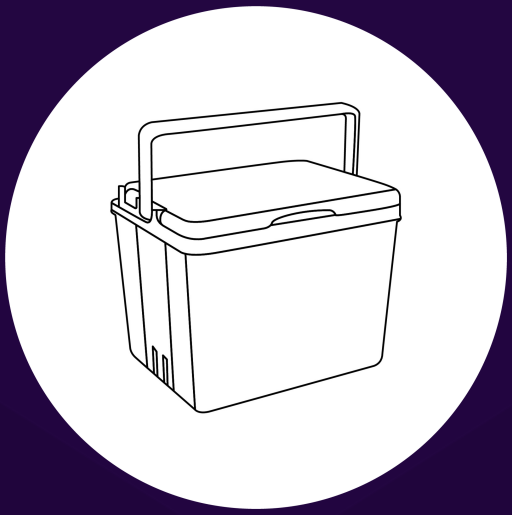
SHERLOCK

# SHERLOCK SECURITY REVIEW FOR



**Prepared for:** Cooler
**Prepared by:** Sherlock
**Lead Security Expert:** jkoppel
**Dates Audited:** August 25 - August 28, 2023
**Prepared on:** September 22, 2023

# Introduction

A peer-to-peer lending protocol allowing a borrower and lender to engage in fixed-duration, fixed-interest lending. Cooler Loans are lightweight, trustless, independent of price-based liquidation.

## Scope

Repository: ohmzeus/Cooler

Branch: main

Commit: c6f2bbe1b51cdf3bb4d078875170177a1b8ba2a3

---

For the detailed scope, see the contest details.

## 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.

## Issues found

| Medium | High |
|---|---|
| 4 | 4 |

## Issues not fixed or acknowledged

| Medium | High |
|---|---|
| 0 | 0 |

## Security experts who found valid issues

Ignite
deth
jkoppel
detectiveking
mert_eren
ubermensch
thekmj
banditx0x
deadrxsezzz

evilakela
klaus
castle_chain
mahdikarimi
Chinmay
pengun
xAlismx
0xMAKEOUTHILL
Negin
0xbepresent
harisnabeel
nmirchev8
ni8mare
libratus
Delvir0
Kow
Mlome
BugHunter101
Vagner
Hama
jah
cats
Breeje
sandy
james_wu
ubl4nk
pep7siup
radevauditor
p-tsanev
0xmurali7
B353N
Kral01
Silvermist
ADM
tvdung94
SanketKogekar
carrotsmuggler
0xMosh
Yanev
jovi
SBSecurity
HChang26
hals

# Issue H-1: Can steal gOhm by calling Clearinghouse.claimDefaulted on loans not made by Clearinghouse

Source: https://github.com/sherlock-audit/2023-08-cooler-judging/issues/28

## Found by

detectiveking, jkoppel, mert_eren

`Clearinghouse.claimDefaulted` assumes that all loans passed in were originated by the Clearinghouse. However, nothing guarantees that. An attacker can wreak havoc by calling it with a mixture of Clearinghouse-originated and external loans. In particular, they can inflate the computed `totalCollateral` recovered to steal excess gOhm from defaulted loans.

## Vulnerability Detail

1. Alice creates a Cooler. 9 times, she calls `requestLoan` (not through the Clearinghouse) to request a loan of 0.000001 DAI collateralized by 2 gOhm. For each loan, she then calls `clearLoan` and loans the 0.000001 DAI to herself.
2. One week later, Bob calls `Clearinghouse.lendToCooler` and takes a loan for 3000 DAI collateralized by 1 gOHM
3. Alice defaults on the loans she made to herself and waits 7 days
4. Bob defaults on his loan
5. Alice calls `Clearinghouse.claimDefaulted`, passing in both her loans to herself and Bob's loan from the Clearinghouse. `Clearinghouse.claimDefaulted` calls `Cooler.claimDefaulted` on each, returning 18 gOhm to Alice and 1 gOhm to the Clearinghouse.
6. For each of Alice's loan, the keeper reward is incremented by the max award of 0.1 gOhm. For Bob's loan, the keeper reward is incremented by somewhere between 0 and 0.05 gOhm, depending on how much time has elapsed since Bob's loan defaulted.
7. The keeper reward is transferred to Alice. Alice's reward will be between 0.9 and 0.95 gOhm, but it should be between 0 and 0.05 gOhm. The contract should recover between 0.95 and 1 gOhm, but it only recovers between 0.05 and 0.1 gOhm. Alice has effectively stolen 0.9 gOhm from the contract

The attack as stated above can steal at most 5% of the collateral. **Note that Alice can get this even without waiting 7 days from loan expiry time.** It further requires the Clearinghouse have some extra gOhm around, as it will burn `totalCollateral -`

`keeperRewards`. This can happen if the treasury or someone sends it some gOhm for some reason, or by calling claimDefault as in #3 .

