AAVE

AAVE Safety Module

Smart Contract Security Review

Version: 2.0

April, 2023

Contents

Introduction 2
Disclaimer 2
Document Structure 2
Overview 2

Security Assessment Summary 3
Findings Summary 3

Detailed Findings 4

Summary of Findings 5
Attacker May Steals Stake in `stakeWithPermit()` 6
Returning Funds when `totalShares` is Zero May Lead to Denial of Service & Lock Funds 8
Slashing Potentially Causes Rounding Error 9
Incorrect Input of `configureAssets()` May Increase Users' Rewards 10
Unusable Functions Cause Deployment Inefficiency 11
Slashing Does Not Impact Rewards Calculation 12
Potential Abuse of `returnFunds()` to Add Snapshots & Disrupt Voting 13
`permit()` May Fail Silently 14
Initialisers Should be Disabled in the Constructor 15
Miscellaneous General Comments 16

A Test Suite 1 18

B Test Suite 2 19

C Vulnerability Severity Classification 20

Introduction

Sigma Prime was commercially engaged to perform a time-boxed security review of the AAVE smart contracts. The review focused solely on the security aspects of the Solidity implementation of the contract, though general recommendations and informational comments are also provided.

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.

Document Structure

The first section provides an overview of the functionality of the AAVE smart contracts 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 Appendices: Test Suite 1 and Test Suite 2).

The appendix provides additional documentation, including the severity matrix used to classify vulnerabilities within the AAVE smart contracts.

Overview

Aave Safety Module (SM) is a component in Aave's money market ecosystem that mitigates shortfall events. The assessed SM consists of several contracts that will be used as an upgrade to the current implementation of `stkAAVE` and `stkABPT`.

The SM allows users to stake tokens to get rewards called Safety Incentives (SI). The reward amount depends on the reward rate (emission per second) and how long the tokens are staked. An admin account reserves the right to slash the users' staked tokens. The staking shares are transferrable and redeemable at any time after calling cooldown and waiting until cooldown period expires (around one week).

Security Assessment Summary

This review was conducted on the files hosted on the AAVE private repository and were assessed at commit e40e4d9.

The list of assessed contracts is as follows.

1. `AaveDistributionManager.sol`
2. `StakedAaveV3.sol`
3. `StakedTokenV3.sol`
4. `StakedTokenV3.sol`

Note: the OpenZeppelin and Aave libraries and dependencies were excluded from the scope of this assessment.

The target commit was updated to 2e9e5a0 on 10 February 2023.

The manual code review section of the report is focused on identifying any and all issues/vulnerabilities associated with the business logic implementation of the contracts. This includes 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 focused 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].

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

- 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.

Findings Summary

The testing team identified a total of 10 issues during this assessment. Categorised by their severity:

- Critical: 1 issue.
- Medium: 1 issue.
- Low: 2 issues.
- Informational: 6 issues.

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the AAVE smart contracts. 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 gas optimisations, 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
ASM-01	Attacker May Steals Stake in `stakeWithPermit()`	**Critical**	**Resolved**
ASM-02	Returning Funds when `totalShares` is Zero May Lead to Denial of Service & Lock Funds	**Medium**	**Resolved**
ASM-03	Slashing Potentially Causes Rounding Error	**Low**	**Resolved**
ASM-04	Incorrect Input of `configureAssets()` May Increase Users' Rewards	**Low**	**Resolved**
ASM-05	Unusable Functions Cause Deployment Inefficiency	**Informational**	**Resolved**
ASM-06	Slashing Does Not Impact Rewards Calculation	**Informational**	**Closed**
ASM-07	Potential Abuse of `returnFunds()` to Add Snapshots & Disrupt Voting	**Informational**	**Resolved**
ASM-08	`permit()` May Fail Silently	**Informational**	**Resolved**
ASM-09	Initialisers Should be Disabled in the Constructor	**Informational**	**Resolved**
ASM-10	Miscellaneous General Comments	**Informational**	**Resolved**

ASM-01	Attacker May Steals Stake in `stakeWithPermit()`		
Asset	`StakedTokenV3.sol, StakedAaveV3.sol` (commit 2e9e5a0)		
Status	**Resolved:** See Resolution		
Rating	Severity: Critical	Impact: High	Likelihood: High

Description

The function `stakeWithPermit()` does not check the destination address, the `to` parameter. An attacker can frontrun a transaction, copy the signature, then replace the `to` address to their own.

