Security Research Labs

Mangata Baseline Security Assurance

Threat model and hacking assessment report

V1.0, May 12, 2022

Regina Biro regina@srlabs.de

Mostafa Sattari mostafa@srlabs.de

Abstract. This report describes the result of the thorough and independent security assurance audit of the Mangata blockchain platform performed by Security Research Labs. During the audit, Mangata provided access to relevant documentation and supported the research team effectively.

The protocol design and rust implementation of Mangata was verified to assure that the main features and promises such as front-running protection are resilient to hacking and abuse risks and that users can use the platform safely.

The research team identified several issues ranging from low to high severity. Mangata, in cooperation with the auditors, has already mitigated a subset of them and is planning to fix the remaining issues in a timely manner.

In addition to mitigating the remaining open issues, Security Research Labs recommends a revision of the weaknesses pointed out for the Themis protocol, as well as implementing best practices, such as extending the already existing documentation.

Content

1 Motivation and scope

Mangata is a decentralized exchange (DEX) blockchain platform that will be deployed on the Polkadot relay chain as a parachain, allowing users to exchange various assets. Mangata strives to implement effective countermeasures against front-running by shuffling and encrypting the extrinsics.

This review assesses the Mangata blockchain system's existing protections against a variety of **likely hacking scenarios** and **points out the most relevant weaknesses**, all with the goal of improving the protection capabilities of the blockchain system.

Threats that could compromise the Mangata network go far beyond the theft of tokens. Notable hacking scenarios include the potential to undermine trust in the blockchain system by member account takeover, influencing community decisions or bypassing staking requirements.

This report details the baseline security assurance results with the aim of creating transparency in four steps:

Threat Model. The threat model [1] is considered in terms of *hacking incentives*, i.e., the motivation to achieve the goals of breaching the integrity, confidentiality, or availability of nodes in future Mangata systems. For each of the hacking incentives, *hacking scenarios* were postulated, by which these goals could be reached. The threat model provides guidance for the design, implementation, and security testing of Mangata.

Security design coverage check. Next, the Mangata design was reviewed for coverage against relevant hacking scenarios. For each scenario, the following two aspects were investigated:

a. **Coverage**. Is each potential security vulnerability sufficiently covered?

b. **Underlying assumptions**. Which assumptions must hold true for the design to effectively reach the desired security goal?

Implementation baseline check. As a third step, the current Mangata implementation and relevant forked libraries were tested for vulnerabilities whereby any of the defined hacking scenarios could be executed.

Remediation support. The final step is supporting Mangata 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.

Mangata node is based on Substrate [2], a blockchain development framework. Both Mangata and Substrate are written in Rust, a memory safe programming language. Mainly, Substrate works with three technologies: a WebAssembly (WASM) based runtime [3], decentralized communication via libp2p [4], GRANDPA finality gadget [5] and the BABE block production engine [6].

Mangata's node implementation depends on Substrate [7], Moonbeam's *staking* and *crowdloan-reward* pallets [8], the Open Runtime Module Library (ORML) [9] and Cumulus [10].

SRLabs conducted the audit focusing on the components indicated as in scope by Mangata but also interactions with 3rd party libraries such as *libm* [11]. Mangata's online wiki [12] and blog [13] provided the testers a design and implementation overview.

Repository	Priority	Component(s)
mangata-node	High	pallets/basic-authorship pallets/encrypted-transactions pallets/issuance pallets/xyk runtime/src
	Medium	pallets/asset-info pallets/asset-registry
	Low	pallets/sudo-origin
	Not ready for review	pallets/bridge
substrate	High	client/basic-authorship client/basic-authorship-ver client/block-builder client/block-builder-ver client/consensus client/service frame/executive frame/vesting primitives/shuffler primitives/ver
moonbeam	High	pallets/crowdload-rewards pallets/parachain-staking
cumulus	Medium	pallets/aura-ext
open-runtime-module-library	Medium	tokens xcm xtokens

Table 1. Components overview.

2 Methodology

To be able to effectively review the Mangata codebase, a threat-model driven code review strategy was employed. For each identified threat, hypothetical attacks that can be used to realize the threat were developed and mapped to their respective threat category as outlined in Chapter 3.

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

1. Identified the relevant parts of the codebase, for example, the relevant pallets.

2. Identified viable strategies for the code review. Manual code audits, fuzz testing and manual tests were performed where appropriate.

3. Ensured the code did not contain any vulnerabilities that could be used to execute the respective attacks, otherwise, verified 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.

During the audit, a hybrid strategy was carried out utilizing a combination of code review and dynamic (e.g., fuzz testing) to assess the security of the Mangata codebase.

