Security Research Labs

Polymesh Baseline Security Assurance

Threat model and hacking assessment report

V1.0.0, October 19th, 2023

Louis Merlin	louis@srlabs.de
Kevin Valerio	kevin@srlabs.de
Gabriel Arnautu	gabriel@srlabs.de
Regina Biro	regina@srlabs.de

Abstract. This work describes the result of the thorough and independent security assurance audit of the Polymesh blockchain 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, Polymesh provided access to relevant documentation and supported the research team effectively. The code of Polymesh was verified to assure that the business logic of the product is resilient to hacking and abuse.

The research team identified several issues ranging from high severity to low, many of which concerned runtime extrinsics.

In addition to mitigating the remaining open issues, Security Research Labs recommends further enhancing the existing documentation and tests around the Polymesh identity system.

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 6 may not ensure all future code to be bug free.

2 Motivation and scope

Polymesh [1] is a specialized blockchain designed for regulated financial assets like security tokens. This layer 1 platform combines robust identity verification and secure governance features, making it noteworthy for security assessments. While ensuring transparency, Polymesh mandates identity checks for all users—issuers, investors, and node operators alike. The native cryptocurrency, POLYX, serves dual roles: incentivizing network participation and enabling secure governance features.

Like other Substrate-based blockchain networks, the Polymesh code is written in Rust, a memory safe programming language. Substrate-based chains utilize three main technologies: a WebAssembly (WASM) based runtime, decentralized communication via libp2p, and a block production engine.

In a trustless, decentralized environment like a blockchain, security challenges are inherent. Therefore, ensuring availability and integrity is a priority for Polymesh as it depends on its users to be incentivised to participate in the network. As such, a security review of the project should not only highlight the security issues uncovered during the audit process, but also bring additional insights from an attacker's perspective, which the Polymesh team can then integrate into their own threat modeling and development process to enhance the security of the product.

2.1 Baseline assurance

The first audit of Polymesh blockchain was conducted by Security Research Labs in 2021. Several issues were reported to the developers, most of which were already mitigated, or their risk was accepted [2].

In this current engagement, the audit team focused on the Polymesh runtime and pallet code. Security Research Labs collaborated with the Polymesh team to create an overview containing the runtime modules in scope and their audit priority [3]. The in-scope components and their assigned priorities are reflected in Table 1. During the audit, Security Research Labs used a CIA triad threat model to guide efforts on exploring potential security flaws and realistic attack scenarios.

During the assessment of the code, security critical parts of the code were identified and security issues in these components were communicated to the Polymesh development team in the form of GitHub issues in a private repository.

Repository	Priority	Component(s)	Reference
Polymesh	High	Contracts pallet	[4]
		Sub-identities	
		Settlements	
		NFTs	
		Runtime configuration	
	Medium	Polymesh pallets	

Table 1. In-scope Polymesh components with audit priority

3 Methodology

This report details the baseline security assurance results for the Polymesh network with the aim of creating transparency in three steps: treat modeling, implementation baseline check and finally remediation support:

Threat Modeling. The threat model is considered in terms of *hacking incentives*, i.e., the motivations to achieve the goals of breaching the integrity, confidentiality, or availability of Polymesh network nodes. For each hacking incentive, hacking *scenarios* were postulated, by which these goals could be achieved. The threat model provides guidance for the design, implementation, and security testing of Polymesh.

Implementation baseline check. As a second step, the Polymesh implementation was tested for openings whereby any of the defined hacking scenarios could be executed.

To effectively review the Polymesh codebase, Security Research Labs derived the code review strategy based on the threat model that was established as the first step [3]. For each identified threat, hypothetical attacks were developed and mapped to their corresponding threat category, as outlined in Chapter 4.

Prioritizing by risk, the codebase was assessed for present protections against the respective threats and attacks as well as the vulnerabilities that make these attacks possible. For each threat, the auditors:

1. Identified the relevant parts of the codebase, for example the relevant crates and the runtime configuration.

2. Identified viable strategies for the code review. Manual code review, fuzz testing, and tests via static analysis tools were performed where appropriate.

3. Ensured the code did not contain any vulnerabilities that could be used to execute the respective attacks, or otherwise ensured that sufficient protection measures against specific attacks were present.

4. Immediately reported any vulnerability that was discovered to the development team along with suggestions around mitigations.

Security Research Labs carried out a hybrid strategy utilizing a combination of code review and dynamic tests (e.g., fuzz testing) to assess the security of the Polymesh codebase.

