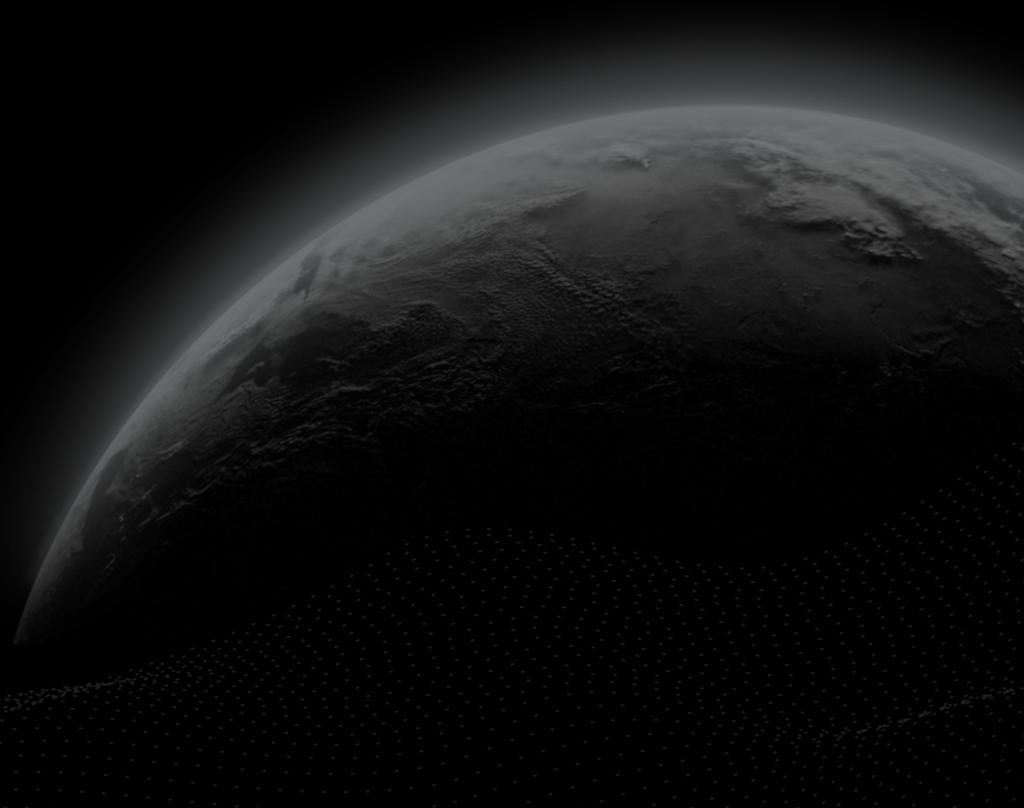
Security Assessment

Tokenwolf - Audit

CertiK Verified on Oct 20th, 2022



CertiK Verified on Oct 20th, 2022

Tokenwolf - Audit

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

Executive Summary

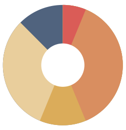
TYPES	ECOSYSTEM	METHODS
Exchange	EVM Compatible	Manual Review, Static Analysis
LANGUAGE	**TIMELINE**	**KEY COMPONENTS**
Solidity	Delivered on 10/20/2022	N/A

CODEBASE

File provided by the client

...View All

Vulnerability Summary



16	14	2	0	0	0	0
Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined	Unresolved

1	Critical	1 Resolved	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.
6	Major	4 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.
2	Medium	2 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
5	Minor	5 Resolved	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.
2	Informational	2 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 | TOKENWOLF - AUDIT

CODEBASE | TOKENWOLF - AUDIT

Repository

File provided by the client

AUDIT SCOPE | TOKENWOLF - AUDIT

4 files audited • 3 files with Mitigated findings • 1 file without findings

ID	File	SHA256 Checksum
BTC	projects/Tokenwolf/contracts/Ballot.sol	4c07ce4cdac6d0bcb53e32f9851bec1e35fdd05cbfa52445cd2f38d0662c6e27
STC	projects/Tokenwolf/contracts/Swap2.sol	66a7e64200600251cae1955639e05f9d46f0afe96056baa6584e1d43d1f6e70c
TTC	projects/Tokenwolf/contracts/Token3.sol	503ad6ed0391ed7df95742628a0b3b152bb3ac6fa8fb9b269eb7f601182f416a
OTC	projects/Tokenwolf/contracts/Owner.sol	83a59a5493d00c3f6f322646e9e6d4f86af71d8e69974e383f0101a32dcbdeb3

APPROACH & METHODS | TOKENWOLF - AUDIT

This report has been prepared for Tokenwolf to discover issues and vulnerabilities in the source code of the Tokenwolf - Audit 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.

