Autonomys Continuous Security Assurance

Threat model and hacking assessment report

V1.2, 29 July 2024

Marc Heuse	marc@srlabs.de
Mostafa Sattari	mostafa@srlabs.de
Daniel Schmidt	schmidt@srlabs.de
Jakob Lell	jakob@srlabs.de

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

During this study, Autonomys provided access to relevant documentation and supported the research team effectively. The code of Autonomys 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 critical to informational level severity, many of which concerned domains, fraud proofs and cross domain messaging. Autonomys, in cooperation with the auditors, already remediated most of the identified issues.

In addition to mitigating the remaining open issues, Security Research Labs recommends integrating more comprehensive security tests and encouraging broader participation as timekeepers to ensure network reliability. Proactively considering congestion control mechanisms is essential to prevent spamming and DoS vulnerabilities. Additionally, conducting thorough threat modeling, further secure development best-practices and economic audits will identify potential risks and ensure the economic integrity of the system. Finally, improving documentation will facilitate better understanding and maintenance of the codebase, contributing to a more robust and secure network.

- 31 pages -

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 Table 1. 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 9 may not ensure all future code to be bug free.

2 Timeline

As shown in Table 1 the Autonomys/Subspace source code has undergone two baseline security audits, one in 2022 and another in 2023 by Security Research Labs. After the last audit, continuous security checks have been in place. During this period, the project has experienced significant architectural changes and code implementations.

Date	Event
April 11, 2022	Initial engagement
July 4, 2022	Report for the baseline security check & consensus whitepaper report audit delivered in one document [1]
March 1, 2023	Subsequent baseline security check and remediation check Consensus whitepaper report audit
June 23, 2023	Report for the second baseline security check delivered [2]
August 1, 2023	Continuous support and remediation start
July 29, 2024	Continuous support and remediation report delivered (this document).

Table 1. Audits timeline

3 Motivation and scope

This report presents the security audit conducted by Security Research Labs as part of our continuous support efforts following the initial baseline security audit performed in June 2023. It is important to note that the findings from the previous engagement are not included in this document. For those initial findings, please refer to the corresponding reports highlighted in Table 1.

Autonomys, formerly known as Subspace, originally focused on scalable blockchain solutions and decentralized storage. It has evolved to become an integral part of a larger platform. Now, Subspace functions as the core protocol within the newly rebranded Autonomys network [3].

It should be noted that the protocol, repository and codebase is still called Subspace, which can be confusing when reading this report.

Autonomys envisions creating a decentralized AI ecosystem called deAI, which will include distributed storage, compute, and a dApp suite. Central to this vision is Autonomys ID, designed to provide secure, verifiable identities for both humans and AI agents. The network has already implemented innovative consensus mechanisms, Proof of Archival Storage, Proof of Stake and Proof of Time, to establish a stable and scalable infrastructure for AI and web3 integration. Under new CEO Labhesh Patel, Autonomys aims to pioneer a future where AI and humans coexist securely and autonomously, with a goal to launch the mainnet in Q3 2024.

On a technical level, the core business logic of Autonomys consists of a base-layer consensus chain, referred to as the Subspace protocol, and a potentially unlimited number of secondary execution chains, called domains. The Subspace protocol is responsible for managing consensus, ensuring data availability, and facilitating the settlement of transaction bundles.

These transaction bundles are executed by operators on their respective domains. Domains function as enshrined rollups, capable of supporting any state transition framework and smart contract execution environment imaginable.

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

Autonomys' runtime consists of multiple modules compiled into a WASM Binary Large Object (blob) that is stored on-chain. Nodes execute the runtime code either natively or will execute the on-chain WASM blob.

Security Research Labs collaborated with the Autonomys team to create an overview containing the runtime modules in scope and their audit priority. The in-scope components and their assigned priorities are reflected in Table 2. During the audit, Security Research Labs used a threat model [4] to guide efforts on exploring potential security flaws and realistic attack scenarios. Additionally, Autonomys' online documentations provided the testers with a good runtime module design and implementation overview.

Repository	Priority	Component(s)	Reference
Subspace	High	Domains XDM Runtime	[5]
	Medium	Farmer	[5]

Table 2. In-scope Autonomys' components with audit priority

4 Methodology

This report details the continuous security assurance results for the Autonomys network with the aim of creating transparency in four steps: threat modeling, security design coverage checks, implementation baseline check and finally remediation support. We applied the following four steps methodology when performing feature and PR reviews.

4.1 Threat modeling and attacks

The goal of the threat model framework is to be able to determine specific areas of risk in Autonomys network. 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 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 3.

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

Table 3. 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 Autonomys network
- Medium: Risk scenarios pose a considerable threat to Autonomys' functionality as a network.
- High: Risk scenarios pose an existential threat to Autonomys network functionality.

Damage and Hacking Value are divided according to Table 4.

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

Table 4. Risk measurement scale

After applying the framework to the Autonomys 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 to steal native tokens from nodes participating in the Autonomys ecosystem and claiming the assets (represented in the token) for themselves.

Integrity:

Integrity threat scenarios threaten to disrupt the functionality of the entire network by undermining or bypassing the rules that ensure that Autonomys transactions/operations are fair and equal for each participant. Undermining Autonomys' 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 Autonomys' functionality and, in turn, its reputation. For example, invalidating already executed domain transactions would violate the core promise that transactions on the domains are irreversible.

