

yAudit Sonne Finance Review

Review Resources:

- [Sonne docs](#)

Auditors:

- engn33r
- spalen

Table of Contents

- 1 [Review Summary](#)
- 2 [Scope](#)
- 3 [Code Evaluation Matrix](#)
- 4 [Findings Explanation](#)
- 5 [Critical Findings](#)
- 6 [High Findings](#)
 - a [1. High - Sonne market parameters have a high risk profile](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
 - d [Developer Response](#)
 - b [2. High - EOA admins control staking rewards](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
 - d [Developer Response](#)
 - c [3. High - EOA swapped staked token rewards to ETH for gas](#)
 - a [Technical Details](#)
 - b [Impact](#)

- c [Recommendation](#)
 - d [Developer Response](#)
- d [4. High - Unclear protection against Hundred Finance attack vector](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
 - d [Developer Response](#)

7 [Medium Findings](#)

- a [1. Medium - No sequencer uptime check before querying Chainlink data on L2](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
 - d [Developer Response](#)
- b [2. Medium - Poor choice of interest rate models in Sonne](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
 - d [Developer Response](#)
- c [3. Medium - Sonne interest rate model math error](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
 - d [Developer Response](#)
- d [4. Medium - No Borrow Caps set](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
 - d [Developer Response](#)
- e [5. Medium - Reserves stored in Sonne adds currency risk](#)

- a [Technical Details](#)
- b [Impact](#)
- c [Recommendation](#)
- d [Developer Response](#)

f 6. Medium - No Chainlink staleness check in `_getLatestPrice()`

- a [Technical Details](#)
- b [Impact](#)
- c [Recommendation](#)
- d [Developer Response](#)

8 Low Findings

a 1. Low - User can accidentally postpone staking release time

- a [Technical Details](#)
- b [Impact](#)
- c [Recommendation](#)
- d [Developer Response](#)

b 2. Low - Small SONNE/USDC liquidity leaves potential for large price shifts

- a [Technical Details](#)
- b [Impact](#)
- c [Recommendation](#)
- d [Developer Response](#)

c 3. Low - Incorrect totalSupply in Comp for on-chain voting

- a [Technical Details](#)
- b [Impact](#)
- c [Recommendation](#)
- d [Developer Response](#)

d 4. Low - SUSD and LUSD Chainlink price feeds are not standardized verified feeds

- a [Technical Details](#)
- b [Impact](#)
- c [Recommendation](#)

- d [Developer Response](#)
- e [5. Low - Functions calls to uninitialized address](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
 - d [Developer Response](#)

- f [6. Low - High default slippage may lose value](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
 - d [Developer Response](#)

9 [Gas Savings Findings](#)

- a [1. Gas - Remove SafeMath import](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- b [2. Gas - Cache variable](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- c [3. Gas - Replace `totalShares` and `shares\[\]`](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- d [4. Gas - Remove duplicate line of code](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- e [5. Gas - Consistently apply unchecked for gas savings](#)

- a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- f [6. Gas - Use Solidity errors in 0.8.4+](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- g [7. Gas - Borrow gas optimizations from Compound](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- h [8. Gas - Initialize variable only if needed](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- 10 [Informational Findings](#)
 - a [1. Informational - Undocumented market creation process](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
 - b [2. Informational - Replace magic numbers with constants](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
 - c [3. Informational - Remove unused code](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
 - d [4. Informational - Add events to Distributor.sol](#)

- a [Technical Details](#)
- b [Impact](#)
- c [Recommendation](#)
- e [5. Informational - Add event to `setWithdrawalPendingTime\(\)` and `burn\(\)`](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- f [6. Informational - Remove unused files](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- g [7. Informational - Outsourcing staking yield generation increases risk](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- h [8. Informational - Use consistent naming for internal functions](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- i [9. Informational - Remove redundant import](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- j [10. Informational - Missing NatSpec](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- k [11. Informational - Simplify `claimInternal\(\)` arguments](#)
 - a [Technical Details](#)

- b [Impact](#)
 - c [Recommendation](#)
- l [12. Informational - Document the end of rewards accumulation when `burn\(\)` is called](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- m [13. Informational - Distributor function `removeToken\(\)` can lose funds](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- n [14. Informational - Unnecessary code from Compound](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- o [15. Informational - Make public functions external](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- p [16. Informational - Use standard implementation approach in SafeMath.sol](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- q [17. Informational - Incorrect price oracle address in Sonne docs](#)
 - a [Technical Details](#)
 - b [Impact](#)
 - c [Recommendation](#)
- r [18. Informational - No comparison against minAnswer or maxAnswer](#)
 - a [Technical Details](#)

b [Impact](#)

c [Recommendation](#)

s [19. Informational - `_getLatestPrice\(\)` timeStamp return value never used](#)

a [Technical Details](#)

b [Impact](#)

c [Recommendation](#)

t [20. Informational - Protocol will stop working after year 2106](#)

a [Technical Details](#)

b [Impact](#)

c [Recommendation](#)

u [21. Informational - Typos](#)

a [Technical Details](#)

b [Impact](#)

c [Recommendation](#)

v [22. Informational - Use interface instead of call function](#)

a [Technical Details](#)

b [Impact](#)

c [Recommendation](#)

w [23. Informational - SNX token risk](#)

a [Technical Details](#)

b [Impact](#)

c [Recommendation](#)

x [24. Informational - Solidity version 0.8.20](#)

a [Technical Details](#)

b [Impact](#)

c [Recommendation](#)

11 [Final remarks](#)

Review Summary

Sonne Finance

Sonne Finance provides a lending market and a staking market. The lending market is a fork of Compound II which allows users to deposit a single token and earn variable yield on their deposit. Users with sufficient collateral can borrow tokens from any market while paying a variable interest rate. At the time of this review, borrowing and depositing were incentivized with SONNE token rewards. After users earn SONNE tokens, they can stake their SONNE in the staking contracts to earn yield. The uSONNE staking contract earns rewards in USDC while sSONNE earns rewards in SONNE. The Sonne staking contracts are forked from Tarot Finance. This combination of lending and staking allows holders of any token supported by a Sonne market to earn yield by lending the token to the market, earn SONNE, and then earn additional yield by staking the earned SONNE token. Sonne Finance is only deployed on Optimism and made some minor modifications to Compound Finance for maximum compatibility with this L2 chain, most notably using `block.timestamp` instead of `block.number`.

The contracts of the Sonne Finance Repositories were reviewed over 21 days. The code review was performed by 2 auditors between May 1 and May 21, 2023. The repository was under active development during the review, but the review was limited to the latest commit at the start of the review. This was commit [36ac087f7e5154e8c4ebf814b42b0e7aba848b5b](#) for the lending-protocol repo, commit [eaefb378243b274b1328f76238ed3bd42dc6e571](#) for the staking-protocol repo, and commit [3e7976fb0a2bc282a7118d4e0f16a231a054d742](#) for the sonne-token repo.

Scope

The scope of the review consisted of the following contracts at the specific commit:

- The entire lending-protocol github repository
- The entire staking-protocol github repository
- The Sonne.sol contract in the sonne-token github repository

The liquidity generation event and other contracts in the sonne-token github repository were out of scope. After the findings were presented to the Sonne Finance team, fixes were made and included in several PRs.

This review is a code review to identify potential vulnerabilities in the code. The reviewers

did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Sonne Finance and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Average	There are privileged addresses, including admin and owner addresses, which can access certain functions that no other account can call.
Mathematics	Average	The same math from Compound is used in Sonne. Because the math has not changed from Compound, it can be assumed to be quite secure, but the collateral factors have changed which could introduce some additional risk and the risk modelling around this change (which is not in the protocol code itself) is complex.
Complexity	Average	Sonne is a fork of compound and changed very little of the Compound logic. The added contracts and functions do not greatly change the complexity of the protocol compared to Compound.
Libraries	Good	The only library is SafeMath, which has an implementation very similar to OpenZeppelin's SafeMath for solidity 0.8.X.
Decentralization	Low	There is very little decentralization in the protocol because most actions relating to market parameters and adding new markets are controlled by the Comptroller admin. The value of rewards sent to the uSONNE and sSONNE contracts is fully controlled by an EOA address, as is the

Category	Mark	Description
		process of swapping some rewards for gas.
Code stability	Good	The protocol is already deployed to Optimism and was not undergoing any major changes at the time of the audit. Sonne is a Compound fork, and the latest commit on the main Compound repository (where this code was forked from) was almost 1 year old at the time of this review.
Documentation	Low	The docs website information did not clearly match the logic in the contract code, as many questions were not answered by the docs. The key roles played by EOAs and the EOA taking some earned fees for gas are important facts to documented. The staking-protocol contracts and custom lending-protocol contracts did not have NatSpec.
Monitoring	Good	Because Sonne is forked from Compound, it has all the same on-chain events as Compound. The developers had a manual process for monitoring protocol-level bad debt at regular intervals, but an active continuous process should be implemented in the future.
Testing and verification	Average	Sonne is forked from Compound and the Compound code has been tested extensively. There were some additional tests to provide coverage for the custom ExternalRewardDistributor contract.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
 - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended consequences/actions that are outside the scope of the requirements
- Gas savings
 - Findings that can improve the gas efficiency of the contracts

- Informational
 - Findings including recommendations and best practices

Critical Findings

None.

High Findings

1. High - Sonne market parameters have a high risk profile

Many risk parameters, including collateral factor and reserve factor values, are controlled by the protocol multisig. Sonne has chosen unusual collateral factor values for certain assets and does not provide a clear explanation for why these custom values were chosen. Due to the lack of evidence of any risk modelling and the inconsistent protocol behavior compared to other lending protocols, the choice of parameters leaves the protocol at risk of bad debt which could negatively impact the overall health of the protocol.

Technical Details

In summary, the collateral limits set by Sonne are unusually high in certain markets. Additionally, the overall amount of borrowed assets in Sonne compared to the total assets deposited is also very high. The combination of these factors indicates that Sonne is at risk of collecting bad debt and may not be in a position to pay off this bad debt, which could permanently impact the protocol's reputation.

For a first data point, Sonne collateral parameters can be compared to [Compound Finance v2](#), [Aave v2 Ethereum](#), and [Aave v3 Optimism](#) market parameters. The table below shows the collateral and reserve parameters for the different underlying assets.

