SYNTHETIX

Solidity Security Review

EtherCollateral Smart Contract

Version: 2.0

February, 2020

Contents

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the latest `EtherCollateral` smart contract, part of the Synthetix platform. This review focused solely on the security aspects of the Solidity implementation of the contract, but also includes general recommendations and informational comments. The more general economic structure of the system and related economic game theoretic attacks on the platform are outside the scope of this assessment.

Disclaimer

Sigma Prime makes all effort but holds no responsibility for the findings of this security review. Sigma Prime does not provide any guarantees relating to the function of the smart contract. Sigma Prime makes no judgements on, or provides any security review regarding, the underlying business model or the individuals involved in the project. This document is based upon a time-boxed analysis of the underlying smart contracts. Statements are not guaranteed to be accurate and do not exclude the possibility of undiscovered vulnerabilities.

Document Structure

The first section provides an overview of the functionality of the (`EtherCollateral` smart contract contained within the scope of the security review. A summary followed by a detailed review of the discovered vulnerabilities is then given, which assigns each vulnerability a severity rating (see Vulnerability Severity Classification), an *open/closed/resolved* status and a recommendation. Additionally, findings which do not have direct security implications (but are potentially of interest) are marked as *informational*. Outputs of automated testing that were developed during this assessment are also included for reference (in the Appendix: Test Suite).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities found within the `EtherCollateral` contract.

Overview

The Synthetix platform aims to produce collateralized stable coins. It uses two types of tokens:

1. **Synths:** Tokens which aim to be stable compared to fiat or crypto currencies (sUSD, sAUD, sEUR, sBTC, etc.);
2. **Synthetix:** Fixed-supply token which receives fees from the exchanging of Synths via synthetix.exchange.

The Synthetix platform has been assessed by Sigma Prime on multiple previous occasions. Since these reviews, the team has considerably revised the system and introduced significant changes to the underlying contracts.

The `EtherCollateral` smart contract allows users to lock Ether (`ETH`) to mint `sETH` (i.e. *Synthetix ETH*), with a collateral ratio of 150%. A minting fee of 50 basis points (0.5%), along with an interest rate of 5% (Annual Percentage Rate). Initially, the supply cap of `sETH` is set to 5000.

The details behind the Ether collateral mechanics are described in a dedicated Synthetix Improvement Proposal (SIP): SIP #35.

Security Review Summary

This review was initially conducted on commit 16cb99f, and targets exclusively the sole file `EtherCollateral.sol`. This contract implements the SIP 35 specification, which was accessed at commit e73479e and is available here. This smart contract inherits the following contracts:

- `Owned`: Smart contract allowing inheriting contracts to implement a 2-step ownership transfer process;
- `Pausable`: Smart contract allowing inheriting contracts to be moved to a *Paused* state;
- `ReentrancyGuard`: *OpenZeppelin* Smart contract preventing re-entrancy vulnerabilities from being exploited.

The manual code-review section of the reports are focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the contracts. Specifically, their internal interactions, intended functionality and correct implementation with respect to the underlying functionality of the Ethereum Virtual Machine (for example, verifying correct storage/memory layout). Additionally, the manual review process focuses on all known Solidity anti-patterns and attack vectors. These include, but are not limited to, the following vectors: re-entrancy, front-running, integer overflow/underflow and correct visibility specifiers. For a more thorough, but non-exhaustive list of examined vectors, see [1, 2].

In prior reviews, Sigma Prime have raised vulnerabilities regarding centralization aspects (i.e. administrative control from the owner account). Synthetix have previously acknowledged and accepted these vulnerabilities, so we omit them from this review and direct the reader to our prior reviews for more information.

Retesting activities targeted commit 7976093. The testing team notes that multiple changes were introduced that are not directly related to this security review. The testing team only performed retesting on the issues shared with the development team in this report.

The testing team identified a total of nine (9) issues during this assessment, of which:

- Two (2) are classified as high risk,
- One (1) is classified as medium risk,
- Three (3) are classified as low risk,
- Three (3) are classified as informational.

All these vulnerabilities have been acknowledged and/or resolved by the development team.

To support this review, the testing team used the following automated testing tools:

- Rattle: `https://github.com/trailofbits/rattle`
- Mythril: `https://github.com/ConsenSys/mythril`
- Slither: `https://github.com/trailofbits/slither`
- Surya: `https://github.com/ConsenSys/surya`

Output for these automated tools is available upon request.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within Synthetix's `EtherCollateral` smart contract. Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Vulnerability Severity Classification.

A number of additional properties of the contracts, including comments not directly related to the security posture of the `EtherCollateral` contract, are also described in this section and are labelled as *"informational"*.