However, #5 extends this attack so that Alice can steal 100% of the collateral, even if the Clearinghouse has no extra gOhm lying around.

For added fun, note that, when setting up her loans to herself, Alice can set the loan duration to 0 seconds. So this only requires setting up 1 block in advance.

## Impact

Anyone can steal collateral from defaulted loans.

## Code Snippet

https://github.com/sherlock-audit/2023-08-cooler/blob/main/Cooler/src/Clearinghouse.sol#L191

Notice the lack of any checks that the loan's lender is the Clearingouse

```
function claimDefaulted(address[] calldata coolers_, uint256[] calldata loans_)
↪  external {
    uint256 loans = loans_.length;
    if (loans != coolers_.length) revert LengthDiscrepancy();

    uint256 totalDebt;
    uint256 totalInterest;
    uint256 totalCollateral;
    uint256 keeperRewards;
    for (uint256 i=0; i < loans;) {
        // Validate that cooler was deployed by the trusted factory.
        if (!factory.created(coolers_[i])) revert OnlyFromFactory();

        // Claim defaults and update cached metrics.
        (uint256 debt, uint256 collateral, uint256 elapsed) =
↪  Cooler(coolers_[i]).claimDefaulted(loans_[i]);
```

`keeperRewards` is incremented for every loan.

```
    // Cap rewards to 5% of the collateral to avoid OHM holder's dillution.
    uint256 maxAuctionReward = collateral * 5e16 / 1e18;
    // Cap rewards to avoid exorbitant amounts.
    uint256 maxReward = (maxAuctionReward < MAX_REWARD)
        ? maxAuctionReward
        : MAX_REWARD;
    // Calculate rewards based on the elapsed time since default.
    keeperRewards = (elapsed < 7 days)
        ? keeperRewards + maxReward * elapsed / 7 days
```

```
        : keeperRewards + maxReward;
}
```

https://github.com/sherlock-audit/2023-08-cooler/blob/main/Cooler/src/Cooler.sol#L318

`Cooler.claimDefaulted` can be called by anyone.

```
function claimDefaulted(uint256 loanID_) external returns (uint256, uint256,
↪  uint256) {
        Loan memory loan = loans[loanID_];
        delete loans[loanID_];

       // Hey look, no checks on sender
}
```

## Tool used

Manual Review

## Recommendation

Check that the Clearinghouse is the originator of all loans passed to claimDefaulted

## Discussion

**0xRusowsky**

- https://github.com/ohmzeus/Cooler/pull/48

**jkoppel**

Note on this:

The link to #3 is meant to be a link to #46

The link to #5 is meant to be a link to #115

In the past, when I linked to issues in my private judging repository, Sherlock would properly update them upon submission. Now it just links them to whatever issue in the public judging repo has the same number.

**jkoppel**

Fix confirmed.

# Issue H-2: At claimDefaulted, the lender may not receive the token because the Unclaimed token is not processed

Source: https://github.com/sherlock-audit/2023-08-cooler-judging/issues/119

## Found by

0xMAKEOUTHILL, Chinmay, Negin, banditx0x, deadrxsezzz, jkoppel, klaus, mahdikarimi, pengun, xAlismx

`claimDefaulted` does not handle `loan.unclaimed` . This preventing the lender from receiving the debt repayment.

## Vulnerability Detail

```
function claimDefaulted(uint256 loanID_) external returns (uint256, uint256,
↪  uint256) {
  Loan memory loan = loans[loanID_];
  delete loans[loanID_];
```

Loan data is deletead in `claimDefaulted` function. `loan.unclaimed` is not checked before data deletead. So, if `claimDefaulted` is called while there are unclaimed tokens, the lender will not be able to get the unclaimed tokens.