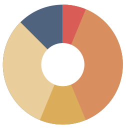
The auditing process pays special attention to the following considerations:

- 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 | TOKENWOLF - AUDIT



16	1	6	2	5	2
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Tokenwolf - Audit. Through this audit, we have uncovered 16 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
BTC-01	Potential Double Voting	Logical Issue	Major	Resolved
BTC-02	Unused Field `deadlineTimestamp`	Control Flow	Medium	Resolved
BTC-03	Missing Input Validation On `voteType`	Control Flow	Minor	Resolved
STC-01	No Upper Limit For Fee Rates	Control Flow	Critical	Resolved
STC-02	Check Effect Interaction Pattern Violated	Volatile Code	Major	Resolved
STC-03	Possible DOS Attack	Volatile Code	Major	Resolved
STC-04	No Access Control On `cancelOffer()`	Control Flow	Major	Resolved
STC-05	Function `showOffers()` May Not Work As Intended	Logical Issue	Medium	Resolved
STC-06	Divide Before Multiply	Mathematical Operations	Minor	Resolved
STC-07	Missing Check `quantity` <= `offer.quantity`	Logical Issue	Minor	Resolved

ID	Title	Category	Severity	Status
STC-08	Missing Check In `cancelOffer()`	Logical Issue	Minor	Resolved
TCK-01	**Centralization Risks**	**Centralization / Privilege**	**Major**	**Mitigated**
TCK-02	Missing Zero Address Validation	Volatile Code	Minor	Resolved
TTC-01	**Initial Token Distribution**	**Centralization / Privilege**	**Major**	**Mitigated**
STC-10	Restore Deleted Offer	Logical Issue	Informational	Resolved
STC-11	Invalid Checks On `amount`	Logical Issue	Informational	Resolved

BTC-01 | POTENTIAL DOUBLE VOTING

Category	Severity	Location	Status
Logical Issue	● Major	projects/Tokenwolf/contracts/Ballot.sol: 68	● Resolved

Description

```
61      function vote(uint256 proposalId, uint8 voteType) public {
62          SecurityToken token = SecurityToken(_tokenAddress);
63          uint voteCount = token.getPastVotes(msg.sender, _proposals[proposalId].blockNumber);
64          require(voteCount > 0, "User has no voting rights");
65          if (voteType == 0) _proposals[proposalId].voteCountContra += voteCount;
66          if (voteType == 1) _proposals[proposalId].voteCountPro += voteCount;
67          if (voteType == 2) _proposals[proposalId].voteCountAbstain += voteCount;
68          _votedOnProposalId[msg.sender][proposalId] = true;
69      }
```

The function `vote()` sets `_votedOnProposalId[msg.sender][proposalId]` to be true at the end, which means `msg.sender` voted for the proposal with `proposalId`. However, because the function does not require `_votedOnProposalId[msg.sender][proposalId]` to be false, an account can vote twice or even unlimited times.

Recommendation

We recommend adding additional logic built into the contract to prevent double voting.

Alleviation

This issue was resolved in the updated code.

BTC-02 | UNUSED FIELD `deadlineTimestamp`

Category	Severity	Location	Status
Control Flow	Medium	projects/Tokenwolf/contracts/Ballot.sol: 13, 61	Resolved

Description

```
    struct Proposal {
        string name;
        uint proposalTimestamp;
        uint deadlineTimestamp;
        uint blockNumber;
        uint voteCountPro;
        uint voteCountContra;
        uint voteCountAbstain;
    }
```

The field `deadlineTimestamp` stands for the deadline for voting. Because the function `vote()` does not check a proposal's `deadlineTimestamp`, a user can vote for an expired proposal.

Recommendation

We recommend adding sanity checks to ensure a user cannot vote for an expired proposal.

Alleviation

This issue was resolved in the updated code.

BTC-03 | MISSING INPUT VALIDATION ON `voteType`

Category	Severity	Location	Status
Control Flow	Minor	projects/Tokenwolf/contracts/Ballot.sol: 61	Resolved

Description

```
    function vote(uint256 proposalId, uint8 voteType) public {
        SecurityToken token = SecurityToken(_tokenAddress);
        uint voteCount = token.getPastVotes(msg.sender,
_proposals[proposalId].blockNumber);
        require(voteCount > 0, "User has no voting rights");
        if (voteType == 0) _proposals[proposalId].voteCountContra += voteCount;
        if (voteType == 1) _proposals[proposalId].voteCountPro += voteCount;
        if (voteType == 2) _proposals[proposalId].voteCountAbstain +=
voteCount;
        _votedOnProposalId[msg.sender][proposalId] = true;
    }
```

The function `vote()` does not require the parameter `voteType` to be only 0, 1, or 2. A vote is invalid if the value of `voteType` is larger than 2.

Recommendation

We advise adding the check for the passed-in value to prevent invalid votes.

Alleviation

This issue was resolved in the updated code.

