CERTIK

Security Assessment

Unreal Finance

CertiK Verified on Oct 13th, 2022

CertiK Verified on Oct 13th, 2022

Unreal Finance

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
DeFi	Ethereum (ETH)	Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 10/13/2022	N/A

CODEBASE
https://github.com/unrealfinance/contracts-v2/
...View All

COMMITS
93b2e0ee5ea0a881fdabbd08ebb74a483d875b16
...View All

Vulnerability Summary

18	14	2	1	1	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
3	Major	1 Resolved, 2 Mitigated	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
1	Medium	1 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
8	Minor	6 Resolved, 1 Partially Resolved, 1 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
6	Informational	6 Resolved	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | UNREAL FINANCE

CODEBASE | UNREAL FINANCE

Repository

https://github.com/unrealfinance/contracts-v2/

Commit

93b2e0ee5ea0a881fdabbd08ebb74a483d875b16

AUDIT SCOPE | UNREAL FINANCE

14 files audited ● 5 files with Acknowledged findings ● 4 files with Mitigated findings ● 4 files with Resolved findings ● 1 file without findings

ID	File	SHA256 Checksum
AFB	contracts/futures/AFuture.sol	d56468f76a7f5d5da0610e194a3f6311bfa8c7748f61e3c3e10eb60f5a2528b7
AVF	contracts/futures/AaveV3Future.sol	2c6b3643d24db9c7ac8a2c4e03488c021787169cf4f93980a896c0f65957fb05
CFB	contracts/futures/CFuture.sol	844af1831876761df6f16633908428bdd7fecb29bed14870981ec591e45872a5
FBB	contracts/futures/FutureBase.sol	eed311c8f78b21572973ddb6f4bcc3ce91b83b844e7c257f013117bb4eb0a3ce
YFB	contracts/futures/YFuture.sol	e9d1d861f8413f6348ec0abe3c208731bb61ed0358de20877bbe127eb8cf6575
OTB	contracts/tokens/OwnershipToken.sol	ecb56627a5ec73bae31eb0cb37c85a1206cb68e3654a3318364c596cafce2bf3
YTB	contracts/tokens/YieldToken.sol	ced2cede8f8d80a591c2f28eb77fc96b73701119b4bc11ee14cb36463a5ec4c0
COR	contracts/Core.sol	04c36c455ddb722aeb049a605432df4cf8d60aed5c4006e7bfb9f6f45d827aeb
TRE	contracts/Treasury.sol	603abc3b2231ec3533ad5d4812178496c2c292411f08af1ff36c45d1bfb515ad
DTB	contracts/libs/DateTime.sol	feddff6e71b0cc8e09fa9992cf76a1514b2df35209af4b733a6a02ae972a35b0
DER	contracts/libs/DetailedERC20.sol	064c4336d55e7990550f58f8fceadde0c3962faa41708ce6f78af0e98b678932
MLB	contracts/libs/MathLib.sol	edec61bf5e7f8b37fac095655a6e37f29f4337bd5bfb1586cb3f36e66548f0de
UTI	contracts/libs/Utils.sol	c2f9e6d21d63796c97476a124d69fb215bf6a16fcc2270e72d28842f51a4975f
DSS	contracts/.DS_Store	c86aa53289f61c13144488d18a9068c1619a6ff49e916656175bcfd3a79bc405

APPROACH & METHODS | UNREAL FINANCE

This report has been prepared for Unreal Finance to discover issues and vulnerabilities in the source code of the Unreal Finance project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

FINDINGS | UNREAL FINANCE

18	0	3	1	8	6
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Unreal Finance. Through this audit, we have uncovered 18 issues ranging from different severity levels. Utilizing the techniques of Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
CON-01	**Centralization Related Risk**	**Centralization / Privilege**	**Major**	**Mitigated**
CON-02	Unused Return Value	Volatile Code	Minor	Resolved
CON-03	Check Effect Interaction Pattern Violated	Logical Issue	Minor	Partially Resolved
CON-04	Missing Input Validation	Volatile Code	Minor	Resolved
COR-01	**Centralized Control Of Contract Upgrade**	**Centralization / Privilege**	**Major**	**Mitigated**
COR-02	Unprotected Upgradeable Contract	Language Specific	Major	Resolved
COR-03	Shadowing State Variable	Coding Style	Medium	Resolved
COR-04	No Validation Check That `streamKey` Is Not `bytes32(0)`	Inconsistency	Minor	Resolved
COR-05	Owner Inputs `_bytecode` For `create2`	Volatile Code	Minor	Resolved
COR-06	No Check `amountBurned` Is Positive Before `claimYield()` Is Called	Inconsistency	Minor	Resolved

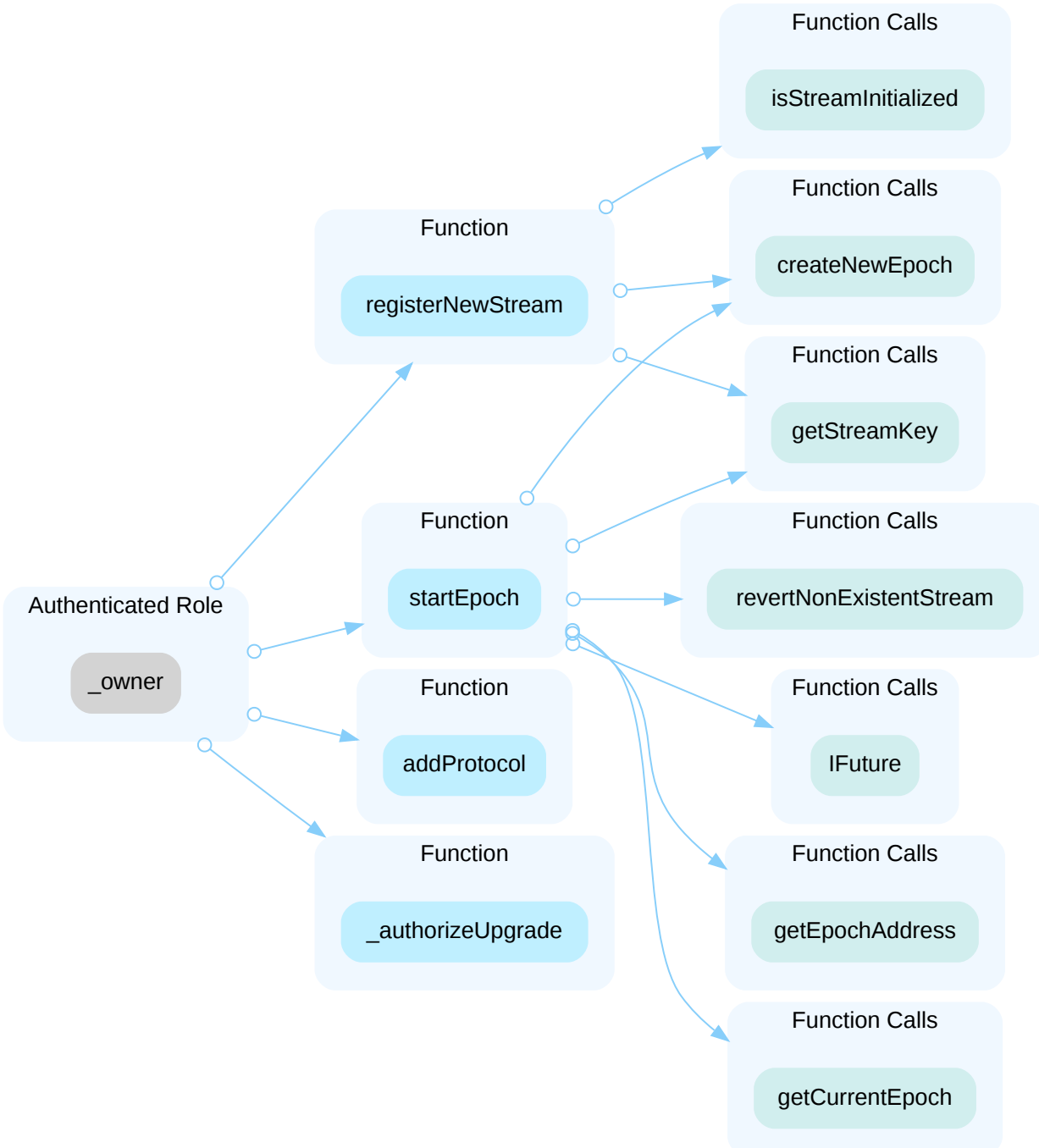
ID	Title	Category	Severity	Status
FUT-01	Third Party Dependency	Volatile Code	Minor	Acknowledged
FUT-02	Unchecked ERC-20 `transfer()` / `transferFrom()` Call	Volatile Code	Minor	Resolved
CON-06	Unlocked Compiler Version	Language Specific	Informational	Resolved
CON-07	Missing Emit Events	Coding Style	Informational	Resolved
COR-08	`_protocol` May Be Different From What `_bytecode` Describes	Coding Style	Informational	Resolved
COR-09	`amountUnderlying` May Be Larger Than `totalSupply`	Mathematical Operations, Logical Issue	Informational	Resolved
DER-01	`_decimals` Can Be Made Private	Language Specific	Informational	Resolved
FUT-03	Incompatibility With Deflationary Tokens	Logical Issue	Informational	Resolved