Each vulnerability is also assigned a **status**:

- ***Open:*** the issue has not been addressed by the project team;
- ***Resolved:*** the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk;
- ***Closed:*** the issue was acknowledged by the project team but no further actions have been taken.

Summary of Findings

ID	Description	Severity	Status
SEC-01	Improper Supply Cap Limitation Enforcement	High	Resolved
SEC-02	Improper Storage Management of Open Loan Accounts	High	Resolved
SEC-03	Contract Owner Can Arbitrarily Change Minting Fees and Interest Rates	Medium	Closed
SEC-04	Redundant and Unused Code	Low	Resolved
SEC-05	Single Account Can Capture All Supply	Low	Closed
SEC-06	Insufficient Input Validation	Low	Resolved
SEC-07	Use of Excessive Digits for Variable Assignments	Informational	Resolved
SEC-08	Unused Event Logs	Informational	Resolved
SEC-09	Miscellaneous General Comments	Informational	Resolved

SEC-01	Improper Supply Cap Limitation Enforcement		
Asset	`EtherCollateral.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

The `openLoan()` function does not check if the loan to be issued will result in the supply cap being exceeded. Indeed, line [251], only enforces that the supply cap is not reached **before** the loan is opened.

As a result, any account can create a loan that exceeds the maximum amount of sETH that can be issued by the `EtherCollateral` contract.

Recommendations

Introduce a `require` statement in the `openLoan()` function to prevent the total cap from being exceeded by the loan to be opened.

Resolution

The recommendation above was implemented by the development team.

SEC-02	Improper Storage Management of Open Loan Accounts		
Asset	`EtherCollateral.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: High	Impact: High	Likelihood: Medium

Description

When loans are open, the associated account address gets added to the `accountsWithOpenLoans` array regardless of whether the account already has a loan/is already included in the array (see `storeLoan()` function called by the `openLoan()` function on line [342]).

As a result, consider the following scenario:

- Alice opens `loanA`, Bob opens `loanB`, and Alice opens `loanC` and `loanD`
- The `accountsWithOpenLoans` array will be `[Alice, Bob, Alice, Alice]`
- When Alice closes `loanA`, the function `_removeFromOpenLoanAccounts` doesn't get called, as Alice still has `loanC` and `loanD`
- When Alice closes `loanC` and `loanD` the `accountsWithOpenLoans` array will be `[Alice, Bob, Alice]`

Additionally, it is possible for a malicious actor to create a denial of service condition exploiting the unbound storage array in `accountsSynthLoans` via the following scenario:

1. Use an issue limit of 5000 ETH, and a minimum loan size of 1 ETH (current contract defaults);
2. Issue 1200 loans with 1 ETH as collateral for each one of them, from the same borrower;
3. Try to close loan number `1199`, and watch the transaction fail due to a gas cost higher than the block gas limit (set to 10 million gas, the mainnet network value at the time of writing).

Refer to our test `test/EtherCollateral-gas-tests.js` for a proof-of-concept.

Recommendations

- Consider changing the `storeLoan` function to only push the `account` to the `accountsWithOpenLoans` array if the loan to be stored is the first one for that particular `account`;
- Introduce a limit to the number of loans each account can have.

Resolution

The development team implemented the recommendations above.

SEC-03	Contract Owner Can Arbitrarily Change Minting Fees and Interest Rates		
Asset	`EtherCollateral.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Medium	Impact: Medium	Likelihood: Medium

Description

The `issueFeeRate` and `interestRate` variables can both be changed by the `EtherCollateral` contract `owner` **after** loans have been opened.

As a result, the owner can control fees such as they equal/exceed the collateral for any given loan.

Recommendations

While *"dynamic"* interest rates are common, we recommend considering the minting fee (`issueFeeRate`) to be a constant that cannot be changed by the owner.

Resolution

The development team acknowledged the issue and provided the following response:

"Synthetix reserves the right to change the Interest and minting fee for the benefit of SNX holders and to use as a mechanism to incentivise of disincentivise loan opening and closing. "

SEC-04	Redundant and Unused Code		
Asset	`EtherCollateral.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `_recordLoanClosure()` function returns a boolean (`loanClosed`) which is never used by the calling function (see `_closeLoan()` , line [312]).

Furthermore, since the `_recordLoanClosure()` function is only called via the `_closeLoan()` function, this means that `synthLoan.timeClosed` is always equal to zero (see `require` statement on line [305]). Therefore, the `if` statement on line [357] is redundant and unnecessary.

Recommendations

Consider the following:

- Using the return value of the `_recordLoanClosure()` function or changing the function definition to stop returning `loanClosed` ;
- Removing the `if` statement in line [357];

Resolution

The recommendation above was implemented by the development team.

