# Smart Contract Audit

Tempus Finance

coinspect

tempus

# Tempus

## Smart Contract Audit

v211013

Prepared for Tempus • October 2021

## 1. Executive Summary

In **September 2021**, Tempus engaged Coinspect to perform a source code review of their on-chain derivatives marketplace. The objective of the project was to evaluate the security of the smart contracts.

Tempus contracts are clearly documented and the tests included in the project provide good coverage.

No high-risk vulnerabilities that would result in stolen users funds were identified. However, one medium-risk issue (high impact, but low likelihood) was reported that could impact user funds if current security assumptions change in the future. Another medium-risk issue was reported related to the power the pool owners possess to update fees without constraints and that could be abused to harm users if the account were compromised.

Additionally, several recommendations are provided with the goal of further increasing the security of the platform.

The following issues were identified during the assessment:

| High Risk | Medium Risk | Low Risk |
|---|---|---|
| 0 | 3 | 0 |
| Fixed | Fixed | Fixed |
| - | 100% | - |

During **October 2021**, Coinspect verified that the issues reported had been correctly fixed by the Tempus team, and this report was updated to reflect that.

## 2. Introduction

The audit started on August 30 and was conducted on the `coinspect-audit-1` branch of the git repository at https://github.com/tempus-finance/tempus-protocol as of commit `ee5964bd416770e00639a4053b45f346e4bf8b93` of **August 30, 2021.**