CON-01 | CENTRALIZATION RELATED RISK

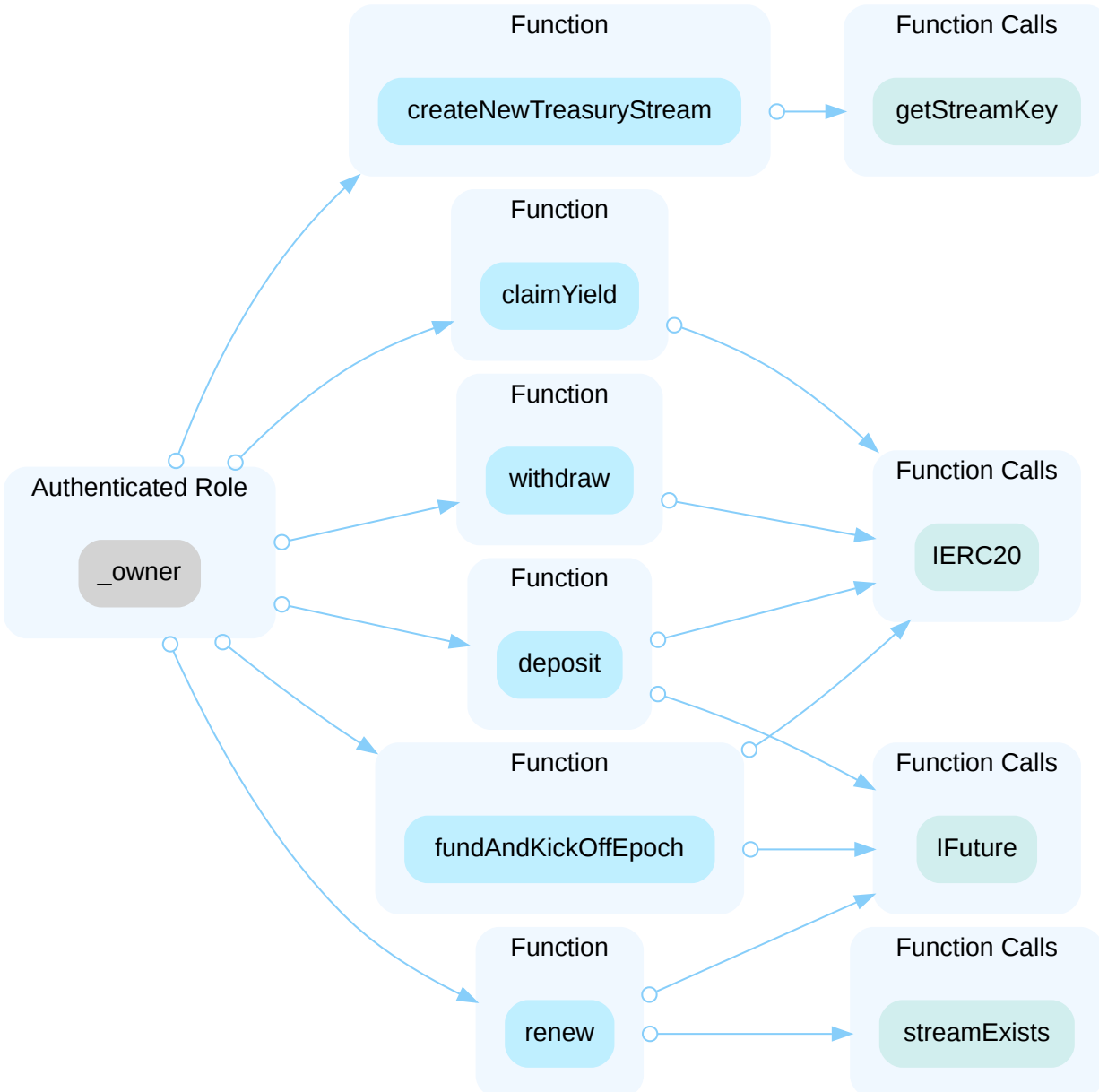
Category	Severity	Location	Status
Centralization / Privilege	Major	contracts/Core.sol: 80, 89, 121, 149; contracts/Treasury.sol: 38, 57, 80, 100, 119, 134; contracts/futures/AFuture.sol: 36; contracts/futures/AaveV3Future.sol: 36; contracts/futures/CFuture.sol: 35; contracts/futures/FutureBase.sol: 77, 82, 86, 92, 96, 102, 124, 129, 228; contracts/futures/YFuture.sol: 32; contracts/tokens/OwnershipToken.sol: 28~29, 32~33; contracts/tokens/YieldToken.sol: 28~29, 32~33	Mitigated

Description

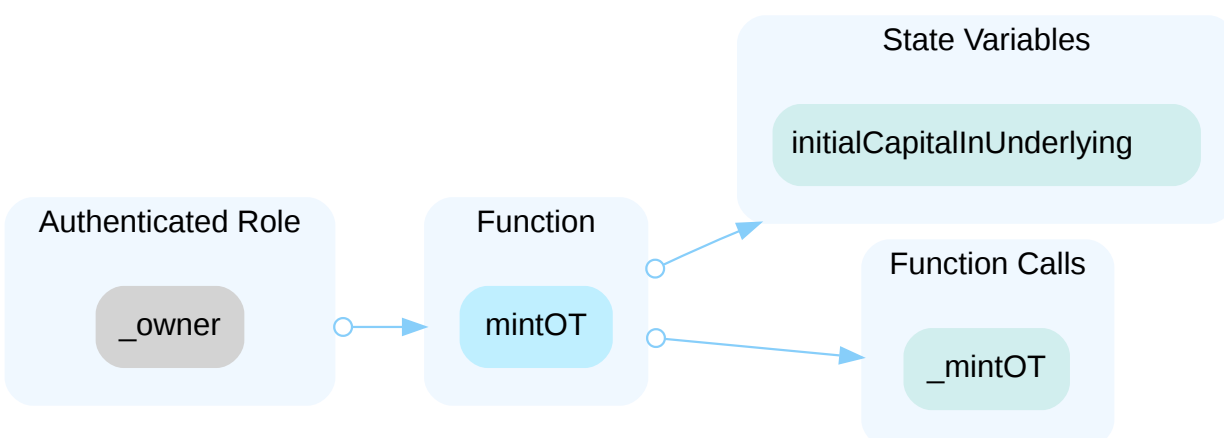
In the contract `Core` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and change the implementation through `upgradeTo()`. In turn, since this contract is the owner of each futures contract and is meant to be the owner of the `Treasury` contract, tokens can be sent to unintended address from each.