STC-01 | NO UPPER LIMIT FOR FEE RATES

Category	Severity	Location	Status
Control Flow	Critical	projects/Tokenwolf/contracts/Swap2.sol: 111~112	Resolved

Description

```
function setFees(uint256 makerFee, uint256 takerFee) public onlyOwner {
    _makerFee = makerFee;
    _takerFee = takerFee;
}
```

The `_owner` can set `_makerFee` and `_takerFee` in the contract and there is no upper limit on what the fee rate can be. In the extreme case, the fee can be higher than 100%, implying that users cannot buy or sell the token.

Moreover, for a "buy offer", the "maker fee" is not deducted from the swapping tokens but is transferred to this contract separately. If the `_owner` is compromised, an attacker can steal all the buyer's balance of `_otherToken`. Here is the attack scenario:

- an account approves all its balance of `_otherToken` and creates a "buy offer" via the function `addOffer()`
- the `_owner` updates `_makerFee` to a high value via the function `setFees()`
- the `_owner` or any account accepts the "buy offer" via `acceptOffer()`
- all or a lot of the offer creator's `_otherToken` would be transferred to this contract, and those tokens can be withdrawn by the `_owner`

Recommendation

We recommend setting reasonable upper limits for `_makerFee` and `_takerFee`, such as 10%.

Alleviation

Resolved by setting the upper limits to 10% for `_makerFee` and `_takerFee` in the updated code.

STC-02 | CHECK EFFECT INTERACTION PATTERN VIOLATED

Category	Severity	Location	Status
Volatile Code	Major	projects/Tokenwolf/contracts/Swap2.sol: 320, 322, 324, 330, 332, 334, 338, 339, 341, 361	Resolved

Description

In the function `acceptOffer()` , `offer.quantity` is decreased after transferring the tokens, which violates the check-effect-interaction pattern. The function `acceptOffer()` triggers the following external function calls. If the `token1` , `token2` , or both have a hook, they can reenter the same function and drain token balance from `msg.sender` or `counterpart` .

External call(s)

```
320                 require(token2.transferFrom(msg.sender, counterpart,
amountOtherToken-makerFee), "Transfer of token2 failed");
```

```
322                 require(token1.transferFrom(counterpart, msg.sender, amountToken),
"Transfer of token1 failed");
```

```
324                 require(token2.transferFrom(msg.sender, address(this),
takerFee+makerFee), "Transfer of fees failed");
```

```
330                 require(token1.transferFrom(msg.sender, counterpart, amountToken),
"Transfer of token1 failed");
```

```
332                 require(token2.transferFrom(counterpart, msg.sender,
amountOtherToken-takerFee), "Transfer of token2 failed");
```

```
334                 require(token2.transferFrom(counterpart, address(this),
makerFee+takerFee), "Transfer of fees failed");
```

State variables written after the call(s)

```
338             offer.quantity -= quantity;
339             if (offer.quantity == 0) cancelOffer(id);
340             // save modified offer to storage
341             offers[id] = offer;
```

Events emitted after the call(s)

```
351            emit NewTrade(block.timestamp, amountToken, amountOtherToken);
```

```
350            emit OffersChanged();
```

```
365            emit OffersChanged();
```

Recommendation

We recommend using the Checks-Effects-Interactions Pattern to avoid the risk of calling unknown contracts or applying OpenZeppelin ReentrancyGuard library - `nonReentrant` modifier for the aforementioned functions to prevent reentrancy attack.

Alleviation

This issue was resolved in the updated code.

STC-03 | POSSIBLE DOS ATTACK

Category	Severity	Location	Status
Volatile Code	Major	projects/Tokenwolf/contracts/Swap2.sol: 277, 382~386	Resolved

Description

```
382     function firstFreeSlot() public view returns (uint8 index) {
383         for (uint8 t = 0; t < 255; t++) {
384             if (usedSlots[t] == false) return t;
385         }
386     }
```

The function `firstFreeSlot()` returns the smallest value between 0 to 254 that is not currently used as an offer's ID, or returns 0 if all numbers are in use. When an offer is created via the function `addOffer()` , it would be assigned the return value of `firstFreeSlot()` as its ID. And a valid offer must have an ID.

However, an attacker can call the function `addProposal()` multiple times to fill up all IDs between 0 to 254. Then the following creators can only constantly overwrite the offer with ID 0.

Recommendation

We recommend reconsidering the logic to prevent the potential DOS attack.

Alleviation

Resolved the issue by adding access controls and zero checks for the inputted values.

STC-04 | NO ACCESS CONTROL ON `cancelOffer()`

Category	Severity	Location	Status
Control Flow	Major	projects/Tokenwolf/contracts/Swap2.sol: 359	Resolved

Description

```
function cancelOffer(uint8 id) public {
    // delete offer from storage and mark the slot as free
    delete(offers[id]);
    usedSlots[id] = false;
    _offerCount--;
    // fire event
    emit OffersChanged();
}
```

The function `cancelOffer()` does not have any access control, any account can call this function to cancel any offer.

