RISK AUDIT

for

FOREPROTOCOL

on

March 07, 2025

FIDESIUM

Executive Summary

Report

7
/100
TOTAL
Low risk
April 14, 2025

13
/100
TOTAL
Low risk
April 07, 2025

34
/100
TOTAL
Low risk
March 07, 2025

Abstract

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

Repository Link:
https://github.com/FOREProtocol/contracts

Initial Commit Hash:

`5155ce07ef24d6178b7e7a0de8b39227ac6a9be0`

Follow on Review:

`536b0e4df44da665d40041cbce4e072301bcbafd`

Follow on Review:

`4fe5c095067ed25c72fa995809cd33aab38a4d37`

Issue Summary

Critical	High	Medium	Low	Info
0 Issues	7 Issues	10 Issues	6 Issues	6 Issues

Caveats

Fore's codebase is generally well written, but does incur a handful of flaws.

Test Approach

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

Methodology

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

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

Severity Definitions

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

Risk Issues

Vulnerability	Description	Risk	Probability	Status
Reentrancy	The `executeTransaction` function in `Timelock.sol` allows reentrancy	High	Low	Resolved
Flash Loan	The `GovernorDelegate` contract does not enforce holding or staking requirements before counting votes	High	Medium	Resolved
Front Running: Deterministic Create2 Address	The `BeaconFactory` contract relies on a deterministic Create2 computation	High	High	Resolved
Missing Token Validation	The `ForeToken` contract in `GovernorDelegate.sol` is presumed to be set correctly	~~High~~Medium	Medium	Resolved
Missing Token Validation	The `token` contract in `ForeUniversalRouter.manageTokens` is presumed to be set correctly	High	Medium	Resolved
Missing Token Validation	The `TokenIncetiveRegistry` does not validate tokens fully	High	Medium	Resolved
Missing Access Control	The `queue` function in `GovernorDelegate.sol` allows access when `moderator` is `address(0)`	High	Medium	Resolved
Missing storage collision protection	The `GovernorDelegator.sol` does not prevent storage collision	High	Low	Resolved
Implementation set before admin	The `GovernorDelegator.sol` contract sets implementation before admin	High	Low	Resolved
Signature Format Assumption	The `Timelock.sol` assumes selector format	High	Low	Resolved
Missing Pausability	Multiple contracts do not implement pausability. This could limit the ability of the developer to respond in an emergency.	Medium	Medium	Acknowledged
Centralization	Multiple priviliged roles have significant modification rights over the contracts and their state.	Medium	Medium	Acknowledged
Missing Ownership Validation	`ForeProtocol.buyPower` does not validate that `msg.sender` owns the `id`	Medium	Low	Acknowledged
Missing Zero Address Validations	Multiple locations in the codebase are missing a zero address validation. This can result in unexpected behavior, and lost assets.	Medium	Low	Remediated
Missing Contract Validation	The `protocolAddress` and `permit2Address`contracts in `ForeUniversalRouter.sol` are presumed to be set correctly	Medium	Medium	Resolved
Missing Contract Validation	The `timelock_` contract in `GovernorDelegate.sol` is presumed to be set correctly	Medium	Medium	Resolved
One Step Ownership Transfer	Multiple contracts apply the `Ownable` pattern. It relies on a one step `transferOwnership` strategy. This exposes these contracts to accidental ownership transfer to malicious or invalid wallets	Medium	Low	Acknowledged
SafeMint Reentrancy	`ForeProtocol.createMarket` relies on `safeMint`	Medium	Low	Acknowledged
Unchecked external calls	`BasicMarketV2` makes unchecked external calls	Medium	Low	Resolved
Reliance on Block Timestamp	Multiple contracts rely on `block.timestamp`, which can be manipulated by miners.	Medium	Low	Acknowledged

