Own - Audit

Security Assessment

CertiK Assessed on Jul 3rd, 2025

CertiK Assessed on Jul 3rd, 2025

Own - Audit

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

Executive Summary

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

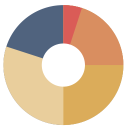
LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 07/03/2025	N/A

CODEBASE
https://github.com/ownprotocol/Own-Smart-Contracts/tree/18df788723bba6fb648dcec89365d708ddd61d9c/contracts/contracts/implementations
View All in Codebase Page

Highlighted Centralization Risks

- Contract upgradeability
- Initial owner token share is 100%

Vulnerability Summary



20	15	0	5	0
Total Findings	Resolved	Partially Resolved	Acknowledged	Declined

🟧	2	Centralization	2 Acknowledged	Centralization findings highlight privileged roles & functions and their capabilities, or instances where the project takes custody of users' assets.
🟥	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.
🟧	2	Major	1 Resolved, 1 Acknowledged	Major risks may include logical errors that, under specific circumstances, could result in fund losses or loss of project control.
🟨	5	Medium	5 Resolved	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
🟨	6	Minor	6 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.

■ 4	Informational	2 Resolved, 2 Acknowledged	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 | OWN - AUDIT

Summary

Executive Summary

Vulnerability Summary

Codebase

Audit Scope

Approach & Methods

Findings

OWA-01 : Attacker Can Drain Stake Tokens By Providing Unlocked Position IDs

OWA-03 : Centralization Related Risks

OWA-20 : Centralized Control of Contract Upgrade

OWA-05 : Attacker Can DoS Claim Functionality For Victim

OWA-21 : Initial Token Distribution

OWA-06 : Incorrect Final Week Reward Handling Leads To Loss Of User Rewards

OWA-07 : Incorrect Week Iteration In Reward Calculation Leads To Loss Of Final Week Rewards

OWA-08 : Potential Insufficient Withdrawable Tokens From Sablier Stream Enables Reward Claim Failures

OWA-09 : Unclaimed Tokens Withdrawable by Owner at Presale End

OWA-24 : Incorrect Reward Calculation When Claiming in Multiple Transactions

OWA-10 : Missing Zero Address Validation

OWA-12 : `addBoostDetails()` Allows Setting Boosts For Current Week

OWA-13 : Incorrect Round Status Before Presale Start

OWA-14 : Cannot Update Rounds Before Presale Starts

OWA-15 : Unmodifiable First Round

OWA-23 : Gas-Heavy Weekly Reward Cache Update

OWA-11 : Inconsistent Stake Time

OWA-17 : Unfinalized Token Name And Symbol

OWA-19 : Allocation Update Without Token Balance Check

OWA-22 : VeOwn Tokens Not Burned Upon Unstake

Appendix

Disclaimer

CODEBASE | OWN - AUDIT

Repository

https://github.com/ownprotocol/Own-Smart-Contracts/tree/18df788723bba6fb648dcec89365d708ddd61d9c/contracts/contracts/implementations

AUDIT SCOPE | OWN - AUDIT

4 files audited ● 4 files with Acknowledged findings

ID	Repo	File	SHA256 Checksum
● OWN	ownprotocol/Own-Smart-Contracts	OWN.sol	d8dbe6e0a5f4689bc10c99f340424f85c7e9aaf146c12dde5ce38b7a9f35c8a1
● POS	ownprotocol/Own-Smart-Contracts	Presale.sol	a35a66af37ca35ccae6faadd7b5a56accc22e1624f1530f268573b98e00538c4
● SOS	ownprotocol/Own-Smart-Contracts	Stake.sol	182c284129c0c4a417fc4e82896bf6e5292f2436a4bd07f8a4c25cb851a4c6f2
● OWO	ownprotocol/Own-Smart-Contracts	veOWN.sol	1d271e4fdcc7811c8a2ca54625f8808bd2b389f7408baf5cca6e066f5c80297e

APPROACH & METHODS | OWN - AUDIT

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

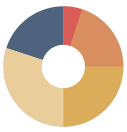
The auditing process pays special attention to the following considerations:

- 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 | OWN - AUDIT



20	1	2	2	5	6	4
Total Findings	Critical	Centralization	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Own - Audit. Through this audit, we have uncovered 20 issues ranging from different severity levels. Utilizing the techniques of Formal Verification, Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
OWA-01	Attacker Can Drain Stake Tokens By Providing Unlocked Position IDs	Logical Issue	Critical	● Resolved
OWA-03	**Centralization Related Risks**	**Centralization**	**Centralization**	● **Acknowledged**
OWA-20	**Centralized Control Of Contract Upgrade**	**Centralization**	**Centralization**	● **Acknowledged**
OWA-05	Attacker Can DoS Claim Functionality For Victim	Denial of Service	Major	● Resolved
OWA-21	**Initial Token Distribution**	**Centralization**	**Major**	● **Acknowledged**
OWA-06	Incorrect Final Week Reward Handling Leads To Loss Of User Rewards	Logical Issue	Medium	● Resolved
OWA-07	Incorrect Week Iteration In Reward Calculation Leads To Loss Of Final Week Rewards	Logical Issue	Medium	● Resolved
OWA-08	Potential Insufficient Withdrawable Tokens From Sablier Stream Enables Reward Claim Failures	Denial of Service	Medium	● Resolved
OWA-09	Unclaimed Tokens Withdrawable By Owner At Presale End	Inconsistency	Medium	● Resolved
OWA-24	Incorrect Reward Calculation When Claiming In Multiple Transactions	Incorrect Calculation	Medium	● Resolved