The scope of the audit was limited to the following Solidity source files, shown here with their sha256sum hash:

```
5b6e4d28629c7a79d4459c20a29a52511a201d250e294128da2a07392a005935  ./protocols/lido/ILido.sol
8a2f219345a683b3e5cbd8e2dcbfee87a247388b4714d530ef90d4393da93a69  ./protocols/compound/ICToken.sol
4502b84ac9b51d5411746c8f22132befb0103bf454309b0b78eabd6970bd5ff5  ./protocols/compound/ICErc20.sol
b85b62c35fdbbcfee563d24d02ec06f2ce79a730e6e226fb3fbee423822a5ab4  ./protocols/compound/IComptroller.sol
6c64c9e97c6c84dc8f647d44ba49b653e4bceb0eb586ca03db2041acf44f5648  ./protocols/aave/IAToken.sol
f0104e96f0dc3f6373b73ab01aae4ce005ebb2c39b5e846eeafcf3252500c3bb  ./protocols/aave/ILendingPool.sol
e53250bd47efc21eeec629c62e833ad30611bcaa4c72f1f6a552337e3dea82f3  ./mocks/lido/LidoMock.sol
0746b6511cd9d16ca569847eb96f89fc951f5463e2373ff2e27030365609d843  ./mocks/lido/StETH.sol
77c28395f5128df496563bb6d8b929143c638c54ce5239d4382facebffdd25b4  ./mocks/compound/CErc20.sol
ede80370a3e7440b84d02b33d507730153d211c8f3c2b256a889d91e0407cee2  ./mocks/compound/CTokenInterfaces.sol
7fdc76a53c42ea71bd369ec22860de6c152cfd494a4f1c02af5623e4ada9f31a  ./mocks/compound/ComptrollerStorage.sol
978177c66abac4768c229b02ea798924be68f8f531a69b43ae6944b80dc583b7  ./mocks/compound/ComptrollerMock.sol
2d9aebda970411162ce08df9d0eb44561782bec84c79925500d1dbf483534622  ./mocks/compound/CTokenMock.sol
92a93a59a6facc7105b48ed9aa476c011ed0f346986540dd066fd68330abb400  ./mocks/compound/ComptrollerInterface.sol
3b2d7bd4a4375f3317089415958844d2402e2821e3a7b745a6807f4aa99e6d95  ./mocks/aave/AavePoolMock.sol
1b51f2c1725f2f6df66be949fa6b60ce75522544d68e68cea01b7ae16a49d325  ./mocks/aave/WadRayMath.sol
c13497111087cfd4cc4b9882fa8066a90119dd478cebbcf06be0b7729248d370  ./mocks/aave/ATokenMock.sol
82a7eae58492d55b2b7a5f5d0324e38a2a93b137215bc0d8e8aa8c19ea411fbe  ./stats/ITokenPairPriceFeed.sol
b42d38dbfcabfa71534374152b3aed995e1a113e770105ae5b1f77f938bc8c0d  ./stats/ChainlinkTokenPairPriceFeed/IENS.sol
4b056b69ae0b7605f72bdeea43c8a2add317389ddb42def9ca9d1bfc3a0977bd
./stats/ChainlinkTokenPairPriceFeed/ChainlinkTokenPairPriceFeed.sol
56d3a1b7bef8cf124887dffccf4f05f14cc015e249cd4fbbc670986cad616820
./stats/ChainlinkTokenPairPriceFeed/IChainlinkAggregator.sol
0847e91fdd10d28a44b1e4dd5a094282163a95f8fbf44c829c05d809557118b9  ./stats/Stats.sol
aff1de973bf67c6df47a340c702753d56aced09196a5c48a7feebf275bac98fe  ./math/Fixed256x18.sol
840d89dbe5d33f8b6da5f8a63952eda85dfb2c34338d3c6be45e8d69aecd9dc5  ./utils/PermanentlyOwnable.sol
6c60930c926c93f3c47250c49168506eecdc00fc95fd6aba6b6d8da3e1d40b46  ./token/PoolShare.sol
0331eb2ae72a57de4bac35c03e0016258b68b91738cc37f8fc448a413308c0fd  ./token/YieldShare.sol
60ac71ccc8c38d97bfe0c6c1a5d5d3d733764bf8bef4c83e02c407c8f759b6ce  ./token/TempusToken.sol
be0351ae2a3aee6aa53a483031e8200f2a11e143a556136a20fcd285992edc59  ./token/ERC20FixedSupply.sol
aef4de5fd37327b01a6b05628da8f6a772b8965e8ab5915421a4dccccc320146  ./token/IPoolShare.sol
29b8b4bd48aa14d335f7054c06ba83903d8f0189768bff17943448e8e12bdb37  ./token/ERC20OwnerMintableToken.sol
36f8adfc19d6ed53a608b57920c5280945b129ac897239853e290b6ddcefb291  ./token/PrincipalShare.sol
25c0d1ef81134c81be850c66157df6f01cda7e51102444e571bb3ec29e62f122  ./pools/CompoundTempusPool.sol
03fe7110033fcc54307200dbde00e5441a68f90961e6ac5f4e29e6edcb6334a3  ./pools/AaveTempusPool.sol
1dca8d086cdba0c1c3d8400314db9dc6dbd5a13077c48de79679a19f41a554e3  ./pools/LidoTempusPool.sol
3253db9ab498d966a48ff892f70a38a9a12aec260ee5d8a8948b981367fe5dbd  ./TempusPool.sol
31fe02244c8e2d0a16a6a671dcef5cd3b90a4a974a45f74ff2b63b6d1d14dc36  ./ITempusPool.sol
f458161e8b8e43737c162b3ddd71e0644a41cf6f77d391dfdafa856072200206  ./amm/mocks/TempusShareMock.sol
aaae15490422332f3ceb18f9a29d2091bc72965d13b7144437246ea27960f847  ./amm/Authorizer.sol
7eab3535167bfcfdde357cfdddc5601ba30b7b01069a6a5f9e7645faa2db8f18  ./amm/interfaces/IVault.sol
450565d938ef48342fcaf449012abcf431137f8b56a52cd00f4833886a1f1706  ./amm/interfaces/IRateProvider.sol
fbd26660d07cbb22e963b45eb6ca76ca269556b0f06d0c96dcdeafeb554503a5  ./amm/interfaces/ITempusAMM.sol
e3755ea2db9796103f1285a0e89516b990c4741ab89b803f1c7fdf66bd2ec51b  ./amm/TempusAMMUserDataHelpers.sol
171da6d69d5d1db4ccc3b2071c5ba5a6c5f22367f7f01658f74a0fe4766bd36f  ./amm/StableMath.sol
0f80c002fdc4a9aacd1fb5e3379ec9401d445db9d41cc88f0d655144296f4768  ./amm/TempusAMMFactory.sol
```

```
9f9b291cfb6bd8cccc0b06b4d5967a053c267d1c09bc61616c23ea6660ce4419  ./amm/TempusAMM.sol
e029a1af1a4e6e615907c5d3a6ec79ae51f9cd16cbc460f1caa544593eca4bad  ./amm/Vault.sol
19c718dd546e90122d7c373d4dfd12d585f2dc646a80380862c4552d69efe81d  ./TempusController.sol
```