Recommendation

We recommend adding appropriate access control.

Alleviation

This issue was resolved in the updated code.

STC-05 | FUNCTION `showOffers()` MAY NOT WORK AS INTENDED

Category	Severity	Location	Status
Logical Issue	Medium	projects/Tokenwolf/contracts/Swap2.sol: 422~423	Resolved

Description

```
398     function showOffers() public view returns (OfferDetails[] memory) {
399         // counter for result array
400         uint8 cnt = 0;
401         // counter for all offers
402         uint8 cnt2 = 0;
403         // create result array
404         OfferDetails[] memory result = new OfferDetails[](_offerCount);
405         // if there are no offers => return an empty array
406         if (_offerCount == 0) return result;
407         // iterate all slots and check for valid offers
408         for (uint8 t = 0; t < 255; t++) {
409             // check validity of offer (counterpart funds and allowance)
410             (bool check,) = checkOfferCounterpart(t,0);
411             if (usedSlots[t] == true) {
412                 OfferDetails memory details;
413                 details.quantity = offers[t].quantity;
414                 details.price = offers[t].price;
415                 details.bid = offers[t].bid;
416                 details.offererAddress = offers[t].offererAddress;
417                 details.id = t;
418                 details.valid = check;
419                 result[cnt] = details;
420                 cnt++;
421             }
422             cnt2++;
423             if (cnt2 == _offerCount) return result;
424         }
425         return result;
426     }
```

The function `showOffers()` uses `cnt2`, which represents how many times the loop is executed rather than how many existing offers are found. When `cnt2` equals `_offerCount`, the number of existing offers, the function returns the result. So the function may not return all the valid offers as intended.

Recommendation

We recommend reconsidering the logic to implement the intended design.

Alleviation

This issue was resolved in the updated code.

STC-06 | DIVIDE BEFORE MULTIPLY

Category	Severity	Location	Status
Mathematical Operations	Minor	projects/Tokenwolf/contracts/Swap2.sol: 129, 131	Resolved

Description

```
127     function calculateFees(uint256 amount, bool maker) public view returns
(uint256) {
128         if (maker == true) {
129             return(_makerFee * (amount/10**decimals2) );
130         } else {
131             return(_takerFee * (amount/10**decimals2) );
132         }
133     }
```

Performing integer division before multiplication truncates the low bits, losing the precision of calculation.

The function `calculateFees()` is used in the function `acceptOffer()` to calculate the fees when trading an offer. The parameter `amount` stands for the amount of `_otherToken` for swapping, the variable `decimals2` is the `_otherToken` 's decimals.

If `amount` < `10**decimals2` , the return value of `calculateFees()` would be 0, which means no fee would be collected in the function call of `acceptOffer()` . Even though `amount` > `10**decimals2` , fees would be less than expected due to the loss of precision.

Recommendation

We recommend applying multiplication before division to avoid loss of precision.

Alleviation

This issue was resolved in the updated code.

STC-07 | MISSING CHECK `quantity` <= `offer.quantity`

Category	Severity	Location	Status
Logical Issue	Minor	projects/Tokenwolf/contracts/Swap2.sol: 210	Resolved

Description

The function `checkOfferCounterpart()` does not require the inputted value of `quantity` <= `offer.quantity` and could return an incorrect value.

Recommendation

We recommend adding sanity checks on the inputted parameter.

Alleviation

This issue was resolved in the updated code.

STC-08 | MISSING CHECK IN `cancelOffer()`

Category	Severity	Location	Status
Logical Issue	Minor	projects/Tokenwolf/contracts/Swap2.sol: 363, 398	Resolved

Description

```
363     function cancelOffer(uint8 id) public {
364         // delete offer from storage and mark the slot as free
365         delete(offers[id]);
366         usedSlots[id] = false;
367         _offerCount--;
368         // fire event
369         emit OffersChanged();
370     }
```

The function does not check if an offer exists and decreases the value of `_offerCount`, which stands for the number of existing offers. This would make the function `showOffers()` that returns all existing offers invalid.

Recommendation

We recommend adding sanity checks to ensure the function `cancelOffer()` only cancels existing offers.