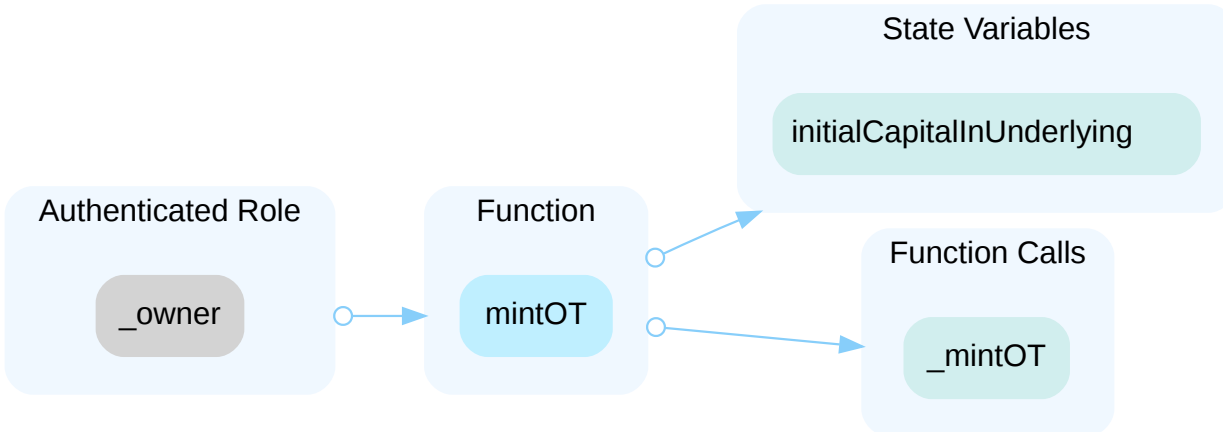
In the contract `Treasury` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and send underlying asset tokens to unintended addresses, draining tokens from the contract.



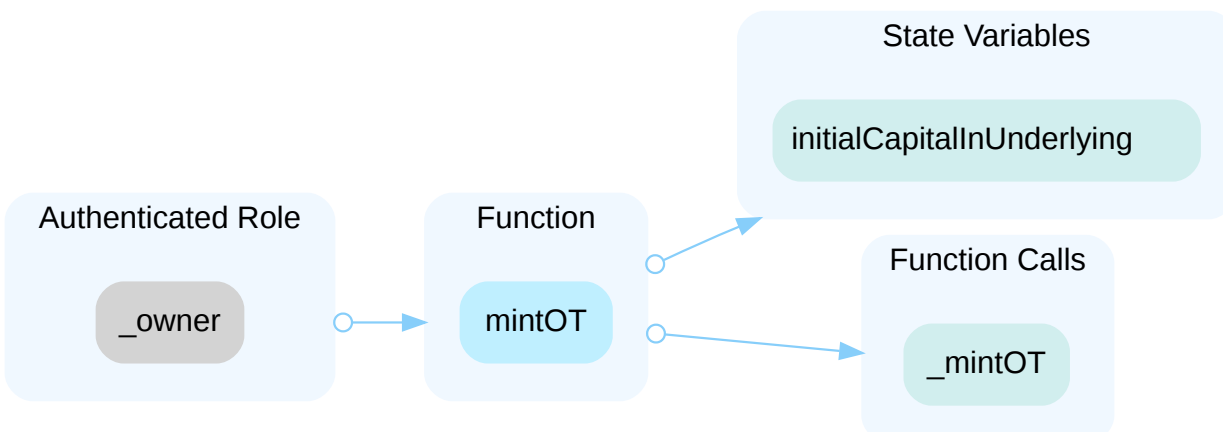
In the contract `AFuture` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and mint an unbounded number of `oT` tokens, then burning them in exchange for the underlying asset token through the `Core` contract.



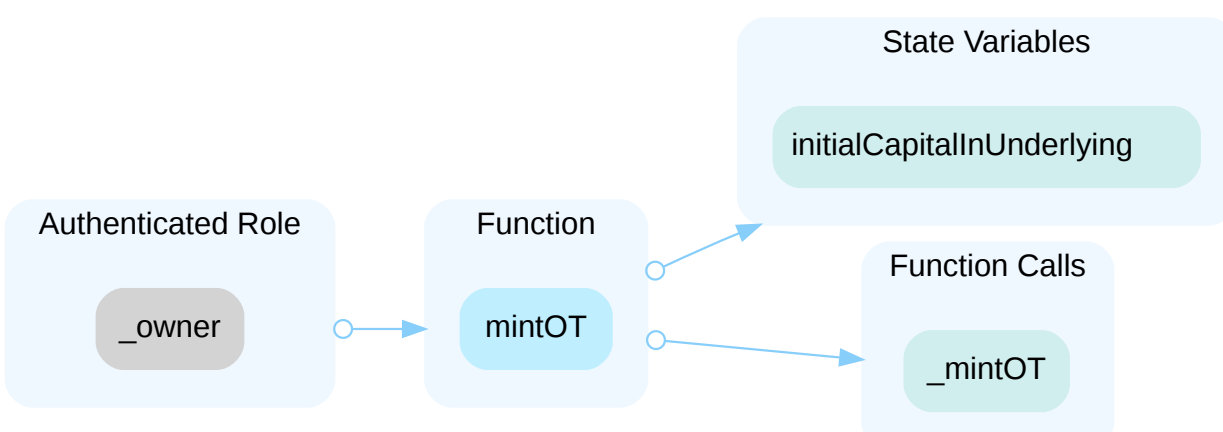
In the contract `AaveV3Future` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and mint an unbounded number of `oT` tokens, then burning them in exchange for the underlying asset token through the `Core` contract.



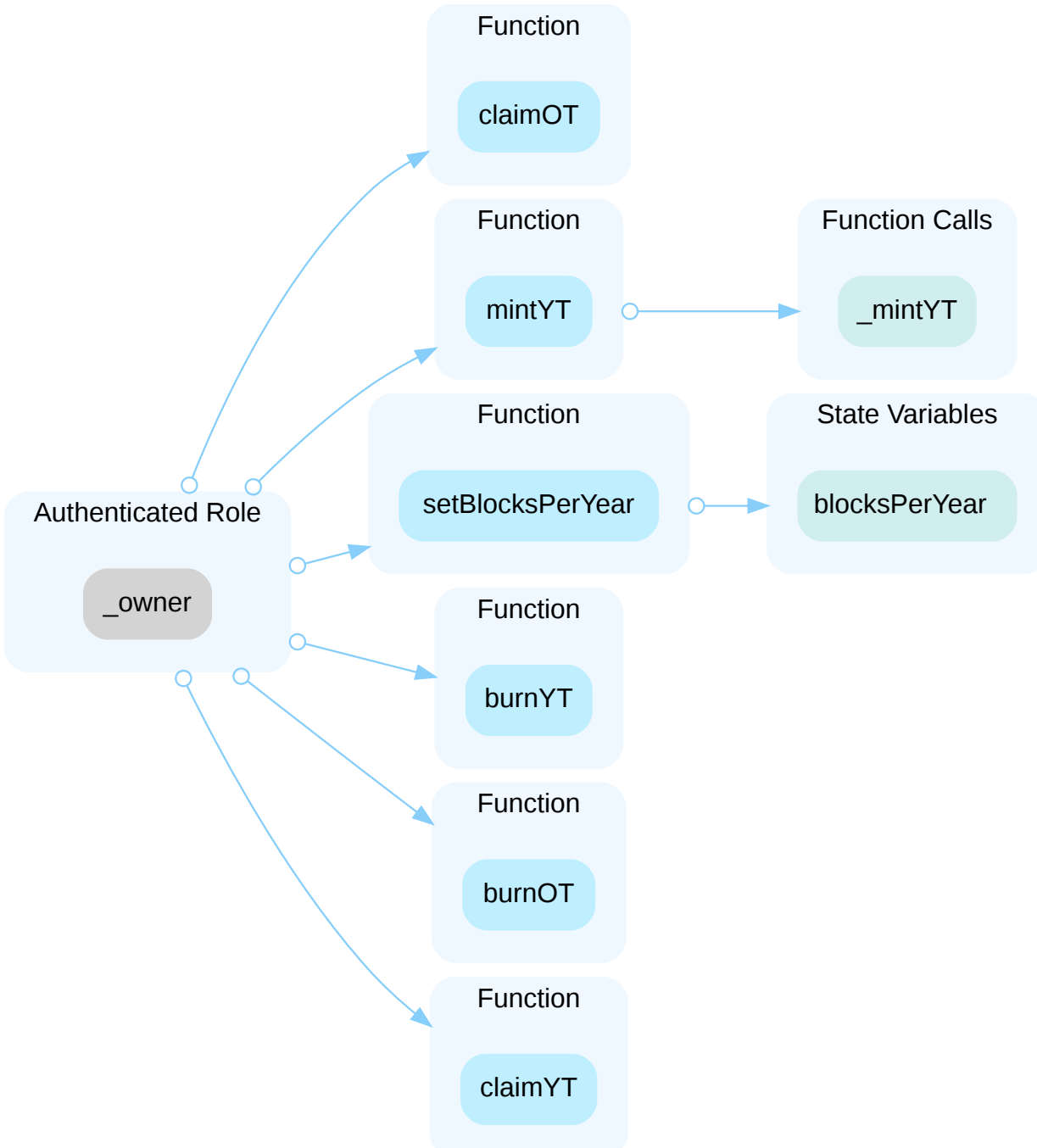
In the contract `CFuture` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and mint an unbounded number of `oT` tokens, then burning them in exchange for the underlying asset token through the `Core` contract.