## Impact

Lender cannot get unclaimed token.

## Code Snippet

https://github.com/sherlock-audit/2023-08-cooler/blob/6d34cd12a2a15d2c92307d44782d6eae1474ab25/Cooler/src/Cooler.sol#L318-L320

## Tool used

Manual Review

## Recommendation

Process unclaimed tokens before deleting loan data.

```
function claimDefaulted(uint256 loanID_) external returns (uint256, uint256,
↪  uint256) {
+   claimRepaid(loanID_)
```

```
    Loan memory loan = loans[loanID_];
    delete loans[loanID_];
```

## Discussion

**0xRusowsky**

- fix: https://github.com/ohmzeus/Cooler/pull/54
- https://github.com/ohmzeus/Cooler/pull/47

**jkoppel**

Fix approved.

# Issue H-3: Clearinghouse.sol#claimDefaulted()

Source: https://github.com/sherlock-audit/2023-08-cooler-judging/issues/176

## Found by

Ignite, deth `Clearinghouse` doesn't approve the `MINTR` to handle tokens in his name, which bricks the entire function.

## Vulnerability Detail

Inside `claimDefaulted` on the last line we call `MINTR.burnOhm` which in turn calls OHM.burnFrom. The docs for MINTR.burnFrom state: "Burn OHM from an address. Must have approval.". We can confirm that this is the case when looking at `OHM` source code and it's `burnFrom`. I found 2 `OHM` tokens that are currently deployed on mainnet, so I'm linking both their addresses: https://etherscan.io/token/0x383518188c0c6d7730d91b2c03a03c837814a899#code, https://etherscan.io/token/0x64aa3364f17a4d01c6f1751fd97c2bd3d7e7f1d5#code. Both addresses use the same `burnFrom` logic and in both cases they require an `allowance`. Nowhere in the contract do we approve the `MINTR` to handle `OHM` tokens in the name of `Clearinghouse`, in fact `OHM` isn't even specified in `Clearinghouse`.

Side note: The test `testFuzz_claimDefaulted` succeeds, because `MockOhm` is written incorrectly. When `burnFrom` gets called `MockOhm` calls the inherited `_burn` function, which burns tokens from `msg.sender`. The mock doesn't represent how the real `OHM.burnFrom` works.

## Impact

`Claimdefault` will always revert.

## Code Snippet

https://github.com/sherlock-audit/2023-08-cooler/blob/6d34cd12a2a15d2c92307d44782d6eae1474ab25/Cooler/src/Clearinghouse.sol#L244

## Tool used

Manual Review

## Recommendation

Add a variable `ohm` which will be the `OHM` address and approve the necessary tokens to the `MINTR` before calling `MINTR.burnOhm`.

## Discussion

**jkoppel**

Seems real

**0xRusowsky**

Confirmed, but disagree with the severity. Defaults could still happen via the Cooler contracts and OHM could be burned ad-hoc by the DAO.

**0xRusowsky**

After discussing it internally, we don't mind if it's labeled as high or medium cause we would need to deploy a new policy (so it would require some extra work on our end)

**0xRusowsky**

- https://github.com/ohmzeus/Cooler/pull/52

**jkoppel**

Fix approved.

# Issue H-4: isCoolerCallback can be bypassed

Source: https://github.com/sherlock-audit/2023-08-cooler-judging/issues/187

## Found by

0xbepresent, BugHunter101, Delvir0, Hama, Ignite, Kow, Mlome, Vagner, banditx0x, castle_chain, deadrxsezzz, detectiveking, evilakela, harisnabeel, jah, klaus, libratus, mert_eren, ni8mare, nmirchev8, ubermensch

The lender can bypass `CoolerCallback.isCoolerCallback()` validation without implements the `CoolerCallback` abstract.

In the provided example, this may force the loan to default.

## Vulnerability Detail

The `CoolerCallback.isCoolerCallback()` is intended to ensure that the lender implements the `CoolerCallback` abstract at line 241 when the parameter `isCallback_` is `true`.

https://github.com/sherlock-audit/2023-08-cooler/blob/main/Cooler/src/Cooler.sol#L233-L275

However, this function doesn't provide any protection. The lender can bypass this check without implementing the `CoolerCallback` abstract by calling the `Cooler.clearRequest()` function using a contract that implements the `isCoolerCallback()` function and returns a `true` value.