While fuzz testing and dynamic tests establish a baseline assurance, the focus of this audit was a manual code review of the Polymesh codebase to identify logic bugs, design flaws, and best practice deviations. The approach of the review was to trace the intended functionality of the runtime modules in scope and to assess whether an attacker can bypass/misuse/abuse these components or trigger unexpected behavior on the blockchain due to logic bugs or missing checks. Since the Polymesh codebase is entirely open source, it is realistic that a malicious actor would analyze the source code while preparing an attack. For the closed-source smart contract that was also in scope, the auditors assumed it would be released to the public before being deployed, hence applied the same principle.

Fuzz testing is a technique to identify issues in code, that handles untrusted input, which in Polymesh's case is extrinsics in the main Polymesh runtime. Fuzz testing works by taking some valid input for a method under test, applying a semi-random mutation to it, and then invoking the method under test again with this semi-valid input. Through repeating this process, fuzz testing can unearth inputs that would cause a crash or other undefined behavior (e.g., integer overflows) in the method under test. The fuzz testing methods written for this assessment utilized the test runtime genesis configuration as well as mocked externalities to execute the fuzz test effectively against the extrinsics in scope.

Remediation support. The final step is supporting Polymesh with the remediation process of the identified issues. Each finding was documented and published with mitigation recommendations. Once the mitigation solution is implemented, the fix is verified by the auditors to ensure that it mitigates the issue and does not introduce other bugs.

4 Threat modeling and attacks

The goal of the threat model framework is to be able to determine specific areas of risk in Polymesh's blockchain system. Familiarity with these risk areas can provide guidance for the design of the implementation stack, the actual implementation of the stack, as well as the security testing. This section introduces how risk is defined and provides an overview of the identified threat scenarios. The *Hacking Value*, categorized into *low*, *medium*, and *high*, considers the incentive of an attacker, as well as the effort required by an attacker to successfully execute the attack. The hacking value is calculated as:

$$Hacking\ Value = \frac{Incentive}{Effort}$$

While *incentive* describes what an attacker might gain from performing an attack successfully, *effort* estimates the complexity of this same attack. The degrees of incentive and effort are defined as follows:

Incentive:

- Low: Attacks offer the hacker little to no gain from executing the threat.
- Medium: Attacks offer the hacker considerable gains from executing the threat.
- High: Attacks offer the hacker high gains by executing this threat.

Effort:

- Low: Attacks are easy to execute. They require neither elaborate technical knowledge nor considerable amounts of resources.
- Medium: Attacks are moderately difficult to execute. They might require bypassing countermeasures, the use of expensive resources or a considerable amount of technical knowledge.

- High: Attacks are difficult to execute. The attacks might require in-depth technical knowledge, vast amounts of expensive resources, bypassing countermeasures, or any combination of these factors.

Incentive and Effort are divided according to Table 2.

Hacking Value	Low incentive	Medium Incentive	High Incentive
High effort	Low	Medium	Medium
Medium effort	Medium	Medium	High
Low effort	Medium	High	High

Table 2. Hacking value measurement scale.

Hacking scenarios are classified by the risk they pose to the system. The risk level, also categorized into low, medium, and high, considers the hacking value, as well as the damage that could result from successful exploitation. The risk of a threat scenario is calculated by the following formula:

$$Risk = Damage \times Hacking\ Value = \frac{Damage \times Incentive}{Effort}$$

Damage describes the negative impact that a given attack, performed successfully, would have on the victim. The degrees of damage are defined as follows:

Damage:

- Low: Risk scenarios would cause negligible damage to the Polymesh blockchain.
- Medium: Risk scenarios pose a considerable threat to Polymesh's functionality as a blockchain.
- High: Risk scenarios pose an existential threat to Polymesh's functionality.

Damage and Hacking Value are divided according to Table 3.

Risk	Low hacking value	Medium hacking	High hacking
Low damage	Low	Medium	Medium
Medium damage	Medium	Medium	High
High damage	Medium	High	High

Table 3. Risk measurement scale

After applying the framework to the Polymesh system, different threat scenarios according to the CIA triad were identified.

The CIA triad describes three security promises that can be violated by a hacking attack, namely confidentiality, integrity, availability.

Confidentiality:

Confidentiality threat scenarios concern sensitive information regarding the blockchain network and its users. Native tokens are units of value that exist on the blockchain - confidentiality threat scenarios include, for example, attackers abusing information leaks and gaining control over network participants' private keys. The private key can then be used to steal user funds or to impersonate them.