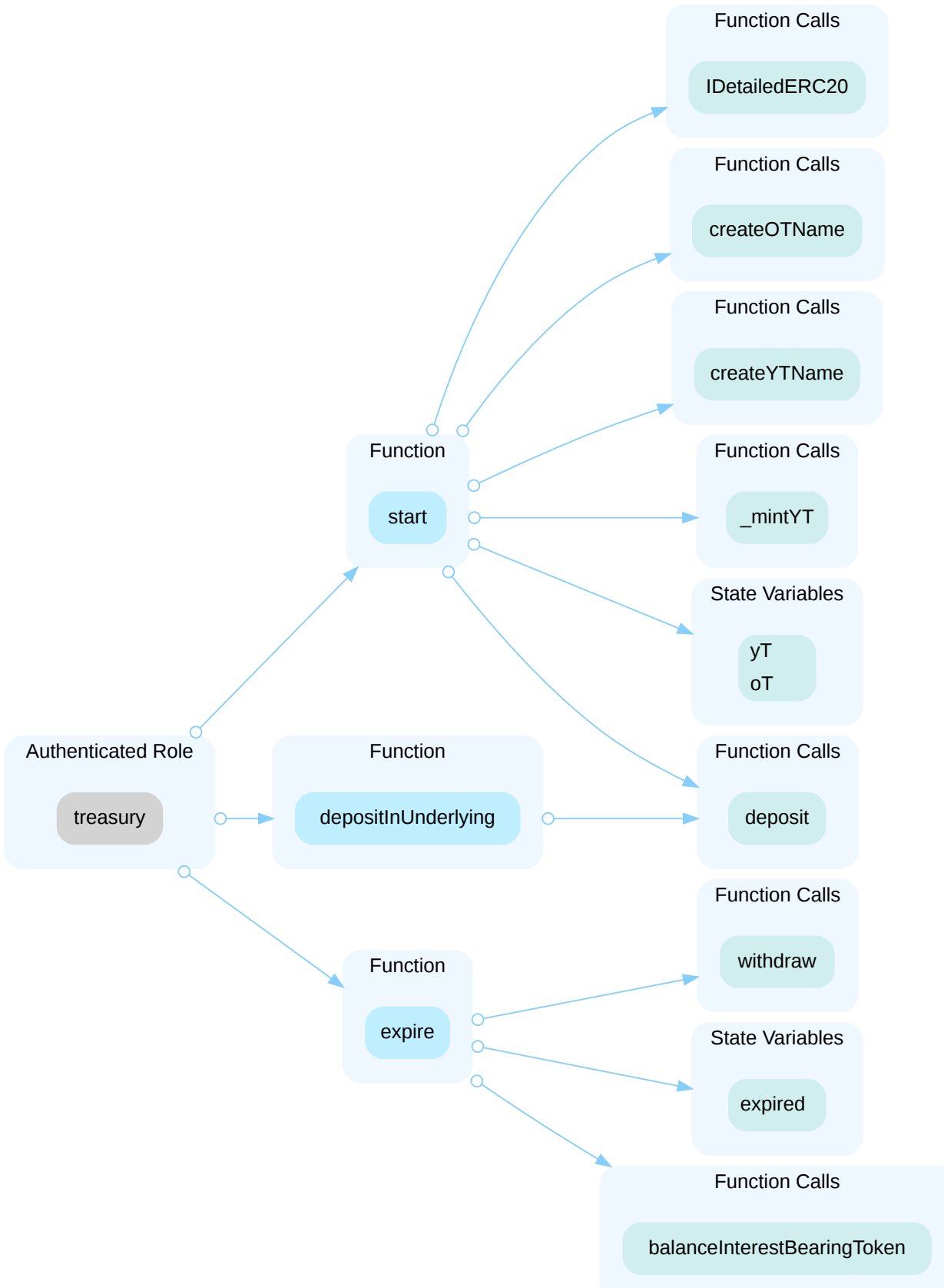
In the contract `YFuture` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and mint an unbounded number of `oT` tokens, then burning them in exchange for the underlying asset token through the `Core` contract.



In the contract `FutureBase` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and mint an unbounded number of `oT` tokens, then burning them in exchange for the underlying asset token through the `Core` contract.



In the contract `FutureBase` the role `treasury` has authority over the functions shown in the diagram below. Any compromise to the `treasury` account may allow the hacker to take advantage of this authority and call `expire()` which calls internal `withdraw()`, sending all underlying asset tokens to the `treasury` address.



In the contract `OwnershipToken` the role `MINTER_ROLE` has authority over the functions `burn()` and `mint()`. Any compromise to the `MINTER_ROLE` account may allow the hacker to take advantage of this authority and mint an unbounded number of tokens to an unintended address, which could then be exchanged for the underlying asset via the `Core` contract.

Moreover, the hacker could burn any amount of tokens from any holding address. The same `MINTER_ROLE` vulnerability occurs in `YieldToken`.

In the contract `OwnershipToken` the role `ADMIN_ROLE` has authority over updating the `MINTER_ROLE`. Any compromise to the `ADMIN_ROLE` account may allow the hacker to take advantage of this authority and change the `MINTER_ROLE` to an unintended address. The same `ADMIN_ROLE` vulnerability occurs in `YieldToken`.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

`[CertiK]`: The `Unreal Finance` team acknowledges the finding and is working toward adding multi-signature wallets to mitigate the risk in the short term.

`[Unreal Finance]`: "For now, we will be using multisig with a timelock contract from openzeppelin and transferring the ownership of Core to timelock. Hence only timelock can interact with core contract. While we are building our DAO for a long-term solution and provide more transparency."

Update 10/10/22

`[CertiK]`: The team has provided all necessary information to mitigate this finding. Please see below.

- **Multi-sign proxy contract address**: 0xfCac5736B08A6c3dA460ba21b4C91441707269c2
- **Internal multi-signature addresses**:
  - 0x83Fabaf7Dd2B44d27b4612B0aCdC09b3a7FE5D1a
  - 0xF5E1cA50Da44bF3CD71856Eb861Bda320AfFd396
  - 0xCB6d5BE2E778D575fD1734946679e8ed60bA4Ee6
- **Time lock contract address**: 0x4ECb095869aBb691aB817c35Fd50a378D27DFD06
- **Transaction proof for transferring ownership to multi-signature proxy**: 0x5367539a944cc6602362cf90f5ef2d6b8bb3657a7741f4c6273b18d626584486
- **Time lock owner transfer transaction hash**: 0x33c80847981aebb62272d646a91632ed49b99fc1098f8aebeea96ed70ed21249
- **Medium article**: https://unrealfinance.medium.com/unreal-finance-gnosis-safe-security-decentralization-339075ba5950

CON-02 | UNUSED RETURN VALUE

Category	Severity	Location	Status
Volatile Code	Minor	contracts/Treasury.sol: 66; contracts/futures/AFuture.sol: 42; contracts/futures/AaveV3Future.sol: 42; contracts/futures/CFuture.sol: 32, 41; contracts/futures/YFuture.sol: 29, 38	Resolved

Description

The return value of an external call is not stored in a local or state variable.

```
66          prevEpochInstance.expire();
```

```
42          ILendingPool(getProtocolFrontend()).withdraw(underlying, _amount,
treasury);
```

```
42          IPool(getProtocolFrontend()).withdraw(underlying, _amount, treasury);
```

```
32          cToken.mint(_amount);
```

```
41          cToken.redeem(_amountCToken);
```

```
29          yVault.deposit(_amount, address(this));
```

```
38          yVault.withdraw(_amount);
```

Recommendation

We recommend checking or using the return values of all external function calls.