SEC-05	Single Account Can Capture All Supply		
Asset	`EtherCollateral.sol`		
Status	**Closed:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `EtherCollateral` smart contract does not rely on a `maxLoanSize` to limit the amount of ETH that can be locked for a loan. As a result, a single account can issue a loan that will reach (or exceed, see SEC-01) the total minting supply.

Recommendations

Make sure this behaviour is understood and consider introducing and enforcing a cap (`maxLoanSize`) on the size of the loans allowed to be opened.

Resolution

The development team acknowledged the issue and provided the following response:

"Synthetix accepts this scenario. The supply cap can be increased to allow more loans."

SEC-06	Insufficient Input Validation		
Asset	`EtherCollateral.sol`		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

The `constructor` of the `EtherCollateral` smart contract does not check the validity of the addresses provided as input parameters. It is possible to deploy an instance of the `EtherCollateral` contract with the `synthProxy`, `sUSDProxy` and `depot` addresses set to zero.

Similarly, the effective interest rate can be equal to zero if `interestRate` is set to any value lesser than 31536000 (`SECONDS_IN_A_YEAR`), as `interestPerSecond` will be `null`.

Recommendations

Consider introducing the following `require` statements:

- in the `constructor`:
  - `require _depot != address(0)`;
  - `require _synthProxy != 0`;
  - `_sUSDProxy != address(0)`.
- in the `setInterestRate()` function:
  - `require interestRate > SECONDS_IN_A_YEAR`.

Resolution

The `require` statement in the `setInterestRate()` function suggested above was implemented by the testing team. Furthermore, dependant addresses are now using a dedicated `AddressResolver` contract.

SEC-07	Use of Excessive Digits for Variable Assignments
Asset	`EtherCollateral.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The `issueFeeRate` and `interestRate` variables are assigned with an excessive number digits. This representation is prone to errors and complicates the review process.

Recommendations

To enhance the overall readability of the smart contract, consider replacing the variable assignments on line [33] and line [37] respectively with:

- `5 * SafeDecimalMath.unit() / 100`;
- `5 * SafeDecimalMath.unit() / 1000`.

Resolution

The recommendation above was implemented by the development team.

SEC-08	Unused Event Logs
Asset	`EtherCollateral.sol`
Status	**Resolved:** See Resolution
Rating	Informational

Description

The following log events are declared but never emitted:

- `LogInt`;
- `LogString`;
- `LogAddress`.

Recommendations

Remove these events from the `EtherCollateral` contract.

Resolution

The recommendation above was implemented by the development team.

SEC-09	Miscellaneous General Comments
Asset	`EtherCollateral.sol`
Status	**Resolved:** See **??**
Rating	Informational

Description

This section details miscellaneous findings discovered by the testing team that do not have a direct security implication:

- The following functions can be made `external` for gas saving purposes:
  - `currentInterestOnLoan()`;
  - `calculateMintingFee()`;
  - `accountsWithOpenLoans()`;
  - `openLoanIDsByAccount()`;
  - `getLoan()`;
  - `loanLifeSpan()`.
- The `_getLoanFromStorage()` function can be made a `view` function;
- The `synthLoans` variable in the `_getLoanFromStorage` function can be declared with the `memory` keyword for gas saving purposes;
- In line [L165], `uint` could be changed to `uint256` for consistency;
- Typographic errors:
  - line [32]: `iterest` -> `interest`
  - line [75]: `toke` -> `token`

Recommendations

Ensure that the comments are understood and acknowledged, and consider implementing the suggestions above.

Resolution

The recommendations above were implemented by the development team.