For example:

By being the `loan.lender` with implement only `onDefault()` function, this will cause the `repayLoan()` and `rollLoan()` methods to fail due to revert at `onRepay()` and `onRoll()` function. The borrower cannot repay and the loan will be defaulted.

After the loan default, the attacker can execute `claimDefault()` to claim the collateral.

Furthermore, there is another method that allows lenders to bypass the `CoolerCallback.isCoolerCallback()` function which is loan ownership transfer.

Normally, the lender who implements the `CoolerCallback` abstract may call the `Cooler.clearRequest()` with the `_isCoolerCallback` parameter set to `true` to execute logic when a loan is repaid, rolled, or defaulted.

But the lender needs to change the owner of the loan, so they call the `approveTransfer()` and `transferOwnership()` functions to the contract that doesn't implement the `CoolerCallback` abstract (or implement only `onDefault()` function to force the loan default), but the `loan.callback` flag is still set to `true`.

Thus, this breaks the business logic since the three callback functions don't need to be implemented when the `isCoolerCallback()` is set to `true` according to the dev note in the `CoolerCallback` abstract below:

> /// @notice Allows for debt issuers to execute logic when a loan is repaid, rolled, or defaulted. /// @dev The three callback functions must be implemented if `isCoolerCallback()` is set to true.

## Impact

1. The lender forced the Loan become default to get the collateral token, owner lost the collateral token.
2. Bypass the `isCoolerCallback` validation.

## Code Snippet

https://github.com/sherlock-audit/2023-08-cooler/blob/main/Cooler/src/Cooler.sol#L241

https://github.com/sherlock-audit/2023-08-cooler/blob/main/Cooler/src/Cooler.sol#L338-L343

https://github.com/sherlock-audit/2023-08-cooler/blob/main/Cooler/src/Cooler.sol#L347-L354

## Tool used

Manual Review

## Recommendation

Only allowing callbacks from the protocol-trusted address (eg., `Clearinghouse` contract).

Disable the transfer owner of the loan when the `loan.callback` is set to `true`.

## Discussion

**Oot2k**

Duplicate of 30

**Oot2k**

Reorder issues

**0xRusowsky**

- https://github.com/ohmzeus/Cooler/pull/51

- https://github.com/ohmzeus/Cooler/pull/57

**MLON33**

From Cooler on Discord: “We’re gonna leave this issue 187 untouched.”

# Issue M-1: `emergency_shutdown` role is not enough for emergency shutdown.

Source: https://github.com/sherlock-audit/2023-08-cooler-judging/issues/1

## Found by

thekmj, ubermensch

There are two protocol roles, `emergency_shutdown` and `cooler_overseer`. The `emergency_shutdown` should have the ability to shutdown the Clearinghouse.

However, in the current contract, `emergency_shutdown` role does not have said ability. An address will need both `emergency_shutdown` and `cooler_overseer` to perform said action.

We have also confirmed with the protocol team that the two roles will be held by two different multisigs, with the shutdown multisig having a lower threshold and more holders. Thereby governance will not be able to act as quickly to emergencies than expected.

## Vulnerability Detail

Let's examine the function `emergencyShutdown()`:

```
function emergencyShutdown() external onlyRole("emergency_shutdown") {
    active = false;

    // If necessary, defund sDAI.
    uint256 sdaiBalance = sdai.balanceOf(address(this));
    if (sdaiBalance != 0) defund(sdai, sdaiBalance);

    // If necessary, defund DAI.
    uint256 daiBalance = dai.balanceOf(address(this));
    if (daiBalance != 0) defund(dai, daiBalance);

    emit Deactivated();
}
```

This has the modifier `onlyRole("emergency_shutdown")`. However, this also calls function `defund()`, which has the modifier `onlyRole("cooler_overseer")`

```
function defund(ERC20 token_, uint256 amount_) public
↪   onlyRole("cooler_overseer") {
```

Therefore, the role `emergency_shutdown` will not have the ability to shutdown the protocol, unless it also has the overseer role.

