PUBLIC

Code Assessment

of the Burners Smart Contracts

November 11, 2024

Produced for

SYMBIOTIC

by

CHAINSECURITY

Contents

1 Executive Summary

Dear all,

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

In this iteration we reviewed the new `BurnerRouter` and its corresponding `BurnerRouterFactory`. `BurnerRouter` is a contract that allows for the redirection of slashed collateral tokens to configurable addresses. It provides the flexibility to change and configure the receiver of slashed tokens.

The most critical subjects covered in our audit are functional correctness and access control. General subjects covered are gas efficiency and trustworthiness. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a high 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	1
• Code Corrected	1
Medium-Severity Findings	1
• Code Corrected	1
Low-Severity Findings	2
• Code Partially Corrected	1
• Acknowledged	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 Burners 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	12 Aug 2024	a7adb9975416a64324afd664dc72a380246265b8	Initial Version
2	26 Aug 2024	ab43c8dbfc9ac353737452acca949c1a0af9267c	After Intermediate Report
3	28 Oct 2024	7896712bb14f56d0c145dda0fcb69f879492acd7	Add Burner Router
4	06 Nov 2024	60a5eb87f196b62d48406b45cc3b19ef4be0e906	After Report with Router

For the solidity smart contracts, the compiler version `0.8.25` was chosen.

In the first review, the following contracts below were in scope:

```
contracts:
    AddressRequests.sol
    SelfDestruct.sol
    UintRequests.sol
    burners:
        ETHx_Burner.sol
        mETH_Burner.sol
        rETH_Burner.sol
        sfrxETH_Burner.sol
        swETH_Burner.sol
        wstETH_Burner.sol
interfaces:
    IAddressRequests.sol
    IUintRequests.sol
    burners:
        ETHx:
            IETHx_Burner.sol
            IStaderConfig.sol
            IStaderStakePoolsManager.sol
            IUserWithdrawalManager.sol
        mETH:
            IMETH.sol
            IStaking.sol
            ImETH_Burner.sol
        rETH:
            IRocketTokenRETH.sol
```

```
            IrETH_Burner.sol
        sfrxETH:
            IFraxEtherRedemptionQueue.sol
            IsfrxETH_Burner.sol
        swETH:
            ISwETH.sol
            ISwEXIT.sol
            IswETH_Burner.sol
        wstETH:
            IWithdrawalQueue.sol
            IWstETH.sol
            IwstETH_Burner.sol
```

In the current assessment of version 3, only the following contracts were in scope and reviewed:

```
contracts:
    router:
        BurnerRouter.sol
        BurnerRouterFactory.sol
interfaces:
    router:
        IBurnerRouter.sol
        IBurnerRouterFactory.sol
```

2.1.1 Excluded from scope

Generally, only the files mentioned above are in scope. Note that the external systems are out of scope and that they are assumed to work correctly and as expected.

2.2 System Overview

This system overview describes the second received version (Version 2) 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.

Symbiotic implements a set of burner contracts that allow for unstaking LSTs to then burn the underlying ETH. A burner, in Symbiotic's core, is the Vault's target address to transfer the slashed collateral to. Note that the provided contracts serve as a default go-to option. However, other burners could be used.

Generally, burners offer a `triggerBurn` that finalizes burning the underlying token of the collateral (here: ETH). For most protocols, a `triggerWithdrawal` is required to prepare for the burning of the underlying token (e.g. withdrawal request). Note that for the burners the signatures of these functions can be distinct. Even in the case of equal function signatures, the meaning of the function parameters might differ. The details for the individual burners are described below.

2.2.1 ETHx Burner

The `ETHx_Burner` burns an ETHx (Stader) balance.

`triggerWithdrawal` creates one withdrawal request given a maximum withdrawal amount in ETHx. Note that the maximum withdrawal amount can be computed off-chain so that the local bound conditions

are satisfied (assuming no exchange rate changes occur until the transaction arrives). Note that in case the given amount exceeds the balance, the full balance will be used. Requests are created with `requestWithdraw`.

`triggerBurn` finalizes the withdrawal request with the request ID `requestId`. For ETHx that is performed with `claim`.

Note that the ETHx Burner is multicallable to ensure that multiple requests can be created with one function call.

2.2.2 mETH Burner

The `mETH_Burner` burns a mETH (Mantle) balance.

`triggerWithdrawal` creates one withdrawal request. Note that it creates a request with the full balance held by the contract. Note that there is no maximum amount per request and that trying to request less than the minimum will lead to a revert (as expected). Requests are created with `unstakeRequest`.