`stakeWithPermit()` enables users to stake tokens to the contract by providing valid ECDSA signatures in `v, r, s`. The signature is validated through `IERC20WithPermit(address(STAKED_TOKEN)).permit()` where the signature should include essential information such as `from, address(this), amount, deadline`. However, the value `to` is not signed and therefore can be replaced without invalidating the signature.

As seen in the following code snippet `to` is not checked when the signature is verified in `permit()`.

```
function stakeWithPermit(
    address from,
    address to,
    uint256 amount,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
  ) external override {
    IERC20WithPermit(address(STAKED_TOKEN)).permit(
      from,
      address(this),
      amount,
      deadline,
      v,
      r,
      s
    );
    _stake(from, to, amount);
  }
```

An attacker can exploit the issue by monitoring transactions in the mempool, then modifying the `to` field of a `stakeWithPermit()` transaction. If the modified transaction is mined first, the shares will be minted to the attacker's address instead of the signer's address yet the underlying tokens will be transferred from the signer's address.

Recommendations

Validate the `to` address to prevent the attack. This can be done by ensuring that `to` is the same address as `from`.

Resolution

A solution has been implemented in PR #14. The solution removes the variable `to` and ensuring tokens will only be minted to the address, `from`, which signs the permit message.

ASM-02	Returning Funds when `totalShares` is Zero May Lead to Denial of Service & Lock Funds		
Asset	`StakedTokenV3.sol` (commit 2e9e5a0)		
Status	**Resolved:** See Resolution		
Rating	Severity: Medium	Impact: High	Likelihood: Low

Description

Due to the lack of check for the `totalShares` in function `returnFunds()`, the `_currentExchangeRate` can be updated to zero, which causes Denial of Service and lock of funds.

Function `returnFunds()` allows any users to return tokens to the system which will alter the exchange rate. When the `totalShares == 0` and a user calls the `returnFunds()` function, the exchange rate will be set to zero on line [**338**]. As a result, no `StakedToken` will be minted for new stakers, and hence, stakers cannot redeem their tokens and the funds will be locked in the contract.

Furthermore, the function `slash()` cannot be called due to a division by zero issue in `previewRedeem()` in line [**318**]. Hence, Denial of Service occurs.

Recommendations

Add a `require` statement in function `returnFunds()` to prevent calling this function when `totalShares` is zero.

Resolution

PR #13 resolves this issue through the following checks. First, `returnFunds()` will revert if either `totalShares` or `amount` is less than a lower bound. Second, a check has been added in `_updateExchangeRate()` to ensure the new rate is non-zero. Finally, `slash()` will revert if the new balance or shares falls below the lower bounds.

ASM-03	Slashing Potentially Causes Rounding Error		
Asset	`StakedTokenV3.sol` (commit e40e4d9)		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Medium	Likelihood: Low

Description

When function `slash()` is called, it updates the value of `_currentExchangeRate`. An odd value of `_currentExchangeRate` potentially causes rounding error. As the result, there will be a token deficit when the last user redeems their staked tokens.

This behaviour contradicts the following statement from line [**556**]: *@dev always rounds up to ensure 100% backing of shares by rounding in favor of the contract*

The cause of this issue is the formula on function `_getExchangeRate()` specifically on line [**566**] as follows:

```
return uint128(((totalShares * TOKEN_UNIT) + TOKEN_UNIT) / totalAssets);
```

In the above formula, `TOKEN_UNIT` is added to the numerator value to provide a small increase to the `_currentExchangeRate`. However, in certain cases, the added amount is insufficient. This is because the value of `totalShares` and `totalAssets` could be much bigger than `TOKEN_UNIT`, such that the addition is negligible.

