]
_check_access_controls(
    find_contract_in_compilation_units("AavePool"),
    ["nonReentrant"],
    accepted
)
```

Figure E.2: `check-nonReentrant.py`

F. Preventing Panics in Substrate Code

This appendix provides recommendations on detecting and eliminating panics in Substrate code, which can be a difficult task.

The Problem

In many situations in which one would expect Substrate to return an error, Substrate instead panics. See the GitHub issue `[enhancement] Executive::execute_block should return error message #6716` for a discussion on this matter.

The problem with this design is that it mixes anticipated errors (e.g., transaction failures) with unanticipated errors (e.g., out-of-bounds array references). In other words, if a panic occurs, there is no easy way to tell whether it was caused by a user or a programmer error.

Clippy Lints

There are at least three Clippy lints that can help to statically identify panicking code:

- `clippy::expect_used`, which checks for `.expect()` calls on `Option`s and `Result`s
- `clippy::panic`, which checks for uses of `panic!`
- `clippy::unwrap_used`, which checks for `.unwrap()` calls on `Option`s and on `Result`s

Note that setting all of these lints to `deny` is not sufficient to prevent all panics, however. None of these lints check for `assert!`, `assert_eq!`, or `assert_ne!`, for example.

`#[no_panic]`

`#[no_panic]` is “a Rust attribute macro to require that the compiler prove a function can't ever panic.” With this macro, “panic detection happens at link time.” As such, to use this macro on a certain function, one must try to link a binary containing that function.

To enable the `#[no_panic]` macro for the `Executive::execute_block` function referenced in the aforementioned GitHub issue, one would add the yellow-highlighted lines in figure F.1 to `substrate/frame/executive/src/lib.rs`.

```
   weights::{DispatchClass, DispatchInfo, GetDispatchInfo},
};
use frame_system::DigestOf;
use no_panic::no_panic;
use sp_runtime::{
   generic::Digest,
   traits::{
...
```

```
    }

    /// Actually execute all transitions for `block`.
    #[no_panic]
    pub fn execute_block(block: Block) {
        sp_io::init_tracing();
        sp_tracing::within_span! {
```

Figure F.1: substrate/frame/executive/src/lib.rs#L127-L361

That same file contains a test that calls `Executive::execute_block` (see figure F.2).

```
    #[test]
    fn block_import_works() {
        new_test_ext(1).execute_with(|| {
            Executive::execute_block(Block {
                ...
            });
        });
    }
```

Figure F.2: substrate/frame/executive/src/lib.rs#L911-L931

Hence, one can try to build (and therefore link) the package test binary using the following command:

```
cargo build -p frame-executive --tests
```

However, that command results in an error message like the following:

```
error: linking with `cc` failed: exit status: 1
  |
  = note: ... undefined reference to `

                ERROR[no-panic]: detected panic in function `execute_block`
                '
                collect2: error: ld returned 1 exit status
```

On the other hand, the following command does *not* generate such an error message, because it does not involve linking a binary that calls `Executive::execute_block`:

```
cargo build -p frame-executive
```

G. Fix Log

After the initial assessment, Advanced Blockchain addressed certain of the Picasso Parachain and Bribe Protocol issues identified in this report, deploying the fixes to the `main` branch of the `composable` repository (`a5a4163`) and the `development` branch of the `bribe-protocol` repository (`418fb2d`), respectively. Trail of Bits reviewed each fix to ensure that it would correctly address the corresponding issue. The results of this review and additional details on each fix are provided below.

Picasso Parachain Findings			
#	**Title**	**Severity**	**Status**
7	Incorrect equality check in do_liquidate	High	Fixed
10	Risk of median price manipulation	High	Not fixed
11	Compromise of a single oracle enables limited price manipulation	High	Not fixed
12	Failure to enforce minimum oracle stake requirement	High	Fixed
13	Lack of force_unreserve weight causes runtime panics	High	Fixed
18	Insufficient error handling prevents oracles from reclaiming stakes and receiving payouts	High	Fixed
17	Slash amount is significantly lower than the minimum stake amount	High	Fixed
3	Continuous integration builds are not enabled on all branches	Medium	Fixed
14	First-in price selection method enables oracle collusion	Medium	Not fixed
9	Lack of SafeMath could lead to incorrect crowd loan bonuses	Low	Not fixed
15	Use of vault ID to specify deposit recipient is error-prone	Low	Not fixed
16	Lack of SafeMath in the dutch-auction pallet could result in incorrect prices	Low	Not fixed

1	Runtime contains a mock orderbook	Low	Fixed
5	Insufficient test coverage	Informational	Not fixed
8	Lack of codebase documentation	Informational	Not fixed
19	Lack of events for critical operations in Vault	Informational	Not fixed
6	Use of static-weight extrinsics with dynamic arguments can cause a denial of service	Undetermined	Fixed
4	Inconsistent use of Polkadot dependencies	Undetermined	Fixed
2	Vulnerable Rust dependencies	Undetermined	Partially fixed

Bribe Protocol Findings

#	Title	Severity	Status
20	Reentrancy risks in interactions with bidAsset	High	Fixed
21	receiptToken balance-tracking system could enable manipulation of reward calculations	High	Partially fixed
22	Project dependencies contain vulnerabilities	Low	Partially fixed
23	Calculation of auction expiration time depends on an approximation of seconds per block	Informational	Not fixed
24	Solidity compiler optimizations can be problematic	Informational	Not fixed
25	Risks associated with EIP-2612	Informational	Not fixed
26	Risks associated with use of ABIEncoderV2	Undetermined	Not fixed

Detailed Fix Log

TOB-ADVD-007: Incorrect equality check in do_liquidate
Fixed. The equality check is now correct.

TOB-ADVD-010: Risk of median price manipulation
Not fixed. The `get_median_price` function has not been changed.

TOB-ADVD-011: Compromise of a single oracle enables limited price manipulation
Not fixed.

TOB-ADVD-012: Failure to enforce minimum oracle stake requirement
Fixed. Oracles must now stake more funds when they submit multiple prices.

TOB-ADVD-013: Lack of force_unreserve weight causes runtime panics
Fixed. The weight for `force_unreserve` has been implemented.

TOB-ADVD-018: Insufficient error handling prevents oracles from reclaiming stakes and receiving payouts
Fixed. Both the `reclaim_stake` and `handle_payout` functions now properly handle errors.

TOB-ADVD-017: Slash amount is significantly lower than the minimum stake amount
Fixed. The slash amount parameter has been removed.

TOB-ADVD-003: Continuous integration builds are not enabled on all branches
Fixed. A `cargo-build` release line was added to the GitHub workflow for all merges to main. We also recommend using a `cron` job to schedule `cargo-build` runs even if there are no merges to main; this will ensure that any failures in the continuous integration pipeline are detected.

TOB-ADVD-014: First-in price selection method enables oracle collusion
Not fixed.

TOB-ADVD-009: Lack of SafeMath could lead to incorrect crowd loan bonuses
Not fixed. The function has been updated, but it still contains a multiplication operation that does not use `SafeMath`.

TOB-ADVD-015: Use of vault ID to specify deposit recipient is error-prone
Not fixed. There have not been any changes to make this behavior less error-prone.

TOB-ADVD-016: Lack of SafeMath in the dutch-auction pallet could result in incorrect prices

Not fixed. The off_chain_run_auctions function has been updated, but it still contains a subtraction operation that does not use SafeMath.

TOB-ADVD-001: Runtime contains a mock orderbook
Fixed. The MockOrderBook data type has been removed.

TOB-ADVD-005: Insufficient test coverage
Not fixed. The test coverage has not been improved.

TOB-ADVD-008: Lack of codebase documentation
Not fixed.

TOB-ADVD-019: Lack of events for critical operations in Vault
Not fixed. The claim_surcharge, add_surcharge, and delete_tombstoned functions still do not emit events.

TOB-ADVD-006: Use of static-weight extrinsics with dynamic arguments can cause a denial of service
Fixed. The variable-length inputs to the extrinsic have been removed and replaced with a benchmarking system that accounts for database reads and writes. We recommend developing additional documentation specifying the machine specifications of the benchmarking.

TOB-ADVD-004: Inconsistent use of Polkadot dependencies
Fixed. The polkadot-primitives dependency is now referred to exclusively by branch, not tag.

TOB-ADVD-002: Vulnerable Rust dependencies
Partially fixed. The vulnerable aes-soft, aesni, and cpuid-bool crates have been addressed. However, the chrono, hyper, time, and net2 dependencies have not been addressed. Additionally, the project now uses two new vulnerable dependencies: lru and thread_local.

TOB-ADVD-020: Reentrancy risks in interactions with bidAsset
Fixed. The reentrancy risks associated with bidAsset have been fixed. The wrapper token does not follow the checks-effects-interactions pattern; however, this token is controlled by the user.

TOB-ADVD-021: receiptToken balance-tracking system could enable manipulation of reward calculations
Partially fixed. The AavePool contract still relies on a user's receiptToken balance, which can be inflated through flash loans, to calculate the amount of rewards owed to the user.

However, the new limit on the amount of rewards that can be paid out in a certain period helps mitigate this issue.

TOB-ADVD-022: Project dependencies contain vulnerabilities
Partially fixed. The `openzeppelin/contracts` dependency has been updated to a version that does not include the vulnerabilities mentioned in this finding. However, the other dependencies have not been updated.

TOB-ADVD-023: Calculation of auction expiration time depends on an approximation of seconds per block
Not fixed. There does not appear to be new documentation warning users about this risk.

TOB-ADVD-024: Solidity compiler optimizations can be problematic
Not fixed. The Solidity compiler optimizations are still enabled.

TOB-ADVD-025: Risks associated with EIP-2612
Not fixed. There does not appear to be new documentation warning users about this risk.

TOB-ADVD-026: Risks associated with use of ABIEncoderV2
Not fixed. Solidity's `ABIEncoderV2` is still being used.