SHERLOCK

Security Review For Noble

Collaborative Audit Prepared For: **Noble**
Lead Security Expert(s): **ParthMandale**
pkqs90

Date Audited: **September 5 - September 6, 2025**
Final Commit: **9aaf923**

Introduction

Noble Dollar (USDN) is a YBS launched in March this year. It is fully backed by short-term U.S. Treasury bills, and holders accrue yield under the rules of the M0 Protocol - no staking or lockups required. What differentiates USDN from other YBSs is that not only do users of USDN get yield by simply holding it in their wallets, but the yield can also be sent cross-chain to other networks that integrate USDN. See here to learn more about USDN's cross-chain yield capabilities.

Scope

Repository: m0-foundation/noble-dollar

Audited Commit: 6e94ce35d60794a8bafa29b8693c22f1530e6db5

Final Commit: 9aaf9235ae7b0cf70ca7189fc138a78f7051d13f

Files:

- contracts/src/NobleDollar.sol
- contracts/utils/IndexingMath.sol
- contracts/utils/UIntMath.sol

Final Commit Hash

9aaf9235ae7b0cf70ca7189fc138a78f7051d13f

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues Found

High	Medium	Low/Info
0	1	2

Issues Not Fixed and Not Acknowledged

High	Medium	Low/Info
0	0	0

Issue M-1: User yield claim may be DoSed due to not enough balance.

Source: https://github.com/sherlock-audit/2025-09-noble/issues/5

Summary

User yield claim may be DoSed due to not enough balance.

Vulnerability Detail

Assume index = 1.2, Bob has 100 wei principal, and 120 wei balance. Bob transfers 119 balance to Alice. The principal here would be roundUp(119 / 1.2) = 100, so Alice will have 100 principal and 119 balance.

Then Alice can call claim() to claim 1 wei, because 100 * 1.2 - 119 = 1. This is effectively minting 1 wei USDN of yield from thin air.

Now, imagine if before transfer Alice originally had a principal of 100, and balanceOf = 100. She could have called claim() and receive 100 * 1.2 - 100 = 20 tokens. After the transfer, she has 200 principal and 219 balance, then when she calls claim(), she will receive 200 * 1.2 - 219 = 21 tokens. However, since there are only 20 yield tokens, Alice's claim() will end up reverting.

In the worst case scenario an attacker can abuse this to DoS yield claim for honest users.

https://github.com/sherlock-audit/2025-09-noble/blob/main/noble-dollar/contracts/src/NobleDollar.sol#L213-L216

```
// Safely round up principal in case of transfers or burning.
uint112 fromPrincipal = $.principal[from];
uint112 principalUp = IndexingMath.getSafePrincipalAmountRoundedUp(value, $.index,
↪  fromPrincipal);
$.principal[from] = fromPrincipal - principalUp;\
```

Impact

User may claim 1 wei more yield than they should, which can cause the yield claim to fully DoS.

Claiming 1 wei more yield is not a big problem, because the user that triggered the transfer will also lose 1 wei of yield (due to rounding), so it's not minting extra tokens from thin air. The bigger problem here is causing DoS.

Recommendation

```
    function claim() public {
-       uint256 amount = yield(msg.sender);
+       uint256 amount = min(balanceOf(address(this), yield(msg.sender));

        if (amount == 0) revert NoClaimableYield();

        _update(address(this), msg.sender, amount);

        emit YieldClaimed(msg.sender, amount);
    }
```

During yield claim, clamp the yield with the remaining balance of `address(this)`.

Issue L-1: Code doesn't compile with solidity version 0.8.20

Source: https://github.com/sherlock-audit/2025-09-noble/issues/6

Summary

The NobleDollar pinpoints the solidity version to 0.8.20, but some openzeppelin dependencies require >= 0.8.22, so the compile would fail.

https://github.com/sherlock-audit/2025-09-noble/blob/main/noble-dollar/contracts/src/NobleDollar.sol#L18

Suggest to use 0.8.22 version.

Issue L-2: Inconsistency rounding for principal, index and balance

Source: https://github.com/sherlock-audit/2025-09-noble/issues/7

Summary

There are certain scenarios where the rounding of `principal * index` is inconsistent with user balance.

Vulnerability Detail

Scenario 1: User initally has 100 wei principal, and index grows from 1e12 to 1.199e12. The user balance after claiming yield would be `roundDown(100 * 1.199) = 119`. This is because when we are calculating yield, we round down.

In this state, balance = roundDown(principal * index).

https://github.com/sherlock-audit/2025-09-noble/blob/main/noble-dollar/contracts/src/NobleDollar.sol#L131

```
    function yield(address account) public view returns (uint256) {
        USDNStorage storage $ = _getUSDNStorage();

@>      uint256 expectedBalance =
↪   IndexingMath.getPresentAmountRoundedDown($.principal[account], $.index);

        uint256 currentBalance = balanceOf(account);

        return expectedBalance > currentBalance ? expectedBalance - currentBalance
        ↪   : 0;
    }
```

Scenario 2: Index is 1.199e12, and user mints 119 wei tokens. The balance is now 119 wei, but the principal is `roundDown(119 / 1.199) = 99 wei`. Now, we're in a state where principal is 99 wei, balance is 119 wei, which means in this state, balance = roundUp(principal * index).

https://github.com/sherlock-audit/2025-09-noble/blob/main/noble-dollar/contracts/src/NobleDollar.sol#L205-L211

```
        if (from == address(0)) {
@>          uint112 principalDown =
↪   IndexingMath.getPrincipalAmountRoundedDown(value, $.index);
            $.principal[to] += principalDown;
            $.totalPrincipal += principalDown;
```

```
            return;
        }
```

Impact

Did not find an attack vector related to this, but it may cause weird UX behavior.

Recommendation

Considering the separately maintaining balance and principal is a fundamental mechanism of the token, I don' think there is a perfect fix. Best to document this behavior.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.