Recommendations

Rather than using an absolute value to increase the numerator, it is recommended to use percentage. For example, to add .0001% to the numerator, we can do:

```
return uint128(((totalShares * TOKEN_UNIT) * 1000001) / (totalAssets * 1000000));
```

Resolution

The issue has been fixed on commit 2e9e5a0. The formula has been changed to:

```
return
(((totalShares * EXCHANGE_RATE_UNIT) + totalAssets - 1) / totalAssets)
.toUint216();
```

The dynamic addition to the numerator would ensure that the contract will always have slightly more tokens than required.

ASM-04	Incorrect Input of `configureAssets()` May Increase Users' Rewards		
Asset	`AaveDistributionManager.sol` (commit e40e4d9)		
Status	**Resolved:** See Resolution		
Rating	Severity: Low	Impact: Low	Likelihood: Low

Description

Under a certain condition, the function `configureAssets()` can increase the asset index which increases the rewards of users.

Function `configureAssets()` gives a freedom to the `EMISSION_MANAGER` to configure assets through the input, namely `assetsConfigInput` which is a variable of type `AssetConfigInput` that includes `emissionPerSecond`, `totalStaked`, and `underlyingAsset` as its components.

While being a free input, `totalStaked` plays a crucial role in the calculation of the asset index. The asset index is calculated in function `_getAssetIndex()` (on line [**240-242**]) as follows:

```
return ((emissionPerSecond * timeDelta * (10**uint256(PRECISION))) / totalBalance) + currentIndex;
```

It is intuitive that when `emissionPerSecond` increases, the users' rewards will increase. However, this is not the case with `totalStaked` that is identified as `totalBalance` in the above formula. The `EMISSION_MANAGER` could mistakenly submit an incorrect value of `totalStaked` (i.e., much smaller than the reality) and it could potentially increase the asset index significantly. When the asset index increases, the users' rewards will increase based on the formula on line [**209-210**] on function `_getRewards()`.

Recommendations

The `totalStaked` value should be calculated on-the-fly to prevent the `EMISSION_MANAGER` from submitting an incorrect value.

Resolution

The issue has been fixed on commit 2e9e5a0. The value of `totalStaked` is assigned from `totalSupply()` which guarantees input accuracy.

ASM-05	Unusable Functions Cause Deployment Inefficiency
Asset	`StakedTokenV3.sol` (commit 2e9e5a0)
Status	**Resolved:** See Resolution
Rating	Informational

Description

Some functions on `StakedTokenV3.sol` are uncallable under the desirable setting where `ABPT` is the staked token.

The functions are as follows:

1. `claimRewardsAndStake()`
2. `claimRewardsAndStakeOnBehalf()`
3. `stakeWithPermit()`

The root cause of the first two functions' revert is identical. On `_claimRewardsAndStakeOnBehalf()`, the functions first claim the rewards that belong to the caller (through `_claimRewards()` on line [459]) and send it to the contract. Then the rewards that belong to the caller on the contract, are staked through function `_stake()`. In this scenario, function `_stake()` tries to transfer the reward from and to itself (on line [495]). This operation will revert because there is no appropriate approval on `STAKED_TOKEN`.

The third function, `stakeWithPermit()` fails because contract `ABPT` does not have attribute `_nonces` needed to generate a signature.

Contract `StakedTokenV3` is intended to be used as an upgrade for the current implementation of `stkABPT`. Additionally, it also becomes a base contract for `StakedAaveV3` which upgrades `stkAAVE`. While the functions mentioned above do not work under `stkABPT` setting, they are executable on `stkAAVE` environment because the required approval (through function `approve()`) is set on line [82] of `StakedAaveV3`.

Recommendations

Make sure this behaviour is intended. If the mentioned functions are not expected to be callable on `stkABPT`, the testing team recommends moving them from `StakedTokenV3.sol` to `StakedAaveV3.sol`. This would potentially reduce `stkABPT`'s deployment/upgrade cost.

Resolution

The recommendation has been implemented in PR #21 by moving the functions from `StakedTokenV3` to `StakedAaveV3`.

ASM-06	Slashing Does Not Impact Rewards Calculation
Asset	`StakedTokenV3.sol` (commit 2e9e5a0)
Status	**Closed:** See Resolution
Rating	Informational

Description

When slashing occurs, the underlying asset for each share is reduced through recalculated exchange rate. However, the rewards calculation is not impacted by it.