Risk Issues

Vunerability	Description	Risk	Probability	Status
Missing existence validation	The `buyPower` and `upgradeTier` function on the `ForeProtocol` contract fail to validate that `id` corresponds to an existing verifier NFT.	Low	Low	Acknowledged
Missing bound validations	Multiple parameters lack upper/lower bound validations. This could result in excessively high fees and other issues.	Low	Low	Remediated
Missing balance validations	`GovernorDelegate` does not validate balance before transfer.	Low	Low	Resolved
Missing allowance check	`GovernorDelegate` does not validate allowance before transfer.	Low	Low	Resolved
Potential for Front Running on Market Creation	`BasicFactory.createMarket` relies on `marketHash` uniqueness.	Low	Low	Acknowledged
Circular/Redundant proposals	`GovernorDelegate` does not validate for circular or redundant proposals.	Low	Low	Remediated
Possible Zero Value transfers	`GovernorDelegate` allows zero value transfers.	Low	Low	Resolved
Gas Optimization: Repeated storage reads	Multiple repeated storage reads	Info	Info	Resolved
Gas Optimization: Unnecessary `uint256`	The contract implements `uint256` for multiple variables and parameters.	Info	Info	Resolved
Gas Cost Inefficiency	The `createMarket` function in `ForeProtocol` could potentially exceed block gas limits if `allMarkets` array grows large.	Info	Info	Acknowledged
Gas Optimization: Early return	The `_closeMarket` function in `BasicMarketV2` can short circuit under certain conditions.	Info	Info	Active
Gas Optimization: Unnecessary computation	The `_closeMarket` function in `BasicMarketV2` always computes `verificatorsFees`.	Info	Info	Active
Gas Optimization: Inefficient struct copying	The `MarketLibV2` copies the entire `market` struct, even when only some fields are accessed.	Info	Info	Resolved
Gas Optimization: Inefficient struct copying	The `ForeVesting._vestedAmount` copies the entire `vesting` struct as `memory` even though no local modifications are made.	Info	Info	Acknowledged

Risk Overview

Team Risk

Low risk: 1

No issues found in founding team

Doxxing Status	Team Experience	Risk Summary
Public	Highly relevant	Low

Smart Contract Risks

Risk summary: ~~47~~14

The contracts are well written, and secure with only a few minor issues..

Vulnerabilities Critical

Current scan criticals Clear

During this scan no critical security vulnerabilities were identified. The assessment covered all key components of the project, including smart contract logic, access controls, and potential attack vectors. While no critical issues were found, we recommend ongoing security monitoring and best practices to maintain the integrity and resilience of the system.

Vulnerabilities High

Reentrancy

Vulnerability severity: **High**

Vulnerability probability: **Medium**

The `executeTransaction` function in `Timelock.sol` allows reentrancy

`target.call{value: value}` to a malicous target could result in reentrancy

Recommendations:

Add a reentrancy guard

Action Taken:

Resolved at commit `86c5de05e6cc86e6719379eeb337979d900a4b75`

Flash Loan

Vulnerability severity: **High**

Vulnerability probability: **Medium**

The `GovernorDelegate` contract does not enforce holding or staking requirements before counting votes

An attacker could execute a flashloan to bypass quorum, sway governance, and manipulate the protocol

Recommendations:

Implement a voting delay after staking, vote locking, token age weighted voting, and snapshot voting using `ERC20Votes`

Action Taken:

Resolved at commit `86c5de05e6cc86e6719379eeb337979d900a4b75`

Vulnerabilities High

Front Running: Deterministic Create2 Address

Vulnerability severity: **High**

Vulnerability probability: **High**

The `BeaconFactory` contract relies on a deterministic Create2 computation

This happens in `_createCategoricalMarket`

An attacker could observe the mempool for calls to `createCategoricalMarket` or `createClassicMarket`, predict the target address and deploy a malicious contract.

In the worst case, this could lead to theft of user funds, and it definitely opens the ecosystem up to griefing attacks

Recommendations:

1. Add a random salt parameter to Create2 deployment

```
function _createCategoricalMarket(
    ...
    bytes32 salt
createdMarket = Create2.deploy(uint256(salt), marketHash, bytecode);
```

   Ensure the salt generation is handled internally by the factory and relies on a timestamp, blocknumber, counter, or ideally combination thereof. Keep in mind this would not fully prevent the vulnerability, but would drastically mitigate it
2. Introduce a commit reveal pattern, where the commit sets, but does not reveal the salt
3. Introduce a user provided, private salt, or rely on a good source of secure randomness such as an oracle
4. Maintain a per creator nonce
5. Implement address reservation
6. Deploy markets through a private transaction channel such as Flashbots

While 3-6 are the most secure options, they represent a more dramatic rearchitect of the system, and 1-2 would probably be sufficient in practice under most conditions. That said, to keep the protocol as safe as possible, we recommend 3 with oracles and 6.