Alleviation

`[CertiK]`: The `Unreal Finance` team made many of the changes described above in commit hash 565ed4d9fb5789f8325aaa4a3f0a5a3c699680dc.

See below for unresolved and newly arising issues.

- In `Treasury`, function `renew()` now uses the return value of `expire()` rather than `totalBalanceUnderlying()` in recording the value for `underlyingForOt[_streamKey][_prevEpoch]`. Note, however, that in the `YFuture` contract, these outputs differ through converting by the exchange rate. The same discrepancy arises for the `CFuture` contract. We encourage the team to review this discrepancy and make changes as needed.
- In `YFuture`, function `deposit()` has no check on the return value for the function call `yVault.deposit()`.

The remaining issues above were resolved in commits 263081922dc00ca811fd9da479267605e0051059 and 710c3c5d74bf866b9d1eccd297a3c1bf802a329a respectively.

CON-03 | CHECK EFFECT INTERACTION PATTERN VIOLATED

Category	Severity	Location	Status
Logical Issue	Minor	contracts/Core.sol: 202~203, 303~304, 343~344; contracts/Treasury.sol: 66~67; contracts/futures/FutureBase.sol: 129~130	Partially Resolved

Description

The order of external call/transfer and storage updates should follow the check-effect-interaction pattern.

Recommendation

We recommend rewriting so that storage updates are made before external calls and transfers.LINK

Alleviation

`[CertiK]`: The `Unreal Finance` team made most of the changes outlined above in commit hash 78b65ef9c717f1bec44f9c75405101dd1ac0a677.

The following issues remain.

- In `Core`, the function `createNewEpoch()` makes external calls to functions in `NewEpochAddr` and `_treasuryAddress` before making updates to state variables through the command

```
streams[streamKey].push(newEpochAddr);
```

- In `Treasury`, the function `renew()` makes an external call to function `expire()` in `prevEpochInstance` before making updates to state variable `underlyingForOt[_streamKey]`

```
underlyingForOt[_streamKey][_prevEpoch] = withdrawnAmount - yield;
```

The first remaining issue was resolved in commit 263081922dc00ca811fd9da479267605e0051059.

Please see the response below for the remaining issue.

`[Unreal Finance]`: "For the point in Treasury for function renew() we need the amountWithdrawn to calculate the final yield generated with respect to the initial capital underlying as there will be some slippage while withdrawing, that is why we have moved yield calculation after the external call."

CON-04 | MISSING INPUT VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	contracts/Core.sol: 151~152, 152~153, 303~304; contracts/Treasury.sol: 104~105, 139~140; contracts/futures/AFuture.sol: 26~27; contracts/futures/AaveV3Future.sol: 26~27; contracts/futures/CFuture.sol: 25~26; contracts/futures/FutureBase.sol: 46~47, 47~48, 48~49, 49~50, 83~84, 87~88, 93~94, 97~98; contracts/futures/YFuture.sol: 22~23	Resolved

Description

- Input `_underlying` is missing a check that it is a non-zero address
- Input `_durationSeconds` is missing a check that it is a non-zero value.
- Input `_amountUnderlying` is missing a check that it is a non-zero value.
- Input `_supply` is missing a check that it is a non-zero value.
- Input `_core` and `_treasuryAddr` are missing a check that they are non-zero addresses.
- State variable `yT` is missing a check that it is a non-zero address.
- State variable `oT` is missing a check that it is a non-zero address.
- Local variables `lendingProvider`, `compToken`, and `yearnVault` are missing a check that they are non-zero addresses.

Recommendation

We recommend adding in the checks described above to prevent unexpected errors.

Alleviation

`[CertiK]`: The `Unreal Finance` team resolved this finding by making the changes outlined above in commit hashes a5e00db4b941785b03eda0d83ef465a555b93463 and 4ee741208e110d45debb75df0a6c12119bf1073f.

COR-01 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization / Privilege	**Major**	**contracts/Core.sol: 15**	**Mitigated**

Description

`Core` is an upgradeable contract; the owner can upgrade the contract without the community's consent. If an attacker compromises the account, he or she can change the implementation of the contract and drain tokens from the contract.

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND

- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

`[CertiK]`: The `Unreal Finance` team acknowledges the finding and is working toward adding multi-signature wallets to mitigate the risk in the short term.

`[Unreal Finance]`: "For now, we will be using multisig with a timelock contract from openzeppelin and transferring the ownership of Core to timelock. Hence only timelock can interact with core contract. While we are building our DAO for a long-term solution and provide more transparency."

Update 10/13/22

`[CertiK]`: The team has shared all necessary information to mitigate this finding. Please see below.

- **Multi-sign proxy contract address**: 0xfCac5736B08A6c3dA460ba21b4C91441707269c2
- **Internal multi-signature addresses**:
  - 0x83Fabaf7Dd2B44d27b4612B0aCdC09b3a7FE5D1a
  - 0xF5E1cA50Da44bF3CD71856Eb861Bda320AfFd396
  - 0xCB6d5BE2E778D575fD1734946679e8ed60bA4Ee6
- **Time lock contract address**: 0x4ECb095869aBb691aB817c35Fd50a378D27DFD06
- **Transaction proof for transferring ownership to multi-signature proxy**: 0x5367539a944cc6602362cf90f5ef2d6b8bb3657a7741f4c6273b18d626584486
- **Time lock owner transfer transaction hash**: 0x33c80847981aebb62272d646a91632ed49b99fc1098f8aebeea96ed70ed21249
- **Medium article**: https://unrealfinance.medium.com/unreal-finance-gnosis-safe-security-decentralization-339075ba5950

COR-02 | UNPROTECTED UPGRADEABLE CONTRACT

Category	Severity	Location	Status
Language Specific	Major	contracts/Core.sol: 69	Resolved

Description

`Core` is an upgradeable contract that does not protect its `initialize()` function. Anyone can delete the contract with: `UUPSUpgradeable.upgradeTo(address)` or `UUPSUpgradeable.upgradeToAndCall(address, bytes)`.

```
    function initialize(address _treasuryAddress) public initializer{
```

```
    function upgradeTo(address newImplementation) external virtual onlyProxy {
```

```
    function upgradeToAndCall(address newImplementation, bytes memory data) external
payable virtual onlyProxy {
```

Recommendation

We recommend adding a constructor with the call `_disableInitializers()` from `Initializable` to ensure `initialize()` cannot be called on the logic contract.

Alleviation

`[CertiK]`: The `Unreal Finance` team heeded the recommendation and made the changes outlined above in commit hash a8bd92ed24b01b928fbfa9eb03c55f46bf23c135.

COR-03 | SHADOWING STATE VARIABLE

Category	Severity	Location	Status
Coding Style	Medium	contracts/Core.sol: 18	Resolved

Description

A state variable is shadowing another component defined in a parent contract.

Variable `_owner` in `Core` shadows the variable `_owner` in `OwnableUpgradeable` .

```
18      address private _owner;
```

```
22      address private _owner;
```

Recommendation

We recommend removing or renaming the state variable that shadows another definition.

Alleviation

`[CertiK]` : The `Unreal Finance` team heeded the recommendation and removed the shadowing state variable in commit hash fe7c3815104576a295ce1cbc23b42ba1a3a705d1.

COR-04 NO VALIDATION CHECK THAT `streamKey` IS NOT `bytes32(0)`

Category	Severity	Location	Status
Inconsistency	Minor	contracts/Core.sol: 130~131	Resolved

Description

In function `startEpoch()`, a check is made that the calculated `streamKey` corresponding to the input for `_protocol`, `_underlying`, and `_durationSeconds` is not `bytes32(0)`. No such check is made in `registerNewStream()`.

Recommendation

We recommend adding a check that the calculated `streamKey` is not `bytes32(0)` for function `registerNewStream()`.

Alleviation

`[CertiK]`: The `Unreal Finance` team heeded the recommendation and made the changes outlined above in commit hash 5690940c6fd320c5e99f291924ede3ba7483f522.

COR-05 | OWNER INPUTS `_bytecode` FOR `create2`

Category	Severity	Location	Status
Volatile Code	Minor	contracts/Core.sol: 220~221, 260~261	Resolved

Description

The construction of a futures contract depends on the correct input for `_bytecode` by the `_owner` of contract `Core`. However, the only check that the intended contract deployment was successful is that the `owner()` of the new futures contract is `address(this)`.