During October 2021 Coinspect audited the changes in the updated coinspect-audit-2 branch of the GitHub repository tempus-finance/tempus-protocol as of commit `f66be61a187423f73cb25ad31934ce3afd3c10f4` of **October 1, 2021**. The following pull requests were in scope as per the client's request:

1. https://github.com/tempus-finance/tempus-protocol/pull/323
2. https://github.com/tempus-finance/tempus-protocol/pull/331
3. https://github.com/tempus-finance/tempus-protocol/pull/314
4. https://github.com/tempus-finance/tempus-protocol/pull/324
5. https://github.com/tempus-finance/tempus-protocol/pull/337
6. https://github.com/tempus-finance/tempus-protocol/pull/310

Tempus interacts with the following external protocols: Lido, AAVE, Compound and Chainlink. These external dependencies were not in scope for this engagement.

TempusAMM is based on Balancer's V2 Vaults. The files imported from Balancer's project's repository were not fully reviewed.

## 3. Assessment

Tempus is an on-chain derivatives marketplace that allows users to optimize their existing exposure to variable yield according to their risk profile. Specifically, Tempus allows users to dynamically adjust risk and convert variable yield rates into fixed rates. In addition, it enables users to earn additional yield as liquidity providers if they desire. Tempus allows users to deposit various yield bearing tokens (YBT) on its platform. The following tokens are currently available:

- cTokens (Compound Interest Bearing Tokens)
- aTokens (Aave Interest Bearing Tokens)
- stETH (Lido Staked ETH)

The YBTs are deposited into pool contracts with specific maturity dates. Users can also deposit the underlying backing tokens (BT) directly into Tempus, which will in turn deposit the funds in the integrated DeFi platforms. These user interactions are handled by the `TempusController` contract.

In exchange for the users' tokens, Tempus mints an equal number of Principals and Yields. Users can trade these tokens against each other on TempusAMM. Depending on the Principal/Yield ratio a user holds, the exposure to the underlying yield changes for the user.

For example, by selling all the Yields and holding only Principals, users obtain a fixed rate until the maturity date which is specified for each pool on Tempus. By selling all Principals to hold only Yields, users are exposed to the underlying yield on a leveraged basis. The TempusAMM is therefore a vehicle for users to re-allocate the risk of interest rate fluctuations between each other. Users can also use these Principals and Yields on Tempus to provide liquidity to the AMM and earn some additional yield. The additional yield comes from the transaction fees that are generated when traders swap the Principals for the Yields.

All contracts are specified to be compiled with Solidity version 0.8.6, except for the AMM component ones that are based on Balancer's contracts. The project includes comprehensive unit tests targeting the contracts functionality with good coverage. In addition, all contracts are clearly documented.

The system is formed by two main components: **TempusPools** and **TempusAMM**. TempusPools have a fixed maturity duration and receive users´ tokens. They allow users to redeem their tokens after the maturity date is due. Alternatively, users can perform an early redemption without penalties, but they must provide the same amount of Principal and Yield tokens to get back their YBTs.

The TempusAMM allows users to swap Principal and Yield tokens. It is based on Curve's StableSwap AMM (which is intended for stable coins), but token balances are scaled. Tempus implementation is based on the Balancer v2 stable pool contract located in github.com/balancer-labs/balancer-v2-monorepo/tree/master/pkg/pool-stable.

Contracts imported from Balancer by the TempusAMM's contracts implement critical functionality (`BaseGeneralPool.sol, Vault.sol, Authorizer.sol, VaultAuthorization.sol, FlashLoans.sol,` and `Swaps.sol`). **These Vault related contracts hold token balances, provide flash loans, and implement the swap functionality**. In Tempus, the tokens exchange rate logic was modified and it is adjusted depending on how close the maturity date of the pool is: when the maturity date is far away, the rate is calculated based on liquidity, but when the maturity date is close, the expected price based on past yields is used.