ID	Title	Category	Severity	Status
OWA-10	Missing Zero Address Validation	Volatile Code	Minor	● Resolved
OWA-12	`addBoostDetails()` Allows Setting Boosts For Current Week	Inconsistency	Minor	● Resolved
OWA-13	Incorrect Round Status Before Presale Start	Logical Issue	Minor	● Resolved
OWA-14	Cannot Update Rounds Before Presale Starts	Logical Issue	Minor	● Resolved
OWA-15	Unmodifiable First Round	Logical Issue	Minor	● Resolved
OWA-23	Gas-Heavy Weekly Reward Cache Update	Denial of Service	Minor	● Resolved
OWA-11	Inconsistent Stake Time	Inconsistency	Informational	● Acknowledged
OWA-17	Unfinalized Token Name And Symbol	Coding Issue	Informational	● Acknowledged
OWA-19	Allocation Update Without Token Balance Check	Inconsistency	Informational	● Resolved
OWA-22	VeOwn Tokens Not Burned Upon Unstake	Design Issue	Informational	● Resolved

OWA-01 | ATTACKER CAN DRAIN STAKE TOKENS BY PROVIDING UNLOCKED POSITION IDS

Category	Severity	Location	Status
Logical Issue	● Critical	Stake.sol: 161	● Resolved

Description

The `Stake::claimRewards()` function checks if the current week is equal to the last week rewards were claimed for a position:

```
if (currentWeek == positionLastWeekRewardsClaimed) {
    continue;
}
```

However, this check is bypassed for unlocked and claimed positions because the `positionLastWeekRewardsClaimed` is set to `finalWeek`, and `currentWeek` is not equal to `finalWeek`. Although the calculated `reward` is zero, the `totalReward` still includes the `positions[positionId].ownAmount`:

```
if (currentWeek > finalWeek) {
    positions[positionId].lastWeekRewardsClaimed = finalWeek;

    uint256 ownAmount = positions[positionId].ownAmount;

>   totalReward += ownAmount;
    totalStakeDeductions += ownAmount;
} else {
```

The vulnerability allows an attacker to drain stake tokens by providing unlocked position IDs.

Proof of Concept

Attacker claims position 1 multiple times with input `positions: [1,1,1]`:

```
describe("Audit - Claim Rewards with Already Claimed Positions", async () => {
  let own: OwnContract;
  let stake: StakeContract;
  let signers: Signers;
  let alice: Signers[0];
  let stake_alice: StakeContract;

  beforeEach(async () => {
    ({ stake, own, signers } = await getContractInstances());
    await stake.write.setDailyRewardAmount([parseEther("5")]);
    await stake.write.startStakingNextWeek();
    await setDayOfWeekInHardhatNode(DayOfWeek.Friday);
    alice = signers[1];
    await own.write.transfer([alice.account.address, parseEther("1000")]);
    stake_alice = await hre.viem.getContractAt(
      "Stake",
      stake.address as `0x${string}`,
      { client: { wallet: alice } }
    );
  });

  it("Should not allow claiming rewards with already claimed positions", async () =>
{
    await setDayOfWeekInHardhatNode(DayOfWeek.Friday);
    const duration = BigInt(7); // 1 week
    // users stake
    await own.write.approve([stake.address, parseEther("1000")], {
      account: alice.account,
    });
    await stake_alice.write.stake([parseEther("1000"), duration]);

    // Attacker stakes tokens for 1 week
    const amount = parseEther("100");
    await own.write.approve([stake.address, amount]);
    await stake.write.stake([amount, duration]);

    // Move to next Saturday
    await setDayOfWeekInHardhatNode(DayOfWeek.Saturday);
    await setDayOfWeekInHardhatNode(DayOfWeek.Saturday);

    // Attempt to claim rewards with positions input [1,1,1]
    // Verify the balance of the stake contract decreased by 300 ethers
    const initialBalance = await own.read.balanceOf([stake.address]);
    await stake.write.claimRewards([[BigInt(1), BigInt(1), BigInt(1)]]);
    const finalBalance = await own.read.balanceOf([stake.address]);
    console.log("initialBalance", initialBalance);
    console.log("finalBalance", finalBalance);
    expect(initialBalance - finalBalance).to.equal(parseEther("300"));
  });
});
```

```
  Stake - claimRewards
    Audit - Claim Rewards with Already Claimed Positions
Own deployed at: 0xe7f1725E7734CE288F8367e1Bb143E90bb3F0512
veOwn deployed at: 0xCf7Ed3AccA5a467e9e704C703E8D87F634fB0Fc9
MockUSDT deployed at: 0xDc64a140Aa3E981100a9becA4E685f962f0cF6C9
Presale deployed at: 0x0165878A594ca255338adfa4d48449f69242Eb8F
Stake deployed at: 0x8A791620dd6260079BF849Dc5567aDC3F2FdC318
initialBalance 110000000000000000000n
finalBalance 80000000000000000000n
      ✔ Should not allow claiming rewards with already claimed positions


  1 passing (1s)
```

Recommendation

Ensure positions that have unstaked cannot claim rewards again by adding a check for unstaked positions.

Alleviation

[Own, 05/20/2025]: Updated check in claimRewards function to early return if they are trying to claim on the last claim week: contracts/contracts/implementations/Stake.sol#L191

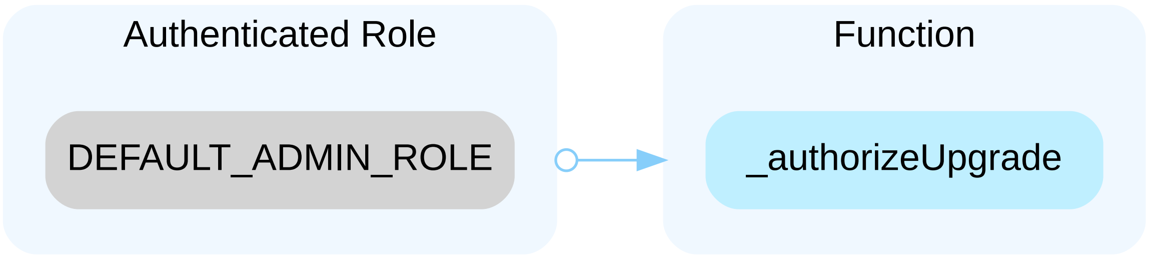
Added extension to existing test case to validate: contracts/test/stake/claimRewards.test.ts#L284 in commit d7c8dcd4648e6a1306851ed45d7e84e7826343ff