TOKEN	Sonne Collateral Factor	Compound Collateral Factor	Aave v2 Collateral Factor	Aave v3 Optimism Collateral Factor	Sonne Reserve Factor

TOKEN	Sonne Collateral Factor	Compound Collateral Factor	Aave v2 Collateral Factor	Aave v3 Optimism Collateral Factor	Sonne Reserve Factor
OP	65%			30.00%	20%
USDC	90%	85.50%	80.00%	80.00%	10%
USDT	90%	0.00%	0.00%	75.00%	10%
DAI	90%	83.50%	75.00%	78.00%	10%
sUSD	60%		0.00%	60.00%	20%
wETH	75%	82.50%	82.50%	80.00%	15%
SNX	45%		46.00%		27%
wBTC	70%	70.00%	72.00%	73.00%	20%
LUSD	60%		0.00%	0.00%	20%
wstETH	60%		72.00%	70.00%	18%

Some of the most obvious cases where the Sonne choices differ from other protocols include:

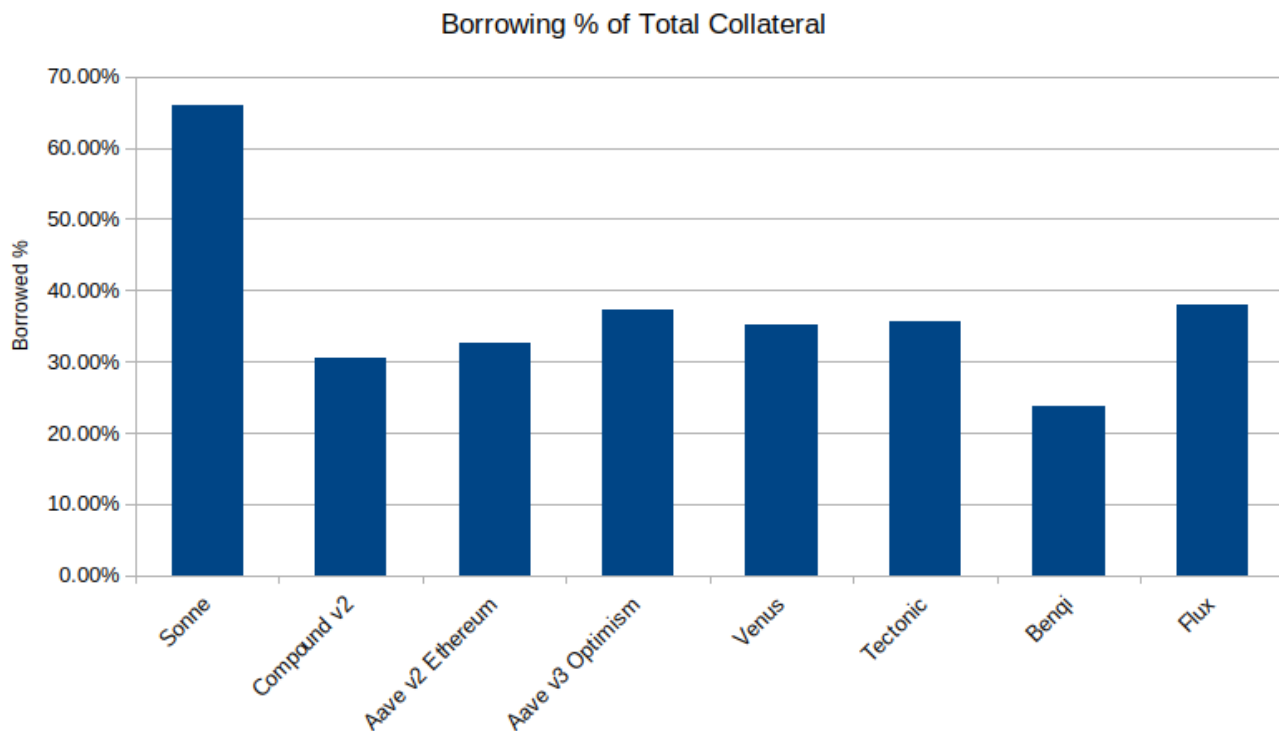
- 1 90% USDT collateral factor (compared to 0%)
- 2 60% sUSD and LUSD collateral factors (compared to 0%)
- 3 75% WETH collateral factor (compared to 80%–82.5%)

Aave and Compound have undergone extensive risk analysis as evidenced by their dashboards on Gauntlet’s website ([1](#), [2](#)), Gauntlet risk assessment reports ([1](#), [2](#)), and regular governance proposals that modify the risk parameters according to market conditions ([1](#), [2](#)). Aave even [disabled borrowing on many tokens in AIP-125](#) in Nov 2022 after an attack on the protocol caused an accumulation of bad debt. Some of the tokens that had borrowing disabled, including SNX (which exists on Sonne), have not yet reenabled borrowing. While [Euler Finance decided to allow borrowing against USDT collateral](#), it is unclear if Euler

Finance has undergone the same level of extensive risk modelling that Compound and Aave have. Comptroller has a hardcoded 90% collateralization factor limit, and because the `liquidationIncentiveMantissa` is 108%, this means that there is less than a 3% margin in price movement ($100\% - (108\% * 90\%) = 2.8\%$) for some markets before a partial liquidation of a position will push the entire position towards liquidation, as outlined [by the “Counterproductive Incentives” high finding](#) in this OpenZeppelin report.

A second data point can be found by comparing the amount of assets lent and borrowed between different lending protocols. A specific focus was placed on comparing Sonne with other Compound forks (Venus, Tectonic, Benqi, Flux) for a more equal comparison. Note that Venus also has a Gauntlet [dashboard](#). The results below show that Sonne is allowing a much higher amount of borrowing than other lending protocols, which means Sonne is at higher risk of accumulating bad debt than other lending protocols. The reason behind such high borrowing might be the added SONNE token incentives, but the specific borrowing interest rate curves may be another reason behind such high borrowing.

Protocol	Total assets (\$M)	Borrowed (\$M)	Borrowed percentage
Sonne	85	56	65.90%
Compound v2	1770	539	30.50%
Aave v2 Ethereum	5130	1670	32.60%
Aave v3 Optimism	94	35	37.20%
Venus	1219	430	35.20%
Tectonic	194	69	35.60%
Benqi	151	36	23.80%
Flux	60	23	38.00%



A final data point that would be helpful to understand the risk for the Sonne protocol is a dashboard that shows liquidations or bad debt in Sonne. Sonne is not listed in the [Risk DAO bad debt dashboard](#), but working with this team to add Sonne would be a good step forward. Creating a Dune dashboard for liquidations and accounts at risk of liquidation would also help. The very high borrowing on Sonne relative to underlying collateral indicates that users are taking higher risk positions in the protocol than other lending protocols. Monitoring should be in place to understand if bad debt is accumulating, as it did in Aave in November 2022, so that governance action can be taken to adjust the protocol parameters as needed.

The combination of the above factors and the lack of any borrow cap means that the risk of a bank run on Sonne is higher than protocols with less risky parameters. There are at least two different ways that a bank run could happen. One is the case where a market is at a high utilization rate, say near 90% utilization like the soOP market currently is. This means that if a whale who has deposited into the soOP pool plans to withdraw 10% of the total supplied assets, the soOP market will reach 100% utilization ratio. When a market is at 100% utilization ratio, there are no more assets that can be withdrawn from the market, because the market physically does not hold any of these assets. This means that some users will not be able to withdraw their assets in this case. The second scenario where a bank run could happen is if bad debt accumulates in the market. The early users to

withdraw will be able to withdraw their funds without any problem, but the late depositors will not. The result is that the late or last users to withdraw their assets will receive 100% of the impact of bad debt accumulation.

Impact

High. Current choices for collateral parameters increase the risk of bad debt accumulation compared to other lending protocols. Sonne has not performed any risk analysis or started monitoring bad debt to inform adjustments to current risk parameters. Bank run risk on specific markets during adverse market conditions is higher than many other lending protocols.

Recommendation

First, the risk profile of the protocol should be clearly described to users. If the only information communicated to users is that Sonne is a Compound fork but Sonne has a much higher risk in certain market conditions, users may not understand that a protocol that is a Compound fork does not have the same risk profile as Compound Finance.

Sonne must perform a detailed risk assessment to account for market conditions. Lower collateral factors and non-zero borrow caps should be implemented accordingly. Because lowering collateral factors can cause some accounts to immediately become liquidated, sufficient public messaging must happen before the collateral factors are changed so that borrowers can adjust their positions before the new values go live on-chain. This is [how Compound Finance approaches such changes](#). Bad debt and liquidations should be monitored in a public or private dashboard, ideally on [the Risk DAO dashboard](#) alongside other major lending protocols, to provide an indication of the protocol's health level. Consider implementing an API [like Compound does](#) to make it easier to query for accounts that are unhealthy.

Developer Response

We acknowledge the concerns raised by the audit team regarding our market parameters and their potential risk profile. Our primary goal at Sonne Finance is to ensure the security and longevity of our platform, while offering competitive advantages that foster growth. We value the importance of an extensive risk modeling exercise, and acknowledge that we can and should do better in this aspect. Here is our plan to address these issues:

- 1 Risk Parameters Explanation: The parameters were initially set to encourage early adoption and usage of the platform, as our protocol is relatively new compared to other

established players like Compound and Aave. We understand the need for more transparency and plan to provide detailed explanations for each parameter setting, including collateral factor and reserve factor, and the risk/reward rationale behind them. This will be published on our website and communicated to the community.

- 2 Risk Modelling: We are working on detailed risk assessments for each asset in our markets, including scenario-based stress tests under different market conditions.
- 3 Parameter Adjustments: Based on the results of the risk modelling exercise, we will propose adjustments to our collateral and reserve factors to better align with industry standards and to ensure the protocol's risk profile is better managed. Any changes will be introduced gradually and communicated well in advance to our users to avoid unexpected liquidations.
- 4 Monitoring and Alerts: We will develop a public dashboard that provides a real-time view of key risk metrics, such as the proportion of borrowed assets and accounts close to liquidation. We're also considering an alert system that will notify users when their collateral is close to being liquidated due to changes in the market.
- 5 Risk DAO integration: Sonne Finance is already on Risk Dao dashboard. Special thanks to Spalen. <https://bad-debt.riskdao.org/>
- 6 Bank run risk mitigation: To mitigate the risk of a bank run, we'll introduce mechanisms such as borrow caps and more conservative utilization thresholds as advised. This will ensure better protection of our users' funds under adverse market conditions.
- 7 Community Engagement: We understand the importance of transparency and communication and it has always been one of the strongest suits of ours. Sonne Finance community will be informed on every step we take forward.