## Proof of concept

To get a coded PoC, make the following modifications to the test case:

- In `Clearinghouse.t.sol`, comment out line 125 (so that `overseer` only has `emergency_shutdown` role) https://github.com/sherlock-audit/2023-08-cooler/blob/main/Cooler/src/test/Clearinghouse.t.sol#L125

```
//rolesAdmin.grantRole("cooler_overseer", overseer);
rolesAdmin.grantRole("emergency_shutdown", overseer);
```

- Run the following test command (to just run a single test `test_emergencyShutdown()`):

```
forge test --match-test test_emergencyShutdown
```

The test will fail with the `ROLES_RequireRole()` error.

## Impact

`emergency_shutdown` role cannot emergency shutdown the protocol

## Code Snippet

https://github.com/sherlock-audit/2023-08-cooler/blob/main/Cooler/src/Clearinghouse.sol#L339 https://github.com/sherlock-audit/2023-08-cooler/blob/main/Cooler/src/Clearinghouse.sol#L360-L372

## Tool used

Manual Review, Foundry/Forge

## Recommendation

There are two ways to mitigate this issue:

- Separate the logic for emergency shutdown and defunding. i.e. do not defund when emergency shutdown, but rather defund separately after shutdown.
- Move the defunding logic to a separate internal function, so that emergency shutdown function can directly call defunding without going through a modifier.

## Discussion

**sherlock-admin**

1 comment(s) were left on this issue during the judging contest.

**0xyPhilic** commented:

> invalid because it can be considered low as roles can be given again and there is no loss of funds

**0xRusowsky**

fair point, but it still should be low as a user can have several roles

**Oot2k**

I have to disagree, a user can indeed have several roles, but that can not be ensured/ if there are two separate roles they should be considered separate.

**ohmzeus**

Fix: https://github.com/ohmzeus/Cooler/pull/50

**jkoppel**

Fix confirmed.

# Issue M-2: Lender is able to steal borrowers collateral by calling rollLoan with unfavourable terms on behalf of the borrower.

Source: https://github.com/sherlock-audit/2023-08-cooler-judging/issues/26

## Found by

0xMosh, 0xbepresent, 0xmurali7, ADM, B353N, Breeje, BugHunter101, Chinmay, Delvir0, HChang26, Kow, Kral01, Mlome, SBSecurity, SanketKogekar, Silvermist, Yanev, banditx0x, carrotsmuggler, castle_chain, cats, deadrxsezzz, detectiveking, deth, evilakela, hals, jovi, libratus, mahdikarimi, ni8mare, nmirchev8, p-tsanev, pengun, sandy, tvdung94 A Lender is able to call provideNewTermsForRoll with whatever terms they want and then can call rollLoan on behalf of the borrower forcing them to roll the loan with the terms they provided. They can abuse this to make the loan so unfavourable for the borrower to repay that they must forfeit their collateral to the lender.

## Vulnerability Detail

Say a user has 100 collateral tokens valued at $1,500 and they wish to borrow 1,000 debt tokens valued at $1,000 they would would call: (values have simplified for ease of math)

```
requestLoan("1,000 debt tokens", "5% interest", "10 loan tokens for each
↪  collateral", "1 year")
```

If a lender then clears the request the borrower would expect to have 1 year to payback 1,050 debt tokens to be able to receive their collateral back.

However a lender is able to call provideNewTermsForRoll with whatever terms they wish: i.e.

```
provideNewTermsForRoll("loanID", "10000000% interest", "1000 loan tokens for
↪  each collateral" , "1 year")
```

They can then follow this up with a call to rollLoan(loanID): During the rollLoan function the interest is recalculated using:

```
function interestFor(uint256 amount_, uint256 rate_, uint256 duration_) public
↪  pure returns (uint256) {
    uint256 interest = (rate_ * duration_) / 365 days;
    return (amount_ * interest) / DECIMALS_INTEREST;
}
```

As rate_ & duration_ are controllable by the borrower when they call provideNewTermsForRoll they can input a large number that the amount returned is much larger then the value of the collateral. i.e. input a rate_ of amount * 3 and duration of 365 days so that the interestFor returns 3,000.

This amount gets added to the existing loan.amount and would make it too costly to ever repay as the borrower would have to spend more then the collateral is worth to get it back. i.e. borrower now would now need to send 4,050 debt tokens to receive their $1,500 worth of collateral back instead of the expected 1050.