Recommendation

We recommend adding in more checks that inputs such as `_durationSeconds`, the underlying asset, the treasury address match the recorded values for that epoch.

Alleviation

`[CertiK]`: The `Unreal Finance` team added the following check at line 223 in commit 2616d54bab750971b2e43f55c34e16af12644a14

```
    if(IFuture(newEpochAddr).owner() != address(this) &&
          IFuture(newEpochAddr).expiry() != block.timestamp + _durationSeconds &&
          IFuture(newEpochAddr).underlying() != _underlying &&
          IFuture(newEpochAddr).treasury() != _treasuryAddress)
revert("ERR_INVALID_EPOCH");
```

Such a check will only revert if *all* parts of the check are unsatisfied. For instance, if the `owner()` is not `address(this)` but the remaining checks pass, then the transaction will not revert. We recommend that the team revisit this logic and make changes as needed. In this case, if the team wants the transaction to revert if any one of the checks fails, then they may consider using `||` (logical OR) between each check.

The issue described above was resolved in commit hash 679f69b70c5af55be96a49c1cb503fc035263d5e.

COR-06 NO CHECK `amountBurned` IS POSITIVE BEFORE `claimYield()` IS CALLED

Category	Severity	Location	Status
Inconsistency	Minor	contracts/Core.sol: 347~348, 350~351	Resolved

Description

Function `redeemYield()` can be called more than once, where all subsequent times will be for an `amountBurned` value of 0. Like function `redeemPrinciple()`, there should be a check that `amountBurned` is positive before proceeding with `claimYield()`.

Recommendation

We recommend adding a check that `amountBurned` is positive in order to call `claimYield()`.

Alleviation

`[CertiK]`: The `Unreal Finance` team heeded the recommendation and made the changes outlined above in commit hash 78b65ef9c717f1bec44f9c75405101dd1ac0a677.

FUT-01 | THIRD PARTY DEPENDENCY

Category	Severity	Location	Status
Volatile Code	Minor	contracts/futures/AFuture.sol: 16; contracts/futures/AaveV3Future.sol: 16; contracts/futures/CFuture.sol: 15; contracts/futures/FutureBase.sol : 20~21; contracts/futures/YFuture.sol: 12	Acknowledged

Description

The contract is serving as the underlying entity to interact with one or more third party protocols. The scope of the audit treats third party entities as black boxes and assume their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

```
20      address public immutable underlying;
```

- The contract `FutureBase` interacts with third party contract via `underlying`.

```
16      ILendingPoolAddressesProvider private provider;
```

- The contract `AFuture` interacts with third party contract with `ILendingPoolAddressesProvider` interface via `provider`.

```
16      IPoolAddressesProvider private provider;
```

- The contract `AaveV3Future` interacts with third party contract with `IPoolAddressesProvider` interface via `provider`.

```
15      CTokenInterface private cToken;
```

- The contract `CFuture` interacts with third party contract with `CTokenInterface` interface via `cToken`.

```
12      IVault private yVault;
```

- The contract `YFuture` interacts with third party contract with `IVault` interface via `yVault`.

Recommendation

We understand that the business logic requires interaction with the third parties. We encourage the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

Alleviation

`[Unreal Finance]` "We constantly monitor any changes occurring in the third-party protocols and with the adaption of ERC 4626 for yield-bearing tokens, we will be following the same standard for third-party protocols."

FUT-02 | UNCHECKED ERC-20 `transfer()` / `transferFrom()` CALL

Category	Severity	Location	Status
Volatile Code	Minor	contracts/futures/CFuture.sol: 43; contracts/futures/FutureBase.sol: 83~84, 93; contracts/futures/YFuture.sol: 40	Resolved

Description

The return value of the transfer()/transferFrom() call is not checked.

```
43          IERC20(underlying).transfer(treasury, underlyingbalance);
```

```
83          yT.transfer(_receiver, _amount);
```

```
 93           oT.transfer(_receiver, _amount);
```

```
40          IERC20(underlying).transfer(treasury, underlyingbalance);
```

Recommendation

Since some ERC-20 tokens return no values and others return a `bool` value, they should be handled with care. We recommend using the OpenZeppelin's `SafeERC20.sol` implementation to interact with the `transfer()` and `transferFrom()` functions of external ERC-20 tokens. The OpenZeppelin implementation checks for the existence of a return value and reverts if `false` is returned, making it compatible with all ERC-20 token implementations.

Alleviation

`[CertiK]`: The `Unreal Finance` team heeded the recommendation and made the changes outlined above in commit hash aef8b711325bf54427b204f155307f4905b64ee5.

CON-06 | UNLOCKED COMPILER VERSION

Category	Severity	Location	Status
Language Specific	Informational	contracts/Core.sol: 2~3; contracts/Treasury.sol: 2~3; contracts/futures/AFuture.sol: 2~3; contracts/futures/AaveV3Future.sol: 3; contracts/futures/CFuture.sol: 2~3; contracts/futures/FutureBase.sol: 2~3; contracts/futures/YFuture.sol: 2~3; contracts/libs/DateTime.sol: 2~3; contracts/libs/DetailedERC20.sol: 2~3; contracts/libs/MathLib.sol: 2~3; contracts/libs/Utils.sol: 2~3; contracts/tokens/OwnershipToken.sol: 2~3; contracts/tokens/YieldToken.sol: 2~3	Resolved

Description

The contracts cited have unlocked compiler versions. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We recommend that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation

`[CertiK]` : The `Unreal Finance` team heeded the recommendation above and made the changes outlined above in commit 900d63bb48bba428456bc4862fc39d746bc1bb88.

CON-07 | MISSING EMIT EVENTS

Category	Severity	Location	Status
Coding Style	Informational	contracts/Core.sol: 89~90; contracts/futures/FutureBase.sol: 77~78	Resolved

Description

Functions that affect the status of sensitive variables should emit events as notifications.

Recommendation

We recommend adding events for state changes or sensitive actions, and emitting them in corresponding functions

Alleviation

`[CertiK]`: The `Unreal Finance` team heeded the recommendation above and made the changes outlined above in commit 423797f43c08c9758e7ac7145876b3b36dbc535c.

COR-08 `_protocol` MAY BE DIFFERENT FROM WHAT `_bytecode` DESCRIBES

Category	Severity	Location	Status
Coding Style	Informational	contracts/Core.sol: 121~122, 149~150	Resolved

Description

The input `_protocol` used to check whether the given protocol is supported does not necessarily have to match the protocol that corresponds to the futures contract implemented through the input `_bytecode`. It is possible bytecode for a futures contract corresponding to one protocol is stored under the `streamKey` of a different protocol.

Recommendation

We recommend clarifying the intent of the design described above.

Alleviation

`[Unreal Finance]`: "The idea is that we can have different future bytecode under the same stream key because we are using futures to interact with third-party protocols and if any changes occur to the third-party protocols we can make the changes accordingly to the future contract under the same protocol/stream key."

COR-09 | `amountUnderlying` MAY BE LARGER THAN `totalSupply`

Category	Severity	Location	Status
Mathematical Operations, Logical Issue	Informational	contracts/Core.sol: 319~320	Resolved

Description

`_amountUnderlying` is the amount of underlying asset the `msg.sender` wishes to deposit, while `totalSupply` refers to the total supply of associated yield tokens for that underlying asset. The yield tokens are in one-to-one ratio with the asset tokens that are deposited, and they have the same decimals as the underlying asset. It is possible for the entire product `yield * _amountUnderlying / totalSupply` to exceed the value of `_amountUnderlying`, causing a revert due to underflow in `amountOT`. This may keep a user from depositing the underlying asset.

Recommendation

We recommend revisiting the formula for `amountOT` and deciding if it needs to be reworked to accommodate the possibilities outlined above.

Alleviation

`[Unreal Finance]`: "The only possible case for yield * _amountunderlying / totalsupply to be greater than _amountunderlying would be when yield becomes greater than or equal to 100%. Currently, we are just targeting stable coins and ETH and this won't be possible with them. Maybe in the later versions, we'll add support for coins like OHM but that's not in the plan as of now."