2. High - EOA admins control staking rewards

The transfer of reward tokens from Velodrome to the staking contracts is performed by an intermediate EOA address. This EOA controls all the staked token value and the Velodrome rewards. Because the movement of rewards is not governed by a smart contract, there are no guarantees that this EOA will always act as described in the Sonne documentation. Trusting an EOA in a DeFi protocol increases the risk of a potential rug and means there is a single point of failure if a private key compromise happens.

Technical Details

The address 0xFb59Ce8986943163F14C590755b29dB2998F2322 is the owner of uSONNE and sSONNE contracts and it is an EOA. This can be confirmed by opening etherscan or querying the contracts directly: `cast call`

```
0x41279e29586eb20f9a4f65e031af09fced171166 "owner()(address)" --rpc-url  
https://mainnet.optimism.io cast call 0xdc05d85069dc4aba65954008ff99f2d73ff12618  
"owner()(address)" --rpc-url https://mainnet.optimism.io
```

This same EOA that is the owner of uSONNE and sSONNE is the admin address for [Sonne LiquidityGenerator](#), which was out of scope of this review. Another role of this EOA is to deploy contracts, including the proxy and logic contract for the Unitroller in the Sonne lending protocol.

Another EOA address, 0x201ECB1C439F92aFd5df5d399e195F73b01bB0F3, plays a key role in manually transferring rewards from Velodrome to the uSONNE and sSONNE contracts by calling `addReward()`. The same EOA is stored as `reservesManager` in Sonne LiquidityGenerator, which was a contract outside the scope of this review. `cast call`

```
0x17063Ad4e83B0aBA4ca0F3fC3a9794E807A00ED7 "reservesManager()(address)" --rpc-url  
https://mainnet.optimism.io
```

EOAs should not play key roles in value transfer operations in a DeFi protocol. There is no way to trust that the EOA is going to act as expected, and without context there is no way to determine that this EOA will not end up rugging depositors of their rewards. Fortunately, in the case of uSONNE and sSONNE, the

0x201ECB1C439F92aFd5df5d399e195F73b01bB0F3 EOA does not have access to the underlying SONNE tokens deposited into the staking contracts, and only has control over future rewards that SONNE generates. But using an EOA for key operations increases the risk of loss of funds due to private key exposure. A multisig would mitigate the risk of this single point of failure.

Impact

High. An EOA is trusted with key operations in the staking protocol, so there is no way to verify the EOA will act as advertised.

Recommendation

At a minimum, replace the two EOAs identified with core roles in this protocol with 2-of-3 multisigs to avoid a single point of failure if private keys are compromised. A better long-

term solution would be to replace these EOAs with smart contracts that have clearly defined functions that can be called at certain times. If there is a need to change certain state variables, the setter functions could be behind timelocks to increase the trust that users have that the protocol is not going to rug them.

Developer Response

Thank you for bringing up this concern about the use of EOAs within the protocol. We understand the risk it presents and agree that changes are needed to enhance the trust, security, and decentralization of our platform. “Reduce reserves” action is done by 3-of-5 multisig wallet, and EOA is only be available to act after this, but we understand the concern and will act according to that.

In the longer term, we plan to replace these multisigs with smart contracts that will govern key operations. These contracts will have clearly defined functions and checks to avoid any misuse.

3. High - EOA swapped staked token rewards to ETH for gas

The EOA that plays a key role in distributing rewards to Sonne stakers swapped some reward tokens for ETH to pay for the EOA gas. This process is not documented anywhere and could, depending on the user’s perspective, be viewed as theft of rewards.

Technical Details

The EOA 0x201ECB1C439F92aFd5df5d399e195F73b01bB0F3 plays a key role in manually transferring rewards from Velodrome to the uSONNE and sSONNE contracts by calling `addReward()`. This same EOA [converted some rewards to ETH](#) to pay for the gas consumed in the EOA transactions. This process of taking some staking rewards for ETH is not documented anywhere. In fact, this process contradicts the [staking documentation which states that users will get 100% of staking rewards](#):

Stakers will get 80% of the protocol revenue and 80% of VELO rewards for the first 3 months. After team tokens start to get unlocked, stakers will start to get 100% of the protocol revenue.

Because the EOA is taking some of these rewards for gas, users are not getting 100% of staking rewards. In fact, there is no explanation for how many reward tokens are spent on gas, so users cannot know what percentage of staking rewards they receive.

At the time of this review, 27% of total SONNE is staked in uSONNE or sSONNE contracts, which means over 25% of all SONNE tokens are indirectly impacted by this finding.

Impact

High. SONNE stakers are receiving less value than expected.

Recommendation

There are many ways to improve the protocol from the current approach. At a minimum, it the current process will continue in the future, make it clear to users how much of the staking rewards will be converted to gas so they can better calculate the rewards that stakers will receive.

A better approach would be to provide stakers with 100% of the rewards as promised, and transfer ETH to the EOA as needed instead of taking reward tokens from users to pay for gas.

The best approach would be to replace the EOA with a smart contract to increase user trust and make the process of swapping and returning rewards more transparent.

Developer Response

We acknowledge that the use of staking rewards to cover gas costs by the EOA was not explicitly mentioned in our documentation, and we understand the importance of full transparency in DeFi.

We would like to clarify that the amount of rewards used for gas is indeed minute compared to the total rewards distributed. However, we recognize that our communication on this matter should have been better, and we regret any confusion or concern this may have caused our users.

Our intent was to ensure the efficient operation of the protocol and the timely distribution of rewards. Given the nature of Ethereum transactions, the requirement for gas is an operational necessity. The decision to use a small portion of the rewards was made with the aim of maintaining a streamlined distribution process.

In hindsight, we realize that this should have been more clearly communicated. We are committed to rectifying this, and will make sure this information is added to our documentation.

We understand the auditor's perspective, even though we don't agree with it being high impact. But thanks for auditing even our oldest transactions. It will ensure trust in the audit more.

4. High - Unclear protection against Hundred Finance attack vector

The Hundred Finance hack was primarily caused by an empty market, which allowed for share price manipulation and resulted in the draining of the protocol's funds. Sonne does not appear to have any mitigations in place to prevent this attack when a new market is deployed and was likely vulnerable to this attack in the past.

Technical Details

The Hundred Finance hack was [largely caused by an empty WBTC market](#). The other requirements for this attack vector are that the market has a non-zero collateral factor (to allow borrowing against this token as collateral) and a totalSupply of the cToken of zero. The result of this is a scenario similar to ERC4626 share price manipulation for the first deposit, with a large donation to skew share price. There is no evidence that Sonne has a clear and consistent mitigation to this attack vector.

One of the more recently created Sonne markets, sowstETH, had a 1 day period between the creation of the market [on February 19](#) and the first deposit [on February 20](#). The same pattern is seen in the soLUSD and soWBTC markets, with the soLUSD market created [on February 2](#) and the first deposit [on February 3](#) and the soWBTC market created [on December 3](#) and the first deposit [on December 4](#). The markets were initialized with a non-zero collateral factor, but the collateral factor was raised above 0 before the first deposit in all three of these cases, meaning there was a point in time when the protocol was likely vulnerable to the same attack as Hundred Finance. For example:

- 1 soWBTC market is deployed at [0x33865E09A572d4F1CC4d75Afc9ABcc5D3d4d867D](#).
- 2 soWBTC market was deployed [in block 45086893](#). Collateral factor in this block and soon after is zero, which is good, because assets deposited immediately cannot be borrowed against.

```
cast call 0x60CF091cD3f50420d50fD7f707414d0DF4751C58 "markets(address)(bool,uint256,bool)" 0x33865E09A572d4F1CC4d75Afc9ABcc5D3d4d867D --block 45086893 --rpc-url https://mainnet.optimism.io
```
- 3 First deposit into soWBTC happened [in block 45448745](#). But in the block before it, we can see the collateralization factor was already non-zero and was set to 70%:

```
cast call 0x60CF091cD3f50420d50fD7f707414d0DF4751C58 "markets(address)(bool,uint256,bool)"
```

```
0x33865E09A572d4F1CC4d75Afc9ABcc5D3d4d867D --block 45448744 --rpc-url
```

<https://mainnet.optimism.io> For further confirmation that this is the correct block to examine, the blockchain data confirms that the totalSupply of soWBTC in the block before the first deposit was zero, and then was non-zero in the following block. Before:

```
cast call 0x33865E09A572d4F1CC4d75Afc9ABcc5D3d4d867D "totalSupply()" --block 45448744
```

```
--rpc-url https://mainnet.optimism.io After: cast call
```

```
0x33865E09A572d4F1CC4d75Afc9ABcc5D3d4d867D "totalSupply()" --block 45448745 --rpc-url
```

<https://mainnet.optimism.io> In fact, after doing a search to find when the collateralization factor changed, it is revealed that the collateralization factor was changed in block 45448654. The before and after can be compared with: Before: `cast`

```
call 0x60CF091cD3f50420d50fD7f707414d0DF4751C58 "markets(address)(bool,uint256,bool)"
```

```
0x33865E09A572d4F1CC4d75Afc9ABcc5D3d4d867D --rpc-url https://mainnet.optimism.io
```

```
--block 45448653 After: cast call 0x60CF091cD3f50420d50fD7f707414d0DF4751C58
```

```
"markets(address)(bool,uint256,bool)" 0x33865E09A572d4F1CC4d75Afc9ABcc5D3d4d867D
```

```
--rpc-url https://mainnet.optimism.io --block 45448654
```

There is no script in [the deploy directory](#) for deploying a new market, which is another data point that there is no consistent documented process to deploying new markets in a way that mitigates this attack vector.

Impact

High. While the protocol on mainnet is not actively vulnerable to this issue, there are past instances where an attack following the Hundred Finance pattern was likely possible.