Integrity:

Integrity threat scenarios threaten to disrupt the functionality of the entire network by undermining or bypassing the rules that ensure that Polymesh transactions/operations are fair and equal for each participant. Undermining Polymesh's integrity often comes with a high monetary incentive, like for example, if an attacker can double spend or mint tokens for themselves. Other threat scenarios do not yield an immediate monetary reward, but rather, could threaten to damage Polymesh's functionality and, in turn, its reputation. For example, if an attacker can modify an asset's identifier or description, they trick users into investing in fraudulent assets and cause reputational damage to Polymesh.

Availability:

Availability threat scenarios refer to compromising the availability of the network to process normal transactions. Important threat scenarios regarding availability for blockchain systems include Denial of Service (DoS) attacks on participating nodes, stalling the transaction queue, and spamming.

Table 4 provides a high-level overview of the hacking risks concerning Polymesh with identified example threat scenarios and attacks, as well as their respective hacking value and effort. The complete list of threat scenarios identified along with attacks that enable them are described in the threat model deliverable [3]. This list can serve as a starting point to the Polymesh developers in order to guide their security outlook for future feature implementations. By thinking in terms of threat scenarios and attacks during code review or feature ideation, many issues can be caught or even avoided altogether.

For Polymesh, the auditors attributed the most hacking value to the integrity class of threats. Undermining the integrity of the Polymesh chain directly impacts their claim of being a regulation-ready and institution-grade blockchain [1] and could have devastating consequences for the entities being impacted by such an attack.

Security promise	Hacking value	Example threat scenarios	Hacking effort	Example attack ideas
Confiden-tiality	Medium	- Compromise a user's private key	High	- Targeted attacks to compromise a user's private key
Integrity	High	- Take over validator nodes - Fabricate false transactions - Mint securities in an unregulated fashion	Medium	- Slash honest executors by submitting false offence reports - Abuse missing check to mint token to the attacker's account

Availa-bility	High	- Stall block production - Spam the blockchain with bogus transactions	Low	- Crash validator nodes

Table 4. Risk overview. The threats for Polymesh's blockchain were classified using the CIA security triad model, mapping threats to the areas: (1) Confidentiality, (2) Integrity, and (3) Availability.

5 Baseline Assurance

5.1 Findings summary

During the analysis of the Polymesh code, Security Research Labs identified six issues for Polymesh, which are summarized in Table 5.

Each finding to the Polymesh project described here was shared with Polymesh developers in a dedicated private GitHub repository [4] as an issue.

Issue	Severity	References	Status
Arithmetic underflow can lead to invalid CUSIP identifier	High	[5]	Closed
Creating many child identities will lead to storage bloat	Medium	[6]	Closed
Overflow in weight definition could lead to DoS	Medium	[7]	Closed
Undervalued weight could cause block execution timeouts	Medium	[8]	Closed
Committee disapproval will leave the call in storage	Low	[9]	Closed
Lack of custom *WeightInfo* for *pallet_contract*	Low	[10]	Closed

Table 5 Code audit issue summary

5.2 Detailed findings

5.2.1 Arithmetic underflow can lead to invalid CUSIP identifier

Attack scenario	**Create an asset with an invalid identifier**
Location	primitives
Tracking	[5]
Attack impact	Attackers can create assets that do not conform to CUSIP.
Severity	High
Status	Closed

Assets in Polymesh are made to represent real-world assets. They contain metadata such as the CUSIP or CINS identifier of the securities issued by the company (these

are North American standards). These identifiers must comply with a specific verifiable format to be considered valid.

During creation of an asset, an arithmetic overflow during verification of the CUSIP or CINS identifier can lead to an invalid identifier being accepted and used. This is detrimental to the correctness of the financial information stored on the chain, which is one of the core promises of the Polymesh blockchain.

This issue was fixed by handling the verification with safe arithmetic functions [11].

5.2.2 Creating many child identities will lead to storage bloat

Attack scenario	Bloat storage to decrease usability
Location	runtime
Tracking	[6]
Attack impact	Attacker can hinder blockchain usability at low cost
Severity	Medium
Status	Closed

On the Polymesh blockchain, a verified primary identity acting as an identified organization can decide to hand over control of certain permissions they have to specific trusted identities (such as subsidiaries or employees).