DER-01 | `_decimals` CAN BE MADE PRIVATE

Category	Severity	Location	Status
Language Specific	Informational	contracts/libs/DetailedERC20.sol: 9~10	Resolved

Description

Variable `_decimals` can be made private to avoid two getter functions `_decimals()` and `decimals()` that return the same value.

Recommendation

We recommend making the above updates so that there is only one getter function for `_decimals` .

Alleviation

`[CertiK]` : The `Unreal Finance` team heeded the recommendation above and made the changes outlined above in commit 6850c6fc66c66b74ca9253fb95b20642fb0d1062.

FUT-03 | INCOMPATIBILITY WITH DEFLATIONARY TOKENS

Category	Severity	Location	Status
Logical Issue	Informational	contracts/futures/AFuture.sol: 31, 33; contracts/futures/AaveV3Future.sol: 31, 33; contracts/futures/CFuture.sol: 30, 32; contracts/futures/FutureBase.sol: 120, 126; contracts/futures/YFuture.sol: 27, 29	Resolved

Description

When transferring deflationary ERC20 tokens, the input amount may not be equal to the received amount due to the charged transaction fee. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee), only 90 tokens actually arrived to the contract. However, a failure to discount such fees may allow the same user to withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

Reference: https://thoreum-finance.medium.com/what-exploit-happened-today-for-gocerberus-and-garuda-also-for-lokum-ybear-piggy-caramelswap-3943ee23a39f

```
31          IERC20(underlying).safeTransferFrom(msg.sender, address(this), _amount);
```

- Transferring tokens by `_amount`.

```
33          ILendingPool(getProtocolFrontend()).deposit(underlying, _amount,
address(this), 0);
```

- The `_amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.
- Note: `deposit` is an external function and its behavior wasn't evaluated.

```
31          IERC20(underlying).safeTransferFrom(msg.sender, address(this), _amount);
```

- Transferring tokens by `_amount`.

```
33          IPool(getProtocolFrontend()).supply(underlying, _amount, address(this),
0);
```

- The `_amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.
- Note: `supply` is an external function and its behavior wasn't evaluated.

```
30          IERC20(underlying).safeTransferFrom(msg.sender, address(this), _amount);
```

- Transferring tokens by `_amount`.

```
32          cToken.mint(_amount);
```

- The `_amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.
- Note: `mint` is an external function and its behavior wasn't evaluated.

```
120             deposit(_amountInUnderlying);
```

- Transferring tokens by `_amountInUnderlying`.
- This function call executes the following operation.
- In `AFuture.deposit`,
    - `IERC20(underlying).safeTransferFrom(msg.sender, address(this), _amount);`

```
120             deposit(_amountInUnderlying);
```

- This function call executes the following operation.
- In `AFuture.deposit`,
    - `ILendingPool(getProtocolFrontend()).deposit(underlying, _amount, address(this), 0);`
    - Note: `deposit` is an external function and its behavior wasn't evaluated.
- The `_amountInUnderlying` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

```
120             deposit(_amountInUnderlying);
```

- Transferring tokens by `_amountInUnderlying`.
- This function call executes the following operation.
- In `YFuture.deposit`,
    - `IERC20(underlying).safeTransferFrom(msg.sender, address(this), _amount);`

```
120             deposit(_amountInUnderlying);
```

- This function call executes the following operation.
- In `YFuture.deposit`,
    - `yVault.deposit(_amount, address(this));`
    - Note: `deposit` is an external function and its behavior wasn't evaluated.
- The `_amountInUnderlying` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

```
126         deposit(_amount);
```

- Transferring tokens by `_amount`.
- This function call executes the following operation.
- In `AFuture.deposit`,
    - `IERC20(underlying).safeTransferFrom(msg.sender, address(this), _amount);`

```
126         deposit(_amount);
```

- This function call executes the following operation.
- In `AFuture.deposit`,
    - `ILendingPool(getProtocolFrontend()).deposit(underlying, _amount, address(this), 0);`
    - Note: `deposit` is an external function and its behavior wasn't evaluated.
- The `_amount` appears to be used for bookkeeping purposes without compensating the potential transfer fees.

Recommendation

We recommend the client regulate the set of tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

`[Unreal Finance]`: "We are not supporting deflationary tokens as of now. The supported tokens will include Stable Coins and ETH. In the future with DAO in place, we will be providing support for Yearn Curve Vaults accordingly."

OPTIMIZATIONS | UNREAL FINANCE

ID	Title	Category	Severity	Status
CON-05	Variables Could Be Declared As `immutable`	Gas Optimization	Optimization	Resolved
COR-07	Unused State Variable	Gas Optimization	Optimization	Resolved

CON-05 | VARIABLES COULD BE DECLARED AS `immutable`

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/futures/AFuture.sol: 16~17; contracts/futures/AaveV3Future.sol: 16~17; contracts/futures/CFuture.sol: 15~16; contracts/futures/YFuture.sol: 12~13; contracts/libs/DetailedERC20.sol: 9	Resolved

Description

The variables `provider`, `yVault`, `cToken`, and `_decimals` assigned in the constructor can declared with `Immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. An advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since will not be stored in storage.

Recommendation

We recommend declaring the cited variables as `immutable`

Alleviation

`[CertiK]`: The `Unreal Finance` team heeded the recommendation above and made the changes outlined above in commit 36c4509b223fed0d9a5c946f7d7e9c2c5e13b222.

COR-07 | UNUSED STATE VARIABLE

Category	Severity	Location	Status
Gas Optimization	Optimization	contracts/Core.sol: 18, 20	Resolved

Description

One or more state variables are never used in the codebase.

Variable `_owner` in `Core` is never used in `Core`.

```
18      address private _owner;
```

```
15  contract Core is Initializable, UUPSUpgradeable, OwnableUpgradeable {
```

Variable `initialized` in `Core` is never used in `Core`.

```
20      bool initialized;
```

```
15  contract Core is Initializable, UUPSUpgradeable, OwnableUpgradeable {
```

Recommendation

We recommend removing unused variables.

Alleviation

`[CertiK]`: The `Unreal Finance` team heeded the recommendation and made the changes outlined above in commit 3a74b8a2cd0bbab2bec2710dcd0352ad9520c330.

FORMAL VERIFICATION | UNREAL FINANCE

Formal guarantees about the behavior of smart contracts can be obtained by reasoning about properties relating to the entire contract (e.g. contract invariants) or to specific functions of the contract. Once such properties are proven to be valid, they guarantee that the contract behaves as specified by the property. As part of this audit, we applied automated formal verification (symbolic model checking) to prove that well-known functions in the smart contracts adhere to their expected behavior.

Considered Functions And Scope

Verification of ERC-20 compliance

We verified properties of the public interface of those token contracts that implement the ERC-20 interface. This covers

- Functions `transfer` and `transferFrom` that are widely used for token transfers,
- functions `approve` and `allowance` that enable the owner of an account to delegate a certain subset of her tokens to another account (i.e. to grant an allowance), and
- the functions `balanceOf` and `totalSupply`, which are verified to correctly reflect the internal state of the contract.

The properties that were considered within the scope of this audit are as follows:

Property Name	Title
erc20-transfer-revert-zero	Function `transfer` Prevents Transfers to the Zero Address
erc20-transfer-correct-amount	Function `transfer` Transfers the Correct Amount in Non-self Transfers
erc20-transfer-succeed-self	Function `transfer` Succeeds on Admissible Self Transfers
erc20-transfer-succeed-normal	Function `transfer` Succeeds on Admissible Non-self Transfers
erc20-transfer-correct-amount-self	Function `transfer` Transfers the Correct Amount in Self Transfers
erc20-transfer-change-state	Function `transfer` Has No Unexpected State Changes
erc20-transfer-exceed-balance	Function `transfer` Fails if Requested Amount Exceeds Available Balance
erc20-transfer-recipient-overflow	Function `transfer` Prevents Overflows in the Recipient's Balance
erc20-transfer-false	If Function `transfer` Returns `false`, the Contract State Has Not Been Changed
erc20-transfer-never-return-false	Function `transfer` Never Returns `false`
erc20-transferfrom-revert-from-zero	Function `transferFrom` Fails for Transfers From the Zero Address