Recommendation

Consider implementing a fix in the contracts themselves to prevent this attack vector, borrowing ideas from [this OpenZeppelin discussion](#) on the related ERC4626 inflation attack vector. The team reported that they are holding soTokens, but they should consider an approach such as burning an initial amount of tokens to prevent unintentionally allowing a critical protocol bug to be attacked.

Developer Response

Regarding the specific attack vector you highlighted, we acknowledge that an empty market, coupled with a non-zero collateral factor and a totalSupply of zero for the corresponding soToken, could lead to share price manipulation and potential draining of funds.

Our approach will involve a multi-step workflow that effectively prevents this attack vector without requiring an upgrade to the existing smart contracts. The steps we have implemented are as follows:

- 1 Addition of the market to the comptroller with a zero collateral factor: Before any borrowing or lending activity can take place, we add the market to the comptroller with a collateral factor of zero.
- 2 Minting and burning of a small amount of soTokens: To eliminate the possibility of an empty market, we mint a small amount of soTokens and subsequently burn them.
- 3 Setting the collateral factor for the market: Once the previous steps have been completed, we proceed to set an appropriate collateral factor for the market.

soToken burn transactions:

- 1 [soDAI](#)
- 2 [soLUSD](#)
- 3 [soOP](#)
- 4 [soSUSD](#)
- 5 [soUSDC](#)
- 6 [soUSDT](#)
- 7 [soWBTC](#)
- 8 [soWETH](#)
- 9 [sowstETH](#)
- 10 [soSNX](#)

Medium Findings

1. Medium - No sequencer uptime check before querying Chainlink data on L2

If the Optimism sequencer is down, the L2 rollup can only be accessed through L1 optimistic rollup contracts.

Technical Details

If a transaction is created on an L2 rollup while the sequencer is down, the transaction [would be added to a queue](#). Because the transaction timestamp is the time when the transaction was queued, by the time the sequencer comes back online to process the queue, the price data could be outdated.

Impact

Medium. If the Optimism sequencer is down, old price data may be used.

Recommendation

Modify `_getLatestPrice()` to use [the Chainlink example code](#) to check sequencer uptime and revert if the sequencer is down.

Developer Response

We acknowledged the issue. We will do necessary research and implement the suggested solution.

2. Medium - Poor choice of interest rate models in Sonne

Compound cTokens each have a unique interest rate model contract to set the slope and kink of the lending and borrowing curve. Sonne uses the same interest rate contract across many soTokens. This means that the incentives for individual soTokens cannot be changed independently with the current protocol setup, which can cause problems when market conditions require protocol updates.

Technical Details

Every cToken has a separate interest rate model contract. This is true even when the cTokens have the exact same interest rate borrowing and lending curves. Some examples are shown below.

cToken	Interest Rate Model Contract
cUSDC	0xD8EC56013EA119E7181d231E5048f90fBbe753c0
cDAI	0xFB564da37B41b2F6B6EDcc3e56FbF523bD9F2012
cETH	0xF9583618169920c544Ec89795a346F487cB5a227

In contrast, Sonne is using the same interest rate model contract for many different tokens.

The results for all soTokens are shown below. The interest rate model curve for WETH and OP is the same as the interest rate model for the stablecoins that aim to maintain a peg to USD.

soToken	Interest Rate Model Contract
soWETH	0xbbbd75383f6A61d5EB5b43e94E6372Df6F7f13c6
soDAI	0xbbbd75383f6A61d5EB5b43e94E6372Df6F7f13c6
soUSDC	0xbbbd75383f6A61d5EB5b43e94E6372Df6F7f13c6
soUSDT	0xbbbd75383f6A61d5EB5b43e94E6372Df6F7f13c6
soOP	0xbbbd75383f6A61d5EB5b43e94E6372Df6F7f13c6
soSUSD	0xbbbd75383f6A61d5EB5b43e94E6372Df6F7f13c6
soLUSD	0xbbbd75383f6A61d5EB5b43e94E6372Df6F7f13c6
soWBTC	0x3F6fB832279AC7db0B4F92b79cBB8Df03702631e
sowstETH	0x3F6fB832279AC7db0B4F92b79cBB8Df03702631e
soSNX	0x7320bD5fA56F8a7Ea959a425F0C0b8cAc56F741E

One of the side effects of the Sonne interest rate model choices is that the rate curve is less steep for tokens with more volatile prices. In order to minimize the risk of bad debt, there should be incentives to avoid excessive borrowing of risky assets. Specifically, WETH and OP are using the same interest rate model curve as stablecoins that should remain pegged to USD. Using a curve with less slope for volatile assets means that borrowing that asset is less expensive, and making it cheaper to borrow volatile assets increases the risk of bad debt accumulating. The table below compares Compound cToken parameters to Sonne soTokens.

Token	multiplierPerSecond	jumpMultiplierPerSecond	baseRatePerSecond
cUSDC	1585489599	34563673262	0

Token	multiplierPerSecond	jumpMultiplierPerSecond	baseRatePerSecond
soUSDC	1981861998	43283866057	0
cETH	7134703196	1553779807204	634195839
soWETH	1981861998	43283866057	0
cwBTC	7134703196	31709791983	634195839
sowBTC	8918378995	39637239979	634195839

Not surprisingly, the largest discrepancy is in WETH, because as mentioned above, soWETH is using the same slope parameters as stablecoins. The soWETH `multiplierPerSecond` slope is 27.7% of the slope value that Compound uses, and the `jumpMultiplierPerSecond` slope is only 2.8% of the value chosen by Compound. Or for a different comparison, the cWETH slope after the kink is 218 times the slope before the kink. For soWETH, the slope only increases 22 times after the kink. This means that the Compound Finance market is providing a strong disincentive to borrow beyond the kink for the WETH market than Sonne, most likely to reduce the risk of bad debt. If Compound v2 had a market for OP tokens, the same result would almost certainly exist because soOP is using the same interest rate model as stablecoins, just like soWETH.

Finally, while it is true that `_setInterestRateModel()` in the cToken contract can be used to set a new interest rate model contract, the admin that is able to trigger this change is the [Timelock Controller at 0x37ff10390f22fabdc2137e428a6e6965960d60b6](#), which means that a minimum 48 hour delay must take place before the change can be implemented.

Impact

Medium. Deployed interest rate model configurations for soTokens increase the protocol risk. Configuration values do not match Compound Finance.

Recommendation

Regular risk assessments should consider market conditions and adjust protocol incentives accordingly, including the interest rate model. Use the same contract deployment configuration as Compound and deploy a separate interest rate model contract for every soToken. Analyze the risk and incentive strategy to target based on the

protocol's goals to choose the proper multiplier (slope before the kink point) and jump rate multiplier (slope after the kink point) for Sonne.

Developer Response

We updated the interest rate model for soOP to use the same interest rate model as soWBTC on [this transactions](#).

We will continue with the common interest rate model for the markets which use common parameters. In future, we will consider to separate the interest rate model and parameters for each market.

3. Medium - Sonne interest rate model math error

Sonne attempts to imitate Compound Finance for some math, but does so incorrectly.

Technical Details

The only non-stablecoin asset in Sonne that is found in Compound Finance with an interest rate model that is almost the same is WBTC. However, Sonne does not properly imitate the WBTC curve from Compound Finance. Consider these example values for a utilization value of 80%.

- cash: 2000000000000000
- borrows: 8000000000000000
- reserves: 0
- reserveMantissa: 2000000000000000000

Entering these values into `getBorrowRate` and `getSupplyRate` in etherscan for interest rate contract defined in [cWBTC](#), the resulting values are 95129375951 and 60882800608 respectively. After multiplying this rate-per-block by the blocks-per-year value of 2628000, we find the borrow rate is 25% and the supply rate is 16%. Meanwhile, entering these values into etherscan for soWBTC returns 7768899035 for `getBorrowRate` and 4972095382 for `getSupplyRate`. After multiplying this rate-per-second by the seconds-per-year value of 31536000, we find the borrow rate is 24.5% and the supply rate is 15.7%. This error may be partially caused by the `baseRatePerBlock` value chosen for soWBTC. The Compound value for `baseRatePerBlock` is 9512937595 and the soWBTC value is $9512937595 / 15 = 634195839$. However, a soWBTC value assuming 12 second blocks instead of 15 second blocks may improve alignment to Compound Finance's cWBTC curve.

Be aware that the Compound UI is inaccurate for the cWBTC curve as detailed in [this open issue for the Compound Finance frontend](#). When hovering over the 80% utilization in the Compound Finance UI, the UI suggests a borrow rate of 24.5% and a supply rate of 18.13%. Therefore, the UI curves do not match the on-chain cWBTC deployment.

Additionally, Sonne uses the same value for constant `borrowRateMaxMantissa` as [Compound](#) but in the comment it is stated that borrow rate value is applied per block. This means that the value should be 12 times lower on Optimism. Another Compound fork on Optimism, Hundred Finance, uses the correct value `0.00004e16`.

Impact

Medium. Configuration values try to match Compound Finance but do not.

Recommendation

`borrowRateMaxMantissa` should be set to the proper value instead of the current value that is 12x higher than what Compound Finance uses. If Sonne intends to borrow Compound Finance's interest rate curves, the resulting on-chain implementation should match Compound Finance's on-chain values.

Developer Response

We tried hard to match the Compound Finance interest rate model, but it is not exactly possible because Compound Finance uses blocks for calculations and Sonne uses seconds. We do not try to match with the on-chain contracts but UI values.

`borrowRateMaxMantissa` is hard-coded in the contract and cannot be changed. We can only consider to change the value for the future markets.

4. Medium - No Borrow Caps set

All soTokens have a zero `borrowCap` value. Compound made significant adjustments to their borrow cap values after the November 2022 CRV short selling impacted Aave.

Technical Details

In November 2022, a widely publicized CRV short selling event aimed at liquidating certain Aave borrowers. Compound Finance took action after this event and made significant changes to their borrow cap approach (1, 2). The result is that Compound Finance has a borrow cap set on every market except for USDC, DAI, and USDT. The borrow caps that are set on markets are set to values that are less than the assets supplied. This means that

even though the interest rate curve for borrowing could work up to 100% supplied assets, the borrow cap keeps the maximum amount that can be borrowed significantly lower. Aave also has non-zero borrow caps, although Aave v3 on mainnet Ethereum has higher borrow caps for many assets relative to the amount deposited.