OWA-03 | CENTRALIZATION RELATED RISKS

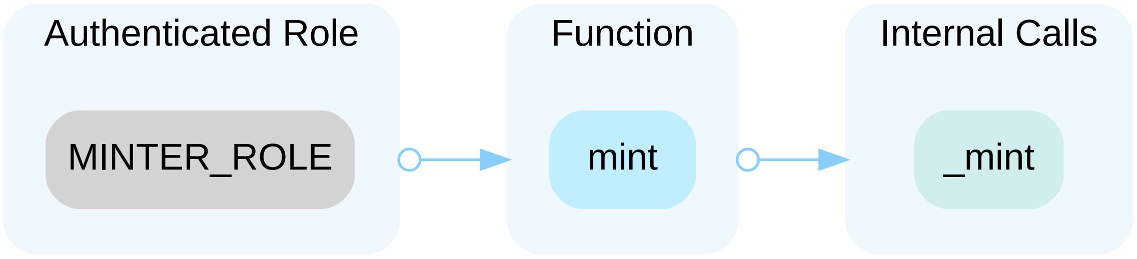
Category	Severity	Location	Status
Centralization	● Centralization		● Acknowledged

Description

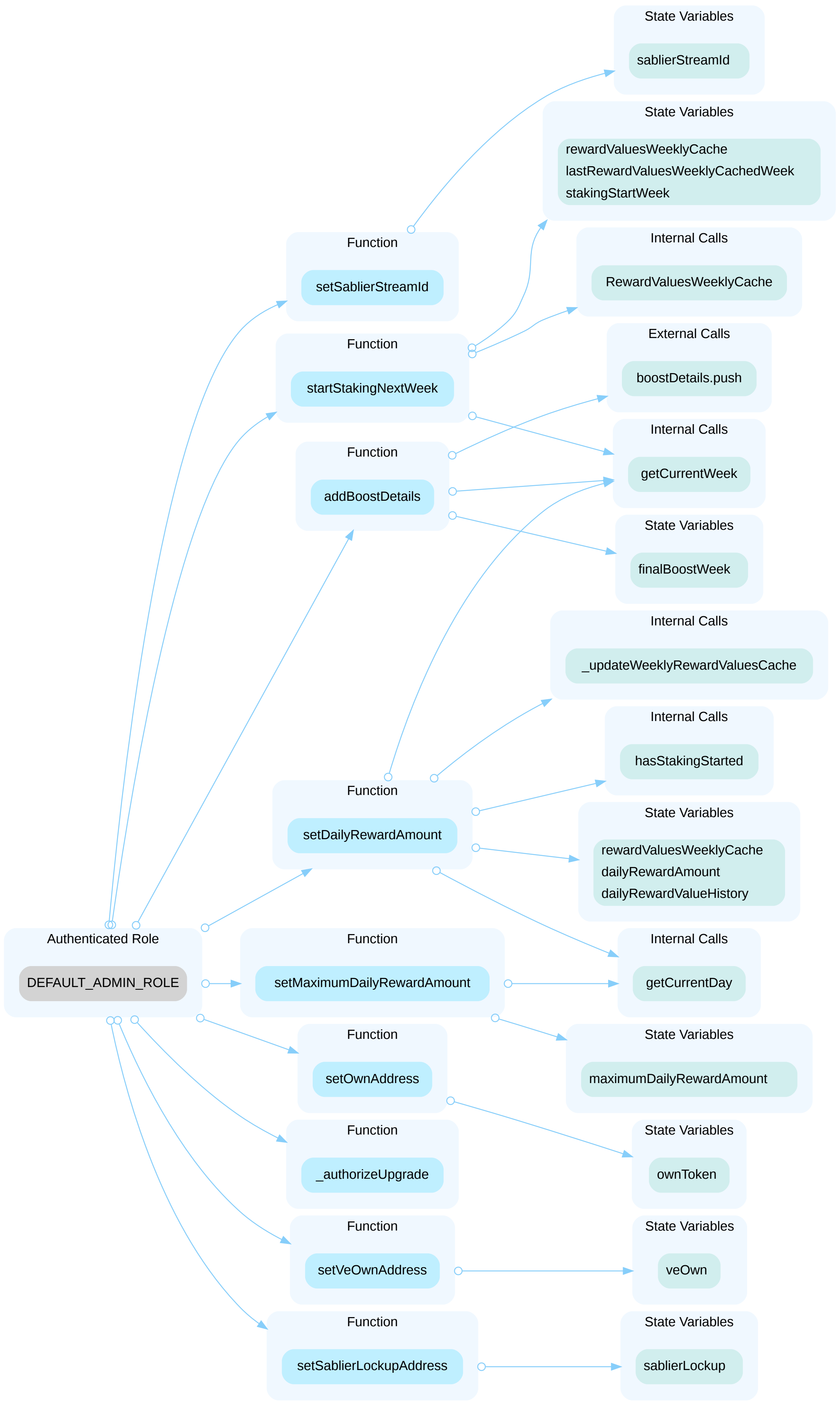
In the contract `VeOwn`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and authorize contract upgrades with the admin role.



