Security Research Labs

Astar lockdrop precompile review

Threat model and hacking assessment report

V1.2, June 4th, 2024

Haroon Basheer haroon@srlabs.de

Rachna Shriwas rachna@srlabs.de

Regina Biro regina@srlabs.de

Abstract. This work describes the result of a thorough and independent security assurance audit of the performed by Security Research Labs. Security Research Labs is a consulting firm that has been providing specialized audit services for Substrate-based blockchains since 2019, including in the Polkadot ecosystem.

During this study, Astar provided access to relevant GitHub repositories and supported the research team effectively. The code of the lockdrop precompile was verified to assure that the business logic of the product is resilient to hacking and abuse.

The research team identified two issues: a benchmarking issue and a memory related issue, leading to potential denial of service and heap overflow respectively. The findings were disclosed and acknowledged with remediations.

In addition to mitigating the issue, Security Research Labs recommends using the runtime environment for benchmarking, utilising heap memory safe types for handling call dispatches and regular code reviews.

Content

1 Disclaimer

This report describes the findings and core conclusions derived from the audit carried out by Security Research Labs within the agreed-on timeframe and scope as detailed in Chapter 2. Please note that this report does not guarantee that all existing security vulnerabilities were discovered in the codebase exhaustively and that following all evolution suggestions described in Chapter 4 may not ensure all future code to be bug free.

2 Motivation and scope

Astar network implemented a precompile dispatch for the lockdrop feature, where users will lock a certain amount of token in smart contracts for a predefined period. The lockdrop mechanism rewards early adopters by paying out tokens proportional to the locked period and the amount of locked tokens. The Astar lockdrop precompile implements the *dispatch_lockdrop_call* function, that dispatches calls on behalf of the SS58 account of the lockdrop account [1].

In this engagement, the code assurance team focused on the security of the precompile implementation of *dispatch_lockdrop_call* function, ensured its logical correctness and verified benchmarking of the dispatch extrinsic call.

During the assessment of the code, security critical parts of the code were identified and security issues in these components were communicated to the development team in the form of GitHub issues [2].

3 Baseline Assurance

3.1 Findings summary

During the analysis of the precompile, Security Research Labs identified two issues (1 info and 1 low severity), which are summarized in Table 1.

Issue	Severity	References	Status
Missing benchmarking for the lockdrop precompile dispatch	Info	[2]	Closed
Unbounded call length limit in lockdrop dispatch call	Low	[3]	Closed

Table 1 Code audit issue summary

3.2 Detailed findings

3.2.1 Missing benchmarking for the lockdrop precompile dispatch

Attack scenario	Static *ref_time* used for weight calculation leading to underestimation of the weights can enable an attacker to perform denial of service
Location	precompiles/dispatch-lockdrop
Tracking	[2]
Attack impact	An attacker may spam and conduct denial of service attacks cheaply in comparison to the actual *weight_to_gas* price
Severity	Info
Status	Closed [4]

The precompile dispatch-lockdrop has un-benchmarked *weight_to_gas* estimation with *ref_time* configured to 1_000_000_000 [5].

```
// Record a fixed amount of weight to ensure there is no free execution
handle.record_cost(Runtime::GasWeightMapping::weight_to_gas(
    Weight::from_parts(1_000_000_000u64, 0),
))?;
```

This estimation doesn't reflect the actual runtime environment and can aid an attacker to spam the chain.

We suggest to appropriately benchmark the precompile dispatch to reflect the accurate *weight_to_gas* estimation for *ref_time* and *POV_size*.

The issue was acknowledged by the Astar team and remediation is currently in progress through collaboration with the Frontier team [6]

3.2.2 Unbounded call length limit in lockdrop dispatch call

Attack scenario	An attacker may create multiple nested calls bloating the call_length before call decoding
Location	precompiles/dispatch-lockdrop
Tracking	[3]
Attack impact	Unbounded call length can aid an attacker to cause heap overflow when call data is moved to the vector
Severity	Low
Status	Closed [7]

The precompile dispatch call uses the *UnboundedBytes* [8] type without any *call_length* being set for the call object parameter. During runtime-call decoding, a stack overflow is prevented through usage of *DecodeLimit* [9] however, a heap overflow might occur even before decoding if an unbounded call with large *call_length* is moved into the u8 vector.

An attacker can use this to create multiple nested calls bloating the *call_length* and cause heap overflow even before call decoding.

We recommend using *BoundedBytes<call_length>* instead of *UnboundedBytes*. A best practice implementation from Moonbeam may be adopted for setting *CallLengthLimit* similar to *GetProposalLimit* [10] and implementing additional guard condition for *call_length* validity similar to *proposal_length* [11] before decoding the call for additional safety.

4 Evolution suggestions

To ensure that precompiles are secure against known and yet undiscovered threats alike, the auditors recommend considering the evolution suggestions and best practices described in this section.

Fix the reported vulnerabilities in a timely manner. Ensure that the reported vulnerabilities from this audit are addressed promptly. Astar has acknowledged these issues, and remediation efforts are currently underway. Given that the lockdrop precompiles have already been merged into the Astar master branch, it's crucial to emphasize the importance of timely resolution. Thereby, it mitigates the risks posed by the reported issues but also prevents the exposure of Astar's broader codebase to potential attack vectors.

Regular code review and continuous fuzz testing. Regular code reviews are recommended to avoid introducing new logic or arithmetic bugs, while continuous fuzz testing can identify potential vulnerabilities early in the development process. Ideally, Astar should continuously fuzz their code on each commit made to the codebase. The substrate-runtime-fuzzer [12] (which uses Ziggy [13], a fuzzer management tool) can be a good starting point.

5 Bibliography

[1] [Online].Available: https://github.com/AstarNetwork/Astar/pull/1142.

[2] [Online].Available: https://github.com/AstarNetwork/dappstaking-v3-audit/issues/7.

[3] [Online].Available: https://github.com/AstarNetwork/dappstaking-v3-audit/issues/8.

[4] [Online].Available: https://github.com/AstarNetwork/dappstaking-v3-audit/issues /7#issuecomment-2034174383.

[5] [Online].Available:https://github.com/AstarNetwork/Astar/blob/ 282485aa2d50f12f42463bba1d393fce4c57c2a3/precompiles/dispatch-lockdrop/src/lib.rs#L88-L90.

[6] [Online].Available: https://github.com/polkadot-evm/frontier/issues/1348.

[7] [Online].Available: https://github.com/AstarNetwork/Astar/pull/1208.

[8] [Online].Available:https://github.com/AstarNetwork/Astar/blob/ 282485aa2d50f12f42463bba1d393fce4c57c2a3/precompiles/dispatch-lockdrop/src/lib.rs#L74.

[9] [Online].Available:https://github.com/AstarNetwork/Astar/blob/ 282485aa2d50f12f42463bba1d393fce4c57c2a3/precompiles/dispatch-lockdrop/src/lib.rs#L104C8-L104C84.

[10] [Online].Available:https://github.com/moonbeamfoundation/moonbeam/blob/ 7f77c13bd20d33da1fabfe55acef75797f5369f1/precompiles/collective/src/lib.rs#L88.

[11] [Online].Available:https://github.com/moonbeamfoundation/moonbeam/blob/ 7f77c13bd20d33da1fabfe55acef75797f5369f1/precompiles/collective/src/lib.rs#L117.

[12] [Online].Available: https://github.com/srlabs/substrate-runtime-fuzzer.

[13] [Online].Available: https://github.com/srlabs/ziggy.