Maple Finance contest Findings & Analysis Report

2022-04-20

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit contest outlined in this document, C4 conducted an analysis of the Maple Finance smart contract system written in Solidity. The audit contest took place between March 17—March 21 2022.

Wardens

20 Wardens contributed reports to the Maple Finance contest:

1. cccz
2. rayn
3. IllIllI
4. WatchPug (jtp and ming)
5. berndartmueller
6. defsec
7. gzeon
8. CertoraInc (danb, egjlmn1, OriDabush, ItayG, and shakedwinder)
9. robee
10. Dravee
11. 0xNazgul
12. 0xwags
13. 0xkatana
14. Tomio
15. Funen

This contest was judged by LSDan.

Final report assembled by liveactionllama.

Summary

The C4 analysis yielded 2 unique MEDIUM severity vulnerabilities. Additionally, the analysis included 8 reports detailing issues with a risk rating of LOW severity or non-critical as well as 13 reports recommending gas optimizations. All of the issues presented here are linked back to their original finding.

Notably, 0 vulnerabilities were found during this audit contest that received a risk rating in the category of HIGH severity.

Scope

The code under review can be found within the C4 Maple Finance contest repository, and is composed of 7 smart contracts written in the Solidity programming language and includes 1272 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website.

Medium Risk Findings (2)

[M-01] Incorrect implementation of Lender can result in lost tokens

Submitted by cccz

MapleLoanInternals._sendFee should check returnData.length == 32 before decoding, otherwise if it returns bytes data, the abi.decode will return 0x20, result in lost tokens.

Proof of Concept

[MapleLoanInternals.sol#L332-L344]

This contract can test that when the function returns bytes data, abi.encode will decode the return value as 0x20.

```
pragma solidity 0.8.7;
contract A{
    address public destination;
    uint256 public number;
    function convertA() external{
        (bool su,bytes memory ret )= address(this).call(abi.enco
        number = ret.length;
        destination = abi.decode(ret, (address));
    }
    function ret() public returns(bytes memory){
        return "0x74d754378a59Ab45d3E6CaC83f0b87E8E8719270";
    }
}
```

Recommended Mitigation Steps

```
function _sendFee(address lookup_, bytes4 selector_, uint256 amo
    if (amount_ == uint256(0)) return true;

    ( bool success , bytes memory data ) = lookup_.call(abi.enco

+       if (!success || data.length != uint256(32)) return false

    address destination = abi.decode(data, (address));

    if (destination == address(0)) return false;

    return ERC20Helper.transfer(_fundsAsset, destination, amount
    }
```

[lucas-manuel (Maple Finance) confirmed, but disagreed with Medium severity and commented]:

> This technically is true, but I consider this informational as this is an extreme edge case. The poolDelegate and mapleTreasury values are almost guaranteed to never change, and if they did it would be to an audited Maple implementation which would never use bytes memory to return the address.

> This is a cool finding and we will address, but it is informational.

[JGcarv (Maple Finance) resolved]:

> Fix: [Check data length (maple-labs/loan#154)]

[LSDan (judge) commented]:

> Disagree with sponsor on the rating. This has external requirements and assets are not at direct risk, but it could result in a loss of funds.

[M-02] Processes refinance operations may call malicious code by re-created refinancer contract

Submitted by rayn

When an attacker (borrower) proposes a new term, the attacker can let a lender accept the malicious term which the lender doesn't expect.

It uses delegatecall in `_acceptNewTerms` of MapleLoanInternals.sol. Though a lender can manually check refinancer contract before calling `acceptNewTerms` , the attacker (borrower) can still re-create a malicious contract on same address before the lender is calling `acceptNewTerms` , and trigger malicious code by delegatecall in `_acceptNewTerms` .

Proof of Concept

In summary, an attacker can use CREATE2 to re-create a new malicious contract on same address. Here is CREATE2 exploit example: https://x9453.github.io/2020/01/04/Balsn-CTF-2019-Creativity/

1. An attacker (borrower) first deploy a refinancer contract with normal refinance actions to cheat lenders. The refinancer have malicious constructor which can be hidden in inherited contracts.
2. The attacker call `proposeNewTerms`, specifying a refinancer contract, and monitor `acceptNewTerms` in Mempool.
3. When the attacker monitored a lender calls `acceptNewTerms`, then quickly pack these transactions:
    1. Destroy refinancer contract by calling selfdestruct
    2. Use CREATE2 to re-deploy a new refinancer contract with malicious code on same address
    3. The lender calls `acceptNewTerms`
4. Then a lender will execute malicious code of new refinancer contract.