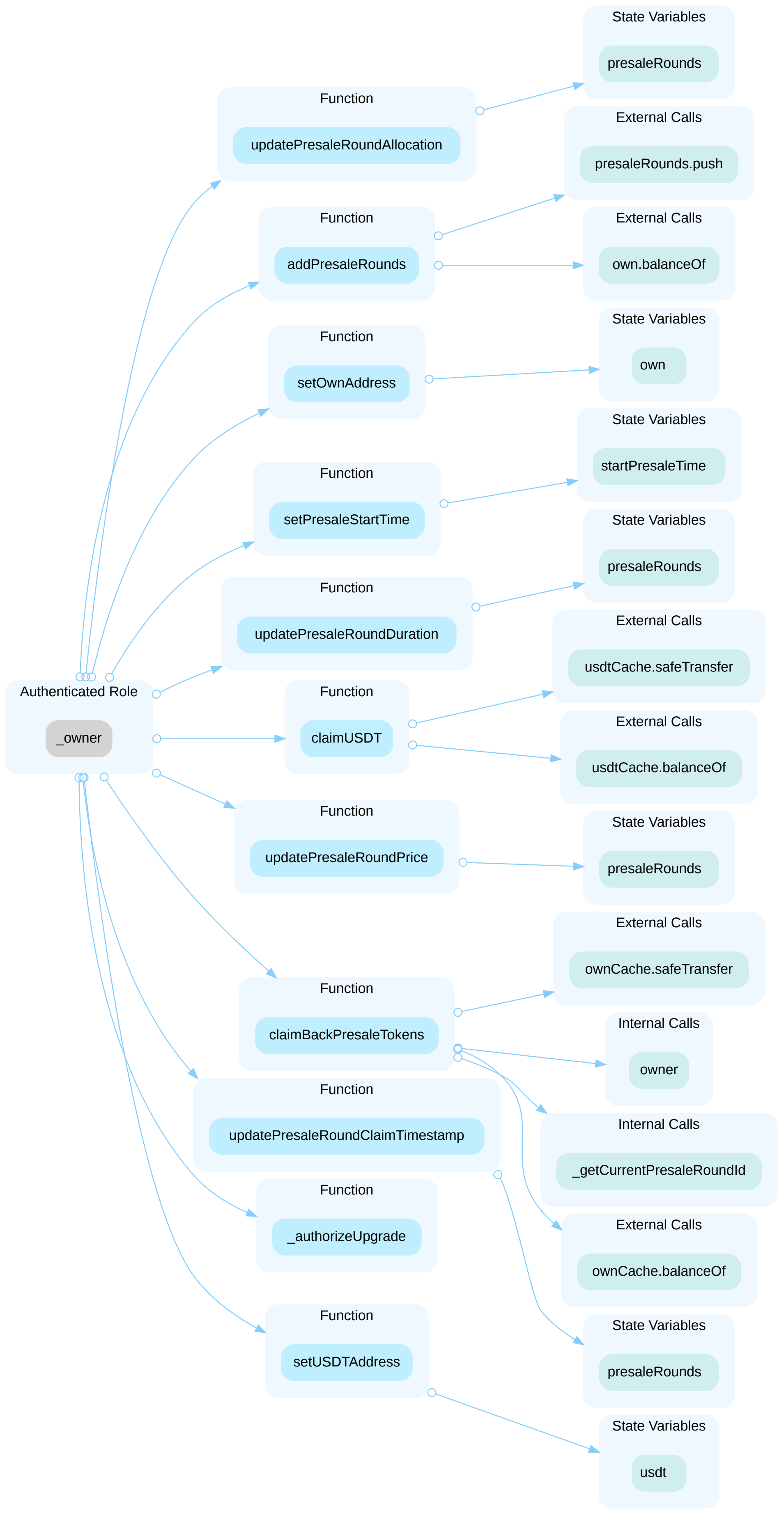
In the contract `VeOwn`, the role `MINTER_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `MINTER_ROLE` account may allow the hacker to take advantage of this authority and mint tokens to a specified address.



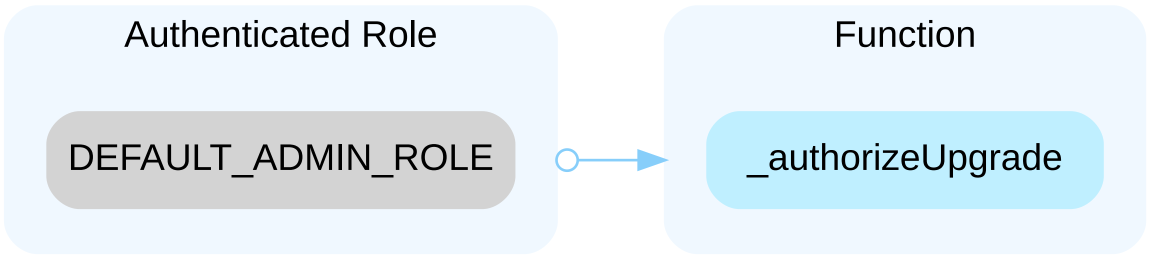
In the contract `Stake`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and set the Sablier stream ID, set the daily reward amount, start staking next week, set the maximum daily reward amount, set own address, authorize the upgrade to new implementation, add boost details with conditions and validations, set the veOwn address, and set the Sablier lockup address.


State Variables
sablierStreamId
State Variables
rewardValuesWeeklyCache
lastRewardValuesWeeklyCachedWeek
stakingStartWeek
Function
setSablierStreamId
Internal Calls
RewardValuesWeeklyCache
Function
startStakingNextWeek
External Calls
boostDetails.push
Internal Calls
getCurrentWeek
Function
addBoostDetails
State Variables
finalBoostWeek
Internal Calls
_updateWeeklyRewardValuesCache
Internal Calls
hasStakingStarted
Function
setDailyRewardAmount
State Variables
rewardValuesWeeklyCache
dailyRewardAmount
dailyRewardValueHistory
Authenticated Role
DEFAULT_ADMIN_ROLE
Function
setMaximumDailyRewardAmount
Internal Calls
getCurrentDay
Function
setOwnAddress
State Variables
maximumDailyRewardAmount
Function
_authorizeUpgrade
State Variables
ownToken
Function
setVeOwnAddress
State Variables
veOwn
Function
setSablierLockupAddress
State Variables
sablierLockup

In the contract `Presale`, 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 update presale round allocation, add new presale rounds, set the own address, set the presale start time, update the presale round duration, claim all USDT balance to the owner, update the presale round price, claim back presale tokens, update presale round claim timestamp, authorize contract upgrade to new implementation, and set the USDT address.