Regarding system privileged roles, it is worth mentioning that the **`TempusController` owner is permanent and cannot renounce ownership.** The same is true for `TempusPools`. The `TempusPool` owner has the ability to arbitrarily change fees (applied during user deposits and redeems) for the pool. Moreover, the contract allows the owner to withdraw the accumulated fees to any account.
On the AMM component side, role-based authentication is used. The TempusAMM contract owner has the ability to start and stop the amplification factor update procedure, which affects how all exchange rate calculations are performed. This **amplification factor update procedure is triggered off-chain** and results in this parameter being slowly updated (the maximum being double the daily rate) over a period of time (minimum of one day), preventing it from being manipulated abruptly.

## General recommendations

1. Currently, if a user transfers tokens directly to the contract instead of using the expected interface (e.g., the deposit functions), those tokens will be accounted as if transferred by the next user calling the deposit functions. In order to improve fairness, the protocol could distribute those funds among all existing users; or could accumulate those funds in a separate contract in order to reimburse the sender. An opportunistic bot could be deployed to wait for funds to be transferred to the pool (maybe induced by social engineering) and steal them. Tempus considered this suggestion but because of the complexity of implementing this suggestion and the unlikely exploitation scenario, it was decided it was not worth it.
2. Beware of utilizing `transfer` as a protection from reentrancy, as opcode costs may change in the future. It is not recommended to rely on opcode gas costs, as protocol upgrades might render the protection provided by `transfer` and `send` useless (as it almost happened with Ethereum Constantinople update and EIP 1283: Constantinople enables new Reentrancy Attack | by ChainSecurity | ChainSecurity).
3. Make TempusPools `pausable` in order to allow a quick reaction in case of a vulnerability in one of the pools. Note the `TempusAMM` is `pausable`, except the `_exitExactBPTInForTokensOut` function, which can always be called by design.
4. Constantly monitor upstream fixes to the files imported from Balancer repository. This should be a continuous process once the contracts are deployed. It has happened before that a fix (silent and not silent as well) in an upstream project was ignored by a project which ended up with vulnerable code not being updated.
5. Implement a stalled price feed detection mechanism to enable users to react if a Chainlink oracle has not been updated in a period of time. As the team explained this is not considered important as only view methods providing stats (TVL) for the frontend use the price feed. However, scenarios must be considered where off-chain code might choose to trust the frontend as the source of data for critical decision making processes, and presenting them stale data without a warning/flag that allows them to know this data might be stale could have bad consequences.
6. Evaluate and remove all TODO comments in the source code.

## 4. Summary of Findings

| Id | Title | Total Risk | Fixed |
|---|---|---|---|
| TEM-1 | Function deposit mints more tokens than it should | Medium | ✔ |
| TEM-2 | Lack of reentrancy protection mechanisms | Medium | ✔ |
| TEM-3 | Pool owner can set arbitrary fees | Medium | ✔ |

# 5. Detailed Findings

## TEM-1 Function deposit mints more tokens than it should

| Total Risk | Impact | Location |
|---|---|---|
| Medium | Medium | `TempusPool.sol` `TempusController.sol` |
| **Fixed** | **Likelihood** | |
| ✓ | Medium | |

### Description

The `TempusPool` and `TempusController` contracts could be abused to mint more tokens than the amount corresponding to the yield-bearing or backing tokens actually deposited.

The `deposit` function does not verify if the yield bearing token `safeTransferFrom` resulted in the expected amount being transferred. Instead, if the call does not revert, it trusts the total amount was transferred.

The total `yieldTokenAmount` is assumed to be transferred after the transfer and used to calculate the number of shares to mint to the depositor.

The same scenario applies to the `depositYieldToken` and `depositBacking` functions in the `TempusController` contract.

There are tokens that transfer less than the specified amount. For example, the USDT token's `transfer` and `transferFrom` functions (Tether: USDT Stablecoin | 0xdac17f958d2ee523a2206206994597c13d831ec7) deduct a fee for each transfer if a fee percentage is configured. While it is not configured to take fees right now, this could change in the future.