`triggerBurn` finalizes the withdrawal request with the request ID `requestId`. For Mantle that is performed with `claimUnstakeRequest`.

2.2.3 rETH Burner

The `rETH_Burner` burns a rETH (Rocket Pool) balance.

`triggerBurn` finalizes burns rETH. Namely, note that this is done with `burn`.

Further, note that Rocket Pool's design does not require users to make requests. Thus, it is expected that there eventually will be an excess ETH balance so that rETH can be burned.

2.2.4 sfrxETH Burner

The `sfrxETH_Burner` burns a sfrxETH (Frax) balance.

`triggerWithdrawal` creates one withdrawal requests. Note that it creates a request with the full balance held by the contract. Note that there is no maximum amount per request. Requests are created with `enterRedemptionQueueViaSfrxEth`.

`triggerBurn` finalizes the withdrawal request with the request ID `requestId`. For Frax that is performed with `burnRedemptionTicketNft`.

2.2.5 swETH Burner

The `swETH_Burner` burns a swETH (Swell) balance.

`triggerWithdrawal` creates withdrawal requests. Note it tries to create as many maximum requests as possible, given Swell's maximum withdraw amount. If the remaining balance would exceed the minimum withdrawal amount, another request will be created. Namely, requests are created with `createWithdrawRequest`.

`triggerBurn` finalizes the withdrawal request with the request ID `requestId`. For swETH that is performed with `finalizeWithdrawal`.

2.2.6 wstETH Burner

The `wstETH_Burner` burns a wstETH (Lido) balance.

`triggerWithdrawal` unwraps wstETH to stETH and creates withdrawal requests. Note it tries to create as many maximum requests as possible, given Lido's maximum withdraw amount. If the remaining balance would exceed the minimum withdrawal amount, another request will be created. Namely, requests are created with `requestWithdrawals`. Note that there might remain an outstanding stETH

balance in the contract. That will however be possible to burn in the next `triggerWithdrawal` execution.

`triggerBurn` finalizes the withdrawal request with the request ID `requestId`. For stETH that is performed with `claimWithdrawal`.

2.2.7 Changes in Version 3

Version 3 introduces the `BurnerRouter` and its corresponding `BurnerRouterFactory`. `BurnerRouter` is a contract that allows for the redirection of slashed collateral tokens to configurable addresses. It provides the flexibility to change and configure the receiver of slashed tokens.

When a slashing event occurs, the Vault transfers the slashed tokens to the `BurnerRouter` contract. After the transfer, the `onSlash` function is called on the `BurnerRouter`. It determines the appropriate address for redirection based on the configured receivers, and it records the amount to be redirected to that address. The actual transfer of the slashed collateral is performed by calling the `triggerTransfer()` function, which transfers the accumulated balance to the designated receiver.

Receiver addresses can be set at different levels: global (`globalReceiver`), network-specific (`networkReceiver`), or operator-specific (`operatorNetworkReceiver`). The receiver resolution follows a hierarchy where the operator-specific receiver takes precedence over the network-specific receiver, which in turn takes precedence over the global receiver

Delayed updates are implemented using pending variables (e.g., `pendingGlobalReceiver`) with timestamps to ensure that changes to receivers and the delay parameter are subject to a configurable delay before taking effect. Changes are proposed using functions `setGlobalReceiver`, `setNetworkReceiver`, `setOperatorNetworkReceiver`, and `setDelay` which store the new values in the pending variables. After the delay period has elapsed, these changes can be finalized by calling the corresponding `accept` functions (`acceptGlobalReceiver`, `acceptNetworkReceiver`, `acceptOperatorNetworkReceiver`, and `acceptDelay`).

We assume all collateral tokens to be trusted and properly tested before added. Testing should ensure that the token has no potentially problematic properties like e.g., deflationary tokens.

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:

- Security: Related to vulnerabilities that could be exploited by malicious actors
- 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	2

5.1 Expensive Storage Operations

Design Low Version 3 Acknowledged

CS-SYMB-BRN-014

The `BurnerRouter` uses storage pointers in most places. Operations with these pointers are quite gas inefficient as they are storage operations.

Acknowledged:

The Symbiotic has acknowledged the issue but has decided to keep the code unchanged, accepting it as a trade-off with other design considerations.

5.2 ETHx DoS by Potential Rounding Errors

Correctness Low Version 1 Code Partially Corrected

CS-SYMB-BRN-002

Under certain conditions rounding errors in Stader's `previewDeposit` and `previewWithdraw` functions might lead to reverts in the system.