State Variables
presaleRounds
Function
updatePresaleRoundAllocation
External Calls
presaleRounds.push
Function
addPresaleRounds
External Calls
own.balanceOf
State Variables
own
Function
setOwnAddress
State Variables
startPresaleTime
Function
setPresaleStartTime
State Variables
presaleRounds
Function
updatePresaleRoundDuration
External Calls
usdtCache.safeTransfer
Authenticated Role
_owner
Function
claimUSDT
External Calls
usdtCache.balanceOf
Function
updatePresaleRoundPrice
State Variables
presaleRounds
External Calls
ownCache.safeTransfer
Function
claimBackPresaleTokens
Internal Calls
owner
Function
updatePresaleRoundClaimTimestamp
Internal Calls
_getCurrentPresaleRoundId
Function
_authorizeUpgrade
External Calls
ownCache.balanceOf
Function
setUSDTAddress
State Variables
presaleRounds
State Variables
usdt

In the contract `Own`, the role `DEFAULT_ADMIN_ROLE` has authority over the functions shown in the diagram below. Any compromise to the `DEFAULT_ADMIN_ROLE` account may allow the hacker to take advantage of this authority and authorize contract upgrades to new implementations.



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

[Own, 07/02/2025]: The team acknowledged this issue.

[CertiK, 07/02/2025]: It is suggested to implement the aforementioned methods to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

OWA-20 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization	● Centralization	OWN.sol: 13; Presale.sol: 13; Stake.sol: 15; veOWN.sol: 11	● Acknowledged

Description

In the upgradable contracts `Presale` , `Stake` , `Own` and `VeOwn` , the role `DEFAULT_ADMIN_ROLE` has the authority to update the implementation contract behind the proxy contract.

Any compromise to the `DEFAULT_ADMIN_ROLE` may allow a hacker to take advantage of this authority and change the implementation contract which is pointed by proxy and therefore execute potential malicious functionality in the implementation contract.

Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

Short Term:

A combination of a time-lock and a multi signature (⅔, ⅗) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 48 hours, for awareness of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.

- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate the contract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;
  AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

Renouncing ownership of the `admin` account or removing the upgrade functionality can *fully* resolve the risk.

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

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[Own, 07/02/2025]: The team acknowledged this issue.

[CertiK, 07/02/2025]: It is suggested to implement the aforementioned methods to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

OWA-05 | ATTACKER CAN DOS CLAIM FUNCTIONALITY FOR VICTIM

Category	Severity	Location	Status
Denial of Service	● Major	Presale.sol: 347	● Resolved

Description

The `claimPresaleRoundTokens()` function iterates over the entire `presalePurchases[msg.sender]` array to find unclaimed tokens. This exposes a denial-of-service (DoS) risk, where an attacker can force victims' claim transactions to exceed gas limits by bloating their purchase history with trivial entries.

In `Presale::claimPresaleRoundTokens()` , the following loop is used:

```
uint256 presalePurchaseLength = presalePurchases[msg.sender].length;
for (uint256 i = 0; i < presalePurchaseLength; ++i) {
    if (!presalePurchases[msg.sender][i].claimed) {
        // logic to check and claim token
    }
}
```

This design assumes that the number of purchases per user is bounded and reasonable. However, since the `purchasePresaleTokens()` function allows any `_receiver` address to be specified, an attacker can flood a victim's account with tiny purchases by setting `_receiver = victim` .

Each such entry increases the length of `presalePurchases[victim]` , making future calls to `claimPresaleRoundTokens()` increasingly gas-heavy. Eventually, the loop will consume more gas than the block limit, preventing the victim from claiming any tokens at all.

Scenario

1. Victim purchases tokens via `purchasePresaleTokens()` and has one valid entry.
2. Attacker repeatedly calls `purchasePresaleTokens(_usdtAmount:1, _receiver:victim)` with small USDT amounts and sets `_receiver = victim` .
3. Victim tries to call `claimPresaleRoundTokens()` , but the transaction runs out of gas due to looping over thousands of entries.
4. Victim cannot claim any tokens until the gas cost of the loop falls below the block limit — which may never happen.
5. Tokens remain locked indefinitely in the contract, effectively freezing the victim's assets.

Recommendation

Consider implementing claim pagination with `from` and `to` parameters.

Alleviation

[Own, 05/20/2025]: Added pagination to the method in commit 99dd72e00e5ade2253a5d8e4150a53c20ab7644c.

OWA-21 | INITIAL TOKEN DISTRIBUTION

Category	Severity	Location	Status
Centralization	● **Major**	**OWN.sol: 37**	● **Acknowledged**

Description

All of the Own tokens are sent to `_recipient` , an externally-owned account (EOA) address. This is a centralization risk because the owner of the EOA can distribute tokens without obtaining the consensus of the community. Any compromise to these addresses may allow a hacker to steal and sell tokens on the market, resulting in severe damage to the project.

Recommendation

It is recommended that the team be transparent regarding the initial token distribution process. The token distribution plan should be published in a public location that the community can access. The team should make efforts to restrict access to the private keys of the deployer account or EOAs. A multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. Additionally, the team can lock up a portion of tokens, release them with a vesting schedule for long-term success, and deanonymize the project team with a third-party KYC provider to create greater accountability.

Alleviation

[Own, 07/02/2025]: The team acknowledged this issue.