Although [the explanation in the proposal](#) for reducing the borrow caps was “insolvency risk from liquidation cascades” and “risk of high utilization”, setting proper borrow cap values may also provide some protection in cases like the [Venus protocol LUNA fallout](#) (Venus is a Compound fork), because a borrow cap below the value of assets deposited to the protocol would prevent all the assets from the protocol from being borrowed if a Chainlink oracle failed.

Impact

Medium. Borrow cap values are not used at all, which increases protocol risk.

Recommendation

Evaluate which borrow cap values make sense for different pools to allow Sonne to safely navigate unexpected conditions in turbulent market conditions.

Developer Response

Borrow caps were set for soOP and soSNX markets at [this transaction](#) according to suggestion. We will consider to set borrow caps for other markets in the future.

5. Medium - Reserves stored in Sonne adds currency risk

The reserves value in Compound Finance is stored in the underlying asset of each cToken and remains in the cToken contract. In comparison, Sonne withdraws the underlying reserve value, converts it into SONNE tokens, and deposits the SONNE into the sSONNE staking contract. Because the reserves are intended to help pay for any bad debt that accumulates in the protocol and the debt is denominated in the underlying token of each soToken market, swapping the reserves into SONNE introduces currency risk. If SONNE accumulates bad debt denominated in a stablecoin like USDC and the SONNE token value drops at the same time, perhaps because speculators have observed the protocol is accumulating bad debt, then the value of the SONNE reserves will drop. This means that SONNE does not necessarily have the amount of token reserves they expect to have because the reserves are denominated in a different token, and specifically a token that is inflationary (due to high rewards distributions) and highly correlated to the success of the

SONNE protocol.

Technical Details

When comparing Sonne and Compound Finance on-chain data, it is obvious that there is a large difference in the amount of reserves stored in each protocol. In Compound's cUSDC market, [the value of totalReserves is over \\$13 million](#). These tokens are owned and stored by the cUSDC contract. In comparison, the Sonne soUSDC market has [a totalReserves value of under \\$1000](#). The reason that soUSDC has such a low totalReserves value is because the reserves are periodically withdrawn by the admin in transactions such as [0xed3ee0eb900779a6c82d474fc12697bb6cc55372960c775a22c42e4931a7d922](#). The value flow of the reserves in Sonne is as follows:

- 1 Reserve tokens accumulate in soToken market contract
- 2 Reserve tokens are withdrawn by EOA admin
- 3 EOA admin swaps reserve tokens for USDC using Velodrome pool (potentially has 2% slippage)
- 4 EOA admin swaps USDC for SONNE tokens (potentially 2% slippage, plus the market price is moved because the USDC/Sonne pool has low liquidity)
- 5 EOA deposits SONNE tokens into sSONNE, where it sits in the sSONNE contract and collect Velodrome rewards

At a minimum, storing the value of the reserves tokens in SONNE introduces [currency risk](#). If bad debt accumulates in a token that is not correlated or pegged to SONNE token value, there is a risk that the value of the reserves could drop in USD terms while the bad debt amount increases in USD terms, making it impossible for the Sonne multisig or admins to pay off the debt. For a specific example, if Sonne collects \$1000 in bad debt denominated in USDC and has \$5000 of reserves value that was converted to SONNE, then if the SONNE token drops in price more than 80% compared to when the USDC was converted to SONNE, the protocol will not have enough reserves to pay off the bad debt.

It is [generally a good practice for DAOs or protocols to diversify their treasury holdings](#). If Sonne holds the original underlying assets instead of converting these assets to Sonne, it demonstrates that Sonne holds real value uncorrelated to the SONNE token. The current approach converts these underlying into SONNE with the goal of propping up the SONNE token price by funding the Velodrome USDC/SONNE market, but it reduces the actual value owned by SONNE in non-SONNE tokens.

A secondary risk that is introduced with the swapping of reserve funds is the total losses due to the swaps. Sonne may be losing 5% of total reserve value by swapping the underlying reserve tokens into SONNE because the default slippage on Velodrome is 2%. Two swaps in Velodrome means around 4% of value can be lost, and as highlighted in a separate low findings, the SONNE/USDC market has low liquidity and large swaps can move the market price. If SONNE tokens need to be converted back to other tokens to pay off debt, then the overall losses due to slippage could be 7% or higher. This means that the reserves collected by the protocol over time does not match the actual value that could pay off bad debt.

Impact

Medium. Currency risk related to storing the protocol reserves in SONNE increases the chances that Sonne will not be able to pay back bad debt if bad debt accumulates in the protocol. Substantial reserves value can be lost due to swapping reserves.

Recommendation

Sonne should not convert the reserve tokens into SONNE due to the currency risks, treasury diversification, and value loss on swaps. The process used by Compound or Aave should be applied, where the reserves remain in the soToken market in the underlying token, which is the same currency used for borrowing from the market.

Developer Response

Within our protocol design, we anticipate a low occurrence of bad debt requiring repayment from the reserves. This led us to strategically deploy these reserves in the sSONNE market. In addition, we've implemented a feature in our uSonne staking that lets users choose to receive rewards in USDC, not just SONNE tokens. This adds a level of flexibility for our users, allowing them to make decisions that best fit their individual needs. While we're aware of the risks involved, we firmly believe that these choices should yield better results for SONNE holders in the long run.

6. Medium - No Chainlink staleness check in `_getLatestPrice()`

`_getLatestPrice()` retrieves price data from Chainlink, but there are no checks to discard data if the oracle returns stale data.

Technical Details

The Chainlink `latestRoundData()` function returns price data along with the roundId and timestamp of the data. If the data is stale, it should be discarded. Otherwise the protocol

will trust outdated data that could lead to a loss of value from using an inaccurate exchange rate. [Chainlink docs recommend to check the roundId and timestamp values](#) that the oracle returns, as shown in other security report findings [here](#) and [here](#).

Impact

Medium. The Chainlink oracle data should be checked for staleness.

Recommendation

Consider modifying `_getLatestPrice()` to the following. Be aware that a different `ORACLE_STALENESS_THRESHOLD` value should be used for different tokens depending on the frequency of price updates for the specific oracle feed:

```
function _getLatestPrice(string memory symbol)
    internal
    view
    returns (uint256, uint256)
{
    IAggregatorV3 oracleAddress = priceFeeds[symbol];
    require(address(oracleAddress) != address(0), "missing priceFeed");

    (
        uint80 roundId,
        int256 price, //uint256 startedAt
        ,
        uint256 timeStamp,
        uint80 answeredInRound
    ) = oracleAddress.latestRoundData();

    if (answeredInRound <= roundId && block.timestamp - timeStamp >
    ORACLE_STALENESS_THRESHOLD) revert InvalidPrice(feedValue);

    IChainlinkAggregator aggregator =
    IChainlinkAggregator(IAggregatorV3(oracleAddress).aggregator());

    require(price > int256(aggregator.minAnswer()) && price <
    int256(aggregator.maxAnswer()), "price cannot be zero");
    uint256 uPrice = uint256(price);
```



```
        return (uPrice, timeStamp);  
    }
```

Developer Response

We will add this suggestion into our price oracle contract in the next release.

Low Findings

1. Low - User can accidentally postpone staking release time

The first step for a user to unstake their staked SONNE tokens is to call `burn()`. But if a user has already unstaked their tokens and is waiting for the release time to arrive, calling `burn()` with a zero amount will restart the release time countdown process so the user must wait another week. There is no reason for a user to make such a call. Moving all the `burn()` logic into the if statement will avoid this case and save gas in the case where a user calls `burn()` with a zero value.

Technical Details

If `burn()` is called with amount zero, no state variable changes should be made. Moving all state variable changes inside the if statement will make sure that less gas is spent on the zero amount case, and will also prevent a user from accidentally postponing their release time.

```
function burn(uint256 amount) public {  
    if (amount > 0) {  
        _burn(msg.sender, amount);  
        Withdrawal storage withdrawal_ = withdrawal[msg.sender];  
        withdrawal_.amount = withdrawal_.amount + amount;  
        withdrawal_.releaseTime = block.timestamp +  
        withdrawalPendingTime;  
    }  
}
```

Impact

Low. There is no need to introduce unexpected code paths in edge cases so the code

should be changed to avoid this from happening.

Recommendation

Modify `burn()` to the implementation shared above.

Developer Response

Staking contracts are not upgradable. We will consider this suggestion in the next staking contract release.

2. Low - Small SONNE/USDC liquidity leaves potential for large price shifts

The SONNE/USDC Velodrome liquidity pool can experience large changes in price from a single swap. Because this SONNE/USDC Velodrome pool is one of the largest holders of SONNE, swaps in this pool can effectively shift the market price for SONNE. This may have unintended consequences and swaps in this pool may become a target for MEV bots.


Technical Details

Let us examine a specific example of the price change before and after [this tx](#) involving the EOA that performs swaps of the Sonne staking rewards. Approximately \$14500 of USDC was swapped for SONNE.

- 1 Check the price of SONNE in the pool before the swap (block 94817489) `cast call`
`0xc899c4d73ed8df2ead1543ab915888b0bf7d57a2 "getAmountOut(uint256,address)(uint256)"`
`"1000000000000000000000" "0x1DB2466d9F5e10D7090E7152B68d62703a2245F0" --block 94817489`
`--rpc-url OPTIMISM_RPC` 1 SONNE = 1516372 USDC
- 2 Check the price of SONNE in the pool after the swap (block 94817490) `cast call`
`0xc899c4d73ed8df2ead1543ab915888b0bf7d57a2 "getAmountOut(uint256,address)(uint256)"`
`"1000000000000000000000" "0x1DB2466d9F5e10D7090E7152B68d62703a2245F0" --block 94817490`
`--rpc-url OPTIMISM_RPC` 1 SONNE = 1561887 USD
- 3 Calculate the price change from this swap $(1561887 - 1516372) / 1561887 \sim 3\%$


This price impact is even displayed in the Velodrome UI when it exceeds a certain threshold.


Balance: 3,582.29



3000

OP





57482.03434628

SONNE

Balance: 0.00

Price Info

0.05

OP per SONNE


19.16

SONNE per OP


Slippage

2 %


Route



Volatile



Volatile



Price Impact: 1.10%

Swap

Impact

Low. Consider the risks involved with an EOA performing swaps that move the market on the protocol's token.