Fuzz testing is a technique to identify issues in code that handles untrusted input, which in Mangata's case is the functions implementing the extrinsics as well as the calculations (e.g., for rewards) within the platform. It works by taking generated valid input for a given method, applying a semi-random mutation to it, and then invoking the tested method again with this semi-valid input. Through repeating this process, the fuzzer can discover inputs that would cause a crash or other undefined behavior (e.g., integer overflows). Fuzzing methods written for this assessment utilized the test runtime genesis configuration as well as mocked externalities.

During the audit, findings were shared via a private Github repository [14]. In addition, weekly jour fixe meetings were held to provide detailed updates and address open questions.

3 Threat modeling and attacks

The goal of the threat model framework is to be able to determine specific areas of risk in Mangata's blockchain system. Familiarity with these risk areas can provide guidance for the design of the implementation stack.

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 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.

A more detailed description of the threat modelling framework is summarized in the threat model shared [1] with Mangata's team.

After applying the framework to the Mangata system, different threat scenarios according to the CIA triad (Confidentiality, Integrity, and Availability) were identified. Table 2 provides a high-level overview of the threat model 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.

Security promise	Hacking value	Example threat scenarios	Hacking effort	Example attack ideas
Confidentiality	Medium	- Decrypt encrypted transactions	High	- Exploit a bug in the transaction encryption implementation
Integrity	High	- Manipulate transactions order in a block to front-run other transactions - Market manipulation - Tamper collator selection - Manipulate pool parameters to increase rewards	Medium	- Liquidity token price manipulation by misbehaving collators - Tampering with the transaction shuffling mechanism - Spamming to achieve front-running - Increase the rewards in the pool configuration
Availability	High	- Censor certain transactions - Stall block production	Low	- Targeted DoS attack against collators - Cheaply fill up blockchain storage

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

4 Findings summary

We identified **11** issues - summarized in Table 3 - during our analysis of the runtime modules in scope in the Mangata codebase that enable the attacks outlined above.

Issue	Severity	References	Status
The VER solution does not protect against front-running	High	[15]	Open
Malicious collators do not face punishment	High	[15]	Open
Integer overflows in the xyk pallet	High	[16]	Mitigated [17]
Spamming risks through free transactions	High	[18]	Mitigated
Malicious collators could spam the chain using unsigned extrinsics	High	[19]	Open
AURA collator selection opens the possibility for DoS attacks	High	[20]	Open
Stack overflow due to missing *DecodeLimit*	High	[21]	Open
Underestimated extrinsic weights	High	[22], [23]	Partially mitigated [24]
Integer overflows in delegation code	Low	[25]	Mitigated [26]
Possible correlation attacks	Low	[27]	Open
The current VED implementation is symmetric	Info	5.1.2	Open, but implementation is in PoC state

Table 3. Issue summary.

5 Detailed findings

This section summarizes the technical analysis of the issues found by SRLabs for Mangata's blockchain implementation.

5.1 Issues concerning the design of the Themis protocol

Mangata introduced the Themis protocol [28] as a main component of their blockchain platform to solve the MEV (Miner Extractable Value) problem, which includes two categories of value extraction performed by malicious collators.

The first category, VER (Value Extraction by Reordering) happens when a block author reorders the transaction in a block to gain profit (e.g., by front-running transactions).

The second category, VED (Value Extraction by Denial) specifies a type of attack where a block author decides to censor certain transactions.

To achieve this goal, the protocol introduces a two-step block production pipeline which performs block building and block execution sequentially. After a doubly encrypted transaction is submitted to the block-builder, they will remove the first layer of encryption and collect and shuffle them into a block which will be decrypted and executed by the block-executors in the next slot. This should prevent users and collators from monitoring on transactions and therefore gaining an unfair economical advantage through front-running.

During the audit of the project, SRLabs found several issues linked to the design and implementation of this protocol.

5.1.1 Value Extraction by Reordering (VER)

The VER scheme introduces transaction shuffling to prevent malicious collators from front-running transactions and other economic attacks [28]. The shuffling logic is implemented by decoupling block production: while block *N* contains its respective extrinsics, they are only executed in block *N+1*, using a Verifiable Random Function (VRF) as a random seed from block N. During the audit, we uncovered several conceptual issues regarding the VER scheme, which are detailed as follows.

The VER solution does not protect against sequential front-running. In case the goal is to run the transaction after another transaction, the current shuffling logic does not protect against it, as the extrinsics are grouped by dispatch order (first the first transactions from each user shuffled, then the second, etc.). Executing buy or sell transactions after a specific transaction was executed on command could also bring direct financial benefit and disproportionate advantage. To achieve such a front-running scenario, an attacker would only need to dispatch a few dummy calls before calling the transaction that should be executed after the victim's transaction.