[CertiK, 07/02/2025]: It is suggested to implement the aforementioned methods to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

OWA-06 | INCORRECT FINAL WEEK REWARD HANDLING LEADS TO LOSS OF USER REWARDS

Category	Severity	Location	Status
Logical Issue	● Medium	Stake.sol: 161	● Resolved

Description

In `Stake::claimRewards()` , when a user claims rewards and `currentWeek < finalWeek` , the position's `lastWeekRewardsClaimed` is set to `currentWeek` . However, if `currentWeek == finalWeek` , this logic still applies, which means that the final week's rewards are never distributed because `_calculateRewardsForPosition()` returns 0 `reward` when `positionLastWeekRewardsClaimed == finalWeek` (L646).

```
// File: Stake.sol
            if (currentWeek > finalWeek) {
...
            } else {
                positions[positionId].lastWeekRewardsClaimed = currentWeek;
            }
```

This breaks the expected behavior where users should be able to claim rewards up until and including the final week of their stake.

Recommendation

Allow rewards to be claimed for the `finalWeek` .

Alleviation

[Own, 05/20/2025]: Fixed in commit d7c8dcd4648e6a1306851ed45d7e84e7826343ff.

OWA-07 INCORRECT WEEK ITERATION IN REWARD CALCULATION LEADS TO LOSS OF FINAL WEEK REWARDS

Category	Severity	Location	Status
Logical Issue	● Medium	Stake.sol: 686~709	● Resolved

Description

In `Stake::_calculateRewardsForPosition()`, when `currentWeek > finalWeek`, the function sets `finalWeekToIterateTo` to either `finalWeek` or `finalWeek - 1` depending on whether the final day falls at the end of the week. However, the subsequent `for` loop only iterates up to (but not including) this value due to using `week < finalWeekToIterateTo`:

```
if (currentWeek > finalWeek) {
    if (finalDayEndOfWeek) {
        finalWeekToIterateTo = finalWeek;
    } else {
        finalWeekToIterateTo = finalWeek - 1;
    }
}

// Iterate over every week, using the cached value for efficiency
for (
    uint256 week = startWeekToIterateFrom;
>   week < finalWeekToIterateTo;
    ++week
) {
```

This means that even though the logic intends to include all weeks up to and including `finalWeek`, it actually skips the last full week of rewards because the loop condition does not allow the index to reach `finalWeekToIterateTo`.

This issue compounds with another bug where `lastWeekRewardsClaimed` is set to `finalWeek`, which prevents future claims — effectively making the user lose out on two separate opportunities to claim their final week's reward.

Recommendation

The week iteration should include the final week if `currentWeek > finalWeek`.

Alleviation

[Own, 05/20/2025]: Fix addresses this and OWA-06:

Instead of using the finalWeek when calculating rewards to calculate up to, instead created a `lastClaimWeek` which is an exclusive upper bound of the last week for claiming rewards. The `lastWeekRewardsClaimed` on a position is set to this new value

Created it here: contracts/contracts/implementations/Stake.sol#L187 and using it when comparing the last week to claim rewards for

- contracts/contracts/implementations/Stake.sol#L191
- contracts/contracts/implementations/Stake.sol#L205

Also using this when calculating rewards for the position: contracts/contracts/implementations/Stake.sol#L710.

Updated final week iteration to include the final week: contracts/contracts/implementations/Stake.sol#L762

Then finally when issuing rewards for the final week, updated the check to include the final week: contracts/contracts/implementations/Stake.sol#L778

Commit d7c8dcd4648e6a1306851ed45d7e84e7826343ff.

OWA-08 | POTENTIAL INSUFFICIENT WITHDRAWABLE TOKENS FROM SABLIER STREAM ENABLES REWARD CLAIM FAILURES

Category	Severity	Location	Status
Denial of Service	● Medium	Stake.sol: 161	● Resolved

Description

The `Stake` contract relies on an externally managed Sablier stream (`sablierStreamId`) to provide tokens for reward claims and unstaking.

In `Stake::claimRewards()` , after calculating the total reward amount, the contract checks whether its balance exceeds the required amount. If not, it attempts to withdraw additional tokens from the Sablier stream using `withdrawMax()` . However, since the contract does not lock tokens into Sablier itself, it cannot guarantee the stream has sufficient withdrawable tokens.

This creates a dependency on external management of the Sablier stream. If the stream is misconfigured or depleted, users may face failed transactions when attempting to claim rewards or unstake their principle. The lack of fallback mechanisms further exacerbates this risk.

Recommendation

Consider implementing an emergency withdrawal function for principle-only claims.

Alleviation

[Own, 05/20/2025]: Added an emergency withdraw function `emergencyWithdrawStakePrinciple` in commit d7c8dcd4648e6a1306851ed45d7e84e7826343ff.

OWA-09 | UNCLAIMED TOKENS WITHDRAWABLE BY OWNER AT PRESALE END

Category	Severity	Location	Status
Inconsistency	● Medium	Presale.sol: 120	● Resolved

Description

In `Presale::claimBackPresaleTokens()` , the function checks whether any presale round is still active:

```
(bool roundsInProgress, ) = _getCurrentPresaleRoundId();

if (roundsInProgress) {
    revert CannotClaimBackPresaleTokensWhilePresaleIsInProgress();
}
```

If no round is active, it proceeds to transfer the entire Own token balance of the contract to the owner.

This assumes that all unclaimed tokens are available for owner withdrawal. However, some of these tokens may belong to users who simply haven't yet claimed them. The function does not verify whether all users have successfully withdrawn their purchased tokens before allowing the owner to sweep the balance.

Recommendation

Update `claimBackPresaleTokens()` to ensure only leftover tokens (not allocated to any purchase) can be reclaimed.

Alleviation

[Own, 05/20/2025]: Updated this method to only transfer tokens that haven't been allocated to presale purchases in commit 99dd72e00e5ade2253a5d8e4150a53c20ab7644c.