Alleviation

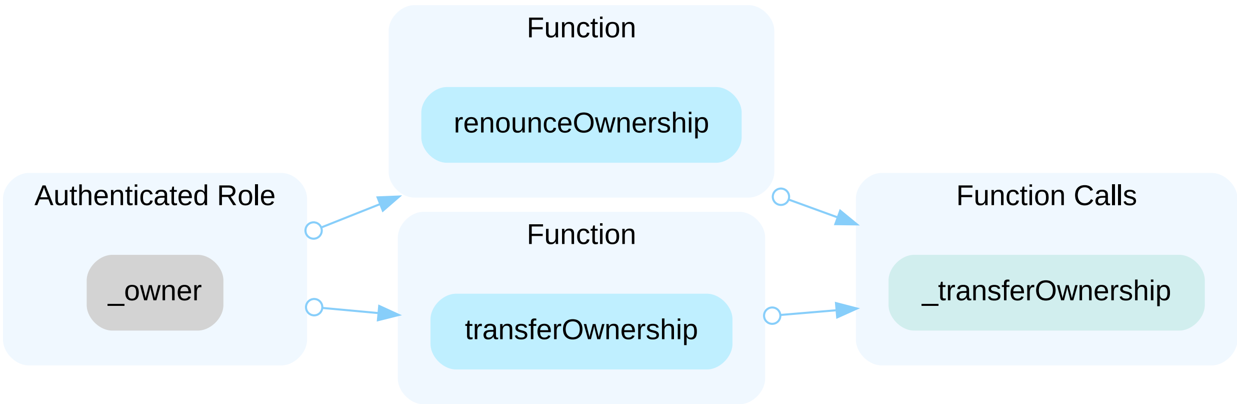
This issue was resolved in the updated code.

TCK-01 | CENTRALIZATION RISKS

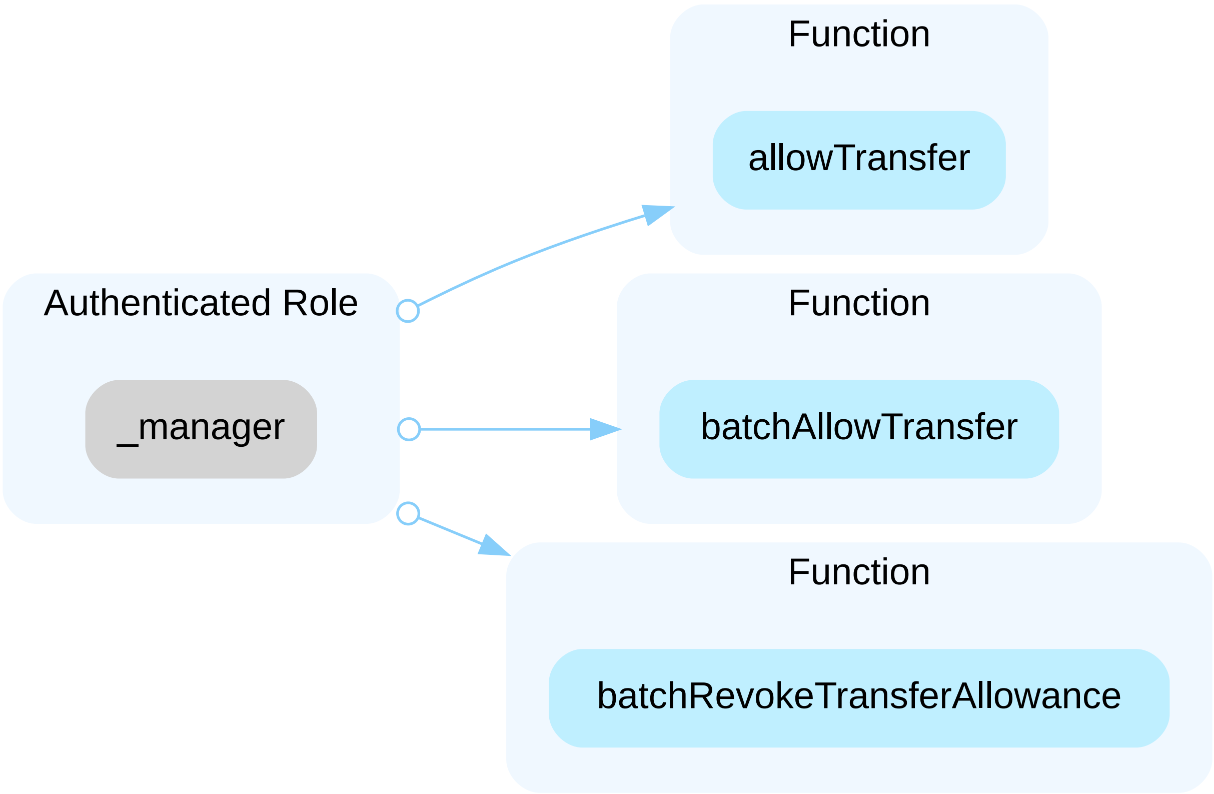
Category	Severity	Location	Status
Centralization / Privilege	**Major**	**projects/Tokenwolf/contracts/Ballot.sol: 41; projects/Tokenwolf/contracts/Swap2.sol: 110, 472; projects/Tokenwolf/contracts/Token3.sol: 84, 130, 138, 148, 158; projects/Tokenwolf/contracts/openzeppelin/access/Ownable.sol: 54, 62**	**Mitigated**