Recommendation

Consider swapping smaller amounts of tokens to avoid moving the market by a significant amount in a single transaction. Even if the exact timing of the swap is irregular, there may be ways for opportunistic just-in-time short-term token holders to profit from the current approach.

Developer Response

Acknowledged. We started to swap smaller amounts in random intervals.

3. Low - Incorrect totalSupply in Comp for on-chain voting

The Compound COMP token is replaced in Sonne with the SONNE token. One difference between COMP and SONNE is the totalSupply value. Sonne did not change this value in one place where the Compound totalSupply was hardcoded into Comp.sol for on-chain voting.

Technical Details

Excluding the 18 decimals of the SONNE token, 100 million SONNE are minted when the token is deployed. Contrast this to only 10 million COMP total supply value. The values can be queried with etherscan:

- COMP totalSupply: <https://etherscan.io/token/0xc00e94cb662c3520282e6f5717214004a7f26888#readContract#F14>
- SONNE totalSupply: <https://optimistic.etherscan.io/address/0x1DB2466d9F5e10D7090E7152B68d62703a2245F0#readContract#F6>

The 10 million total supply value [is hardcoded in Comp.sol](#) and was not updated to the 100 million value that should be used in Sonne. This means that on-chain voting could be problematic if Sonne uses Comp.sol for this reason. The Sonne.sol contract clearly shows [a total supply of 100 million Sonne](#).

Impact

Low. On-chain voting is not going to be used, but if it were, the mismatched numbers could cause issues.

Recommendation

The Sonne Comptroller is deployed behind a proxy, so it can be upgraded to fix this issue in Comp.sol.

Developer Response

We add this to our backlog. We will consider remove Comp.sol usage and use EIP20Interface instead in the next release of Comptroller.

4. Low - SUSD and LUSD Chainlink price feeds are not standardized verified feeds






The SUSD and LUSD Chainlink data feeds are monitored feeds, not a verified feeds, which means they carry additional risk.

Technical Details

The lowest risk and highest quality tier for Chainlink oracles are verified feeds. Monitored Chainlink feeds are the second highest quality tier of oracles, but they carry additional risk and are still under review. Because a [common weakness for Compound forks](#) is oracle manipulation leading to the draining of many markets, the Sonne protocol is only as robust as its weakest oracle. Using Chainlink oracles that introduce extra risk is problematic. Screenshots of the Chainlink documentation at the time of the review is below.

Chainlink documentation about data feed quality

Data feed categories

-  **Verified Feeds** : Feeds that follow a standardized data feeds workflow
-  **Monitored Feeds** : Feeds under review by the Chainlink Labs team to support the stability of the broader ecosystem
-  **Custom Feeds** : Feeds built to serve a specific use case and might not be suitable for general use
-  **Specialized Feeds** : Purpose-built feeds that might rely on contracts maintained by external entities and require in-depth understanding of composition methodology before use
-  **Deprecating** : These feeds are scheduled for deprecation. See the [Deprecation](#) page to learn more.

See the [Selecting Quality Data Feeds](#) page for complete details about each category.











Monitored Feeds

Feeds under the monitored category are *under review* by the Chainlink Labs team to support the stability of the broader ecosystem. While generally resilient and distributed, these feeds carry additional risk.











Data feeds might be under review for the following reasons:

- The token project or asset is in early development
- The project is going through a market event such as a token or liquidity migration
- The token or project is being deprecated in the market
- The asset has a high spread between data providers or low liquidity in the market

The [LUSD data feed](#) is a monitored feed

	LOOKS / USD	LooksRare	Crypto	0.5%	86400s	8		0xd682c5f1A8eaA2385
	LTC / USD	Litecoin	Crypto	0.5%	86400s	8		0x45954efBD01f5A124
	LUSD / USD	Liquity USD	Crypto	0.5%	86400s	8		0x9dfc79Aaeb5bb0f96
	MATIC / USD	Polygon	Crypto	0.2%	1200s	8		0x0ded608AFc23724f
	MIMATIC / USD	MAI	Crypto	1%	86400s	8		0x73A3919a69eFCd5b

The [SUSD data feed](#) is a monitored feed

 SOL / USD	Solana	Crypto	0.2%	1200s	8		0xC663315f7aF904fbb
 STETH / USD	Lido Staked Ether	Crypto	0.5%	86400s	8		0x41878779a3885855C
 SUSD / USD	SUSD	Crypto	0.5%	86400s	8		0x7f99817d87baD03ea
 Total Marketcap USD	Total currency market cap	Currency	1%	86400s	8		0x15772F61e4cDC81c7
 UNI / USD	Uniswap	Crypto	0.2%	1200s	8		0x11429eE838cC01071

Impact

Low. Risky Chainlink oracles are used in production, which increases the risk to all Sonne depositors.

Recommendation

Review [the Chainlink guide for selecting data feeds](#). Only create new markets for tokens with standard verified price oracles to minimize risk to the protocol.

Developer Response

We will consider this suggestion in upcoming market creations. We will add markets with verified feeds first.

5. Low - Functions calls to uninitialized address

[Comptroller](#) is calling functions on an uninitialized address on Optimism.

Technical Details

Because the address is not initialized, the attacker could deploy a harmful contract with the same function calls but with harmful logic. Also, function calls should be done using a contract interface.

Impact

Low. The owner can easily spot if the address is initialized or not.

Recommendation

First, deploy the wanted contract and then initialize the address as a constant within Comptroller.sol. Use interface for function calls to other contracts.

Developer Response

After this report, we will deploy the ExternalRewardDistributor contract and initialize the address as a constant within Comptroller.sol. Comptroller will be upgraded to use new version.

6. Low - High default slippage may lose value

Velodrome has a default slippage of 2% in the frontend. If the Sonne EOA is using the default frontend to perform swaps, this can reduce the value held by the protocol over time. This loss of value due to slippage can impact rewards distributed to the SONNE stakers in sSONNE and uSONNE, but also impacts the value held by protocol reserves that get deposited into sSONNE and uSONNE.

Technical Details

The velodrome frontend has an unusually high 2% slippage setting. The slippage in AMMs like Uniswap is automatically set in a dynamic way.

The screenshot displays the Velodrome Swap interface. At the top, the Velodrome logo is on the left, and navigation links (Swap, Pools, Vest, Vote, Rewards, Bribe) are in the center. On the right, a user's wallet balance is shown as 0.491 ETH. The main swap area features a large 'Swap' title and a diagram of a swap network. Below the title, it states: 'Take advantage of Velodrome's minimal slippage, low swapping fees, and deep liquidity'. The swap details show 0.1 ETH being swapped for 2028.69195085 SONNE. The slippage is set to 2%. The price info section shows the current rates: < 0.01 ETH per SONNE and 20,286.92 SONNE per ETH. The route is indicated as Volatile. A large blue 'Swap' button is at the bottom.

Asset	Balance
ETH	0.491
SONNE	0.00

Asset	Amount
ETH	0.1
SONNE	2028.69195085

Price Info
< 0.01 ETH per SONNE
20,286.92 SONNE per ETH

Slippage
2 %

Route
Volatile

Based on conversations with the development team, the default slippage has been used in the past. While Optimism is not at risk of MEV right now, a high slippage tolerance can

still lose value if the token weights in the Velodrome pool do not favor the direction of the swap.

Impact

Low. High slippage tolerance may cause loss of value.

Recommendation

Consider calculating the expected slippage of the swap, using an off-chain API or other scripted calculations, to determine the ideal slippage tolerance to apply. Uniswap has an SDK for this process, but it is unclear whether Velodrome has a similar SDK or whether a custom calculator must be made.

Developer Response

We will use calculated or fixed small amount of slippage tolerance for further swaps.

Gas Savings Findings

1. Gas - Remove SafeMath import

The contracts are using solidity version 0.8.X, which includes overflow and underflow protection, making the SafeMath library import unnecessary.

Technical Details

Solidity 0.8.0 introduced a breaking change to [implement overflow and underflow protection](#). The original code from Tarot Finance was using solidity 0.6.6 which did not have this feature.

This means the SafeMath imports can be removed to save gas on deployment for [Distributor.sol](#).

Impact

Gas savings.

Recommendation

Remove SafeMath imports from contracts. Modify the code to use standard arithmetic operators like `*` and `/` instead of `.mul()` and `.div()`.

2. Gas - Cache variable

Save roughly 300 gas by storing `tokens.length` in `claimAll()` in a temporary variable instead of querying this state variable twice.

Technical Details

Obtaining the length of an array consumes gas, so storing this value in a temporary memory variable when the value is needed more than once saves gas, like in `claimAll()`.

Impact

Gas savings.

Recommendation

```
+ uint256 token_length = tokens.length;
- amounts = new uint256[](tokens.length);
+ amounts = new uint256[](token_length);
- for (uint256 i = 1; i < tokens.length; i++) {
+ for (uint256 i = 1; i < token_length; i++) {
```

3. Gas - Replace `totalShares` and `shares[]`

The `totalShares` state variable is unnecessary because it is always equal to `totalSupply`. The same applies to `shares[]`, which duplicates that value stored by `_balances[]`. Because `totalSupply` and `_balances` are provided by default in any compliant ERC20, the duplicate `totalShares` and `shares[]` variables can be removed to save gas.

Technical Details

Etherscan shows that the `totalShares` and `totalSupply` values for `uSONNE` and `sSONNE` are the same. Reasoning about the `totalShares` math in `_editRecipientInternal()` reaches the conclusion that these values should remain the same. `_editRecipientInternal()` increases `totalShares` when `uSONNE` or `sSONNE` is minted, decreases `totalShares` when `uSONNE` or `sSONNE` is burned, and doesn't modify `totalShares` when `uSONNE` or `sSONNE` is transferred between non-zero addresses.

The same applies to `shares[]`. In fact, because `_editRecipientInternal()` is always called with `shares_` set to `balanceOf(account)`, the `shares[]` mapping is duplicating exactly what `_balances[]` already stores.

The only reason that Tarot Finance has a `totalShares` state variable and stores the shares

count in `recipient.shares` (the equivalent of `shares[]`) in its Distributor contract (the inspiration for Sonne's Distributor) is because [the Tarot Finance contract is not an ERC20](#) and does not have a `totalSupply` variable or `_balances` mapping.