The rewards calculation is based on the number of shares and does not use exchange rate (`_currentExchangeRate`). As the result, slashing has no impact on how much the users will receive as rewards, as long as they have enough shares.

When a user calls `StakedTokenV3.claimRewards()` , it calls `StakedTokenV3._claimRewards()` that passes the user's shares amount to `StakedTokenV2._updateCurrentUnclaimedRewards()` (on line [**421**] of `StakedTokenV3.sol`). This shares amount is then passed to `AaveDistributionManager._updateUserAssetInternal()` as `userBalance` (on line [**264**] of `StakedTokenV2.sol`) or `stakedByUser` (on line [**114**] of `AaveDistributionManager.sol`). The value is then passed to `AaveDistributionManager._getRewards()` to compute the accrued rewards.

Along the process to compute accrued rewards as described above, there is no conversion from shares to underlying assets through exchange rate. As a result, slashing events modify exchange rates but do not impact reward calculations.

Recommendations

Make sure this behaviour is intended.

Resolution

The development team are aware that slashing does not impact reward calculations. It is a deliberate design feature, therefore it does not require a mitigitation.

ASM-07	Potential Abuse of `returnFunds()` to Add Snapshots & Disrupt Voting
Asset	`StakedAaveV3.sol` (commit 2e9e5a0)
Status	**Resolved:** See Resolution
Rating	Informational

Description

Contract `StakedAaveV3` overrides function `_updateExchangeRate()` to record changes of exchange rate in snapshots. The exchange rate snapshots are used in function `_binarySearchExchangeRate()` (through function `_searchByBlockNumber()`) that implements binary search operation.

Intuitively, the more snapshot data to search, the more expensive the operation will be, either paid in gas cost (on-chain transaction) or API calls (for off-chain transaction).

A malicious user can abuse function `returnFunds()` to increase the size of `_exchangeRateSnapshots` data such that any call to function `StakedAaveV3._binarySearchExchangeRate()` would be expensive.

Our experiments indicate roughly 10% increase in gas cost when executing `AaveGovernanceV2.submitVote()` with 1,000 exchange rate snapshots. For the attacker, it costs $O(n)$, while for users/voters, the cost would be $O(k \times log(n))$, where n is the number of snapshots and k is the number of `submitVote()` transactions.

Given that the cost is more significant than the result, such attack with a larger scale is unlikely.

Recommendations

Enforce a minimum amount when calling `returnFunds()` such that the attack would be less economical.

Resolution

The attack surface has been further reduced by enforcing a lower bound on the amount that can be sent to `returnFunds()` , thereby increasing the cost to the attacker, of calling `returnFunds()` multiple times. The resolution can be seen in PR #13.

ASM-08	`permit()` May Fail Silently
Asset	`StakedTokenV3.sol` (commit 2e9e5a0)
Status	**Resolved:** See Resolution
Rating	Informational

Description

It is possible for the external call `permit()` to fail silently, thereby allowing tokens to be transferred without permission.

If `permit()` is called on a contract which does not implement the function `permit()` and has a fallback, the fallback function will be triggered. If the fallback function executes without reverting then it is assumed `permit()` has succeeded.

This is an issue if `STAKED_TOKEN` ad hears to these conditions as `_stake(from, to, amount)` will be called without verifying any of the parameters. `_stake()` will transfer funds out of the `from` address and mint shares for `to` .

```
function stakeWithPermit(
    address from,
    address to,
    uint256 amount,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
  ) external override {
    IERC20WithPermit(address(STAKED_TOKEN)).permit(
      from,
      address(this),
      amount,
      deadline,
      v,
      r,
      s
    );
    _stake(from, to, amount);
  }
```

There are some prominent ERC20 tokens such as `WETH` which are vulnerable to this attack. However, this contract is only intended to be deployed for the tokens $Aave$ and $ABPT$ neither of which fit the criteria required for this attack. Thus, this issue has been raised as informational.

Recommendations

Consider adding an `isPermit` constructor parameter which only allows `stakeWithPermit()` to be called for certain assets.