Description

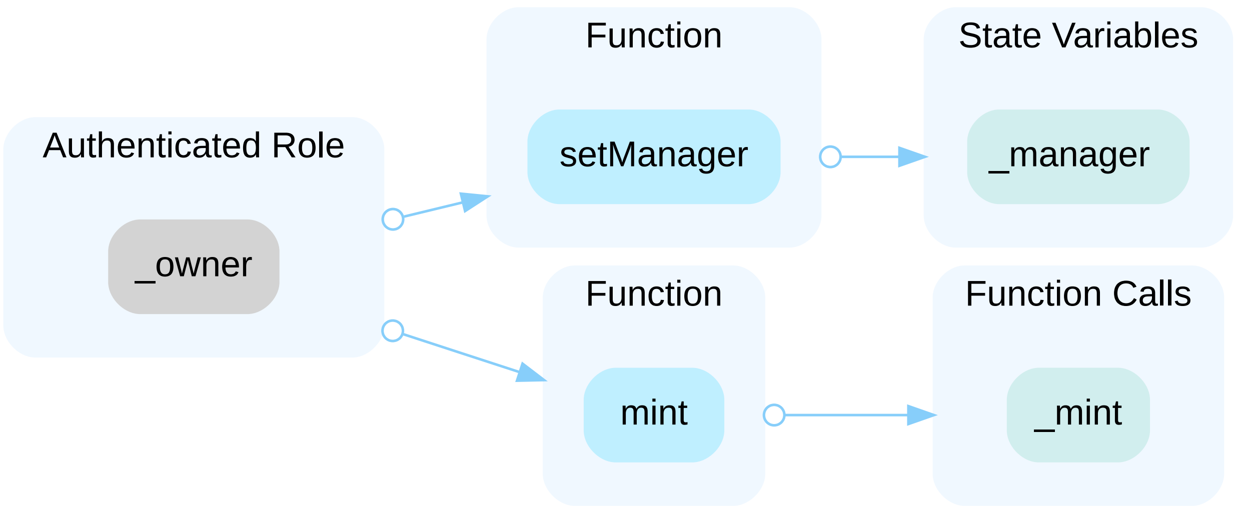
In the contract `Ownable` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and renounce or transfer the ownership.



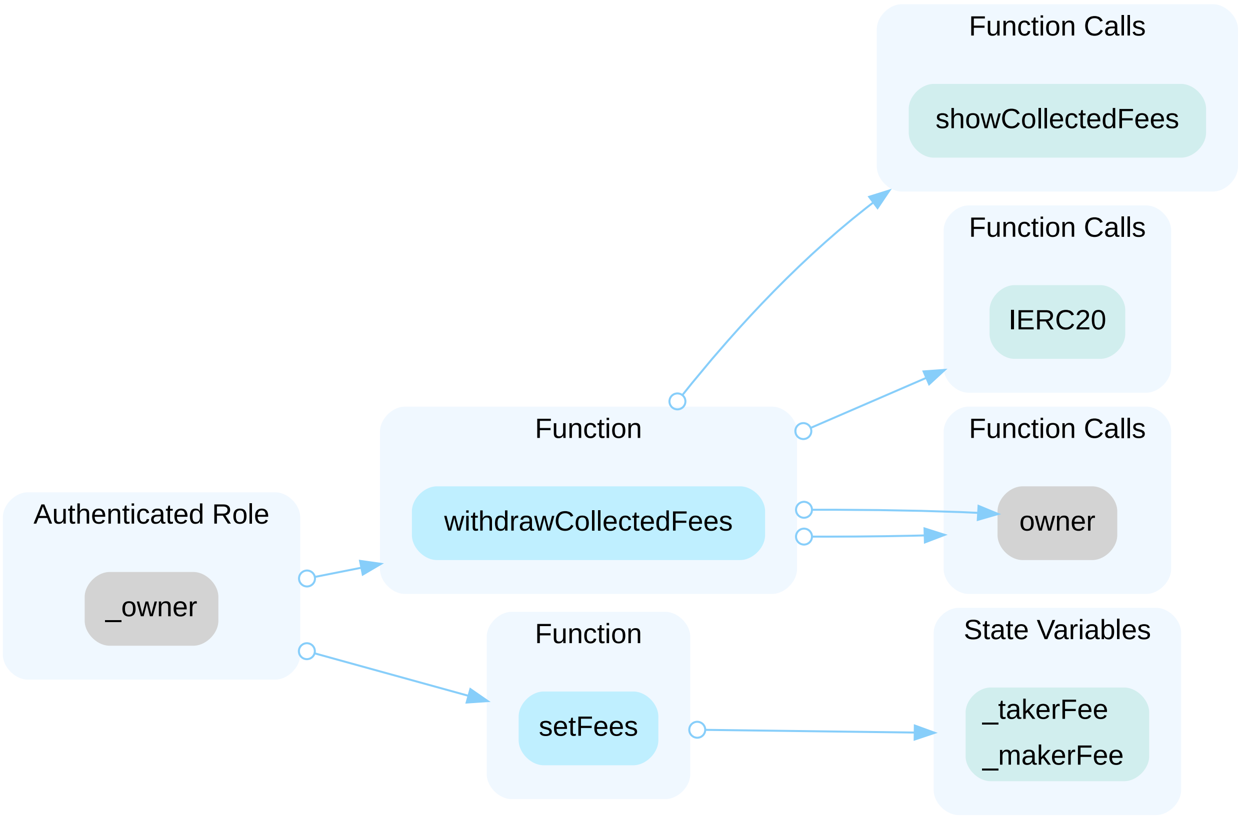
In the contract `SecurityToken` the role `_manager` has authority over the functions shown in the diagram below. Any compromise to the `_manager` account may allow the hacker to take advantage of this authority and allow or disallow users to transfer tokens.