Tools Used

ethers.js

Recommended Mitigation Steps

Also check refinancer contract bytecodes in `_getRefinanceCommitment`:

```
function _getRefinanceCommitment(address refinancer_, uint256 de
    return keccak256(abi.encode(refinancer_, deadline_, calls_,
}

function at(address _addr) public view returns (bytes memory o_c
    assembly {
        // retrieve the size of the code, this needs assembly
        let size := extcodesize(_addr)
        // allocate output byte array - this could also be done
        // by using o_code = new bytes(size)
        o_code := mload(0x40)
        // new "memory end" including padding
```

```
            mstore(0x40, add(o_code, and(add(add(size, 0x20), 0x1f),
            // store length in memory
            mstore(o_code, size)
            // actually retrieve the code, this needs assembly
            extcodecopy(_addr, add(o_code, 0x20), 0, size)
        }
    }
```

lucas-manuel (Maple Finance) disputed and commented:

> Refinancer contracts are vetted by the smart contracts team. Any custom refinancer that is used will be able to have devastating consequences on the Loan as it performs custom delegatecalls. For this reason the assumption is made that Borrowers and Lenders will only use audited immutable Refinancer contracts that are deployed by the Maple Labs smart contracts team, and if not, they must be diligent enough to audit themselves or must accept any consequences.

> This is not a valid issue.

LSDan (judge) commented:

> The report is valid. Given the potential danger involved, you should whitelist the allowable Refinancer contracts to protect your lenders. Issue stands.

lucas-manuel (Maple Finance) disagreed with High severity and commented:

> We're providing smart contracts team validated refinancers to the application for the lenders and borrowers to use. If they choose to use externally developed refinancers (something we want to support as it will allow for highly customizable logic between lenders and borrowers which is a feature not a bug), they must audit themselves.

> If there is a selfdestruct contained in the contract, it can be immediately assumed to be malicious because of the exploit outlined above. Same goes for proxied refinancers and stateful refinancers.

> I do not agree with this being a High Risk issue due to its very high degree of difficulty. The borrower would have to develop a smart contract with this exploit,

> submit it outside of the application, and convince the Pool Delegate to accept the refinancer address also outside of the application without an audit.

> I think that this is an interesting finding and am not discounting it, but I highly disagree with severity.

LSDan (judge) decreased severity to Medium and commented:

> I see your point. This exploit has external requirements and so should be medium, but I do think this represents a significant attack vector should a lender act accidentally against their own best interests.

Low Risk and Non-Critical Issues

For this contest, 8 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by warden IllIllI received the top score from the judge.

The following wardens also submitted reports: WatchPug, defsec, gzeon, CertoraInc, berndartmueller, cccz, *and* robee.

[L-01] Treasury fees are given to the lender on failure, rather than reverting

```
        if (!_sendFee(_mapleGlobals(), IMapleGlobalsLike.mapleTr
            _claimableFunds += treasuryFee_;
        }
```

MapleLoanInternals.sol#L321-L323

[L-02] Inconsistent `approve()` behavior between `ERC20` and `RevenueDistributionToken`

`RevenueDistributionToken` considers an approval value of `type(uint256).max` as ‘allow all amounts’:

```
    if (callerAllowance == type(uint256).max) return;
```

RevenueDistributionToken.sol#L279

whereas `ERC20` considers it as a numerical amount:

```
    _approve(owner_, msg.sender, allowance[owner_][msg.sender] - amo
```

[ERC20.sol#L110](ERC20.sol#L110)

These inconsistences will likely lead to confusion at some point in the future.

[L-03] Incorrect revert string in `setEndingPrincipal()`

```
    require(endingPrincipal_ <= _principal, "R:DP:ABOVE_CURRENT_PRIN
```

Refinancer.sol#L43

It should be `"R:SEP:ABOVE_CURRENT_PRINCIPAL"`.

[L-04] IERC20 should be named IERC20Permit

File: erc20-1.0.0-beta.2/contracts/interfaces/IERC20.sol (lines [4-5](4-5))

There may be cases in the future where you may not want EIP-2612 functionality due to deployment costs, and having the name `IERC20` taken will cause problems.

```
    /// @title Interface of the ERC20 standard as defined in the EIP
    interface IERC20 {
```

[L-05] IERC20 incorrectly includes `PERMIT_TYPEHASH`

`PERMIT_TYPEHASH` is not part of the requirements for EIP-2612, so it shouldn't appear in the interface.

```
    /**
     *  @dev    Returns the permit type hash.
     *  @return permitTypehash_ The permit type hash.
     */
    function PERMIT_TYPEHASH() external view returns (bytes32 permit
```

IERC20.sol#L134-L138
OpenZeppelin has it as a `private constant`: OpenZeppelin/draft-ERC20Permit.sol#L28.