However, creating a child identity on-chain is not priced properly. Because the existential deposit is set to 0, only the transaction costs must be paid to create an identity. These identities take up quite some space in the blockchain's underlying database storage, once created.

An attacker could batch-create as many of these child identities as they can, adding gigabytes to the storage, thus increasing lookup execution times, and making it harder to operate a node and use the chain.

The mitigation that has been implemented by the Polymesh team consists of charging a one-time fee when creating a child identity. This solves the issue by making such an attack prohibitively expensive. However, the fee has been set to zero for now, until the team decides the threat warrants a fee to be set.

5.2.3 Overflow in weight definition could lead to DoS

Attack scenario	Committee majority might trigger node DoS with lengthy extrinsic
Location	Pallets/committee
Tracking	[7]
Attack impact	Attack could cause widespread node denial-of-service
Severity	Medium
Status	Closed

In Substrate-based blockchains, the weight of an extrinsic is used to properly apply fees to a transaction to reflect its execution time and to schedule it inside of a block.

An arithmetic overflow in the weight computation of the *vote_or_propose* extrinsic could be abused to force nodes to execute a time-consuming extrinsic without

properly paying for its weight. This extrinsic would lead to a timeout in the validator nodes trying to execute a block containing the transaction.

This would require any of the chain's committees to approve the extrinsic in question, which greatly reduces the severity of this issue. Still, this would allow a rogue majority in a minor committee to cause denial of service for everybody until every validator node applies a patch.

Using safe arithmetic in the weight computation code fixed this issue [11].

5.2.4 Undervalued weight could cause block execution timeout

Attack scenario	Attacker abuses flawed weight calculation to trigger DoS with cheap batch calls
Location	pallets/identity
Tracking	[8]
Attack impact	Nodes might experience block timeouts and transaction delays
Severity	Medium
Status	Closed

In Substrate-based blockchains, the weight of an extrinsic is used to properly apply fees to a transaction to reflect its execution time and computational complexity; and to schedule it inside of a block.

The weight benchmarking for the *add_authorization* extrinsic within the identity pallet is inadequate and does not account for the worse-case scenario, leaving it susceptible to abuse. Exploiting this flaw, an attacker can execute a series of low-cost batch calls to overwhelm the network, resulting in block timeouts and transaction delays.

The mitigation involved properly accounting for the works-case scenario inside of the benchmarking code [11]. There are other parts of the code that were also modified that were related to this issue, such as in the settlement pallet [12].

5.2.5 Committee disapproval will leave the call in storage

Attack scenario	Attacker creates many heavy calls that will never be removed from storage
Location	pallets/committee
Tracking	[9]
Attack impact	Bloat of the chain storage and hindered usability
Severity	Low
Status	Closed

The *committee* pallet allows users to submit proposals, which are then voted upon by members of the committee.

Such proposals are runtime calls that will have arbitrary size. In order to prevent these calls from accumulating, the code will remove them from storage once they have been processed.

The bug that can be abused here is that the proposal that is voted against, instead of just expiring, is never removed from storage due to a programming error. An attacker could then propose many heavy calls to any committee, that will be voted against and left in the blockchain's storage in perpetuity.

Leaving unused items in storage is bad practice, as they will bloat the chain storage, hindering useability for node operators and increasing lookup times for the whole chain.

The mitigation was to patch the error in the code, effectively removing the call that was voted against from the blockchain's storage [11].

5.2.6 Lack of custom WeightInfo for pallet_contract

Attack scenario	Incorrect weights might lead to incorrect extrinsic prioritization
Location	runtime
Tracking	[10]
Attack impact	Inaccurate extrinsic weights could cause performance issues
Severity	Low
Status	Closed

In Substrate-based blockchains, the weight of an extrinsic is used to properly apply fees to a transaction and to schedule it inside of a block.

Benchmarks most often rely on specific chain configuration values. Hence, it is good practice to run the same pallet's benchmarks individually for each chain it is used for.

Polymesh uses a modified version of *pallet_contracts*, where the weights were benchmarked using the substrate-node template. This configuration does not match the actual Polymesh runtime, potentially resulting in imbalanced extrinsic execution and bad transaction prioritization.

Mitigation involved executing the *pallet_contracts* benchmarks for the Polymesh runtime and importing the results from this computation [13].