Action Taken:

Resolved in commit: `536b0e4df44da665d40041cbce4e072301bcbafd`

Vulnerabilities High

Missing Token Validation

Vulnerability severity: ~~High~~Medium

Vulnerability probability: Medium

The `ForeToken` contract in `GovernorDelegate.sol` is presumed to be set correctly

The contract then relies directly on ERC20 `transfer` calls

This could lead to silent transaction failures, or, in the event of malicious misconfiguration, the injection of malicious tokens and protocol failure

Recommendations:

- Rely on `SafeERC20` and `safeTransferFrom`/`safeTransfer`
- Validate `codesize` in initialize

```
uint256 codeSize;
assembly {
    codeSize := extcodesize(fore_)
}
require(codeSize > 0, "Governor::initialize: Fore is not a contract");
```

- Validate that `fore` conforms to erc20 in `initialize`

```
try IERC20(fore_).totalSupply() returns (uint256 supply) {
    // Check that total supply isn't unreasonably large
    require(supply < 2**200, "Governor::initialize: Suspiciously large total supply");
}
```

Actions Taken:

Resolved in commit: `86c5de05e6cc86e6719379eeb337979d900a4b75` by introducing SafeERC20 and applying validations

Vulnerabilities High

Missing Token Validation

Vulnerability severity: **High**

Vulnerability probability: **Medium**

The `token` contract in `ForeUniversalRouter` is presumed to be set correctly

This occurs in `manageTokens` and `initialize`

Recommendations:

- Validate `codesize`

```
uint256 codeSize;
assembly {
    codeSize := extcodesize(token)
}
require(codeSize > 0, "token is not a contract");
```

- Validate that `token` conforms to erc20

```
try IERC20(token).totalSupply() returns (uint256 supply) {
    // Check that total supply isn't unreasonably large
    require(supply < 2**200, "Governor::initialize: Suspiciously large total supply");
}
```

Action Taken:

Resolved at commit `86c5de05e6cc86e6719379eeb337979d900a4b75`

Vulnerabilities High

Missing Token Validation

Vulnerability severity: **High**

Vulnerability probability: **Medium**

The `TokenIncetiveRegistry` does not validate tokens fully

Recommendations:

- Validate `codesize`

```
uint256 codeSize;
assembly {
    codeSize := extcodesize(token)
}
require(codeSize > 0, "token is not a contract");
```

- Validate that `token` conforms to erc20

```
try IERC20(token).totalSupply() returns (uint256 supply) {
    // Check that total supply isn't unreasonably large
    require(supply < 2**200, "Governor::initialize: Suspiciously large total supply");
}
```

Action Taken:

Resolved at commit `86c5de05e6cc86e6719379eeb337979d900a4b75`

Vulnerabilities High

Missing Access Control

Vulnerability severity: **High**

Vulnerability probability: **Medium**

The `queue` function in `GovernorDelegate.sol` allows access when `moderator` is `address(0)`

```
require(
    moderator == address(0) || msg.sender == moderator,
    "Governor::queue: moderator only"
);
```

Recommendations:

Remove the first condition and ensure that `msg.sender == moderator`

Action Taken:

Resolved at commit `86c5de05e6cc86e6719379eeb337979d900a4b75`

Missing storage collision protection

Vulnerability severity: **High**

Vulnerability probability: **Medium**

The `GovernorDelegator.sol` does not prevent storage collision

This could allow a malicious deployer to take over the contract by overwriting proxy variables, or extract all funds

Recommendations:

- Specify specific slots for storage variables

```
bytes32 private constant IMPLEMENTATION_SLOT =
bytes32(uint256(keccak256('eip1967.proxy.implementation')) - 1);
```

- Rely on battletested proxy libraries such as `OpenZeppelen.TransparentUpgradeableProxy`
- Reserve a dedicated storage gap for future upgrades

Action Taken:

Resolved at commit `86c5de05e6cc86e6719379eeb337979d900a4b75`

Vulnerabilities High

Implementation set before admin

Vulnerability severity: **High**

Vulnerability probability: **Medium**

The `GovernorDelegator.sol` contract sets implementation before admin

This means the deployer executes setImplementation with escalated privileges, allowing the deployer to potentially set a malicious implementation

Recommendations:

Set admin before implementation

Action Taken:

Resolved at commit `86c5de05e6cc86e6719379eeb337979d900a4b75`

Vulnerabilities High

Signature Format Assumption

Vulnerability severity: **High**

Vulnerability probability: **Low**

The `Timelock.sol` assumes selector format

`bytes4(keccak256(bytes(signature)))` is used for function selectors.

