SHERLOCK

SHERLOCK SECURITY REVIEW FOR

Prepared For:	Tempus Protocol
Prepared By:	Sherlock Protocol
Lead Security Expert:	Pauliax
Security Researcher:	Mukesh Jaiswal, Evert Kors
Dates Reviewed:	Jan. 17th - Feb. 7th, 2022
Report Delivered:	February 9th, 2022

Introduction

Tempus is a future yield tokenization and fixed rate protocol. Most forms of yield farming return a variable rate of yield. This means that depositors can be subject to unpredictable fluctuations in their returns. Currently, there is no easy, capital efficient way to obtain a fixed yield or otherwise speculate on the receivable rewards.

This is where Tempus steps in. Tempus has three different use cases, each of which offers a unique value proposition:

1. Fix your future yield using any supported Yield Bearing Token (such as stETH, cDai).

2. Speculate on the rate of future yield of any supported Yield Bearing Token.

3. Provide liquidity to earn additional swap fees (on top of yield earned through yield farming protocols) by depositing any supported Yield Bearing Token.

Scope

Commit: 33a1f33e65efbcff6fa5908395fe78f169f4bdf9
Contracts:

- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/amm/interfaces/ITempusAMM.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/amm/interfaces/IVault.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/amm/TempusAMM.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/amm/TempusAMMUserDataHelpers.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/amm/VecMath.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/math/Fixed256xVar.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/pools/AaveTempusPool.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/pools/CompoundTempusPool.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/pools/LidoTempusPool.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/pools/RariTempusPool.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/pools/YearnTempusPool.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/protocols/aave/IAToken.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/protocols/aave/ILendingPool.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/protocols/compound/ICErc20.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/protocols/compound/ICToken.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/protocols/compound/IComptroller.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/protocols/lido/ILido.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/protocols/rari/IRariFundManager.sol

- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/protocols/rari/IRariFundPriceConsumer.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/protocols/yearn/IYearnVaultV2.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/stats/ChainlinkTokenPairPriceFeed/ChainlinkTokenPairPriceFeed.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/stats/ChainlinkTokenPairPriceFeed/IChainlinkAggregator.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/stats/ChainlinkTokenPairPriceFeed/IENS.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/stats/ITokenPairPriceFeed.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/stats/Stats.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/token/vesting/ERC20Vesting.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/token/vesting/IERC20Vesting.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/token/ERC20FixedSupply.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/token/ERC20OwnerMintableToken.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/token/IPoolShare.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/token/PoolShare.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/token/PrincipalShare.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/token/TempusToken.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/token/YieldShare.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/utils/AMMBalancesHelper.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/utils/IOwnable.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/utils/IVersioned.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/utils/Ownable.sol

- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/utils/UntrustedERC20.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/utils/Versioned.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/ITempusPool.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/TempusController.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/TempusPool.sol
- https://github.com/tempus-finance/tempus-protocol/blob/release-1.1/contracts/amm/interfaces/IRateProvider.sol

Overall Health

Overall the codebase is pretty clean and structured, as the team did a good job at testing and documenting the codebase. Also, members of the team were helpful and professional with communicating and explaining the intentions of different parts of the code.

The pools do not have a huge dependency on an admin and all the configurable variables have reasonable constraints. All the integrated protocols are pretty well-known and battle-tested in the DeFi world with a good reputation.

System Overview

TempusController is the main entry point to the Tempus Protocol. It is used to trigger all user actions in the protocol, such as deposit and redeem. It is also used to simplify the interface for users, especially those who want to carry out atomic transactions like **Deposit and provide liquidity, Deposit and fix yield, Complete exit and redeem.**

The **TempusAMM** smart contract is an essential component of the Tempus Protocol. It allows users to swap Yields and Principals for each other, or to provide liquidity to the platform to earn liquidity provider fees. Each TempusAMM is paired with a corresponding TempusPool

TempusPool is the main smart contract in Tempus Protocol. It holds all locked YieldBearingToken. It is also used to mint Principals and Yields on deposits, and to burn them on redemption in exchange for YieldBearingToken

Each pool in Tempus has certain unique properties:
They are defined by: The supported **YieldBearingToken** (such as *stETH* or *cDai*); and **Maturity Time**

Privileged Roles

Owner: It is automatically set to the deployer of the given contract

Owner Privilege

1. Owner can set authorized contracts through **tempusController.register()**
2. Owner can Set fees configuration through **tempuspool.config()**
3. Owner can transfer fees to the given address through **tempuspool.transferFees()**
4. Owner can Mint tokens through **tempusToken.mint()**, which is capped at 2% of total supply annually.