6 Evolution suggestions

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

6.1 Core improvement suggestions to improve security posture

Document and clarify the identity system. Polymesh's identity system is highly permissioned and very restrictive, which shields the blockchain from many vulnerabilities. Nevertheless, care should be taken in order to clarify and document the identity primitives and pallets. Many methods contain duplicated logic, which could be the source of security vulnerabilities in the future. These duplications should be removed, and the code simplified where possible.

Include concrete examples in documentation. The existing documentation could benefit from improvements as it is very high-level. The team has created exhaustive

SDK-level examples, but the project lacks rust-level technical examples. These examples could include processes outlining which extrinsics to use for typical interactions with Polymesh, such as creating a child identity, managing portfolios and adding claims. They could also list the different possibilities available to users when using these extrinsics, as they can be complex and include many options.

Strengthen testing process. The infrastructure with regards to testing is opaque. This will create blind spots in security testing of certain functionalities and might lead to bugs being introduced that would have been caught otherwise. Code coverage is already measured by the Polymesh CI/CD tools, but the team should make sure the most critical parts of the system (identity, claims, governance, assets) are sufficiently covered by the existing tests, and take steps to create more tests if it is not the case. Besides, some additional code-comments in the existing tests would greatly help internal and external entities trying to collaborate on the project.

6.2 Further recommended best practices

Regularly review the code and continuously fuzz test. Security Research Labs recommends having regular code reviews by security-focused professionals (internal or external to Polymesh) to avoid introducing new logic or arithmetic bugs. Continuous fuzz testing can also identify potential vulnerabilities early in the development process. Ideally, Polymesh should continuously fuzz their code on each commit made to the codebase. The Polkadot codebase provides a good example of multiple fuzzing harnesses [14]. In addition to these, Security Research Labs has released some example substrate runtime fuzzer harnesses [15].

Regularly update. New releases of Substrate may contain fixes for critical security issues. Since Polymesh is a product that heavily relies on Substrate, updating to the latest version as soon as possible whenever a new release is available is recommended. Security Research Labs recommend paying special attention to security fixes, specifically Substrate related ones, as well as setting up a review process for every new main version of Substrate to be incorporated into the update process of Polymesh.

Continue improving best practice review process. Finding vulnerabilities is only the start of the remediation process. To ensure that no issue goes unfixed, Security Research Labs recommends continuing to improve upon the team's review process, establishing a set of guidelines and criteria for the review to ensure consistency and standardization.

Avoid forking pallets and libraries. While it may not always be possible to contribute upstream to popular pallets, forking should be avoided in most cases. Having a fork of a known pallet makes getting upstream fixes a manual process and harder to maintain. Moreover, the adapted fix for the forked pallet may not mitigate the underlying security issue, or it may introduce new vulnerabilities.

7 Bibliography

[1] [Online]. Available: https://polymesh.network/.

[2] [Online]. Available: https://github.com/PolymeshAssociation/Polymesh/blob/616de82c2f3df82b8ca59c516b041106641fb8e2/audit/Polymesh%20Audit%20-%20SRLabs.pdf.

[3] [Online]. Available: https://securityresearchlabs.sharepoint.com/:x:/s/Polymath/EalL-yfX1HxArABy8hxNV_kBnFKH1vljBoar0s9d4xcWvA?e=9HsdYt.

[4] [Online]. Available: https://github.com/PolymeshAssociation/Polymesh/.

[5] [Online]. Available: https://github.com/PolymeshAssociation/polymesh-audit-2023/issues/1.

[6] [Online]. Available: https://github.com/PolymeshAssociation/polymesh-audit-2023/issues/5.

[7] [Online]. Available: https://github.com/PolymeshAssociation/polymesh-audit-2023/issues/3.

[8] [Online]. Available: https://github.com/PolymeshAssociation/polymesh-audit-2023/issues/4.

[9] [Online]. Available: https://github.com/PolymeshAssociation/polymesh-audit-2023/issues/2.

[10] [Online]. Available: https://github.com/PolymeshAssociation/polymesh-audit-2023/issues/6.

[11] [Online]. Available: https://github.com/PolymeshAssociation/Polymesh/pull/1504.

[12] [Online]. Available: https://github.com/PolymeshAssociation/Polymesh/pull/1521.

[13] [Online]. Available: https://github.com/PolymeshAssociation/Polymesh/pull/1516.

[14] [Online]. Available: https://github.com/paritytech/polkadot-sdk/tree/b13a3187f2b18d1ed1821670f11dc0c70399bb50/polkadot/xcm/xcm-simulator/fuzzer.

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