Malicious collators can collaborate and not face punishment. Using a VRF as a seed for shuffling provides some level of protection from tampering the shuffling order, as only the next collator will know the VRF output in advance. Hence, the block-executor cannot arrange the transactions so that the ordering is "right" after the shuffling. However, if two consecutive collators are malicious and cooperating, the first one can manipulate the extrinsics in a way that they will be ordered as intended after shuffling with the VRF seed from the second collator. There is also the possibility that a block producer intentionally skips a block production slot if the transaction ordering isn't "right" – by this they can double the chance of getting a favorable transaction ordering. In addition, misbehaving collators are not subject to any slashing according to the current implementation.

The VER solution does not protect against probabilistic front-running. The VER protection implementation does not protect against probabilistic front-running attempts where a malicious user could create limit orders that will only succeed if they are executed before a specific transaction. This attack (with a 50% success chance) is relatively cheap for the attacker as only the transaction fees would have to be paid in case of unsuccessful attack.

Possible front-running through proxies. It is possible for an attacker to spam the transaction queue by dispatching calls through many proxies. This could increase the chance of frontrunning a certain transaction, as the order of the transactions will be shuffled, and the attacker would have a higher chance of getting their call executed before the target transaction. As a note, such a front-running attack scenario is

rendered unlikely by the introduction of the VED scheme, as the encryption prevents users seeing others' transactions.

5.1.2 Value Extraction by Denial (VED)

Mangata's VED solution aims at preventing collators from gaining an unfair advantage from monitoring other users' transactions. The way it works is that a user encrypts their transactions twice, once with the block-executor's private key and once with the block-builder's private key. Later, the block-builder decrypts the second encryption layer and includes the now singly encrypted transactions in their block. As the next step in the following block, the block-executor decrypts the first encryption layer and executes the actual transaction. This way collators could not spy on or tamper with users' transactions.

Cryptographic overhead not properly calculated. The weight calculation does not consider the computational overhead of the transaction encryption and database storage when processing the doubly and singly encrypted transactions.

The *submit_doubly_encrypted_transaction* extrinsic has a default weight assigned and it is not trivial to benchmark due to the encryption layer on the transaction: a transaction submitted for block *N* will only be executed in the next block by the next block author in line. Users do submit an estimated a weight for executing the transaction upon calling this extrinsic, however it cannot be verified until the transaction is executed in the next block. If an attacker exploits it by calling underweighted extrinsics, it could lead to overweight blocks and possibly disrupt block production [22].

Correlation attack based on the encrypted transactions' length. However, even with the knowledge of the type of transactions (for example, determining whether it is a buy or sell call), it would be possible for malicious collators to harm users financially by omitting their transactions while building the blocks. Due to the nature of the current encryption implementation, it is possible to infer to type of transaction simply by the length of the transaction without knowledge of the content and censor such transactions [27].

The current encryption scheme used for VED is symmetric. Symmetric encryption uses a shared key to encrypt and decrypt the payloads, meaning that the protection of this key is of utmost importance for the security of the protocol. Moreover, as the collators will also be sharing the key, they will be in capacity of decrypting all transactions which defies the purpose of the encryption. Mangata is planning to implement an asymmetric cryptographic scheme for the VED protocol in the future.

5.1.3 AURA

Mangata uses AURA to determine which collator should produce a block at a given time. However, AURA's deterministic execution queues are known to be subject to Denial of Service (DoS) attacks [20]. An attacker could perform a "rolling" DoS attack by constantly switching between the target nodes. This is amplified by Mangata's two-step block production mechanism where the block-builder and executor need to be consecutive. Therefore, an attacker only needs to target every second collator to be able to successfully stall the chain.

Since AURA lacks redundancy, if a collator misses its block authorship slot, due to the nature of Mangata's encrypted transaction implementation, no other node can

decrypt the transactions that are bound to that collator (using their encryption keys). The transactions will not be processed until the next round where the same collator is eligible to produce a block.

Furthermore, in the current PoC implementation of VED, an even more straightforward way to perform DoS attacks against a specific collator is by always submitting its ID as the block-builder and block-executor parameters to the *submit_doubly_encrypted_transaction* extrinsic. Since there is no validation on these parameters, an attacker may use the same ID as the builder and the executor node or even wrong non-existing IDs which will lead to storing encrypted transactions on the chain that will not get cleaned up until the session ends, as no-one is able to decrypt them.