Resolution

The development team have decided not to fix this issue as it does not apply to either of the tokens in active use, $Aave$ and $ABPT$.

ASM-09	Initialisers Should be Disabled in the Constructor
Asset	`StakedTokenV3.sol` & `StakedAaveV3.sol` (commit 2e9e5a0)
Status	**Resolved:** See Resolution
Rating	Informational

Description

During a proxy-implementation set up the initialiser functions should be disabled in the implementation to prevent abuse.

The most common attack on implementations is causing a self-destruct either directly or via a delegate call. It is not possible for an attacker to cause a self-destruct of the implementation contract in the current protocol. Hence, this issue is raised as informational.

However, there is still the potential for scams or unforseen attacks on implementation contracts.

Recommendations

It is recommended disabling the initialiser in the implementation contract. This can be achieved in the constructor by adding the following line `_disableInitializers()`.

Note the constructor is only executed in the implementation contract and not by the proxy.

Resolution

The initialisers have been disabled by updating the variable `lastInitializedRevision` in the constructor, thereby preventing `initialize()` from being called in the implementation contracts. The update can be seen in PR #18.

ASM-10	Miscellaneous General Comments
Asset	contracts/*
Status	**Resolved:** See Resolution
Rating	Informational

Description

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

1. **Typo.**

   Related Asset(s): AaveDistributionManager.sol (commit 2e9e5a0)

   - On line [104]: Updates the state of an user in a distribution -> Updates the state of a user in a distribution
   - On line [211]: Calculates the next value of an specific distribution index, with validations -> Calculates the next value of a specific distribution index, with validations

2. **Division By Zero on Function `returnFunds().`**

   Related Asset(s): StakedTokenV3.sol (commit 2e9e5a0)

   A `Division or modulo by zero` revert occurs when the following conditions hold:

   - There is no assets staked (zero `assets`).
   - Function `returnFunds()` is called with zero `amount`

   This is caused by function `_getExchangeRate()` that uses the following formula on line [579-581]:

   ```
   return
   (((totalShares * EXCHANGE_RATE_UNIT) + totalAssets - 1) / totalAssets)
   .toUint216();
   ```

   The value `assets + amount` in function `returnFunds()` will be passed to function `_getExchangeRate()` as `totalAssets`. If `totalAssets` is zero, the division by zero occurs. Then, the value conversion through `toUint216()` raises an `Integer overflow` revert.

   The testing team recommends checking that the `totalAssets` and `amount` are not zero when calling function `returnFunds()`

3. **Solidity Version for `ProposalPayload.sol`**

   Related Asset(s): ProposalPayload.sol (commit 2e9e5a0)

   When compiled with Solidity compiler (`solc`) prior to v0.8.16, the compilation failed with the following error:

   ```
   TypeError: Initial value for constant variable has to be compile-time constant...
   ```

   The error above is caused by the codes in line [23-27], where `solc` is unable to read constant values during compilation because the values are taken from another contract or library. On `solc` version 0.8.16, they added some new features, including:

   TypeChecker: Support using library constants in initializers of other constants.

   This means that the contract `ProposalPayload.sol` can only be compiled using `solc` version 0.8.16 or above.

   The testing team recommends modifying the pragma to ^0.8.16.

4. **Contract Code Size Exceeds Limit Without Optimizer.**

   Related Asset(s): StakedTokenV3.sol, StakedAaveV3.sol ProposalPayload.sol (commit 2e9e5a0)

   If the related contracts are compiled without optimizer, the code size of each contract exceeds the limited introduced in EIP-170.

   The testing team recommends utilising optimizer with 200 runs.

5. **Incorrect `@dev` Comment.**

   Related Asset(s): StakedTokenV3.sol (commit 2e9e5a0)

   The comment on line [**436**] indicates that function `_claimRewardsAndStakeOnBehalf()` can only be called by the claim helper. However, this function is also called by function `claimRewardsAndStake()` which is callable by any user.