Similar scenarios are possible in the future depending on the ERC20 yield-bearing and bearing tokens that are allowed to be deposited in TempusPools.

For examples of this issue being exploited in production, refer to:

- Hacker Drains $500K From DeFi Liquidity Provider Balancer
- Incident with non-standard ERC20 deflationary tokens | by Mike McDonald | Balancer Protocol

These examples detail how this vulnerability has been exploited to drain funds from the Balancer contracts.

### Recommendation

It is advised to check the balance of the contract before and after the `transferFrom` call is performed in order to determine the exact amount that was received and to bulletproof TempusPools for future protocols and token integrations.

### Status

This issue was addressed by the Tempus team during the engagement in https://github.com/tempus-finance/tempus-protocol/pull/310.

This issue should be clearly documented for developers and reviewers of future integrations to keep in mind when interacting with external contracts.

## TEM-2 Lack of reentrancy protection mechanisms

|  |  |  |
|---|---|---|
| Total Risk<br>**Medium** | Impact<br>High | Location<br>`TempusController.sol` |
| Fixed<br>✓ | Likelihood<br>Low |  |

### Description

Tempus Controller functions have no protection against reentrancy attacks. This could allow attackers to affect the protocol integrity by exploiting cross-function reentrancy.

It is strongly recommended to prevent reentrant calls in Tempus, without making assumptions about any other external contract's ability to reenter Tempus.

In several functions such as the external functions `exitTempusAMMAndRedeem` and `completeExitAndRedeem`, the checks-effects-interactions pattern is not respected.

Even if the currently integrated protocols might not allow attackers to reenter the contract right now, this security relevant assumption could change in the future. For example, the two following scenarios are possible:

a. New protocol integrations that rely on tokens that include callback functions for transfers (ERC771, ERC721, other custom tokens) are added to Tempus.
b. One of the existing integrations upgrades its contracts and callback functions are introduced

This scenario was the exact cause for the recent CREAM protocol exploit that resulted in lost funds when the AMP token was integrated: "*The AMP token contract implements ERC777, which has the _callPostTransferHooks hook that triggers tokensReceived() function that was implemented by the recipient.*" as detailed in CREAM Finance Post Mortem: AMP Exploit | by CREAM | CREAM Finance | Aug, 2021.

## Recommendation

Coinspect recommends that all user interactions triggered via public and external interfaces that lead to an external call (including token and ETH transfers) be protected from reentrancy by the utilization of a reentrancy guard. For example, Open Zeppelin Contracts provides `ReentrancyGuard` and the `nonReentrant` modifier that can be used to prevent nested function calls.

## Status

This issue was addressed by the Tempus team in the following PRs:

- https://github.com/tempus-finance/tempus-protocol/pull/323
- https://github.com/tempus-finance/tempus-protocol/pull/324

## TEM-3 Pool owner can set arbitrary fees

| Total Risk | Impact | Location |
|---|---|---|
| Medium | High | `TempusPool.sol` |
| Fixed ✓ | Likelihood Low | |

### Description

The pool owners possess the ability to update fees arbitrarily without constraints, and that could be abused to harm users if an owner's account is compromised.

A rogue pool owner could front run users' deposits and redeems in order to steal all the funds by setting a 100% fee.

This is aggravated by the fact that those fees can be transferred to any account with the `transferFees` function.

### Recommendation

Consider restricting fee updates by enforcing a maximum fee percentage. This would limit the damage in case the pool owner account is compromised. Another option would be to impose a delay for fee updates to take effect.

Consider only allowing fees to be withdrawn to a separate account so the attacker must control two different accounts in order to steal funds by resetting fees.

### Status

This issue was addressed by the Tempus team in the following PRs:

- https://github.com/tempus-finance/tempus-protocol/pull/331
- https://github.com/tempus-finance/tempus-protocol/pull/314

## 6. Disclaimer

The information presented in this document is provided "as is" and without warranty. The present security audit does not cover any off-chain systems or frontends that communicate with the contracts, nor the general operational security of the organization that developed the code.