A malicious or careless admin could provide a malicious or malformed signature that appears innocuos but actually generates a selector for a damaging function

This could lead to catastrophic protocol damage

Recommendations:

Depending on risk tolerance:

1. Use function signatures directly
2. Implement a selector whitelist
3. Use a standardized library such as `ERC165`
4. Validate signature format for, e.g. open and close parens
5. Require each call to include the full calldata

We recommend the selector whitelist and direct signature usage at a minimum.

Action Taken:

Resolved at commit `86c5de05e6cc86e6719379eeb337979d900a4b75`

Vulnerabilities Medium

Missing Pausability

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

Multiple contracts do not implement pausability. This could limit the ability of the developer to respond in an emergency.

- GovernorDelegator
- ForeProtocol
- ForeVesting
- BasicMarketV2
- AccountWhitelist
- TokenIncentiveRegistry

Recommendations:

Use `Pausable` from OpenZeppelin

Centralization

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

Multiple priviliged roles have significant modification rights over the contracts and their state.

Contract	Role
GovernorDelegate	admin
GovernorDelegate	moderator
GovernorDelegator	admin
GovernorModerator	moderator
AccountWhitelist	initialAuthority
ForeProtocol	owner
ForeProtocol	operator
ProtocolConfig	highGuard
ProtocolConfig	owner
MarketLibV2	highGuard
ForeUniversalRouter	admin

Recommendations:

Ensure that these roles are tied to well maintained Multisig wallets.

Vulnerabilities Medium

Missing Ownership Validation

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

`ForeProtocol.buyPower` does not validate that `msg.sender` owns the `id`.

Recommendations:

Add an explicit ownership check.

Missing Zero Address Validations

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

Multiple locations in the codebase are missing a zero address validation. This can result in unexpected behavior, and lost assets.

Contract	Function	Parameter
ForeProtocol	createMarket	creator
ForeProtocol	createMarket	receiver
ForeProtocol	createMarket	marketAddress
ForeProtocol	mintVerifier	receiver

Recommendations:

Use `!= address(0)` to validate these parameters are not zero addresses

Action Taken:

Partially remediated at commit `86c5de05e6cc86e6719379eeb337979d900a4b75`

Vulnerabilities Medium

Missing Contract Validation

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

The `timelock_` contract in `GovernorDelegate.sol` is presumed to be set correctly

Recommendations:

- Validate `codesize` in initialize

```
uint256 codeSize;
assembly {
    codeSize := extcodesize(timelock_)
}
require(codeSize > 0, "Governor::initialize: Fore is not a contract");
```

- Validate that `timelock_` conforms to TimelockInterface abi in `initialize`

Action Taken:

Resolved at commit `86c5de05e6cc86e6719379eeb337979d900a4b75`

Vulnerabilities Medium

Missing Contract Validation

Vulnerability severity: **Medium**

Vulnerability probability: **Medium**

The `protocolAddress` and `permit2Address`contracts in `ForeUniversalRouter.sol` are presumed to be set correctly

Recommendations:

- Validate `codesize` in initialize

```
uint256 codeSize;
assembly {
    codeSize := extcodesize(protocolAddress)
}
require(codeSize > 0, "Router::initialize: protocol is not a contract");
```

- Validate that the contracts conforms to expected abi in `initialize`

Action Taken:

Resolved at commit `86c5de05e6cc86e6719379eeb337979d900a4b75`

One Step Ownership Transfer

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

Multiple contracts apply the `Ownable` pattern. It relies on a one step `transferOwnership` strategy. This exposes these contracts to accidental ownership transfer to malicious or invalid wallets.

- ForeProtocol
- ProtocolConfig
- ForeVesting
- ForeVerifiers

Recommendations:

Implement `Ownable2Step` to drive a two step ownership transfer. This will require applying `Upgradeable` independently.

Vulnerabilities Medium

SafeMint Reentrancy

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

`ForeProtocol.createMarket` relies on `safeMint`

safeMint has a potential vulnerability, whereby a malicious minting contract could provide a callback triggering reentrancy, or calling other functions on Fore's contracts.

Recommendations:

- Apply the `ReentrancyGuard` pattern from Openzeppelin.
- Complete all state changes before calling `safeMint`, by moving `allMarkets.push(marketAddress);` above `safeMint`

Unchecked external calls

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

`BasicMarketV2` makes unchecked external calls

This could lead to silent failures, inconsistent state, reentrancy, or denial of service

Recommendations:

Implement a `try/catch` pattern on external calls

Action Taken:

Resolved at commit `86c5de05e6cc86e6719379eeb337979d900a4b75`

Vulnerabilities Medium

Reliance on Block Timestamp

Vulnerability severity: **Medium**

Vulnerability probability: **Low**

Multiple contracts rely on `block.timestamp`, which can be manipulated by miners.