6. **Claiming Zero Amount**

   Related Asset(s): StakedTokenV3.sol (commit 2e9e5a0)

   Function `claimRewards()` forwards the inputs to `_claimRewards()` to conduct the desired activity. In function `_claimRewards()`, there is a requirement that `amount` should not be zero or otherwise the transaction reverts. However, if `amountToClaim` is zero, the transaction still conducts the following operations:

   (a) Assigns a new value (of zero) to `stakerRewardsToClaim[from]`.

   (b) Transfers zero amount of `REWARD_TOKEN` from `REWARDS_VAULT` to the user.

   (c) Emits `RewardsClaimed` event.

   This indicates that the current approach is gas-inefficient when the user has nothing to claim.

7. **Gas Improvement on Function `_claimRewardsAndStakeOnBehalf()`**

   Related Asset(s): StakedTokenV3.sol (commit 2e9e5a0)

   The implementation of function `_claimRewardsAndStakeOnBehalf()` simply calls two internal functions, namely `claimRewards()` and `_stake()` when the required condition holds. In this case, there are inefficient operations such as calling function `_updateCurrentUnclaimedRewards()` and `_updateUserAssetInternal()` twice and also transferring `STAKED_TOKEN` from and to the same address.

   Our experiment indicates that optimisation by removing unnecessary operations improves the gas consumption by 9.4%.

8. **Accuracy Improvement When Getting Underlying Asset Balance.**

   Related Asset(s): StakedTokenV3.sol (commit 2e9e5a0)

   Function `slash()` and `returnFunds()` retrieve underlying assets by calling function `previewRedeem()` on line [**318,338**]. The result of this function may not be accurate because it uses `_currentExchangeRate` which was rounded up in favor of the contract. To improve accuracy, the testing team recommends calling the underlying balance directly through `STAKED_TOKEN.balanceOf(address(this));`

Recommendations

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

Resolution

The development team have acknowledged these findings, addressing each of issues 1 through 7 and leaving 8 unchanged as it prevents the ability to rescue tokens.

Appendix A Test Suite 1

A non-exhaustive list of tests were constructed to aid this security review and are provided alongside this document. The `brownie` framework was used to perform these tests and the output is given below.
Network: local testnet

```
test_constructor                               PASSED   [1%]
test_initialize                                PASSED   [3%]
test_transfer                                  PASSED   [5%]
test_get_power                                 PASSED   [7%]
test_get_power_slash                           PASSED   [8%]
test_claim_rewards_and_stake                   PASSED   [10%]
test_claim_rewards_and_stake_on_behalf         PASSED   [12%]
test_return_funds_abuse                        PASSED   [14%]
test_stake_with_permit_others                  PASSED   [16%]
test_transfer_pbt                              SKIPPED  [17%]
test_constructor                               PASSED   [19%]
test_initialize                                PASSED   [21%]
test_initialize_zero                           PASSED   [23%]
test_stake                                     PASSED   [25%]
test_stake_with_permit                         PASSED   [26%]
test_stake_with_permit_invalid                 PASSED   [28%]
test_stake_with_permit_others                  PASSED   [30%]
test_redeem                                    PASSED   [32%]
test_redeem_cooldown_on_behalf_of              PASSED   [33%]
test_redeem_on_behalf                          PASSED   [35%]
test_claim_rewards                             PASSED   [37%]
test_claim_rewards_zero                        PASSED   [39%]
test_claim_rewards_redeemed                    PASSED   [41%]
test_claim_rewards_on_behalf                   PASSED   [42%]
test_claim_rewards_and_stake                   XFAIL    (No...)
test_claim_rewards_and_stake_on_behalf         XFAIL    [46%]
test_claim_rewards_and_redeem                  PASSED   [48%]
test_claim_rewards_and_redeem_on_behalf        PASSED   [50%]
test_claim_rewards_slashed                     PASSED   [51%]
test_transfer_claim                            PASSED   [53%]
test_transfer_claim_stake                      SKIPPED  [55%]
test_slash                                     PASSED   [57%]
test_slash_redeem                              PASSED   [58%]
test_slash_all_redeem                          PASSED   [60%]
test_slash_return_funds_redeem                 PASSED   [62%]
test_slash_redeem_multi                        SKIPPED  [64%]
test_return_funds_no_stake                     PASSED   [66%]
test_return_funds_with_stake                   PASSED   [67%]
test_return_funds_zero                         XFAIL    (Integero...)
test_stake_after_return_funds_zero_totalshares XFAIL    [71%]
test_set_max_slashable_percentage              PASSED   [73%]
test_set_cooldown_seconds                      PASSED   [75%]
test_permit                                    PASSED   [76%]
test_delegate_by_type_by_sig                   PASSED   [78%]
test_delegate_by_sig                           PASSED   [80%]
test_delegate_by_type                          PASSED   [82%]
test_delegate                                  PASSED   [83%]
test_get_power                                 PASSED   [85%]
test_get_power_slash                           PASSED   [87%]
test_configure_assets                          PASSED   [89%]
test_claim_rewards_configure_assets            PASSED   [91%]
test_get_user_asset_data                       PASSED   [92%]
test_stake_pbt                                 SKIPPED  [94%]
test_stake_with_permit_pbt                     SKIPPED  [96%]
test_slash_pbt                                 SKIPPED  [98%]
test_set_cooldown_seconds_pbt                  SKIPPED  [100%]
```