[L-06] Missing checks for `address(0x0)` when assigning values to `address` state variables

File: revenue-distribution-token-1.0.0-beta.1/contracts/RevenueDistributionToken.sol (line 73)

```
    pendingOwner = pendingOwner_;
```

[L-07] Open TODOs

There are many open TODOs throughout the various test files, but also some among the code files

```
./revenue-distribution-token-1.0.0-beta.1/contracts/RevenueDistr
./revenue-distribution-token-1.0.0-beta.1/contracts/RevenueDistr
```

[L-08] Incorrect Natspec

```
     *  @dev    Emits an event indicating that one account has se
```

IERC20.sol#L12

The natspec doesn’t mention that the event is also emitted when `transferFrom()` is called, even though the natspec for `transferFrom()` explicitly mentions it.

[N-01] `_processEstablishmentFees()` should emit events when fee processing fails

```
    function _processEstablishmentFees(uint256 delegateFee_, uint256
        if (!_sendFee(_lender, ILenderLike.poolDelegate.selector, de
            _claimableFunds += delegateFee_;
        }

        if (!_sendFee(_mapleGlobals(), IMapleGlobalsLike.mapleTreasu
            _claimableFunds += treasuryFee_;
        }
    }
```

MapleLoanInternals.sol#L316-L324

[N-02] Multiple `address` mappings can be combined into a single `mapping` of an `address` to a `struct`, where appropriate

File: erc20-1.0.0-beta.2/contracts/ERC20.sol (lines 32-34)

```
    mapping(address => uint256) public override balanceOf;

    mapping(address => mapping(address => uint256)) public override
```

[N-03] Use scientific notation (e.g. `10e18`) rather than exponentiation (e.g. `10**18`)

File: loan-3.0.0-beta.1/contracts/MapleLoanInternals.sol (line 14)

```
    uint256 private constant SCALED_ONE = uint256(10 ** 18);
```

[N-04] `public` functions not called by the contract should be declared `external` instead

Contracts [are allowed](#) to override their parents' functions and change the visibility from `external` to `public`.

File: loan-3.0.0-beta.1/contracts/MapleLoanFactory.sol (lines [16-18](#))

```
    function createInstance(bytes calldata arguments_, bytes32 salt_
        override(IMapleProxyFactory, MapleProxyFactory) public retur
            address instance_
```

[N-05] Use a more recent version of solidity

Use a solidity version of at least 0.8.12 to get `string.concat()` to be used instead of `abi.encodePacked(<str>,<str>)`.

File: erc20-1.0.0-beta.2/contracts/ERC20.sol (line [2](#))

```
    pragma solidity ^0.8.7;
```

[N-06] Typos

`owner => owner_`

[IERC20.sol#L129](#)
[IERC20.sol#L132](#)
[Migrator.sol#L24](#)
[Migrator.sol#L26](#)
[Migrator.sol#L27](#)
[IOwnable.sol#L17](#)

`account => account_`

[IOwnable.sol#L11](#)

`Emits an event => Emitted when`

[IERC20.sol#L12](#)

IERC20.sol#L20

`ERC-2612 => EIP-2612`

IERC20.sol#L4

[N-07] Grammar

Throughout the various interfaces, most of the comments have fragments that end with periods. They should either be converted to actual sentences with both a noun phrase and a verb phrase, or the periods should be removed.

lucas-manuel (Maple Finance) confirmed, but disagreed with severity and commented:

> [L-01] Intentional
> [L-02] We can address, informational
> [L-03] We can address, informational
> [L-04] We are always going to want `permit` , dismissed
> [L-05] We would like to keep this public
> [L-06] `pendingOwner` does not need a zero check as it is a two step process
> [L-07] TODOs is duplicate
> [L-08] Incorrect, Natspec only mentions events emitted in functions
> [N-01] This will be monitored in tenderly, no event needed
> [N-02] Won't implement this
> [N-03] Won't implement this
> [N-04] Has to match visibility of overriden function
> [N-05] Won't implement
> [N-06] Will implement typo changes

> All issues are informational.

JGcarv (Maple Finance) resolved:

> Fix typos (maple-labs/erc20#36)
> Fix typos (maple-labs/mpl-migration#13)
> Fix typos (maple-labs/loan#152)
> Fix revert message (maple-labs/loan#155)

Gas Optimizations

For this contest, 13 reports were submitted by wardens detailing gas optimizations. The report highlighted below by warden team **WatchPug** received the top score from the judge.