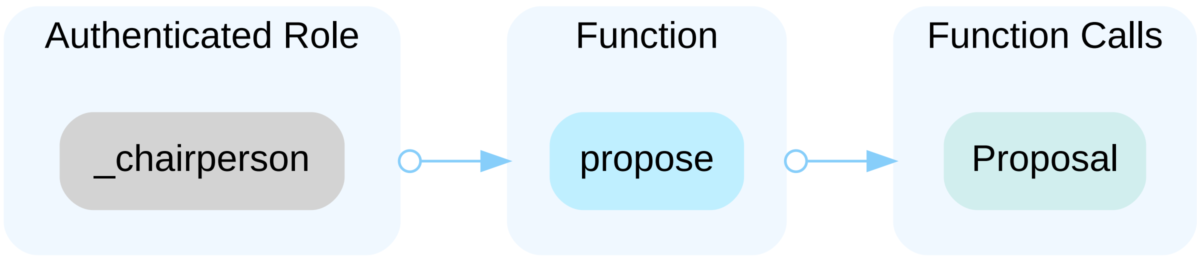
In the contract `SecurityToken` the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and set the role `_manager` or mint tokens.



In the contract `SWAP` 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 set fees or withdraw fees.



In the contract `Ballot` the role `_chairperson` has authority over the function shown in the diagram below. Any compromise to the `_chairperson` account may allow the hacker to take advantage of this authority and add a proposal.



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

Tokenwolf Team:

Because our platform is intended to be used for regulated Security Tokens (where only users may be allowed to trade who have made a KYC/AML process) and the tokens are issued by a BaFIN regulated company a a certain degree of centralization is necessary.

We are aware of the risks about the private key. Therefore, such transactions are only carried out with cold wallet

We decided to use the multisig implementation of paxos (https://github.com/paxosglobal/simple-multisig).

We will transfer _owner, _manager and _chairperson to multisig contracts after contract creation.

We will use 2/3 Multisign for the first contracts deployed.

TCK-02 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	Minor	projects/Tokenwolf/contracts/Ballot.sol: 31; projects/Tokenwolf/contracts/Swap2.sol: 97, 98; projects/Tokenwolf/contracts/Token3.sol: 40, 159	Resolved

Description

Addresses should be checked before assignment or external call to make sure they are not zero addresses.

```
31          _tokenAddress = _token;
```

- `_token` is not zero-checked before being used.

```
97          _token = token;
```

- `token` is not zero-checked before being used.

```
98          _otherToken = otherToken;
```

- `otherToken` is not zero-checked before being used.

```
40          _ownershipContract = ownershipContract_;
```

- `ownershipContract_` is not zero-checked before being used.

```
159         _manager = newManager;
```

- `newManager` is not zero-checked before being used.

Recommendation

We advise adding a zero-check for the passed-in address value to prevent unexpected errors.

Alleviation

This issue was resolved in the updated code.

TTC-01 | INITIAL TOKEN DISTRIBUTION

Category	Severity	Location	Status
Centralization / Privilege	**Major**	**projects/Tokenwolf/contracts/Token3.sol: 36**	**Mitigated**

Description

```
    constructor(string memory name_, string memory symbol_, uint8 decimals_,
uint256 totalSupply_, address ownershipContract_) ERC20(name_, symbol_)
ERC20Permit(name_) {
        _decimals = decimals_;
        // Total Supply mint
        _mint(msg.sender, totalSupply_);

        _owner = msg.sender;
        _manager = msg.sender;
        _ownershipContract = ownershipContract_;
    }
```

`totalSupply_` amount of tokens are sent to the contract deployer when deploying the contract `SecurityToken`. This could be a centralization risk as the deployer can distribute those tokens without obtaining the consensus of the community.

Recommendation

We recommend the team to be transparent regarding the initial token distribution process, and the team shall make enough efforts to restrict the access of the private key.