Note that `getMaxWithdrawAmount` and `getMinWithdrawAmount` will be converted to ETHx amounts. Namely, the range of values to withdraw will be ranging from `previewDeposit(getMinWithdrawAmount)` to `previewDeposit(getMaxWithdrawAmount)`. To ensure proper rounding, 1 is added to the first expression ("increasing the minimum amount") while the second one is rounded down. Thus, values will be between (and including the boundaries) `previewDeposit(getMinWithdrawAmount)+1` and `previewDeposit(getMaxWithdrawAmount)`.

Note that the check on Stader's side will do the following comparisons:

1. `getMinWithdrawAmount <= previewWithdraw(previewDeposit(getMinWithdrawAmount)+1)`
2. `previewWithdraw(previewDeposit(getMinWithdrawAmount)+1) <= getMaxWithdrawAmount`
3. `getMinWithdrawAmount <= previewWithdraw(previewDeposit(getMaxWithdrawAmount))`
4. `previewWithdraw(previewDeposit(getMaxWithdrawAmount)) <= getMaxWithdrawAmount`

Note that 1 and 4 can be shown effectively to be correct. Now, for 2, assume perfect rounding. Then, the left-hand-side will be `getMinWithdrawAmount + 1`. For 3, assume in both preview-functions a rounding error. Assume that the rounding error is 1 wei, we end up with result `getMaxWithdrawAmount-1`.

Now assuming that `getMinWithdrawAmount == getMaxWithdrawAmount`, it is trivially shown that 2 and 3 do not necessarily hold.

Note that further, one can state that the range of values should be ranging from `previewWithdraw(previewDeposit(getMinWithdrawAmount)+1)` to (inclusive) `previewDeposit(getMaxWithdrawAmount)`. Assuming the equality above, it becomes evident that rounding errors are improperly handled as otherwise would would need to hold.

Code partially corrected:

In Version 2 of the protocol the withdrawal is no longer split between withdrawal request. Instead one withdrawal request with `maxWithdrawalAmount` is created, where `maxWithdrawalAmount` is the maximum amount of ETHx that can be withdrawn. The Symbiotic enforces that the value passed to the function `maxWithdrawalAmount` is such that:

1. previewWithdraw(maxWithdrawalAmount) <= maxETHWithdrawAmount
2. previewWithdraw(maxWithdrawalAmount + 1) > maxETHWithdrawAmount

This ensures that all values passed to the function are within the correct range as long as `getMinWithdrawAmount<getMaxWithdrawAmount`. If both values are equal, the Symbiotic's code will revert.

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	1

- Missing onlyOwner Modifier on setDelay Function Code Corrected

Medium-Severity Findings	1

- DoS for ETHx Burner Code Corrected

Low-Severity Findings	0
Informational Findings	2

- Burner Contract Ignores Return Value Code Corrected
- Implementation Can Be Initialized Code Corrected

6.1 Missing onlyOwner Modifier on setDelay Function

Correctness High Version 3 Code Corrected

CS-SYMB-BRN-006

The `setDelay` function lacks the `onlyOwner` modifier, allowing any user to call it. This means that:

- **Unauthorized Delay Changes:** Any user can set the delay to a new value.
- **Denial of Service (DoS):** Users can cancel pending delay changes or repeatedly set new delays, causing a denial of service against the contract owner. This prevents the owner from making legitimate delay updates and disrupts the contract's intended operation.

Code corrected:

In Version 4 an `onlyOwner` modifier is added to the `setDelay` function to restrict access to the contract owner.

6.2 DoS for ETHx Burner

Security Medium Version 1 Code Corrected

CS-SYMB-BRN-001

By creating many withdrawal requests on behalf of the ETHx burner, the ETHx burner could be DoSed due to the impossibility of claiming such requests and due to the enforcing of a maximum number of unprocessed withdrawal requests.

In detail, ETHx' `UserWithdrawalManager.requestWithdraw` will only succeed as long as a maximum number of requests for a given user is respected. Further, the function allows anyone to withdraw to anyone and thus increases the recipient's number of unprocessed requests. Assuming this, consider the following example:

1. An attacker creates 1000 withdrawal requests for the Burner with the minimum amount (0.1 ETH cost - not including gas).
2. `triggerWithdrawal` will always fail due to the imposed maximum number of withdrawals.
3. `triggerBurn` for the given request IDs will always fail due to `_removeRequestId` reverting for untracked request IDs.

Ultimately, the ETHx Burner could be DoSed.