The extra amount should result in more collateral needing to be sent however it is calculated using loan.request.loanToCollateral which is also controlled by the lender when they call provideNewTermsForRoll, allowing them to input a value that will result in newCollateralFor returning 0 and no new collateral needing to be sent.

```
function newCollateralFor(uint256 loanID_) public view returns (uint256) {
    Loan memory loan = loans[loanID_];
    // Accounts for all outstanding debt (borrowed amount + interest).
    uint256 neededCollateral = collateralFor(loan.amount,
↪   loan.request.loanToCollateral);
    // Lender can force neededCollateral to always be less than loan.collateral

    return neededCollateral > loan.collateral ? neededCollateral -
↪   loan.collateral : 0;
}
```

As a result a borrower who was expecting to have repay 1050 tokens to get back their collateral may now need to spend many multiples more of that and will just be forced to just forfeit their collateral to the lender.

## Impact

Borrower will be forced to payback the loan at unfavourable terms or forfeit their collateral.

## Code Snippet

Cooler.sol#L192-L217 Cooler.sol#L282-L300

## Tool used

Manual Review

## Recommendation

Add a check restricting rollLoan to only be callable by the owner. i.e.:

```
function rollLoan(uint256 loanID_) external {
        Loan memory loan = loans[loanID_];

        if (msg.sender != owner()) revert OnlyApproved();
```

Note: unrelated but rollLoan is also missing its event should add:

```
factory().newEvent(reqID_, CoolerFactory.Events.RollLoan, 0);
```

## Discussion

**jkoppel**

Whether this is medium or high depends on how likely borrowers are to make massively over-collateralized loans

**0xRusowsky**

imo a Medium

**Oot2k**

escalate split frontrunning and access control into own issues

**sherlock-admin2**

> escalate split frontrunning and access control into own issues

You've created a valid escalation!

To remove the escalation from consideration: Delete your comment.

You may delete or edit your escalation comment anytime before the 48-hour escalation window closes. After that, the escalation becomes final.

**0xRusowsky**

- fix: https://github.com/ohmzeus/Cooler/pull/54
- https://github.com/ohmzeus/Cooler/pull/60
- https://github.com/ohmzeus/Cooler/pull/61

**Oot2k**

Following issues are not duplicates of 26 and should be grouped together and treaded as another issue: 16 (https://github.com/sherlock-audit/2023-08-cooler-judging/issues/16) 18 (https://github.com/sherlock-audit/2023-08-cooler-judging/issues/18) 72 (https://github.com/sherlock-audit/2023-08-cooler-judging/issues/72) 99 (https://github.com/sherlock-audit/2023-08-cooler-judging/issues/99) 130 (https://github.com/sherlock-audit/2023-08-cooler-judging/issues/130) 137

(https://github.com/sherlock-audit/2023-08-cooler-judging/issues/137) 150
(https://github.com/sherlock-audit/2023-08-cooler-judging/issues/150) 204
(https://github.com/sherlock-audit/2023-08-cooler-judging/issues/204) 221
(https://github.com/sherlock-audit/2023-08-cooler-judging/issues/221) 243
(https://github.com/sherlock-audit/2023-08-cooler-judging/issues/243) 271
(https://github.com/sherlock-audit/2023-08-cooler-judging/issues/271)

226 -> Invalid

**Oot2k**

Addition: 226 shows attack path and root cause, mentions tokens that are not supported -> sherlock has to decide if valid/invalid 231 is not duplicate of this issue and should be grouped with the other ones mentioned above

**hrishibhat**

Result: Medium Has duplicates The respective set of issues has been separated

**sherlock-admin2**

Escalations have been resolved successfully!

Escalation status:

- Oot2k: accepted

**jkoppel**

Fix confirmed. Sponsor agreed to accept some economic concerns with the fix, but no security concerns were identified.

# Issue M-3: gOhm stuck forever if call claimDefaulted on Cooler directly

Source: https://github.com/sherlock-audit/2023-08-cooler-judging/issues/46

## Found by

castle_chain, detectiveking, evilakela, jkoppel

Anyone can call Cooler.claimDefaulted. If this is done for a loan owned by the Clearinghouse, the gOhm is sent to the Clearinghouse, but there is no way to recover or burn it.