Centralization Risk

We have discussed the privileges of the owner in the above section, and any compromise to the owner account may allow hackers to perform all actions that can be done through “owner”.

Code Attributes

Proper formatting: Yes

Readability: High
Commenting: Good

Upgradeable: No
Use of battle-tested libraries where possible: High

Size of Codebase: Medium
Code Complexity: Medium

Test Suite

Test coverage: 93%
Quality of tests: High

Blockchain: Ethereum
Multi-chain: No, but a possible expansion to Fantom
L2s: No

Tokens used:

Backing/Yeld pair of tokens of the integrated protocols can be used, e.g. Lido ETH/stETH. Which tokens are used is specified upon the deployment of each pool.

External contracts/interfaces used:

@balancer-labs/v2-solidity-utils: FixedPoint, InputHelpers.
@balancer-labs/v2-pool-utils: BaseGeneralPool, BaseMinimalSwap InfoPool.
@balancer-labs/v2-pool-stable: StableMath.
@openzeppelin: ERC20, ERC20 Metadata, SafeERC20, ReentrancyGuard, SafeCast, ERC20Votes.

External Dependencies and Trust Assumption

Composability with other protocols: Lido, Rari, Aave, Compound, Yearn & Chainlink.
Use of oracles: Chainlink (only for statistics)

Findings

Each issue has an assigned severity:

- **Informational** issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgement as to whether to address such issues.
- **Low** issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- **Medium** issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- **High** issues are directly exploitable security vulnerabilities that need to be fixed.

Total Issues

Informational	Low	Medium	High
4	6		

SHERLOCK

Issue 01

Summary
Function redeemToBacking is payable for no reason. Even if this function redeems the native asset, it is not supposed to receive it from the initial caller.

Severity
Low

Vulnerability Detail
If a user calls redeemToBacking with a non-zero msg.value, that amount will be locked in the contract because this function does not handle the native asset in any meaningful way.

Impact
Native asset can get stuck

Code Snippet:
```
 TempusPool line 227:
 function redeemToBacking(
        ...
        )external
         payable{}
```

Tool used
Manual Review

Recommendation
RedeemToBacking should not be payable.

Team
[Comment here]

Sherlock
[Comment here]

SHERLOCK

Issue 02

Summary
Consider adding token rescue functions so an authorized sender (admin) can retrieve stuck tokens in case someone accidentally sends them directly to the contract, e.g. TempusController.

Severity
Low

Vulnerability Detail
The contracts might receive tokens they are not supposed to handle, e.g. users can accidentally send the tokens directly to the contract, or there might be left some dust due to a small precision loss in calculations. Also, in case of an airdrop, you can claim it and distribute additional rewards to your users, thus boosting their yield.

Impact
Tokens can get stuck

Tool used
Manual Review

Recommendation
It should be safe to have token rescue functions in contracts that are not escrowing users' funds, e.g. TempusController, or TempusAMM.

Team
[Comment here]

Sherlock
[Comment here]

SHERLOCK

Issue 03

Summary

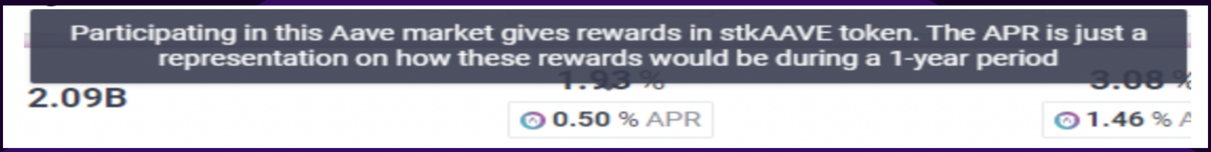
AaveTempusPool should also handle stkAAVE incentive rewards.