OWA-24 | INCORRECT REWARD CALCULATION WHEN CLAIMING IN MULTIPLE TRANSACTIONS

Category	Severity	Location	Status
Incorrect Calculation	● Medium	Stake.sol: 682~684	● Resolved

Description

In the ·_calculateRewardsForPosition()· function, a logical flaw exists when handling reward claims across multiple transactions for positions that start mid-week (i.e., startDay % 7 != 0). Specifically, the function incorrectly skips the entire week immediately following the lastWeekRewardsClaimed due to an unconditional increment:

```
uint256 startWeekToIterateFrom = positionLastWeekRewardsClaimed;
if (!enteredAtStartOfWeek) {
    ++startWeekToIterateFrom;
}
```

This assumes that the previous claim call has already fully accounted for the first partial week of staking via the function `_rewardPerTokenForDayRange` . However, in multi-claim scenarios, only the first few days of that week may have been claimed in the first transaction, while the remainder of the week is still unclaimed. The unconditional increment then skips over this week entirely in future claims.

Scenario

1. User stakes on Day 1 (Week 1, Day 2). The startDay = 1, and the startWeek = 1. The finalDay is 35, and the finalWeek is 6.
2. The first claim occurs in Week 4 (currentWeek = 4): The first partial week (Day 1–6) is processed via daily reward logic. Weeks 2 and 3 are calculated using weekly loop. The `lastWeekRewardsClaimed` is updated to 4.
3. The second claim occurs in Week 9 (currentWeek = 9): `startWeekToIterateFrom` = 4 + 1 = 5. `finalWeekToIterateTo` = 5 (Week 6 is partial).

Week 4 is skipped entirely, even though it was never processed. This results in permanent reward loss for Week 4.

Recommendation

We recommend the team to only increment `startWeekToIterateFrom` when both `!enteredAtStartOfWeek` and `startWeek == positionLastWeekRewardsClaimed` conditions are met.

Alleviation

[Own, 06/30/2025]: Nice find! Pushed a fix with this commit hash 5ac5c929e422922e0d28347d1093e9040b8940c0

OWA-10 | MISSING ZERO ADDRESS VALIDATION

Category	Severity	Location	Status
Volatile Code	● Minor	Presale.sol: 43	● Resolved

Description

The cited address input is missing a check that it is not `address(0)` .

Recommendation

We recommend adding a check the passed-in address is not `address(0)` to prevent unexpected errors.

Alleviation

[Own, 05/20/2025]: Added zero address validation to all initializer methods in commit 9229f1f86d14d48e82cf12fb417da53ed2176f4d

OWA-12 | `addBoostDetails()` ALLOWS SETTING BOOSTS FOR CURRENT WEEK

Category	Severity	Location	Status
Inconsistency	● Minor	Stake.sol: 808	● Resolved

Description

In `Stake::addBoostDetails()` , the function checks if `_boostDetails[i].startWeek` is less than `weeksSinceStakingStarted` . Since `weeksSinceStakingStarted` represents how many weeks have passed since staking began, this check was designed to prevent boosts from being applied retroactively or to currently ongoing weeks.

However, when `currentWeek == stakingStartWeek + weeksSinceStakingStarted` , meaning the admin is trying to add a boost for the current week, the condition:

```
if (currentWeek >= stakingStartWeekCache && _boostDetails[i].startWeek <
weeksSinceStakingStarted)
```

allows `_boostDetails[i].startWeek == weeksSinceStakingStarted` , because it only reverts if the start week is strictly less than `weeksSinceStakingStarted` .

As a result, admins can assign boosts to the current week, change rewards mid-week and violates the requirement: "If staking has started and trying to update a boost that has already started, revert".

Recommendation

Prevent boost settings for current and past weeks.

Alleviation

[Own, 05/20/2025]: Updated to prevent boost settings for current and past weeks in commit 60518751f9ea3549d905c02fc7324c58fe74c119

OWA-13 | INCORRECT ROUND STATUS BEFORE PRESALE START

Category	Severity	Location	Status
Logical Issue	● Minor	Presale.sol: 436	● Resolved

Description

In `Presale::_getCurrentPresaleRoundId()` , the logic begins by checking `hasPresaleStarted()` . If false, `presaleTimeElapsed` remains at zero:

```
uint256 presaleTimeElapsed;
if (hasPresaleStarted()) {
    presaleTimeElapsed = block.timestamp - startPresaleTime;
}
```

It then proceeds to loop through presale rounds:

```
for (uint256 i = 0; i < presaleRoundLength; ++i) {
    uint256 presaleRoundDuration = presaleRounds[i].duration;
    if (presaleTimeElapsed < presaleRoundDuration) {
        return (true, i);
    }

    presaleTimeElapsed -= presaleRoundDuration;
}
```

When `presaleTimeElapsed == 0` and `i == 0` , the condition `presaleTimeElapsed < presaleRoundDuration` will always be true for any duration > 0. As a result, the function returns `(true, 0)` even though the presale has not started.

This leads to misleading information from `getCurrentPresaleRoundDetails()` , which uses this function to determine the active round. Off-chain tools or integrations may interpret this as round 0 being active and allow users to attempt actions like purchasing tokens before the presale officially starts.

Recommendation

`Presale::_getCurrentPresaleRoundId()` should return `(false, 0)` when presale has not started.

Alleviation

[Own, 05/20/2025]: Updated to return false when the presale hasn't started in commit a030d2bb942b0cb465f3ff6eb5e29f6e2c4ab03f.

OWA-14 | CANNOT UPDATE ROUNDS BEFORE PRESALE STARTS

Category	Severity	Location	Status
Logical Issue	● Minor	Presale.sol: 163	● Resolved

Description

In `Presale::updatePresaleRound()` , the logic checks:

```
163  if (!roundsInProgress) {
164      revert AllPresaleRoundsHaveEnded();
165  }
```

This assumes that `roundsInProgress == false` implies that all rounds have ended and no further updates should be allowed. However, this condition can also be true when the presale has **not yet started**, even though future rounds may still be editable.

Recommendation

Consider updating the modifier to allow updates only if the presale has started and the target round hasn't already passed.

Alleviation

[Own, 05/20/2025]: Issue acknowledged and handling has been added to address this in commit a030d2bb942b0cb465f3ff6eb5e29f6e2c4ab03f.