## Vulnerability Detail

1. Bob calls `Clearinghouse.lendToCooler` to make a loan collateralized by 1000 gOhm.
2. Bob defaults on the loan
3. Immediately after default, Eve calls `Cooler.claimDefaulted` on Bob's loan.
4. The gOhm is transferred to the Clearinghouse
5. There is no way to burn or transfer it. (In fact, `defund()` can be used to transfer literally any token *except* gOhm back to the treasury.)

However, the gOhm can now be stolen using the exploit in #1, potentially in the same transaction as when Eve called `Cooler.claimDefaulted()`.

## Impact

Anyone can very easily make all defaulted gOhm get stuck forever.

## Code Snippet

`Cooler.claimDefaulted` sends the collateral to the lender, calls `onDefault`

https://github.com/sherlock-audit/2023-08-cooler/blob/main/Cooler/src/Cooler.sol#L325

`Clearinghouse.onDefault` does nothing

https://github.com/sherlock-audit/2023-08-cooler/blob/main/Cooler/src/Clearinghouse.sol#L265

Although `Clearinghouse.defund()` can be used to send any other token back to the treasury, it cannot do so for gOhm

https://github.com/sherlock-audit/2023-08-cooler/blob/main/Cooler/src/Clearinghouse.sol#L340

## Tool used

Manual Review

## Recommendation

Unsure. Perhaps add a flag disabling claiming by anyone other than `loan.lender`? Or just allow `defund()` to be called on gOhm?

## Discussion

**jkoppel**

This is not a duplicate of #28. #28 involves Clearinghouse.claimDefaulted, but this involves Cooler.claimDefaulted.

**Oot2k**

Not a duplicate

**0xRusowsky**

Despite it is not a duplicate, since gOHM would be stuck in CH instead of the being OHM burn. It wouldn't be a big deal (we could ammend the calculations based on that) because it doesn't have any operational/economical impact as long as that supply is removed from the backing calculations.

On top of that, there is an economical incentive to call it from the CH, as the caller is rewarded.

Disagree with severity, imo at max it should be a medium.

Will think about how to deal with it.

**0xRusowsky**

we will finally add a permissionless `burn` function despite this logic is unlikely to happen

**0xRusowsky**

- https://github.com/ohmzeus/Cooler/pull/57

**jkoppel**

Fix approved.

## Issue M-4: Lender can front-run `rollLoan` and call `provideNewTermsForRoll` with unfavorable terms

Source: https://github.com/sherlock-audit/2023-08-cooler-judging/issues/243

### Found by

0xbepresent, Breeje, banditx0x, cats, deadrxsezzz, detectiveking, evilakela, harisnabeel, james_wu, pep7siup, radevauditor, sandy, ubl4nk Lender can front-run `rollLoan` and result in borrower accepting unfavorable terms.

### Vulnerability Detail

After a loan is created, the lender can provide new loan terms via `provideNewTermsForRoll`. If they are reasonable, the user can then accept them. However this opens up a risky scenario:

1. User A borrows from lender B
2. Lender B proposes new suitable terms
3. User A sees them and calls `rollLoan` to accept them
4. Lender B is waiting for this and sees the pending transaction in the mempool
5. Lender B front-runs user A's transaction and makes a new call to `provideNewTermsForRoll` will an extremely high interest rate
6. User A's transaction now executes and they've accepted unfavorable terms with extremely high interest rate

### Impact

User may get mislead in to accepting unfavorable terms and overpaying interest

### Code Snippet

https://github.com/sherlock-audit/2023-08-cooler/blob/main/Cooler/src/Cooler.sol#L192 https://github.com/sherlock-audit/2023-08-cooler/blob/main/Cooler/src/Cooler.sol#L282

### Tool used

Manual Review

## Recommendation

When calling `rollLoan` let the user pass a parameter consisting of the max interest rate they are willing to accept to prevent from such incidents.

## Discussion

**0xRusowsky**

- https://github.com/ohmzeus/Cooler/pull/63

**jkoppel**

This is moot because rollLoan no longer exists.

**MLON33**

From @0xRusowsky: Cooler says the fix for this issue has been validated by @jkoppel. The protocol team acknowledges this issue: "...he (@jkoppel) validated it afterwards in discord and another issue (#119)."