Finally, as already described in Section 5.1.1, as the order of block authors is well-known in advance, a malicious collator could collaborate with either the previous or the next block author to tamper the shuffling logic.

5.1.4 Functional issues

The introduction of the Themis protocol affects the liveliness of the chain. In the Themis protocol, users encrypt their transaction twice for the block-builder and the block-executor with the appropriate public keys bound to the two specific nodes. However, this could become an obstacle for scaling: in case a collator receives too many transactions to process into one block, the transactions left out will only be included and executed one round later. When there is a consistent high throughput of transactions, some users might experience significant delays for their calls to be executed. Furthermore, since all submitted transactions are saved in the storage and cleared upon a new session, some transactions may never get executed if the chain is in high demand.

Reduced block production throughput up to 50 percent. Due to the nature of the doubly encrypted transactions, it is not possible to weigh the incoming transactions when building a block and therefore there the weight calculation can only be estimated. Furthermore, as the Mangata team pointed out, the extra work of storing and decrypting doubly and singly encrypted transactions on chain is consuming some of the collators' block production capacity and reduces the blockchains' throughput.

The current decoupled block execution scheme does not consider hooks. As the Mangata team pointed out to the auditors, an additional concern with the current scheme is not considering hooks when recording transactions to be executed in the following block. As some pallet hooks (e.g. *on_initialize*) are dependent on the block number, in certain edge cases, like on session starts, executing *on_initialize* or other hooks could prevent block execution when the total execution time considering hooks and extrinsics would exceed the block limit. As an interim mitigation, Mangata does not allow including transactions in blocks preceding a new session. In the future, Mangata intends to alter the block production scheme to separate the initialization process between the extrinsics built in the previous block and the hooks to be executed in the current block.

5.2 Underestimated weights for call execution

The current *encrypted_transactions* pallet lacks validation for the origin for the unsigned extrinsics *submit_singly_encrypted_transaction* and for *submit_decrypted_transaction*. Those unsigned extrinsics allow collators to decrypt

and execute transactions without having to pay for a transaction fee. This allows malicious collators to spam the chain with these unsigned extrinsics without paying any fees [19].

Moreover, at the time of the audit, several extrinsics across the Mangata codebase were assigned default weights and lacked benchmarking [23]. Default weights generally underestimate the computational complexity of extrinsics and thus allow for attackers to spam the chain and bloat the blocks. Mangata partially mitigated this issue by applying benchmarking to most of the extrinsics in their codebase [24]. However, the following components remain un-benchmarked:

- *bootstrap* pallet
- *submit* extrinsic (in *bridges* pallet, *0* as assigned weight)
- *issuance* pallet

5.3 Spamming risk resulting from free calls

In the *xyk* pallet, *buy* and *sell* transaction can be called without transaction fees (*Pays::No*). This could open the possibility for a malicious user to spam the chain with those transactions [18].

The issue was mitigated by the Mangata team by charging fees for the respective extrinsics. As a future plan, Mangata is planning to allow free calls to users, in exchange to requiring a deposit for them.

5.4 Overflow issues

In the *parachain-staking* pallet of the Moonbeam fork, there are multiple integer overflows that could cause service interruptions in the node or functional issues in the staking logic. Those overflows originate from the use of unsafe math functions [25]. The issues were mitigated by the Mangata team [26].

In the *xyk* pallet we identified integer overflow vulnerabilities in the *buy_asset* and *sell_asset* extrinsics that can be triggered by an attacker via a sequence of extrinsic calls [16]. This issue was mitigated in the PR 212 [17].

5.5 Highly nested calls can cause a stack overflow

In the *encrypted-transaction* pallet the decrypted calls in *submit_decrypted_transaction* are decoded without a limit. This could cause a stack overflow leading to crashing nodes in the case where the call is highly nested, for example by nesting batched calls [21]. As a consequence of the high number of recursive calls, the stack size will be insufficient during decoding and could result in a panic in the runtime. A runtime panic could potentially cause collators to fail at block production and miss their slots.

6 Conclusion and further steps

Mangata's Themis protocol implementation adds modifications in security-sensitive Substrate core components including *basic_authorship* and *block_builder*.

Due to Substrate's own complexity, changes in its codebase would require additional functional and security testing to avoid undefined or unexpected behavior. At the

time of the audit, the VED protection was in development and not thoroughly tested by Mangata.

According to our analysis the proposed implementation of the Themis protocol has fundamental security issues that are challenging to solve. A subset of these issues was already known by the development team; others were pointed out by SRLabs team in Chapter 5.

All in all, **the current implementation of the Themis protocol requires revision and significant changes before being security mature.**

Additionally, we recommend the following evolution suggestions to be considered to enhance the security of Mangata.