OWA-15 | UNMODIFIABLE FIRST ROUND

Category	Severity	Location	Status
Logical Issue	● Minor	Presale.sol: 167	● Resolved

Description

In `Presale::updatePresaleRound()` , the following check is used:

```
if (currentRoundId >= _roundId) {
    revert CannotUpdatePresaleRoundThatHasEndedOrInProgress();
}
```

When the presale has not yet started, `_getCurrentPresaleRoundId()` should return `(false, 0)` (after fixes) — indicating no round is active. However, if `_roundId == 0` , this condition becomes true (`currentRoundId >= _roundId`) and blocks the update, even though the presale hasn't begun and the round is not in progress.

Recommendation

Allow updating first round when presale has not started:

```
- if (currentRoundId >= _roundId) {
+ if (currentRoundId >= _roundId && hasPresaleStarted()) {
    revert CannotUpdatePresaleRoundThatHasEndedOrInProgress();
}
```

Alleviation

[Own, 05/20/2025]: Updated the modifier to handle updating the first round if the presale hasn't started in commit a030d2bb942b0cb465f3ff6eb5e29f6e2c4ab03f.

OWA-23 | GAS-HEAVY WEEKLY REWARD CACHE UPDATE

Category	Severity	Location	Status
Denial of Service	● Minor	Stake.sol: 527	● Resolved

Description

In `Stake::_updateWeeklyRewardValuesCache()` , the contract calls `_getValuesToUpdateWeeklyRewardValuesCache()` , which iterates from `fromWeek` to `currentWeek - 1` . For each week, it processes all 7 days.

If no staking or claim rewards activity has occurred for months, the update could involve many of weeks. Each week requires up to 7 daily iterations, increasing computation time exponentially. This leads to transactions calling `stake()` or `claimRewards()` may fail due to out-of-gas errors if they trigger this update.

Recommendation

Consider adding a public function to update the cache incrementally.

Alleviation

[Own, 05/20/2025]: Added public function (updateWeeklyRewardValuesCache) to update the cache in commit 95c742477c9bc56ba3f82603223fa84052299228

OWA-11 | INCONSISTENT STAKE TIME

Category	Severity	Location	Status
Inconsistency	● Informational	Stake.sol: 127, 199	● Acknowledged

Description

In `Stake::claimRewards()` , the `finalWeek` is derived from `finalDay` using integer division by 7:

```
199  uint256 finalWeek = positions[positionId].finalDay / 7;
```

However, if `finalDay` does not correspond to the last day of the week, the calculated `finalWeek` will not reflect the actual staking period. For example:

- Given `currentDay = 20218` , `currentWeek = 2888` , and `stakingStartWeek = 2886` .
- A user stakes for 12 days: `{startDay: 20219, finalDay: 20230}` .
- Calculated `finalWeek = 20230 / 7 = 2890` , implying a stake period of 2 weeks.
- User-provided stake period in days is 12, which spans parts of 2 weeks but does not fully cover them.

This inconsistency arises because `finalWeek` is treated as the absolute end of the stake, regardless of whether `finalDay` falls on the last day of the week.

Recommendation

Consider aligning `finalDay` with the last day of the final week.

Alleviation

[Own, 05/20/2025]: This was an intentional decision to not align `finalDay` with the last day of the week as that would force users to stake longer than they requested. This is only used in internal calculations so it doesn't seem like an issue.

OWA-17 | UNFINALIZED TOKEN NAME AND SYMBOL

Category	Severity	Location	Status
Coding Issue	● Informational	OWN.sol: 31, 33	● Acknowledged

Description

In `Own::initialize()` , the following line is used:

```
__ERC20_init("testToken", "testToken");
```

This sets both the token name and symbol to `"testToken"` , which is clearly a placeholder intended for testing. These values should be updated before deploying to production.

Recommendation

Set correct token name and symbol before deployment.

Alleviation

[Own, 07/02/2025]: The team acknowledged the issue.

OWA-19 | ALLOCATION UPDATE WITHOUT TOKEN BALANCE CHECK

Category	Severity	Location	Status
Inconsistency	● Informational	Presale.sol: 206	● Resolved

Description

The `updatePresaleRoundAllocation()` function enables modifying the token allocation for a specific presale round. However, it does not validate whether the contract actually holds sufficient Own tokens after the update, which could result in over-allocation and unfulfilled claims.

Recommendation

Ensure the contract contains enough tokens to support the updated allocation.

Alleviation

[Own, 05/20/2025]: Added validation to the method to ensure it can't be over-allocated in commit a030d2bb942b0cb465f3ff6eb5e29f6e2c4ab03f.

OWA-22 | VEOWN TOKENS NOT BURNED UPON UNSTAKE

Category	Severity	Location	Status
Design Issue	● Informational	Stake.sol: 15	● Resolved

Description

The `Stake` contract allows users to unstake their Own tokens without burning the corresponding veOwn tokens issued during staking. This enables users to retain voting power or other governance privileges even after fully withdrawing their stake.

Recommendation

Consider implementing relevant logic based on project's requirement.

Alleviation

[Own, 05/27/2025]: Fixed in PR by calculating a user's voting power (veOwn balance) based solely on their active stake positions and the amount of veOwn those positions currently generate.

APPENDIX | OWN - AUDIT

Finding Categories

Categories	Description
Coding Issue	Coding Issue findings are about general code quality including, but not limited to, coding mistakes, compile errors, and performance issues.
Incorrect Calculation	Incorrect Calculation findings are about issues in numeric computation such as rounding errors, overflows, out-of-bounds and any computation that is not intended.
Denial of Service	Denial of Service findings indicate that an attacker may prevent the program from operating correctly or responding to legitimate requests.
Inconsistency	Inconsistency findings refer to different parts of code that are not consistent or code that does not behave according to its specification.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

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

Elevating Your Entire Web3 Journey

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.