Alleviation

Tokenwolf Team:

Because the Tokens we want to emit are regulated Security Tokens they are bound to real securities. e.g. 1 Token represents 1 share in a company or 1/10000 of a property.

There are "real" contracts between token holders and the company issuing the tokens who guarantee that mapping ("profit sharing rights").

So the initial distribution is defined within these regulatory approved contracts. An additional distribuition is only possible if the company raises new capital (like a company issuing new shares)

This process (minting) will be done by a BaFIN (German financial regulator) regulated company.

The importance of a good secured private key is known. We plan to use cold wallets for these transactions.

Multisig will be implemented at the start of the project.

Ownership of the token will be transferred to a multisig contract based on this: https://github.com/paxosglobal/simple-multisig/blob/master/contracts/SimpleMultiSig.sol

STC-10 | RESTORE DELETED OFFER

Category	Severity	Location	Status
Logical Issue	● Informational	projects/Tokenwolf/contracts/Swap2.sol: 339~341	● Resolved

Description

```
339         if (offer.quantity == 0) cancelOffer(id);
340         // save modified offer to storage
341         offers[id] = offer;
```

```
359     function cancelOffer(uint8 id) public {
360         // delete offer from storage and mark the slot as free
361         delete(offers[id]);
362         usedSlots[id] = false;
363         _offerCount--;
364         // fire event
365         emit OffersChanged();
366     }
```

In the function `acceptOffer()` , the mapping `offers` restores an offer after removing it when `offer.quantity` == 0.

Recommendation

We recommend not restoring deleted offers in the mapping `offers` .

Alleviation

This issue was resolved in the updated code.

STC-11 | INVALID CHECKS ON `amount`

Category	Severity	Location	Status
Logical Issue	● Informational	projects/Tokenwolf/contracts/Swap2.sol: 259~260	● Resolved

Description

The function `addOffer()` checks if the offer creators have enough balance of corresponding tokens and if they approve enough allowance for this contract.

For buy offers, because the following require checks do not check the "taker fee" which is charged out of `amount` , requiring both the balance and the allowance >= `amount` cannot ensure that the buyer has enough tokens for trade.

```
257        uint256 amount = quantity * price / 10**decimals1;
258        // allowance and balance_of must be sufficient
259        require(token2.balanceOf(msg.sender) >= amount, "Not enough funds");
260        require(token2.allowance(msg.sender, address(this)) >= amount, "Not
enough allowance");
```

Recommendation

We recommend adding or modifying the require checks to ensure the offer creator has enough balance and approves enough allowance for this contract.

Alleviation

This issue was resolved in the updated code.

OPTIMIZATIONS | TOKENWOLF - AUDIT

ID	Title	Category	Severity	Status
STC-09	Gas Optimization In `showOffers()`	Gas Optimization	Optimization	Resolved
TCK-03	Variables That Could Be Declared As Immutable	Gas Optimization	Optimization	Resolved

STC-09 | GAS OPTIMIZATION IN `showOffers()`

Category	Severity	Location	Status
Gas Optimization	Optimization	projects/Tokenwolf/contracts/Swap2.sol: 410	Resolved

Description

The function call of `checkOfferCounterpart()` in the function `showOffers()` can be called in the condition of `if (usedSlots[t] == true)` for gas saving.

Recommendation

We recommend calling `checkOfferCounterpart()` for only exiting offers.

Alleviation

This issue was resolved in the updated code.

TCK-03 | VARIABLES THAT COULD BE DECLARED AS IMMUTABLE

Category	Severity	Location	Status
Gas Optimization	Optimization	projects/Tokenwolf/contracts/Ballot.sol: 20, 22; projects/Tokenwolf/contracts/Swap2.sol: 71, 73, 75, 77; projects/Tokenwolf/contracts/Token3.sol: 20, 23, 25	Resolved

Description

The linked variables assigned in the constructor can be declared as `immutable`. Immutable state variables can be assigned during contract creation but will remain constant throughout the lifetime of a deployed contract. A big advantage of immutable variables is that reading them is significantly cheaper than reading from regular state variables since they will not be stored in storage.

Recommendation

We recommend declaring these variables as immutable.

Alleviation

This issue was resolved in the updated code.

FORMAL VERIFICATION | TOKENWOLF - AUDIT

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-succeed-self	Function `transfer` Succeeds on Admissible Self Transfers
erc20-transfer-correct-amount	Function `transfer` Transfers the Correct Amount in Non-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`

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

Property Name	Title
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-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 projects/Tokenwolf/contracts/openzeppelin/token/ERC20/ERC20.sol)

Detailed results for function `transfer`

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

APPENDIX | TOKENWOLF - AUDIT

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.
Control Flow	Control Flow findings concern the access control imposed on functions, such as owner-only functions being invoke-able by anyone under certain circumstances.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

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.