Appendix B Test Suite 2

A non-exhaustive list of tests were constructed to aid this security review and are provided alongside this document. The `brownie` framework was used to perform these tests and the output is given below.
Network: Ethereum Mainnet fork

```
test_constructor                         PASSED   [2%]
test_constructor                         PASSED   [4%]
test_initialize                          PASSED   [6%]
test_transfer                            PASSED   [8%]
test_get_power                           PASSED   [10%]
test_get_power_slash                     PASSED   [12%]
test_claim_rewards_and_stake             PASSED   [14%]
test_claim_rewards_and_stake_on_behalf   PASSED   [16%]
test_transfer_pbt                        SKIPPED  [18%]
test_constructor                         PASSED   [20%]
test_initialize                          PASSED   [22%]
test_initialize_zero                     PASSED   [24%]
test_stake                               PASSED   [26%]
test_stake_with_permit                   PASSED   [28%]
test_stake_with_permit_invalid           PASSED   [30%]
test_redeem                              PASSED   [32%]
test_redeem_cooldown_on_behalf_of        PASSED   [34%]
test_redeem_on_behalf                    PASSED   [36%]
test_claim_rewards                       PASSED   [38%]
test_claim_rewards_redeemed              PASSED   [40%]
test_claim_rewards_on_behalf             PASSED   [42%]
test_claim_rewards_and_stake             XFAIL    (No...)
test_claim_rewards_and_stake_on_behalf   XFAIL    [46%]
test_claim_rewards_and_redeem            PASSED   [48%]
test_claim_rewards_and_redeem_on_behalf  PASSED   [51%]
test_claim_rewards_slashed               PASSED   [53%]
test_slash                               PASSED   [55%]
test_slash_redeem                        PASSED   [57%]
test_slash_return_funds_redeem           PASSED   [59%]
test_slash_redeem_multi                  SKIPPED  [61%]
test_return_funds_no_stake               PASSED   [63%]
test_return_funds_with_stake             PASSED   [65%]
test_return_funds_zero                   XFAIL    (Integero...)
test_set_max_slashable_percentage        PASSED   [69%]
test_set_cooldown_seconds                PASSED   [71%]
test_permit                              PASSED   [73%]
test_delegate_by_type_by_sig             PASSED   [75%]
test_delegate_by_sig                     PASSED   [77%]
test_delegate_by_type                    PASSED   [79%]
test_delegate                            PASSED   [81%]
test_get_power                           PASSED   [83%]
test_get_power_slash                     PASSED   [85%]
test_configure_assets                    PASSED   [87%]
test_claim_rewards_configure_assets      PASSED   [89%]
test_get_user_asset_data                 PASSED   [91%]
test_stake_pbt                           SKIPPED  [93%]
test_stake_with_permit_pbt               SKIPPED  [95%]
test_slash_pbt                           SKIPPED  [97%]
test_set_cooldown_seconds_pbt            SKIPPED  [100%]
```

Appendix C 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.

		Likelihood		
Impact		Low	Medium	High
	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Low	Low	Medium

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].