PUBLIC

# Code Assessment

## of the Kill Switch Smart Contracts

March 18, 2024

Produced for


by

# Contents

# 1 Executive Summary

Dear all,

Thank you for trusting us to help Sparklend with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Kill Switch according to Scope to support you in forming an opinion on their security risks.

Sparklend implements a switch allowing arbitrary addresses to disable borrowing in case of a depeg of a pegged asset.

The most critical subjects covered in our audit are functional correctness, access control and integration with the core protocol. Security regarding all the aforementioned subjects is high.

The general subjects covered are testing and documentation. Security regarding all the aforementioned subjects is high. However, testing could be improved to test the ability to repay and to top-up.

In summary, we find that the codebase provides a good level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

## 1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 1 |
| • Code Corrected | 1 |
| Low-Severity Findings | 2 |
| • Code Corrected | 1 |
| • Specification Changed | 1 |

# 2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

## 2.1 Scope

The assessment was performed on the source code files inside the Kill Switch repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

| V | Date | Commit Hash | Note |
|---|---|---|---|
| 1 | 11 March 2024 | 0788632748f0faf7eaf3f5a7721fb1e4d0f443ed | Initial Version |
| 2 | 15 March 2024 | 0f0be3f7a0eaf41a04abf9cfc3b50a2315184262 | After Intermediate Report |
| 3 | 18 March 2024 | 1acd6901379bff6f7a59e63b0129d416a9f7161c | Change in trigger logic |

For the solidity smart contracts, the compiler version `0.8.20` was chosen.

The following contracts are in the scope of this review:

```
IKillSwitchOracle.sol
KillSwitchOracle.sol
```

### 2.1.1 Excluded from scope

The Sparklend protocol and all files not explicitly listed above are out of the scope of this review.

## 2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Sparklend implements a switch, disabling further borrowing, that can be triggered by arbitrary users if a registered asset depegs.

Namely, the `KillSwitchOracle` implements the following functionality:

1. `setOracle()`: Allows the owner to register a price feed of a pegged asset and the base asset (e.g. ETH and stETH). A threshold is specified below which `trigger()` will be executable (e.g. 0.95 stETH/ETH as a depeg scenario). Further, it allows for updating the threshold for the oracle.
2. `disableOracle()`: Allows the owner to disable the oracle.

3. `trigger()`: An arbitrary user can set the LTV of each collateral token to 0 to disable borrowing if an oracle indicates a depeg. However, the function can be called only once and a `reset()` is required to allow calling it again.
4. `reset()`: The owner can reactivates the possibility to trigger the switch again.

Further, the contract has ownership-transfer functionality and getter functions to access the set of oracles and the threshold.

### 2.2.1 Changes in version 2

In version 2, borrowing is disabled by setting the corresponding flag in the config. Hence, the LTV remains untouched and borrow-only reserves are not frozen.

### 2.2.2 Changes in version 3

In version 3, the trigger can be executed as long as an oracle is below the execution threshold or if the trigger has been executed previously (without a reset). Thus, a trigger can now be executed multiple times (in contrast to previous versions). As a consequence, `reset()` now disables calling `trigger()` again if no oracle threshold has been reached.

### 2.2.3 Trust Model & Roles

- Owner: Trusted. Can add oracles that could trigger the kill switch. Can further disable further borrowing on Spark Lend. Expected to be trusted to not unnecessarily hinder the protocol's operation, expected to be set to the Spark SubDAO Proxy.
- Users: Untrusted: Can call `trigger()` to disable borrowing on the core. However, that is only possible under certain circumstances.

Note that it is expected that an address exists that will call the `trigger()` function when needed. Namely, in case a pegged token depegs or in case a governance action adds a new asset or similar (see the resolved issue Pending Governance Spells for further details on possible scenarios). Further, it is expected that the `KillSwitchOracle` holds the needed roles in the core, so that it can operate.

# 3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

# 4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

| Likelihood | Impact | | |
|---|---|---|---|
| | High | Medium | Low |
| High | Critical | High | Medium |
| Medium | High | Medium | Low |
| Low | Medium | Low | Low |

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

# 5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 0 |
| Low-Severity Findings | 0 |

# 6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

| | |
|---|---|
| Critical-Severity Findings | 0 |
| High-Severity Findings | 0 |
| Medium-Severity Findings | 1 |

- Disable Borrowing Code Corrected

| | |
|---|---|
| Low-Severity Findings | 2 |

- Pending Governance Spells Code Corrected
- Specification Mismatch Specification Changed

## 6.1 Disable Borrowing

Design Medium Version 1 Code Corrected

*CS-SKW-002*

The main intention of Kill Switch is to put SparkLend into a lockdown mode preventing further borrowing but still allow users to top up collateral and repay/withdraw. While this can be achieved with the current design setting LTV to zero for assets being used as collateral and freezing borrow only assets, SparkLend offers a more direct way to achieve this by simply using `setBorrowEnable()` to disable borrowing for each asset.

Compared to setting LTV for each collateral to zero the differences are:

`ValidationLogic.validateAutomaticUseAsCollateral` will return `false` in case LTV=0 since `ValidationLogic.validateUseAsCollateral` will return the same. This has effects in for example `SupplyLogic.executeSupply`. Namely, if `isFirstSupply` is `true`, the newly supplied token cannot be used as collateral. Hence, top ups can only be done with used collateral tokens. In contrast, `setBorrowingEnabled(asset, false)` does not have this behaviour and allows for topups where the collateral token is new for the user.

---

**Code corrected:**

`setBorrowEnable()` is now used.

## 6.2 Pending Governance Spells

Design Low Version 1 Code Corrected

*CS-SKW-003*

The `KillSwitchOracle` iterates over all markets to perform the required operations to disable borrowing. However, in case governance has plotted a plan in the `DSPause` e.g. to add a new asset to the Sparklend lending market or to change the LTV of an existing asset, a market could be activated after

the trigger has been called due to the permissionless `DSPause.exec()` function. Ultimately, borrowing for such an asset could be enabled.

Governance should be aware that such scenarios could occur.

---

**Code corrected:**

The code has been adjusted so that `trigger()` can be called repeatedly as long as `reset()` has not been called.

## 6.3 Specification Mismatch

Correctness | Low | Version 1 | Specification Changed

*CS-SKW-001*

The README specifies the following:

```
[...]
anyone can permissionlessly trigger to set SparkLend into lockdown mode in which all collateral
assets have their LTVs set to 0 and all borrowable assets are frozen.
[...]
```

However, note that not all borrowable assets are frozen but only the borrow-only assets are frozen (also documented in the comments in code).

The NatSpec description of function `reset()` incorrectly states:

```
Resets the contract, clearing all set oracles and thresholds
```

The implementation of `reset()` only resets the contract.

---

**Specification changed:**

1. The specification now specifies that:

   ```
   [...] lockdown mode [...] prevents new borrows on all assets.
   ```

which is consistent with the changes implemented in Version 2.

2. The NatSpec of `reset()` was corrected and now reads:

   ```
   Resets the trigger, allowing the kill switch to be triggered again.
   ```