Severity

Low

Vulnerability Detail

When depositing on Aave, you get a base reward, but you also get some stkAAVE rewards, e.g.:



Participating in this Aave market gives rewards in stkAAVE token. The APR is just a representation on how these rewards would be during a 1-year period

The current implementation of AaveTempusPool does not handle these rewards, thus the users can't enjoy these extra rewards.

Impact

Unclaimed stkAAVE rewards

Tool used

Manual Review

Recommendation

An example solution:

```
function claimRewards(address _receiver) external onlyOwner {
   address[] memory assets = new address[](1);
   assets[0] = address(yieldBearingToken);
    aaveIncentivesController.claimRewards(assets, type(uint256).max,
_receiver)
```

Issue 04

Summary

TempusPool relies on the controller to protect from re-entrancy.

Severity

Low

Vulnerability Detail

Some functions in the TempusPool contract that don't really need nonReentrant modifier have it, but those that are more vulnerable do not, e.g.:
transferFees has nonReentrant even though it is only invokable by the owner and it follows the Check-Effects-Interaction pattern, so it is not really vulnerable to re-entrancy:

```
    function transferFees(address recipient) external override
nonReentrant onlyOwner {
        uint256 amount = totalFees;
        totalFees = 0;
        IERC20 token = IERC20(yieldBearingToken);
        token.safeTransfer(recipient, amount);
    }
```

On the other hand, onDepositBacking is invokable by the controller, and it performs external calls in the middle of execution (e.g. depositToUnderlying) but has no re-entrancy protection:

```
    function onDepositBacking(uint256 backingTokenAmount, address
recipient)
       onlyController
    {
       depositedYBT = depositToUnderlying(backingTokenAmount);
        (mintedShares, , fee, rate) = mintShares(depositedYBT, recipient);
    }
```

If there was no re-entrancy protection here, it would basically have to trust the integrated pools (Aave, Compound, Yearn, Lido, Rari, etc) and backing tokens (ETH, USDC, etc) not to exploit this. However, re-entrancy protection is expected to be applied from the caller, that is TempusController, and the current versions are safe.

Impact

While the current implementations are not vulnerable, you must be careful when deploying new pools in the future, as the controller is injected in the constructor and there are no guarantees it will be the same TempusController. Considering that it is Versioned, and the contract is not upgradeable, it can be expected there will be different controllers in the future, and these controllers might not necessarily have re-entrancy protection on these functions.

Tool used

Manual Review

Recommendation

A more robust approach would be that each contract separately protects from re-entrancy and does not rely on the outside caller to do that.

Issue 05

Summary
VecMath library’s mul and div functions do not use SafeMath operations.

Severity
Low

Vulnerability Detail
The VecMath library has declared a compiler version that is not protected from overflows/underflows by default: pragma solidity 0.7.6;
Functions mul and div use regular operations, thus it possesses a risk of unexpected results in case the values go out of boundaries.
The client double-checked every usage of these functions and assured that none of the current usages are vulnerable to this.

Impact
Current contracts are not vulnerable but it is a concern in case this library will be used later or if someone forks it without understanding the risks. Thus, this issue is still left as a reminder with the severity of low because the likelihood is very small but the impact would be huge.

Tool used
Manual Review

Recommendation
Consider applying SafeMath operations to make the library consistent and safe from unexpected behavior, or at least add comments to disincentive reckless usage of this library.

Issue 06

Summary
Not all tokens use safe transfers.

Severity
Informational

Vulnerability Detail
The project utilizes safe ERC20 transfers for all tokens except tempusAMM and PrincipalShare and YieldShare. These tokens are transferred using regular transfer functions.
tempusAMM is a BalancerPoolToken with a known transferFrom() implementation, however, theoretically this could be any valid token that is registered and implements ITempusAMM interface because the register() function in the Controller basically allows any contract to be registered.