6.1 Extend the existing documentation

Documentation is the backbone of projects. It allows present and future developers to know what the code should be doing and how. Establishing an implementer's guide and documentation of the architecture that is consistent with the in-code documentation allows anyone working with the platform to project their understanding onto the code. It is essential for the stability of the project in the future. We recommend enhancing the already existing documentation, especially for critical sections (for example, reward calculation).

6.2 Implement a slashing mechanism

Currently, there is no slashing mechanism in place if a collator misbehaves. As demonstrated in the issues above in Chapter 5, such a risk is possible if multiple collator nodes are collaborating to gain unfair financial interest. We recommend implementing a slashing mechanism to disincentivize malicious behavior.

6.3 Minimize the number of forked dependencies

Mangata has forked most of its major dependencies including Cumulus, Substrate, Polkadot, ORML and Moonbeam repositories. Although forking is a common process to build upon open-source projects, it requires higher level of maintenance and back-porting newer features into Mangata. This creates a significant overhead that could lead to create security vulnerabilities, instability, or missing security patches. We suggest to only fork the bare necessary.

6.4 Use safe math functions

When using Rust math functionalities, we recommend using safe math functions wherever sensible. This best practice helps minimizing the risk of common bugs such as overflows or underflows.

6.5 Consider making the rewards parameters runtime configurable

If in the future, Mangata desires to change their economic model or some parts of it, making the reward calculation parameters runtime configurable would facilitate the process. Currently these values are hardcoded and require more effort to be updated. As a result, modification of these parameters would be a manual process, that requires changing the value in every line where the hardcoded parameter is used and a runtime upgrade which lacks flexibility and is prone to errors.

6.6 Address all TODOs and known issues

Currently, Mangata's code has a lot of open ends tagged as "TODO" or "known issue". Considering those are part of the team's workflow, we recommend assessing these tasks to prevent avoidable bugs becoming security vulnerabilities.

7 References

[1] [Online]. Available: https://securityresearchlabs.sharepoint.com/:x:/s/Mangata/ETWh9s6oujpDtDF7X9OyqDABlenGPmpI49Ux6pGeSWYlyQ?e=MstBRg.

[2] [Online]. Available: https://substrate.io/.

[3] [Online]. Available: https://webassembly.org/.

[4] [Online]. Available: https://libp2p.io/.

[5] [Online]. Available: https://wiki.polkadot.network/docs/learn-consensus#finality-gadget-grandpa.

[6] [Online]. Available: https://polkadot.network/blog/polkadot-consensus-part-3-babe/.

[7] [Online]. Available: https://github.com/mangata-finance/substrate.

[8] [Online]. Available: https://github.com/mangata-finance/moonbeam.

[9] [Online]. Available: https://github.com/mangata-finance/open-runtime-module-library.

[10] [Online]. Available: https://github.com/mangata-finance/cumulus.

[11] [Online]. Available: https://github.com/rust-lang/libm.

[12] [Online]. Available: https://github.com/mangata-finance/wiki/wiki.

[13] [Online]. Available: https://blog.mangata.finance/.

[14] [Online]. Available: https://github.com/mangata-finance/mangata-audit/issues.

[15] [Online]. Available: https://github.com/mangata-finance/mangata-audit/issues/1.

[16] [Online]. Available: https://github.com/mangata-finance/mangata-audit/issues/2.

[17] [Online]. Available: https://github.com/mangata-finance/mangata-node/pull/212.

[18] [Online]. Available: https://github.com/mangata-finance/mangata-audit/issues/4.

[19] [Online]. Available: https://github.com/mangata-finance/mangata-audit/issues/6.

[20] [Online]. Available: https://github.com/mangata-finance/mangata-audit/issues/7.

[21] [Online]. Available: https://github.com/mangata-finance/mangata-audit/issues/8.

[22] [Online]. Available: https://github.com/mangata-finance/mangata-audit/issues/9.

[23] [Online]. Available: https://github.com/mangata-finance/mangata-audit/issues/3.

[24] [Online]. Available: https://github.com/mangata-finance/mangata-node/pull/214.

[25] [Online]. Available: https://github.com/mangata-finance/mangata-audit/issues/5.

[26] [Online]. Available: https://github.com/mangata-finance/moonbeam/commit/bdceeecfbbf14e62137df5c52fe99210b99a0a0a.

[27] [Online]. Available: https://github.com/mangata-finance/mangata-audit/issues/10.

[28] [Online]. Available: https://blog.mangata.finance/blog/2021-10-10-themis-protocol/.