The following wardens also submitted reports: berndartmueller, rayn, Dravee, robee, IllIllI, 0xNazgul, gzeon, CertoraInc, 0xwags, 0xkatana, Tomio, *and* Funen.

[G-01] `ERC20.sol#transferFrom()` Do not reduce approval on transferFrom if current allowance is type(uint256).max

Note: suggested optimation, save a decent amount of gas without compromising readability.

The Wrapped Ether (WETH) ERC-20 contract has a gas optimization that does not update the allowance if it is the max uint.

The latest version of OpenZeppelin's ERC20 token contract also adopted this optimization.

ERC20.sol#L109-L113

```
    function transferFrom(address owner_, address recipient_, ui
        _approve(owner_, msg.sender, allowance[owner_][msg.sende
        _transfer(owner_, recipient_, amount_);
        return true;
    }
```

See:

- OpenZeppelin/ERC20.sol#L336
- OpenZeppelin/openzeppelin-contracts#3085

Recommended Mitigation Steps

Change to:

```
    function transferFrom(address owner_, address recipient_, ui
        uint256 currentAllowance = allowance[owner_][msg.sender]
        if (currentAllowance != type(uint256).max) {
            _approve(owner_, msg.sender, currentAllowance - amou
        }

        _transfer(owner_, recipient_, amount_);
        return true;
    }
```

[G-02] Use immutable variables can save gas

Note: suggested optimation, save a decent amount of gas without compromising readability.

ERC20.sol#L25-L26

```
    string public override name;
    string public override symbol;
```

ERC20.sol#L50-L54

```
    constructor(string memory name_, string memory symbol_, uint
        name     = name_;
        symbol   = symbol_;
        decimals = decimals_;
    }
```

In `ERC20.sol`, `name` and `symbol` will never change, use immutable variable instead of storage variable can save gas.

[G-03] Validation can be done earlier to save gas

Note: minor optimation, the amount of gas saved is minor, change when you see fit.

ERC20.sol#L75-L102

```
    function permit(address owner_, address spender_, uint256 am
        require(deadline_ >= block.timestamp, "ERC20:P:EXPIRED")

        // Appendix F in the Ethereum Yellow paper (https://ethe
        // the valid range for s in (301): 0 < s < secp256k1n ÷
        require(
            uint256(s_) <= uint256(0x7FFFFFFFFFFFFFFFFFFFFFFFFFFF
            (v_ == 27 || v_ == 28),
            "ERC20:P:MALLEABLE"
        );

        // Nonce realistically cannot overflow.
        unchecked {
            bytes32 digest = keccak256(
                abi.encodePacked(
                    "\x19\x01",
                    DOMAIN_SEPARATOR(),
                    keccak256(abi.encode(PERMIT_TYPEHASH, owner_
                )
            );

            address recoveredAddress = ecrecover(digest, v_, r_,

            require(recoveredAddress == owner_ && owner_ != addr
        }

        _approve(owner_, spender_, amount_);
    }
```

Check if `owner_ != address(0)` earlier can avoid unnecessary computing when this check failed.

Recommended Mitigation Steps

Change to:

```
    function permit(address owner_, address spender_, uint256 am
        require(deadline_ >= block.timestamp, "ERC20:P:EXPIRED")
        require(owner_ != address(0), "...");

        // Appendix F in the Ethereum Yellow paper (https://ethe
        // the valid range for s in (301): 0 < s < secp256k1n ÷
        require(
            uint256(s_) <= uint256(0x7FFFFFFFFFFFFFFFFFFFFFFFFFFI
            (v_ == 27 || v_ == 28),
            "ERC20:P:MALLEABLE"
        );

        // Nonce realistically cannot overflow.
        unchecked {
            bytes32 digest = keccak256(
                abi.encodePacked(
                    "\x19\x01",
                    DOMAIN_SEPARATOR(),
                    keccak256(abi.encode(PERMIT_TYPEHASH, owner_
                )
            );

            address recoveredAddress = ecrecover(digest, v_, r_,

            require(recoveredAddress == owner_, "ERC20:P:INVALII
        }

        _approve(owner_, spender_, amount_);
    }
```

[lucas-manuel (Maple Finance) commented](#):

> **[G-01]** `ERC20.sol#transferFrom()` **Do not reduce approval on transferFrom if current allowance is type(uint256).max**
> Less clean, not added

> **[G-02] Use immutable variables can save gas**
> Valid, will add

> **[G-03] Validation can be done earlier to save gas**
> Less clean, won’t add

> Acknowledge G-01 and G-03.
> Confirm G-02 is valid.

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top