Contract	Function
GovernorDelegate	getBlockTimestamp
GovernorDelegate	startForeRewardsCampaign
GovernorDelegate	withdrawForeReward
GovernorDelegate	withdrawForeStake
GovernorDelegate	getNewStakeData
GovernorDelegate	getHypotheticalVotes
GovernorDelegate	propose
GovernorDelegate	queue
GovernorDelegate	state
GovernorDelegate	castVoteInternal
GovernorDelegate	isWhitelisted
Timelock	getBlockTimestamp
Timelock	queueTransaction
Timelock	executeTransaction
BasicMarketV2	verify
BasicMarketV2	_openDispute
MarketLibV2	init
MarketLibV2	_predict
MarketLibV2	_verify
MarketLibV2	openDispute
MarketLibV2	beforeClosingCheck
ForeVesting	withdraw
ForeVesting	_vestedAmount

Recommendations:

- Use block numbers instead of timestamps.
- If timestamps are necessary, use trusted external oracles.

Vulnerabilities Low

Missing existence validations

Vulnerability severity: **Low**

Vulnerability probability: **Low**

The `buyPower` and `upgradeTier` function on the `ForeProtocol` contract fail to validate that `id` corresponds to an existing verifier NFT.

Recommendations:

Valiate the return value from `foreVerifiers.powerOf(id)` is not 0 and the NFT exists.

Vulnerabilities Low

Missing Bound Validations

Vulnerability severity: **Low**

Vulnerability probability: **Low**

Multiple parameters lack upper/lower bound validations. This could result in excessively high fees and other issues.

Contract	Function	Parameter	Bound Missing
GovernorDelegate	propose	targets.length	Lower
GovernorDelegate	propose	description	Both
GovernorDelegate	propose	title	Both
BeaconFactory	_createClassicMarket	amountB	Lower
BeaconFactory	_createClassicMarket	amountA	Lower
BeaconFactory	_createCategoricalMarket	amounts[x]	Lower
ForeVesting	addVestingEntries	_timestampEnd	Lower
ForeVesting	addVestingEntries	_timestampStart	Lower
TokenIncentiveRegistry	initialize	predictionDiscountRate	Upper
TokenIncentiveRegistry	initialize	marketCreatorDiscountRate	Upper
TokenIncentiveRegistry	initialize	verificationDiscountRate	Upper
TokenIncentiveRegistry	initialize	foundationDiscountRate	Upper
TokenIncentiveRegistry	initialize	marketCreationFee	Upper
TokenIncentiveRegistry	addToken	predictionDiscountRate	Upper
TokenIncentiveRegistry	addToken	marketCreatorDiscountRate	Upper
TokenIncentiveRegistry	addToken	verificationDiscountRate	Upper
TokenIncentiveRegistry	addToken	foundationDiscountRate	Upper
TokenIncentiveRegistry	addToken	marketCreationFee	Upper

Recommendations:

Implement validations

Action Taken:

Partially remediated at commit `86c5de05e6cc86e6719379eeb337979d900a4b75`

Vulnerabilities Low

Missing balance validations

Vulnerability severity: **Low**

Vulnerability probability: **Low**

`GovernorDelegate` does not validate balance before transfer.

This could lead to gas waste, failing transactions, bad UX, stuck limbo states, or delayed voting failures

Recommendations:

Validate contract balance before transfer using `SafeERC20.balanceOf` and a `require` check

Resolved at commit `86c5de05e6cc86e6719379eeb337979d900a4b75`

Missing allowance check

Vulnerability severity: **Low**

Vulnerability probability: **Low**

`GovernorDelegate` does not validate allowance before transfer.

This could lead to gas waste, failing transactions, bad UX, and griefing via allowance front running

Recommendations:

Validate allowance before transfer using `SafeERC20.allowance` and a `require` check

Actions Taken:

Resolved at commit `86c5de05e6cc86e6719379eeb337979d900a4b75`

Vulnerabilities Low

Potential for Front Running on Market Creation

Vulnerability severity: **Low**

Vulnerability probability: **Low**

`BasicFactory.createMarket` relies on `marketHash` uniqueness.

A malicious script could identify a transaction in the mempool and submit creation with the same `marketHash` to grief or as part of targeted disruption.