Impact

Gas savings.

Recommendation

Remove the `totalShares` state variable and use `totalSupply` instead. Remove the `shares[]` mapping and replace it with `_balances[]`. Make these changes to

`_editRecipientInternal()`:

```
function _editRecipientInternal(address account, uint256 shares_) internal {
    for (uint256 i = 1; i < tokens.length; i++) {
        updateCredit(tokens[i], account);
    }

-    //updateCredit(token, account);
-    uint256 prevShares = shares[account];
-    uint256 _totalShares = shares_ > prevShares
-        ? totalShares.add(shares_ - prevShares)
-        : totalShares.sub(prevShares - shares_);
-    totalShares = _totalShares;
-    shares[account] = shares_;
    emit EditRecipient(account, shares_, _totalShares);
}
```

4. Gas - Remove duplicate line of code

When `claimInternal()` is called, it checks `tokenIndexes[token] > 0` before calling `updateCredit()`. But the first line of `updateCredit()` is to do the same check `tokenIndexes[token] > 0`, so this check is duplicated.

Technical Details

[This line](#) of `claimInternal()` can be removed because `updateCredit()` [performs the same check](#) and `updateCredit()` is called in the 2nd line of `claimInternal()`.

Impact

Gas savings.

Recommendation

Remove duplicate check to save gas.

5. Gas - Consistently apply unchecked for gas savings

`sub()` and `div()` in SafeMath do not apply unchecked in places that could save gas. This is externally inconsistent with the OpenZeppelin SafeMath implementation and internally inconsistent with how unchecked is applied in `add()` and `mul()`.

Technical Details

[This line](#) of `sub()` and [this line](#) of `div()` can be unchecked. This would follow the approach applied in `trySub()` and `tryDiv()` in the OZ SafeMath library for solidity 0.8.X and the approach in `add()` and `mul()` of the Sonne SafeMath library.

Impact

Gas savings.

Recommendation

Use unchecked consistently for gas savings.

6. Gas - Use Solidity errors in 0.8.4+

Using [solidity errors](#) is a new and more gas efficient way to revert on failure states.

Technical Details

Require statements are found throughout the protocol, but especially in `Comptroller.sol`, `CToken.sol`, and `ExternalRewardDistributor.sol`. Replacing require with solidity errors can provide gas savings.

Impact

Gas savings.

Recommendation

Add errors to replace each `require()` with `revert errorName()` for greater gas efficiency.

7. Gas - Borrow gas optimizations from Compound

[ExternalRewardDistributorV1.sol](#) is a completely new contract that is heavily inspired by

Comptroller.sol. Specifically, `notifyBorrowIndexInternal()` and `notifyBorrowIndexInternal()` are based on Comptroller's `updateCompBorrowIndex()`. `updateCompBorrowIndex()` has some gas optimizations that are not applied in `ExternalRewardDistributorV1`.

Technical Details

`notifyBorrowIndexInternal()` and `notifyBorrowIndexInternal()` can benefit from gas optimizations by:

- caching state variable storage values retrieved more than once
- removing duplicate `>` or `<` checks

The specific differences compared to Compound are:

- `notifyBorrowIndexInternal()` [checks if](#) `blockNumber > marketState.borrowBlock` and then performs the math `blockNumber - marketState.borrowBlock` with `SafeMath`. The subtraction can be unchecked to save gas, or [the logic from Compound](#) where the subtraction is performed before the if statement can be used.
- Two state variable values are queried twice, `marketState.supplyBlock` and `marketState.supplySpeed`. These values can be cached to save gas in the case where they are queried twice. Compound [only caches](#) `supplySpeed` because the supply block is queried only once because of [the previous optimization](#) (subtracting the values before the if statement).

Impact

Gas savings.

Recommendation

Consider borrowing gas optimizations from Compound's implementation.

8. Gas - Initialize variable only if needed

In some cases where variables are initialized, the variables won't be used. This is inefficient and variables should only be initialized when they are used.

Technical Details

Variable `marketBorrowIndex` is initialized before the if statement but it's only used inside the second if block.

Impact

Gas savings.

Recommendation

Initialize variable `marketBorrowIndex` inside the second if block, before it is used, [L228](#).

Informational Findings

1. Informational - Undocumented market creation process

There is no clear public documentation describing how new markets are added to Sonne. Because of the risk that new markets can add to the entire protocol, this process should be documented so that 1. the protocol team has a strict process to follow to avoid common problems and 2. so users can determine the risk associated with this process.

Technical Details

One of the [top causes of Compound fork hacks](#) is reentrancy bugs, because Compound does not follow the checks-effects-interaction pattern. This means that if a Sonne market is created with a token that allows reentrancy, such as an ERC777 token, this can put the protocol funds at risk. Compound has a clear process for adding new markets that includes creating [a public governance proposal](#). [Compound is aware of this risk](#) but has chosen not to fix it in their code.

Impact

Informational.

Recommendation

Clearly document the steps for adding a new market to Sonne. Perform this process in public so that depositors can assess the risk of each new market and have time to withdraw their funds if the risk increases above their preferred threshold.

2. Informational - Replace magic numbers with constants

Constant variables should be used in place of magic numbers to prevent typos. For one example, the magic number `2**160` is found in multiple places in `Distribution.sol` and should be replaced with a constant. Using a constant also adds a description to the value to explain the purpose of the value.

Technical Details

`Distribution.sol` uses magic number `2**160` in several places. Consider replacing these magic numbers with a constant internal variable. This will not change gas consumption.

Impact

Informational.

Recommendation

Use constant variables instead of magic numbers.

3. Informational - Remove unused code

Remove commented code that won't be used in the future.

Technical Details

`Distribution.sol` has commented code that is not used, [L105](#) and [L116](#). Removing it will improve readability.

Impact

Informational.

Recommendation

Remove commented code.

4. Informational - Add events to Distributor.sol

In the contract [Distribution.sol](#), admin function for adding and removing reward tokens are not emitting events. This makes it difficult to track when these actions are performed.

Technical Details

Add events for changing the list of rewards tokens `tokens`. One for each of the functions `addToken()` and `removeToken()` in the file `Distribution.sol`.

Impact

Informational.

Recommendation

Add events to `addToken()` and `removeToken()` functions in the file `Distribution.sol`.

5. Informational - Add event to `setWithdrawalPendingTime()` and `burn()`

`setWithdrawalPendingTime()` modifies the value of `withdrawalPendingTime` which is a state

variable. Also, `burn()` modifies the value of `Withdrawal` which can be valuable to track.

Technical Details

It can be helpful to add events for any action that modifies state variables to make it easier to trace when the value change happened and to add monitoring of such changes more easily.

Impact

Informational.

Recommendation

Add event to `setWithdrawalPendingTime()` and `burn()` functions in the file `StakeDistributor.sol`.

6. Informational - Remove unused files

In the project `sonne-protocol`, there is unused interface [ISonne.sol](#), which can be removed.

Technical Details

Removing unused files will improve readability.

Impact

Informational.

Recommendation

Remove unused file `ISonne.sol`.

7. Informational - Outsourcing staking yield generation increases risk

Sonne staking protocol funds are deposited into Velodrome to earn yield. Outsourcing the yield source for staking assets is an unusual choice and choosing to deposit in Velodrome specifically may add additional risks to staked funds.

Technical Details

Sonne heavily relies on Velodrome for the Sonne staking protocol. Unlike some other staking protocols, Sonne does not directly provide rewards to stakers but instead outsources the reward generation to Velodrome. Relying on a protocol that [does not have an active bug bounty program](#), [has not gotten an audit from a high quality firm](#), is forked from Solidly which [had several known bugs](#), and [has a top 30 TVL](#) introduces risk to the funds that Sonne users stake. Because Velodrome is likely one of the highest TVL

protocols without an active bug bounty program, they likely are a target for bad actors while they do not have much incentive for white hat hackers to look at their code.

Impact

Informational.

Recommendation

Consider adding some monitoring of Velodrome to closely watch for large changes in value and pushing their development team to start a bug bounty program to incentivize security researchers to check their code for issues. Consider an incident response plan if there is any security issues that arise from Velodrome.

8. Informational - Use consistent naming for internal functions

Distributor.sol defines two internal functions, `claimInternal()` and `_editRecipientInternal()`. Consider consistently adding a `_` to the beginning of all internal function names.

Technical Details

`_editRecipientInternal()` and `claimInternal()` are defined in Distributor.sol. Consider renaming `claimInternal()` to `_claimInternal()` for consistency to avoid confusion about the context of certain functions.

Impact

Informational.

Recommendation

Consider renaming `claimInternal()` to `_claimInternal()` for consistency.

9. Informational - Remove redundant import

The SafeToken.sol import in StakedDistributor.sol can be removed because it is imported from the import of Distributor.sol.

Technical Details

[Distributor.sol](#) imports [SafeToken.sol](#) and [StakedDistributor.sol](#) also imports [SafeToken.sol](#).

The SafeToken.sol import in StakedDistributor.sol can be removed because StakedDistributor.sol already imports Distributor.sol.

Impact

Informational.

Recommendation

Remove SafeToken.sol import in StakedDistributor.sol.

10. Informational - Missing NatSpec

The functions in the staking-protocol repository contracts are missing detailed NatSpec and inline comments. The same applies to all custom contracts and functions in the lending-protocol repository. This can make it more difficult for users of the code to determine whether the code is implemented in a way that matches what the docs advertise.

Technical Details

NatSpec is a good way to explain your code to other developers modifying or forking a project, to users who want to understand what the contracts are doing, and to auditors who are trying to determine whether the contract logic is implemented properly. The contracts of staking-protocol have a severe lack of detailed NatSpec comments which makes it harder to understand the developer's intentions.

Impact

Informational.

Recommendation

Add NatSpec to all functions, or at a minimum, all public and external functions. Consider using [GitHub Copilot](#) or [slither-docs-action](#) to leverage AI to speed up the process.

11. Informational - Simplify `claimInternal()` arguments

`claimInternal()` can be simplified by removing the `account` function argument because this value is always `msg.sender`.

