SatLayer BVS ERC20 Staking

Smart Contract Security Assessment

August 9, 2025

ABSTRACT

Dedaub was commissioned to perform a security audit of SatLayer's staking and slashing of ERC20 tokens.

BACKGROUND

SatLayer is a restaking infrastructure enabling stakers to provide cryptoeconomic security to multiple validation services simultaneously. The focus of this audit is a newly developed ERC20 liquid staking system. ERC4626-compliant tokenized vaults are used where users deposit ERC20 assets that are delegated to registered operators who validate for validated services.

The core architecture includes a registry managing bilateral service-operator relationships and slashing parameters, vaults implementing asynchronous ERC7540 redemptions with withdrawal delays for security, and a router coordinating slashing processes with guardrail oversight. The protocol's security model relies on economic penalties where misbehaving operators have vault assets transferred to affected services, with historical parameter tracking preventing retroactive rule changes and resolution windows allowing operator disputes before asset liquidation.

SETTING & CAVEATS

This audit report mainly covers the Solidity contracts of the satlayer-bvs repository at commit `5a0877730a98fd0dc57ff95b75236d822bef506b`.

Audit Start Date:	**July 30, 2025**
Report Submission Date:	**August 9, 2025**

2 auditors worked on the following contracts:

```
contracts/src/
├── interface/
│   ├── ISLAYRegistryV2.sol
│   ├── ISLAYRewardsV2.sol
│   ├── ISLAYRouterSlashingV2.sol
│   ├── ISLAYRouterV2.sol
│   ├── ISLAYVaultFactoryV2.sol
│   └── ISLAYVaultV2.sol
├── MerkleProof.sol
├── RelationshipV2.sol
├── SLAYBase.sol
├── SLAYRegistryV2.sol
├── SLAYRewardsV2.sol
├── SLAYRouterV2.sol
├── SLAYVaultFactoryV2.sol
└── SLAYVaultV2.sol
```

The audit's main target is security threats, i.e., what the community understanding would likely call "hacking", rather than the regular use of the protocol. Functional correctness (i.e. issues in "regular use") is a secondary consideration. Typically it can only be covered if we are provided with unambiguous (i.e. full-detail) specifications of what is the expected, correct behavior. In terms of functional correctness, we often trusted the code's calculations and interactions, in the absence of any other specification. Functional correctness relative to low-level calculations (including units, scaling and quantities returned from external protocols) is generally most effectively done through thorough testing rather than human auditing.

PROTOCOL-LEVEL CONSIDERATIONS

ID	Description	STATUS
P1	Self-slashing	**ACKNOWLEDGED**

Malicious operators can register services, and use these to self-slash before other services can perform slashing.

Consider that such an operator can use `SLAYRegistryV2.registerAsService` to register a service, and create an active relationship with it. Then when it performs a slashable action for another service, it can preempt slashing with a slashing request and lock from its own Sybil service. In this manner, the operator could offset fully the effects of slashing.

ID	Description	STATUS
P2	Slashing Race Condition	**ACKNOWLEDGED**

Operators who have active relationships with services are liable to slashing, based on a maximum percentage specific to the service (represented in mbips, see `SlashParameter.maxMbips`). Consider that an operator has an active relationship with multiple services, and more than one service attempts to slash the operator. This creates a race condition, since slashing is not done proportionally to the current assets of the operator. The service who manages to lock the slash request first (`lockSlashing`) gets a percentage of the original assets of the operator's vaults, while the second gets a percentage of the original assets of the vaults reduced by the amount locked by the first service.

With P1, this behaviour can be exploited by a malicious operator to only partially self-slash, letting other services succeed in slashing. That the subsequent slashing is successful increases the odds of this malicious behaviour going unnoticed.

VULNERABILITIES & FUNCTIONAL ISSUES

This section details issues affecting the functionality of the contract. Dedaub generally categorizes issues according to the following severities, but may also take other considerations into account such as impact or difficulty in exploitation:

Category	Description
CRITICAL	Can be profitably exploited by any knowledgeable third-party attacker to drain a portion of the system's or users' funds OR the contract does not function as intended and severe loss of funds may result.
HIGH	Third-party attackers or faulty functionality may block the system or cause the system or users to lose funds. Important system invariants can be violated.
MEDIUM	Examples: • User or system funds can be lost when third-party systems misbehave. • DoS, under specific conditions. • Part of the functionality becomes unusable due to a programming error.
LOW	Examples: • Breaking important system invariants but without apparent consequences. • Buggy functionality for trusted users where a workaround exists. • Security issues which may manifest when the system evolves.

Issue resolution includes "dismissed" or "acknowledged" but no action taken, by the client, or "resolved", per the auditors.

CRITICAL SEVERITY:

[No critical severity issues]

HIGH SEVERITY:

ID	Description	STATUS
H1	Funds locked upon Guardrail rejection in slashing workflow	**ACKNOWLEDGED**

***Acknowledgement**: A vault can have multiple stakers. If the funds are returned after the `_guardrail` contract rejects a slashing request, the returned funds will be considered as a donation to all the stakers of the vault, even if the stakers become stakers only since the locking of the funds. This current behaviour is intended to avoid this scenario.*

The developers expect this behaviour to be phased out eventually, along with the use of `_guardrail`.

Slashing is handled through a three stage approach in `SLAYRouterV2`:

1. A service requests slashing of an operator with `requestSlashing`.
2. The service can cancel the request (`cancelSlashing`), or If the resolution passes it can lock the funds to be slashed (`lockSlashing`). The funds to be slashed are given to the router address.
3. If the funds are locked, the service can finalise the slashing (`finalizeSlashing`), and the router routes the slashed funds to the destination specified in the request.