Recommendations:

- Implement a Commit Reveal Scheme on market creation.
- Incorporate sender address into marketHash calculation.
- Implement a Nonce based system.
- Implement a refunds system for creation fees that fail due to front running.

Circular/Redundant proposals

Vulnerability severity: **Low**

Vulnerability probability: **Low**

`GovernorDelegate` does not validate for circular or redundant proposals.

Recommendations:

Track parameters changed by proposals, being sure to clean them up when expired or defeated. Validate new proposals against historic and active proposals.

Actions Taken:

Remediated at commit `86c5de05e6cc86e6719379eeb337979d900a4b75` by introducing per user proposal limits

Possible Zero Value transfers

Vulnerability severity: **Low**

Vulnerability probability: **Low**

`GovernorDelegate` allows zero value transfers.

This could lead to gas waste, log pollution, and in extreme cases uncovered attack vectors

Recommendations:

Ensure that `stakeForeForVotes` and `startForeRewardsCampaign` conduct explicit non zero value checks before `transfer`

Action Taken:

Resolved at commit: `4fe5c095067ed25c72fa995809cd33aab38a4d37`

Vulnerabilities Info

Gas Optimization: Repeated storage reads

Vulnerability severity: **Info**

Vulnerability probability: **Info**

Multiple repeated storage reads

Contract	Function	Repeated Read
GovernorDelegate	stakeForeForVotes	ForeStakes[msg.sender]
GovernorDelegate	withdrawForeStake	ForeStakes[msg.sender]
GovernorDelegate	propose	targets.length
AccountWhitelist	initialize	initialAccounts.length
MarketLibV2	calculateVerificationReward	m.result
MarketLibV2	calculateVerificationReward	v.side
MarketLibV2	closeMarket	m.confirmed
MarketLibV2	closeMarket	m.result
MarketLibV2	beforeClosingCheck	m.startVerificationTimestamp

This consumes unnecessary gas

Recommendations:

Use local variables to enforce single reads

Action Taken:

Resolved at commit `4fe5c095067ed25c72fa995809cd33aab38a4d37`

Gas Optimization: Unnecessary `uint256`

Vulnerability severity: **Info**

Vulnerability probability: **Info**

The contracts implement `uint256` for multiple variables and parameters.

This consumes unnecessary gas

Recommendations:

Check business logic and reduce `uint` size as appropriate

Action Taken:

Resolved at commit `4fe5c095067ed25c72fa995809cd33aab38a4d37`

Vulnerabilities Info

Gas Cost Inefficiency

Vulnerability severity: **Info**

Vulnerability probability: **Info**

The `createMarket` function in `ForeProtocol` could potentially exceed block gas limits if `allMarkets` array grows large.

Recommendations:

Since random access to `allMarkets` is never required, a `mapping` should be utilized. This would require tracking `length` separately.

Gas Optimization: Early return

Vulnerability severity: **Info**

Vulnerability probability: **Info**

The `_closeMarket` function in `BasicMarketV2` can short circuit under certain conditions.

Recommendations:

```
function _closeMarket(MarketLibV2.ResultType result) private {
    if (result == MarketLibV2.ResultType.INVALID) {
        MarketLibV2.closeMarket(
            _market,
            0,
            0,
            0,
            result
        );
        return;
    }
    ...
```

Vulnerabilities Info

Gas Optimization: Inefficient struct copying

Vulnerability severity: **Info**

Vulnerability probability: **Info**

The `MarketLibV2` copies the entire `market` struct, even when only some fields are accessed.

- _verify
- verify
- openDispute
- resolveDispute
- closeMarket

Recommendations:

Selectively read necessary fields as function parameters

Action Taken:

Resolved at commit `4fe5c095067ed25c72fa995809cd33aab38a4d37`

Gas Optimization: Inefficient struct copying

Vulnerability severity: **Info**

Vulnerability probability: **Info**

The `ForeVesting._vestedAmount` copies the entire `vesting` struct as `memory` even though no local modifications are made.

Recommendations:

Use `storage`

Gas Optimization: Unnecessary computation

Vulnerability severity: **Info**

Vulnerability probability: **Info**

The `_closeMarket` function in `BasicMarketV2` always computes `verificatorsFees`.

Recommendations:

Move the computation into the if condition where it's result is used to avoid computing when unnecessary

Disclaimer

Disclaimer

This report is governed by the Fidesium terms and conditions.

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

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

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

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

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