Technical Details

`claimInternal()` takes two function arguments of type address. But this function is an internal function, and the only two places where it is called (1, 2) set the `account` argument to `msg.sender`. This means `claimInternal()` does not require this argument because `account` can be replaced with `msg.sender`.

Impact

Informational.

Recommendation

Replace `account` with `msg.sender` in `claimInternal()`.

12. Informational - Document the end of rewards accumulation when `burn()` is called

Withdrawing SONNE from the staking contracts is a 2-step process. The first step is to call `burn()`, then after one week `withdraw()` can be called. The Sonne documentation fails to mention that the one week waiting period between calling `burn()` and `withdraw()` does not accumulate rewards for the deposited SONNE balance. This should be fixed so that user expectations match what the code implements.

Technical Details

Every SONNE depositor into uSONNE or sSONNE must leave their tokens in the contract for one week without reward accumulation. This is [not documented in the Sonne staking docs](#), but should be. Otherwise if the documentation does not properly describe this behavior, users may not expect the code to be implemented in the way that it is.

Impact

Informational.

Recommendation

Document how rewards only accumulate on tokens until `burn()` is called to start with unstaking process.

13. Informational - Distributor function `removeToken()` can lose funds

Distributor.sol contract has a function `removeToken()` that allows the contract owner to remove reward token from the list of reward tokens. This function can be called before all rewards are collected, which can result in the loss of funds.

Technical Details

Removing reward token from the list of reward tokens will result in locked reward tokens which cannot be collected by users or contract owner.

Impact

Informational.

Recommendation

An ideal solution would be to disable removing tokens before all reward tokens are collected, the reward token balance is zero. Waiting for all users to collect all rewards is highly unlikely and will result in a higher gas cost for users interacting with StakedDistributor tokens. Inform users they need to collect all rewards before removing the reward token.

14. Informational - Unnecessary code from Compound

`fixBadAccruals()` in `Comptroller.sol` has the NatSpec “Delete this function after proposal 65 is executed”. This indicates the function is specific to Compound’s fix of bad accruals for specific addresses and is irrelevant to Sonne. Similarly, the `liquidateBorrowVerify()` function serves no purpose in `Comptroller` and can be removed.

Technical Details

[Compound proposal 65](#) is related to fixing a past issue that Compound had related to distributing rewards early. Sonne does not need to fix such an issue and therefore [this function](#) is unnecessary outside of Compound. `liquidateBorrowVerify()` is also unnecessary. Remove the functions without purpose or implement if needed: `sizeVerify()`, `transferVerify()`, `mintVerify()`, `borrowVerify()`, `repayBorrowVerify()`, `liquidateBorrowVerify()`. Also, remove errors that are not used: `TransferTooMuch` and `LiquidateRepayBorrowFreshFailed`.

Impact

Informational.

Recommendation

Remove unnecessary code. In the case of `fixBadAccruals()`, the function has the ability to change the token balances for certain addresses and could be a rug vector for the admin, although the admin is `TimelockController` so the rug cannot happen instantly.

15. Informational - Make public functions external

A public function can be called internally and externally. An external function can only be called externally. If a public function does not need to be called internally, make it an external function. This can also save gas on deployment in some solidity versions.

Technical Details

Declare these public functions as external because they do not need to be called

internally:

- 1 `getSupplyRate()` in [JumpRateModelV4.sol](#)
- 2 `isOwner()` in [Ownable.sol](#)

Impact

Informational.

Recommendation

Declare public functions as external when possible.

16. Informational - Use standard implementation approach in SafeMath.sol

SafeMath.sol has duplicate code for `add()` and `mul()`. This is inconsistent with the implementation for `sub()` and `div()`. Use a consistent implementation for the entire contract.

Technical Details

There are two functions for each SafeMath operation. One of the functions supports an arbitrary error message by providing a function argument for this message. The other function has a hardcoded error message. The way that `sub()` and `div()` are implemented is by having the hardcoded error message implementation calling the arbitrary error message implementation with a hardcoded error string. But `add()` and `mul()` are not implemented in this way. Instead, `add()` and `mul()` reimplement the function with the arbitrary error message but replace the function argument with a hardcoded string. Consider using the same approach from `sub()` and `div()` in `add()` and `mul()` to avoid reimplementing the same logic.

Impact

Informational.

Recommendation

Implement functions consistently.

17. Informational - Incorrect price oracle address in Sonne docs

The address for the price oracle in Sonne Docs is incorrect.

Technical Details

Sonne docs show the price oracle address is

0xEFc0495DA3E48c5A55F73706b249FD49d711A502. But [the Comptroller oracle state variable](#) has the value of 0x8d0db2bd9111e35554b8152e172451c80dff22b7. The older Sonne price oracle does not contain data for tokens soWBTC, soLUSD, and sowstETH.

Impact

Informational.

Recommendation

Update the Sonne docs to match on-chain implementation.

18. Informational - No comparison against minAnswer or maxAnswer

The Chainlink data feed aggregator has a `minAnswer` and `maxAnswer` value. Checking that the price returned by the oracle is within this range or near the endpoints of this range may provide extra safety against unexpected market events, hacks, or extreme oracle manipulation.

Technical Details

The Chainlink data is verified [to be greater than zero](#), but a stricter check would use the aggregator `minAnswer` and `maxAnswer` values. Chainlink [documents this approach](#) in their docs. Many tokens, like USDC or USDT, have a `minAnswer` value equivalent of \$0.01.

Impact

Informational.

Recommendation

Consider adding comparisons with `minAnswer` and `maxAnswer`.

```
- require(price > 0, "price cannot be zero");
+ require(price > int256(aggregator.minAnswer()) && price <
+ int256(aggregator.maxAnswer()), "price outside of range");
```

19. Informational - `_getLatestPrice()` `timeStamp` return value never used

`_getLatestPrice()` can be simplified because one of the return values from this internal function is never used.

Technical Details

`_getLatestPrice()` is an internal function [with two return values](#), `uPrice` and `timeStamp`.

`timeStamp` is not used when this internal function is called so it can be removed.

Impact

Informational.

Recommendation

Simplify `_getLatestPrice()` by removing the `timeStamp` return value.

20. Informational - Protocol will stop working after year 2106

The function `getBlockNumber()` is modified from Compound to return `block.timestamp` instead of `block.number`. This value is safecast to uint32, and `block.timestamp` will exceed the size of uint32 in the year 2106.

Technical Details

`getBlockNumber()` is safecast to uint32 in `Comptroller.sol` (1, 2) and will revert in the year 2016 when `block.timestamp` is too large for this type.

Impact

Informational.

Recommendation

Comptroller is behind a proxy and can be updated when necessary.

21. Informational - Typos

BasicLens has a typo in a function name.

Technical Details

`compAccued()` in `BasicLens` should be `compAccrued()`.

Also consider modifying the function and variable names that include the word block because the Sonne implementation of all of these functions and variables uses seconds, not blocks.

- `getBlockNumber()` -> `getBlockTimestamp()`
- `blocksPerYear` -> `secondsPerYear`
- `multiplierPerBlock` -> `multiplierPerSecond`
- `jumpMultiplierPerBlock` -> `jumpMultiplierPerSecond`

- `baseRatePerBlock` -> `baseRatePerSecond`

Impact

Informational.

Recommendation

Fix typos. Consider renaming functions and variables to properly describe their purpose.

22. Informational - Use interface instead of call function

In [Comptroller.sol](#) calls to ExternalRewardDistributor are made using call function.

Technical Details

Calling the contract using an interface is more secure than using call function. File [ExternalRewardDistributor.sol](#) contains the interface for ExternalRewardDistributor contract. Change call function defined at [L1458](#), [L1590](#), [L1639](#), [L1700](#) and [L1792](#).

Impact

Informational.

Recommendation

Call contracts in a safer way using interface instead of call function.

23. Informational - SNX token risk

There was a previous bug in the SNX token that has since been fixed. However, the SNX token code on Optimism is different from the SNX token code on mainnet Ethereum. Because the SNX token code is more complex than a more simplistic and straightforward ERC20, this token may carry additional risks.

Technical Details

SNX [had a double entrypt point issue](#) that was found in Balancer. The issue was originally reported for mainnet but the same bug was applicable on Optimism. Even after this fix, the [mainnet SNX code](#) and [the Optimism SNX code](#) are different. The SNX token on mainnet and Optimism is using a form of proxy, and the implementation on Optimism has assembly code that differs [from the OpenZeppelin implementation](#). The code of this token is out of scope of the Sonne audit, but risks to Sonne can be introduced by supporting markets for tokens that have added risks.

Impact

Informational.

Recommendation

Consider monitoring for SNX token upgrades.

24. Informational - Solidity version 0.8.20

Solidity version [0.8.20](#) switches default target EVM version to Shanghai. This may cause problems on some chains.

Technical Details

New opcode `PUSH0` is introduced in Solidity version 0.8.20 which may not be supported on a chain other than mainnet like L2 chains.

Impact

Informational.

Recommendation

Select the appropriate EVM before deploying to a chain other than mainnet.

Final remarks

The Sonne protocol is a Compound fork that wisely chose to change very little logic to minimize the differences between Sonne and Compound. Despite the similar on-chain contract logic, there are several decisions in the parameters chosen by Sonne that increase the risk of the protocol when compared to Compound or other DeFi protocols.

One unusual design choice is to include an EOA as a key component in the design of the protocol for distributing rewards. Because the logic behind the EOA operations is not coded on-chain like a smart contract, users must place a substantial amount of trust in the EOA to operate as advertised in the documentation, and at least one discrepancy between the documentation and EOA actions was observed.

Another area where Sonne introduces substantial risk compared to other Compound forks is the choices made around interest rate models, collateral ratios, and general incentive choices that lead to large amounts of borrowing. When compared to many similar protocols, Sonne has a much larger amount of borrowed assets, which implies a greater risk of bad debt. Although a detailed risk analysis was not necessarily in-scope for this review, the discrepancy between Sonne and other protocols that put substantial resources

into risk modelling indicates that Sonne is likely unaware of the exact risk level of the protocol under different market conditions.

A final design choice that is unusual is the outsourcing of staking yield generation to Velodrome. While the Velodrome codebase was out of scope of this review, Velodrome does not have an active bug bounty and does not have any public audit report from well-known audit firms (excluding the crowdsourced code4rena competition, which had a short timeframe for such a large codebase).