Code corrected:

In Version 2, `triggerBurn` does not revert when the request ID is untracked. This allows the withdrawal request created by the attacker to be finalized, thereby preventing the DoS attack.

6.3 Implementation Can Be Initialized

Informational | Version 3 | Code Corrected

CS-SYMB-BRN-009

To deploy the `BurnerRouter`, the Symbiotic uses a minimal proxy design. As such, the implementation contract is not supposed to hold any state. To ensure this, the Symbiotic disables the initializer function in other contracts, such as the `BurnerRouterFactory` (inheriting from `Entity`), by calling the `_disableInitializers` function in the constructor.

However, the `BurnerRouter` has an empty constructor, which means anyone can initialize the implementation contract. While this is not critical, as the implementation contract cannot be self-destructed, it allows writing to dirty state.

Code corrected:

Symbiotic has added a constructor to `BurnerRouter` that is calling the `_disableInitializers` function in Version 4.

6.4 Reset Pending Changes Is Undocumented INFO

Version 3 | Specification Changed

CS-SYMB-BRN-007

In the `BurnerRouter` contract, when a pending change exists, calling `setDelay` or `_setReceiver` with the same value as the pending value resets the delay.

This behavior is undocumented and can be perceived as unintuitive. Users might not expect that re-setting the same value will reset the pending changes' timestamp.

Specification changed:

Symbiotic has extended specification to explain the effects of calling `setDelay` or `_setReceiver` with pending values.

6.5 Burner Contract Ignores Return Value

Informational **Version 1** **Code Corrected**

CS-SYMB-BRN-003

The ETHx `UserWithdrawManager` contract returns the request ID on `requestWithdraw`. However, the `ETHx_Burner` contract ignores this return value and computes the request ID itself from the last request ID. In the current implementation, both methods yield the same result, but a contract upgrade to the `UserWithdrawManager` that changes the way the withdrawal ID is computed could break the `ETHx_Burner` contract.

Code corrected:

In Version 2, the `ETHx_Burner` contract uses the return value of `requestWithdraw`.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Accept Events Lack Receiver Addresses

Informational **Version 3** **Acknowledged**

CS-SYMB-BRN-008

The accept functions (`acceptGlobalReceiver`, `acceptNetworkReceiver`, `acceptOperatorNetworkReceiver`, and `acceptDelay`) emit events upon accepting pending receiver changes. However, these accept events (`AcceptGlobalReceiver`, `AcceptNetworkReceiver`, `AcceptOperatorNetworkReceiver`, and `AcceptDelay`) do not include the new receiver addresses in their event parameters. Including the receiver addresses in the events would enhance transparency, making it easier for observers to track which receivers have been updated through event logs.

Acknowledged:

Symbiotic has acknowledged the issue, but has decided to keep the code unchanged. They have provided the following justification:

```
It is possible to get such information from the previous event.
```

7.2 Frontrunning Batch Operations

Informational **Version 1** **Acknowledged**

CS-SYMB-BRN-004

`triggerBurnBatch` takes multiple IDs as input. Note that such a call could be front-run with a `triggerBurn` call to make the full call revert. Consider the following example:

1. `triggerBurnBatch` is called with IDs 1, 2, 3, 4 and 5.
2. However, it is front-run with `triggerBurn` with ID 5.
3. Now, `triggerBurnBatch` reverts. 1, 2, 3 and 4 have not been executed.
4. Ultimately, gas was wasted on the execution as the function call has only reverted on ID 5.

Note that technically `triggerBurn` could be front-run, too. However, the result would the same as the goal (ID x has been processed) has been reached. In contrast, for the batch function, the goal has not been reached and gas was forcefully wasted.

Acknowledged:

Client has acknowledged the issue.

7.3 Inconsistent Trust Model

Informational **Version 3** **Acknowledged**

CS-SYMB-BRN-010

The level of trust that networks must place in the `BurnerRouter` owner is not clearly defined.

On one hand, if the networks fully trust the `BurnerRouter` owner, implementing delays for updating receivers might be unnecessary, as the owner is expected to act in the networks' best interests.

Conversely, if the `BurnerRouter` owner is not fully trusted, networks need a way to verify that the router is not acting maliciously. They must ensure that the receivers are correctly set and that the slashed funds are appropriately handled—such as being genuinely burned—rather than being redirected back to the Vault owner or any unauthorized party.

The ability of the `BurnerRouter` owner to change receivers at any time (even with a delay) introduces uncertainty. Networks would need to regularly monitor the `BurnerRouter` contract for any changes to the receivers or listen for specific events to detect malicious updates. This continuous verification process places an additional operational burden on the networks.