Appendix A Test Suite

A non-exhaustive list of tests were constructed to aid this security review and are given along with this document. The `truffle` framework was used to perform these tests and the output is given below.

```
Contract: EtherCollateral
✓ should deploy successfully with relevant contracts initialized (20ms)
after successful deployment, we expect default values
  ✓ should have a minLoanSize of 1 (23ms)
  ✓ should have no issued synths or loans (24ms)
  ✓ should not have an open loan Liquidation state (13ms)
  ✓ should have a collateralizationRatio of 150% (14ms)
  ✓ should have an issuanceRatio of 66.6666666666666667% (16ms)
  ✓ should have an issueFeeRate of 50 bips (12ms)
  ✓ should have an interestRate of 5% (31ms)
  ✓ should have an issueLimit of 5000 (99ms)
  ✓ should have a minLoanSize of 1 (20ms)
  ✓ should not be in a loanLiquidatio period (12ms)
  ✓ should have a liquidationDeadline is set after 92 days (29ms)
after successful deployment, the owner
  ✓ should be able to set a valid collateralisation ratio (55ms, 33043 gas)
  ✓ should not be able to set up an invalid collateralisation ratio (68ms, 25267 gas)
  ✓ should be able to change the interest rate (51ms, 36585 gas)
  ✓ should be able to change the minting fee (issueFeeRate) (44ms, 29069 gas)
  ✓ should be able to set minLoanSize (41ms, 28695 gas)
  ✓ should be able to set issueLimit (49ms, 28697 gas)
  ✓ should not be able to immediately open loan liquidation (60ms, 23998 gas)
after successful deployment, a non-owner
  ✓ should not be able to set a valid collateralisation ratio (65ms, 23480 gas)
  ✓ should not not be able to set up an invalid collateralisation ratio (46ms, 23480 gas)
  ✓ should not be able to change the interest rate (55ms, 22962 gas)
  ✓ should not be able to change the minting fee (issueFeeRate) (66ms, 23248 gas)
  ✓ should not not be able to set minLoanSize (210ms, 22874 gas)
  ✓ should not be able to set issueLimit (50ms, 22876 gas)
  ✓ should not be able to open loan liquidation (66ms, 23260 gas)
  with enough ETH
    ✓ should be able to open a loan with a value > minLoanSize (118ms, 299721 gas)
    ✓ should be able to open a loan with a value == minLoanSize (97ms, 299721 gas)
    ✓ should not be able to open a loan with a value < minLoanSize (33ms, 29502 gas)
    ✓ should be able to open a loan to the maximum cap (96ms, 299721 gas)
    ✓ should not be able to open a loan that exceeds the maximum cap (129ms, 37121 gas)
    ✓ should be able to open multiple loans (within the issueLimit) (812ms, 2322210 gas)
    ✓ should allow borrower to close a loan (803ms, 1513755 gas)
    ✓ should prevent anyone from opening a loan when the liquidation period is on (86ms,
    299721 gas)
    ✓ should prevent anyone from closing a loan that is not theirs (871ms, 1583429 gas)
    ✓ should prevent anyone from closing a loan when their balance in sETH is insufficient
    (605ms, 1263110 gas)
    ✓ should return valid loan details (101ms, 299721 gas)
    ✓ should compute collateral from loan correctly (83ms, 299721 gas)
    after minLoanSize is increased
      ✓ should not be able to open a loan with a value < the new minLoanSize (34ms, 29502 gas)
      ✓ should be able to open a loan with a value == the new minLoanSize (71ms, 299721 gas)

40 passing (51s)
```

Appendix B Vulnerability Severity Classification

This security review classifies vulnerabilities based on their potential impact and likelihood of occurance. The total severity of a vulnerability is derived from these two metrics based on the following matrix.

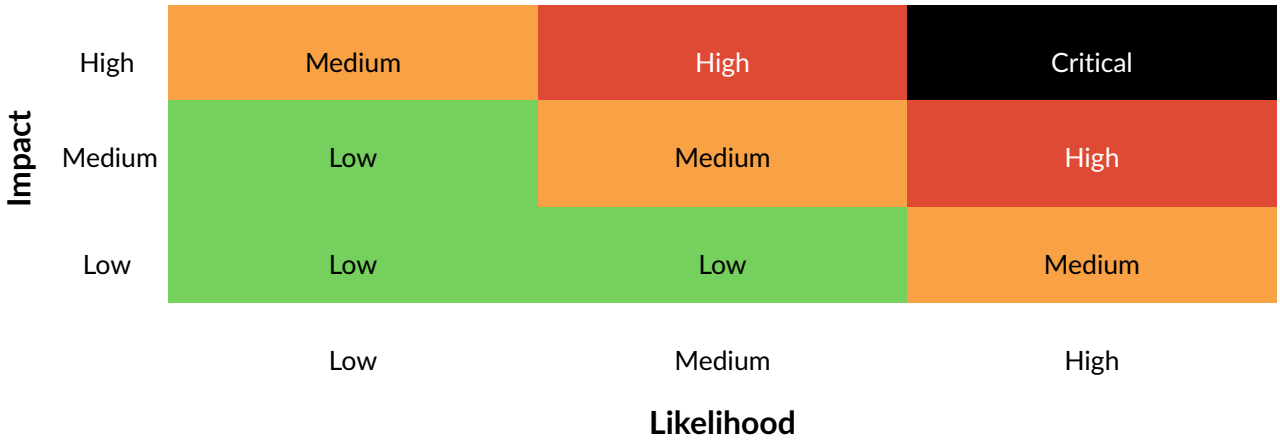

Table 1: Severity Matrix - How the severity of a vulnerability is given based on the *impact* and the *likelihood* of a vulnerability.

References

[1] Sigma Prime. Solidity Security. Blog, 2018, Available: https://blog.sigmaprime.io/solidity-security.html. [Accessed 2018].

[2] NCC Group. DASP - Top 10. Website, 2018, Available: http://www.dasp.co/. [Accessed 2018].