Property Name	Title

erc20-transferfrom-correct-amount	Function `transferFrom` Transfers the Correct Amount in Non-self Transfers
erc20-transferfrom-correct-amount-self	Function `transferFrom` Performs Self Transfers Correctly
erc20-transferfrom-succeed-normal	Function `transferFrom` Succeeds on Admissible Non-self Transfers
erc20-transferfrom-succeed-self	Function `transferFrom` Succeeds on Admissible Self Transfers
erc20-transferfrom-fail-exceed-balance	Function `transferFrom` Fails if the Requested Amount Exceeds the Available Balance
erc20-transferfrom-correct-allowance	Function `transferFrom` Updated the Allowance Correctly
erc20-transferfrom-fail-exceed-allowance	Function `transferFrom` Fails if the Requested Amount Exceeds the Available Allowance
erc20-transferfrom-false	If Function `transferFrom` Returns `false`, the Contract's State Has Not Been Changed
erc20-transferfrom-fail-recipient-overflow	Function `transferFrom` Prevents Overflows in the Recipient's Balance
erc20-totalsupply-succeed-always	Function `totalSupply` Always Succeeds
erc20-totalsupply-correct-value	Function `totalSupply` Returns the Value of the Corresponding State Variable
erc20-transferfrom-never-return-false	Function `transferFrom` Never Returns `false`
erc20-totalsupply-change-state	Function `totalSupply` Does Not Change the Contract's State
erc20-balanceof-succeed-always	Function `balanceOf` Always Succeeds
erc20-balanceof-correct-value	Function `balanceOf` Returns the Correct Value
erc20-balanceof-change-state	Function `balanceOf` Does Not Change the Contract's State
erc20-allowance-succeed-always	Function `allowance` Always Succeeds
erc20-allowance-correct-value	Function `allowance` Returns Correct Value
erc20-allowance-change-state	Function `allowance` Does Not Change the Contract's State
erc20-approve-revert-zero	Function `approve` Prevents Giving Approvals For the Zero Address
erc20-approve-succeed-normal	Function `approve` Succeeds for Admissible Inputs
erc20-approve-correct-amount	Function `approve` Updates the Approval Mapping Correctly
erc20-transferfrom-change-state	Function `transferFrom` Has No Unexpected State Changes
erc20-approve-change-state	Function `approve` Has No Unexpected State Changes
erc20-approve-false	If Function `approve` Returns `false`, the Contract's State Has Not Been Changed
erc20-approve-never-return-false	Function `approve` Never Returns `false`

Verification Results

For the following contracts, model checking established that each of the 38 properties that were in scope of this audit (see scope) are valid:

Contract ERC20 (Source File contracts/Treasury.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	True	
erc20-transfer-correct-amount	True	
erc20-transfer-succeed-self	True	
erc20-transfer-succeed-normal	True	
erc20-transfer-correct-amount-self	True	
erc20-transfer-change-state	True	
erc20-transfer-exceed-balance	True	
erc20-transfer-recipient-overflow	True	
erc20-transfer-false	True	
erc20-transfer-never-return-false	True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	True	
erc20-transferfrom-revert-to-zero	True	
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-correct-amount-self	True	
erc20-transferfrom-succeed-normal	True	
erc20-transferfrom-succeed-self	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-false	True	
erc20-transferfrom-fail-recipient-overflow	True	
erc20-transferfrom-never-return-false	True	
erc20-transferfrom-change-state	True	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-change-state	True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	
erc20-allowance-change-state	True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	True	
erc20-approve-succeed-normal	True	
erc20-approve-correct-amount	True	
erc20-approve-change-state	True	
erc20-approve-false	True	
erc20-approve-never-return-false	True	

Contract OwnershipToken (Source File contracts/Treasury.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-correct-amount	True	
erc20-transfer-revert-zero	True	
erc20-transfer-succeed-normal	True	
erc20-transfer-succeed-self	True	
erc20-transfer-correct-amount-self	True	
erc20-transfer-change-state	True	
erc20-transfer-exceed-balance	True	
erc20-transfer-recipient-overflow	True	
erc20-transfer-false	True	
erc20-transfer-never-return-false	True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	True	
erc20-transferfrom-revert-to-zero	True	
erc20-transferfrom-succeed-normal	True	
erc20-transferfrom-succeed-self	True	
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-correct-amount-self	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-change-state	True	
erc20-transferfrom-false	True	
erc20-transferfrom-fail-recipient-overflow	True	
erc20-transferfrom-never-return-false	True	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-change-state	True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	
erc20-allowance-change-state	True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	True	
erc20-approve-correct-amount	True	
erc20-approve-succeed-normal	True	
erc20-approve-change-state	True	
erc20-approve-false	True	
erc20-approve-never-return-false	True	

Contract YieldToken (Source File contracts/futures/FutureBase.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	True	
erc20-transfer-correct-amount	True	
erc20-transfer-succeed-normal	True	
erc20-transfer-succeed-self	True	
erc20-transfer-correct-amount-self	True	
erc20-transfer-change-state	True	
erc20-transfer-exceed-balance	True	
erc20-transfer-recipient-overflow	True	
erc20-transfer-false	True	
erc20-transfer-never-return-false	True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	True	
erc20-transferfrom-revert-to-zero	True	
erc20-transferfrom-succeed-self	True	
erc20-transferfrom-succeed-normal	True	
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-correct-amount-self	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-change-state	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-false	True	
erc20-transferfrom-never-return-false	True	
erc20-transferfrom-fail-recipient-overflow	True	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-change-state	True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-correct-value	True	
erc20-allowance-succeed-always	True	
erc20-allowance-change-state	True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	True	
erc20-approve-correct-amount	True	
erc20-approve-succeed-normal	True	
erc20-approve-change-state	True	
erc20-approve-false	True	
erc20-approve-never-return-false	True	

Contract DetailedERC20 (Source File contracts/tokens/OwnershipToken.sol)

Detailed results for function `transfer`

Property Name	Final Result	Remarks
erc20-transfer-revert-zero	True	
erc20-transfer-correct-amount	True	
erc20-transfer-succeed-normal	True	
erc20-transfer-succeed-self	True	
erc20-transfer-correct-amount-self	True	
erc20-transfer-change-state	True	
erc20-transfer-exceed-balance	True	
erc20-transfer-recipient-overflow	True	
erc20-transfer-false	True	
erc20-transfer-never-return-false	True	

Detailed results for function `transferFrom`

Property Name	Final Result	Remarks
erc20-transferfrom-revert-from-zero	True	
erc20-transferfrom-revert-to-zero	True	
erc20-transferfrom-correct-amount	True	
erc20-transferfrom-correct-amount-self	True	
erc20-transferfrom-succeed-normal	True	
erc20-transferfrom-succeed-self	True	
erc20-transferfrom-fail-exceed-balance	True	
erc20-transferfrom-correct-allowance	True	
erc20-transferfrom-change-state	True	
erc20-transferfrom-fail-exceed-allowance	True	
erc20-transferfrom-false	True	
erc20-transferfrom-fail-recipient-overflow	True	
erc20-transferfrom-never-return-false	True	

Detailed results for function `totalSupply`

Property Name	Final Result	Remarks
erc20-totalsupply-succeed-always	True	
erc20-totalsupply-correct-value	True	
erc20-totalsupply-change-state	True	

Detailed results for function `balanceOf`

Property Name	Final Result	Remarks
erc20-balanceof-succeed-always	True	
erc20-balanceof-correct-value	True	
erc20-balanceof-change-state	True	

Detailed results for function `allowance`

Property Name	Final Result	Remarks
erc20-allowance-succeed-always	True	
erc20-allowance-correct-value	True	
erc20-allowance-change-state	True	

Detailed results for function `approve`

Property Name	Final Result	Remarks
erc20-approve-revert-zero	True	
erc20-approve-succeed-normal	True	
erc20-approve-correct-amount	True	
erc20-approve-change-state	True	
erc20-approve-false	True	
erc20-approve-never-return-false	True	

APPENDIX | UNREAL FINANCE

Finding Categories

Categories	Description
Centralization / Privilege	Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.
Gas Optimization	Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.
Mathematical Operations	Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.
Logical Issue	Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.
Language Specific	Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.
Coding Style	Coding Style findings usually do not affect the generated byte-code but rather comment on how to make the codebase more legible and, as a result, easily maintainable.
Inconsistency	Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a constructor assignment imposing different require statements on the input variables than a setter function.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE

FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

CertiK | Securing the Web3 World

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.