An important feature of `finalizeSlashing` is that it requires the approval of the `_guardrail` contract to succeed. This contract is set by the owner of the system, to provide final approval or rejection of slashing requests. However, if there is a rejection, there is no mechanism to unlock the funds that were locked in step 2 above. In fact, once a request is approved or rejected by a call to `guardrailApprove(bytes32 slashId, bool approve)`, this determination cannot be modified, see below:

`SLAYRouterV2::guardrailApprove:442-448`

```
if (_guardrailApproval[slashId] != 0) {
    revert ISLAYRouterSlashingV2.GuardrailAlreadyApproved();
}
_guardrailApproval[slashId] = approve ? 1 : 2;
```

MEDIUM SEVERITY:

ID	Description	STATUS
M1	Operators and services withdrawal delay mismatch	**RESOLVED**

Resolution: *Resolved in commit a919abc, by adding validation of the withdrawal delay before the final registration step.*

Services can change their minimum withdrawal delay attribute using `SLAYRegistryV2.setMinWithdrawalDelay(..)`, and similarly operators with `SLAYRegistryV2.setWithdrawalDelay(..)`. During the execution of both, it is checked that the new delay is consistent with entities (operators/services) with which there is an active relationship, reverting otherwise.

However, this invariant (*if an operator has an active relationship with a service, the operator's withdrawal delay must not be less than the minimum withdrawal delay of the service*) can be violated. When an active relationship (through `registerServiceToOperator` and `registerOperatorToService`) between the two entities is established, no such check is done on-chain.

LOW SEVERITY:

[No low severity issues]

CENTRALIZATION ISSUES:

It is often desirable for DeFi protocols to assume no trust in a central authority, including the protocol's owner. Even if the owner is reputable, users are more likely to engage with a protocol that guarantees no catastrophic failure even in the case the owner gets hacked/compromised. We list issues of this kind below. (These issues should be considered in the context of usage/deployment, as they are not uncommon. Several high-profile, high-value protocols have significant centralization threats.)

ID	Description	STATUS
N1	Guardrail contract providing final slashing determination	**ACKNOWLEDGED**
	Acknowledgement: *Guardrails are a temporary measure pending further research and development.* For slashing to be finalised, the system requires the approval of a `_guardrail` contract, controlled by the owner (see `SLAYRouterV2._guardrail`). No slashing request can go through without this approval, while currently if it is rejected the slashable amount remains locked (see H1). The behaviour and security of `_guardrail` is critical to the health of the system.	
N2	Owner controls important system variables	**ACKNOWLEDGED**
	Owners of different contracts control important parameters and behaviour: `SLAYRouterV2`, `SLAYBase`, and `SLAYRegisterV2`.	

OTHER / ADVISORY ISSUES:

This section details issues that are not thought to directly affect the functionality of the project, but we recommend considering them.

ID	Description	STATUS
A1	Unnecessary `_guardrail` address check	**RESOLVED**

***Resolution**: Resolved in commit a919abc, by removing the first if-statement.*

In `SLAYRouterV2.guardrailApprove` there are two checks on the address:

`SLAYRouterV2::guardrailApprove:428-433`

```
if (_guardrail == address(0)) {
    revert ISLAYRouterSlashingV2.Unauthorized();
}
if (_msgSender() != _guardrail) {
    revert ISLAYRouterSlashingV2.Unauthorized();
}
```

Given that the second if-statement checks that the caller is the guardrail contract, the first check is redundant since the message sender can never be the zero address.

ID	Description	STATUS
A2	Misnamed error	**RESOLVED**

***Resolution**: Resolved in commit a919abc, by renaming to `GuardrailHasDetermined()`.*

`ISLAYRouterSlashingV2.GuardrailAlreadyApproved()` is used whenever `_guardrail` has made a determination (see `SLAYRouterV2L::443-445`), not necessarily an approval.

ID	Description	STATUS
A3	Maximum active relationships for services and operators	**RESOLVED**

***Resolution**: Resolved in commit [a919abc](), by using two new variables.*

The maximum amount of active relationships between services and operators is governed by the same variable `SLAYRegistryV2._maxActiveRelationships`. This is limiting; having instead two separate variables to constrain the number of operators per service, and the number of services per operator would allow more flexibility to adapt to changing conditions post-deployment.

A4	Operators and services are not mutually exclusive	**ACKNOWLEDGED**

There is no requirement that an operator cannot itself register as a service, or vice-versa. In particular this allows an operator to register itself as a service and create an active relationship with itself, and perform self-slashing (P1).

A5	Permissive compiler pragma	**RESOLVED**

***Resolution**: Upgraded to `^0.8.24`, see [129ee23]().*

The compiler version specified for the contracts is not very strict, `pragma solidity ^0.8.0`. We recommend using the latest version of Solidity, which has no [known bugs]().

DISCLAIMER

The audited contracts have been analyzed using automated techniques and extensive human inspection in accordance with state-of-the-art practices as of the date of this report. The audit makes no statements or warranties on the security of the code. On its own, it cannot be considered a sufficient assessment of the correctness of the contract. While we have conducted an analysis to the best of our ability, it is our recommendation for high-value contracts to commission several independent audits, a public bug bounty program, as well as continuous security auditing and monitoring through Dedaub Security Suite.

ABOUT DEDAUB

Dedaub offers significant security expertise combined with cutting-edge program analysis technology to secure some of the most prominent protocols in DeFi. The founders, as well as many of Dedaub's auditors, have a strong academic research background together with a real-world hacker mentality to secure code. Protocol blockchain developers hire us for our foundational analysis tools and deep expertise in program analysis, reverse engineering, DeFi exploits, cryptography and financial mathematics.