Acknowledged:

Symbiotic has provided the following reasoning for their design decision:

```
We don’t force vault curators, networks, or users to trust the owner of the router
(the owner can be anyone including some committees with curators and networks inside).
We just provide a component that can be extended or locked
(via the owner’s ownership renouncing) to a desirable configuration.
```

7.4 Misleading Name of Network Operator Mapping

Informational **Version 3** **Acknowledged**

CS-SYMB-BRN-011

The receivers of funds can be defined for each network and operator in a mapping called `operatorNetworkReceiver`.

```
mapping(address network => mapping(address operator => address receiver)) public operatorNetworkReceiver;
```

However, the mapping maps from network to operator, and not the other way around as the name suggests.

Acknowledged:

Symbiotic has provided the following reasoning for their design decision:

```
We follow the same namings as in the core modules.
```

7.5 Missing Events

Informational | Version 3 | Acknowledged

CS-SYMB-BRN-012

The `onSlash` function does not emit any events when it updates receiver balances. Emitting an event within this function could be useful for monitoring and allowing observers to know when `triggerTransfer` should be called.

Acknowledged:

Symbiotic answered:

> `We have an onSlash event in each core module and believe that it is enough.`

7.6 rETH and ETHx Related Addresses Might Change

Informational | Version 1 | Acknowledged

CS-SYMB-BRN-005

rETH's common practice is to store addresses in a storage contract. Similarly, ETHx's practice is to store addresses in a config contract. For both, the respective protocol's codebase indicates that addresses should be retrieved on-chain. However, the burners do not do that.

However, it is rather unlikely that the rETH token's address will change (e.g. due to other integrations). Similarly, the likelihood of ETHx changing is similarly low (e.g. as the contracts are upgradeable). Further, it is unspecified how such a migration should be handled properly by integrators.

Acknowledged:

Client is aware of this behavior, but has decided to keep the code unchanged.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Burner Considerations

Note Version 1

The burners integrate with external protocols. Below are some considerations:

1. There is a finalization possibility. Ultimately, if no funds flow in, an unburnable remainder might remain in the contract.
2. The rETH burner could technically estimate the amount it could burn. Nevertheless, estimating it off-chain and passing it as a parameter works as well.
3. Similarly, the rETH burner might never be able to burn (as there is no request mechanism, but it is relied upon that that enough funds are available at all times).
4. The sfrxETH burner does not enforce a minimum withdraw amount. Thus, it could be possible to create dusty withdrawal requests for which there is no incentive at all to burn them. However, there is no impact to this.

8.2 Cancelled mETH Withdrawal Requests Are Not Deleted

Note Version 1

mETH withdrawal requests are stored in the `UnstakeRequestsManager` contract. An admin can delete unfinalized requests, sending the funds back to the originator. The `mETH_Burner` contract will recycle these funds into a new withdrawal request on the next call to `triggerWithdrawal`. However, the request IDs for the cancelled requests will remain stored in the `_requestIds` set. Users must be aware that the `requestIds` function might return request IDs that are not processable.

8.3 More Could Be Withdrawn

Note Version 1

The wstETH, ETHx and swETH burner create multiple withdrawal requests on `triggerWithdrawal`. While the current "withdrawal allocation" creates maximum withdrawals and optionally one remainder withdrawal (if the protocols' minimum amount can be respected), an alternative solution could drop the balance to zero more often.

For example, assume a maximum amount and a minimum amount of 32 and 19 respectively. Assume an amount of 70 to be withdrawn. For the current implementations, two requests would be created with 32 amounts, totaling the withdrawals to 64. However, an alternative solution could create one 32 and two 19 withdrawals so that the end balance would drop to zero.

However, also note that this might unnecessarily create more withdrawal requests than needed.

8.4 Security Considerations for Integrating the Router

Note **Version 3**

As the protocol is permissionless, any project integrating with a `BurnerRouter` should consider the following points:

1. The delay of the Router is a critical parameter. If the delay is short (i.e. less than the epoch duration in the Vault), then funds can be redirected to a new receiver before the slashing has occurred.
2. Any transfer of funds to the Router must be immediately followed by a call to `onSlash`. If another caller gains control over the execution, they could account these funds towards another receiver then intended.
3. Integrators must be aware of any potential transfer restrictions of the collateral token (e.g., whitelist, address `0` restriction) and must ensure that the receiver will not be set to a restricted address.