Similarly, PrincipalShare and YieldShare tokens are also transferred with non-safe functions and while the current implementations are not vulnerable, theoretically another implementation of TempusPool could be used, and make this a valid concern.

Impact
The likelihood of this is very low, but nevertheless, it still possesses a risk considering that register() function can add new AMM and Pool contracts.

Tool used
Manual Review

Recommendation
It would make the codebase more coherent and robust if safe transfers were used for all the tokens, including tempusAMM and Principal/Yield share. Also, special attention needs to be taken to the register() function, because it allows whitelisting of both new AMMs and Pools, and usually it is not a good practice to use the same registry for different kinds of contracts.

Issue 07

Summary
TempusAMM line 32:
Contract TempusAMM does not explicitly implement the ITempusAMM interface. contract TempusAMM is BaseMinimalSwapInfoPool, StableMath, IRateProvider

Severity
Informational

Vulnerability Detail
TempusAMM is an implementation of ITempusAMM, but it does not explicitly declare this. It does not cause any direct issues because all the required functions are present. However, it is always a nice practice to enforce this, as all the contracts that rely on ITempusAMM, expect that the implementation complies with this interface.

Impact
Even though there are no problems with this now, it is always a good practice to enforce a compile-time validation that the implementation matches the interface.

Tool used
Manual Review

Recommendation
Declare like TempusAMM is ITempusAMM

Issue 08

Summary
Insufficient validation of values from Chainlink Oracle.
ChainlinkTokenPairPriceFeed line 26:
(, int256 latestRate, , ,) = chainLinkAggregator.latestRoundData();

Severity
Informational

Vulnerability Detail
There is no check if the return values indicate stale data. This could lead to stale prices according to the Chainlink documentation:
“if answeredInRound < roundId could indicate stale data.”
“A timestamp with zero value means the round is not complete and should not be used.”

Impact
An oracle in this protocol is not playing a significant role, it is only used for statistics, so the impact is minimal.

Tool used
Manual Review

Recommendation
Add missing checks for stale data. An example solution:

```
(
     uint80 roundID,
     int256 latestRate,
     ,
     uint256 timeStamp,
     uint80 answeredInRound
   ) = chainLinkAggregator.latestRoundData();
   require(
     timeStamp != 0,
     "ChainlinkTokenPairPriceFeed::getLatestAnswer: round is not
complete"
   );
   require(
     answeredInRound >= roundID,
     "ChainlinkTokenPairPriceFeed::getLatestAnswer: stale data"
   );
```

Issue 09

Summary
Inject IENS address in the constructor.
ChainlinkTokenPairPriceFeed line 13-15:

```
    // See https://docs.chain.link/docs/ens/. This may need to be
updated should ChainLink deploy
    // on other networks with a different ENS address.
    IENS private constant ENS =
IENS(0x00000000000C2E074eC69A0dFb2997BA6C7d2e1e);
```

Severity
Informational

Vulnerability Detail
As the codebase is growing, you might forget to review every comment and even though the usage of Oracle is not of paramount importance, it would make it more robust if you inject the address of an Oracle in a constructor instead of hardcoding it.

Impact
An oracle in this protocol is not playing a significant role, it is only used for statistics, so the impact is minimal.

Tool used
Manual Review

Recommendation
Inject hardcoded parameters in the constructor.

Issue 10

Summary
Decimals of backing tokens might change.

Severity
Informational

Vulnerability Detail
This is not a very likely scenario but could happen if the decimals value of the backing token changes, then values like backingTokenONE will start misbehaving because it is only initialized once in the constructor. However, as mentioned this is more of a theoretical issue and would make it pretty expensive to always fetch an up-to-date value, but you can consider adding an admin function to update the decimals value.

Impact
More of a theoretical issue, nice to have but not mandatory.

Tool used
Manual Review

Recommendation
An example improvement:

```
function updateBackingDecimals() external onlyOwner {
    uint8 backingDecimals = _backingToken != address(0) ?
IERC20Metadata(_backingToken).decimals() : 18;
    backingTokenONE = 10**backingDecimals;
}
```