Availability:

Availability threat scenarios refer to compromising the availability of data stored by the Autonomys Network as well as the availability of the network itself to process normal transactions. Important threat scenarios regarding availability for blockchain systems include Denial of Service (DoS) attacks on the domain operators, stalling the transaction queue, and spamming.

Table 5 provides a high-level overview of the hacking risks concerning Autonomys 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. This list can serve as a starting point to the Autonomys developers 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. Undermining the availability of the Autonomys chain renders the main functionalities of the project unavailable. Hindering block production on the Autonomys chain will have an impact on every domain ecosystem tied to the main chain.

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	- Domain take over - Fabricate false domain transactions	Medium	- Slash honest domain operators by submitting false fraud proofs - Spoof XDM messages
Availa-bility	High	- Stall block production - Spam the blockchain with bogus transactions	Low	- Spam a target domain via fraudulent XDM messages - Fill up XDM channels' in/out box - Crash domain operators

Table 5. Risk overview. The threats for Autonomys' blockchain were classified using the CIA security triad model, mapping threats to the areas: (1) Confidentiality, (2) Integrity, and (3) Availability.

4.2 Security design coverage check.

Next, the Autonomys 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?

security design coverage checks, implementation baseline check and finally remediation support.

4.3 Implementation check

As a third step, the current Autonomys implementation was tested for openings whereby any of the defined hacking scenarios could be executed.

To effectively review the Autonomys codebase, we derived our code review strategy based on the threat model that we established as the first step. For each identified threat, hypothetical attacks were developed and mapped to their corresponding threat category, as outlined in Chapter 4.1.

Prioritizing by risk, the code 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 pallets and the runtime configuration.

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, 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.

We carried out a hybrid strategy utilizing a combination of code review, static testis and dynamic tests (e.g., fuzz testing) to assess the security of the Autonomys codebase.

While static testing, fuzz testing and dynamic tests establish a baseline assurance, the focus of this audit was a manual code review of the Autonomys codebase to identify logic bugs, design flaws, and best practice deviations. We reviewed the Subspace repository which contains Autonomys implementation up to commit *e352e08 from the 31st of May 2024.* 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 Autonomys codebase is entirely open source, it is realistic that a malicious actor would analyze the source code while preparing an attack.

Fuzz testing is a technique to identify issues in code that handles untrusted input, which in Autonomys' case are extrinsics in the runtime. (Note that the network part is handled by Substrate, which was not in scope for this review, but is built with a strong emphasis on security and where fuzz testing is also used). 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.

4.4 Remediation support

The final step is supporting Autonomys 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.

During the audit, findings were shared via their GitHub repository [5]. We also used a private slack channel for asynchronous communication and status updates – in addition, bi-weekly jour fixe meetings were held to provide detailed updates and address open questions.

5 Evolution suggestions

The overall impression of the auditors was that Autonomys has been continuously improving their security metrics and undertaken security recommendations that were outlined in the previous audits to improve their code security measures. To ensure that Autonomys is secure against further unknown or yet undiscovered threats, we recommend considering the evolution suggestions and best practices described in this section.

5.1 Business logic improvement suggestions

Encourage broader participation as timekeepers. As timekeepers perform a crucial role in the Proof of Time calculations, it is essential to introduce incentives to run timekeeper nodes. One incentive is implementing token rewards proportional to the amount of work performed. In addition, it is important to develop a reputation system that rewards nodes with higher reliability and accuracy. Timekeeper nodes with a good track record could receive higher rewards and greater responsibilities, creating a competitive environment that drives the quality of the service.

Proactively consider congestion control mechanisms. One common theme among the uncovered issues was spamming and DoS vulnerabilities. These issues can severely degrade network performance, disrupt services, and compromise the overall reliability of the system. To mitigate such risks, we recommend integrating congestion prevention mechanisms into the design phase before starting the implementation. This could include introducing more strict fee systems for transactions to disincentivize spamming.

5.2 Future proof design reevaluation

The Autonomys network should ensure it is prepared for high load and traffic and should continually evaluate whether current design choices can support future network growth. For example, the transaction shuffling mechanism, which aims to prevent intentional reordering of transactions within a bundle, can cause significant spam and network pressure on the peer-to-peer layer once Autonomys domains handle substantial volume. These designs should be reevaluated constantly to ensure the Autonomys network withstands the pressure of large amounts of traffic in the future.

5.3 Secure development improvement suggestions

We recommend to further strengthen the security of the blockchain by implementing the following recommendations:

Adopt pair programming. Adopt pair programming practices to identify security vulnerabilities early in the development cycle. Adopting pair programming practices is beneficial for better code security because it involves immediate code review, allowing vulnerabilities to be caught in real-time. It fosters continuous knowledge transfer, reduces the risk of oversights, and encourages adherence to coding standards. This collaborative approach also boosts problem-solving, morale, and accountability, ensuring early detection of security issues and fostering continuous learning and improvement.

Perform threat modeling. Performing threat modeling for all new features and major updates before coding is crucial for better code security. This practice allows developers to identify potential security threats and vulnerabilities early in the design phase, enabling them to implement appropriate mitigations from the outset. Including the threat model in the pull request description ensures that the entire team is aware of the identified risks and the measures taken to address them, promoting a proactive security culture and enhancing the overall robustness of the codebase. Additionally, it helps the audit team to identify gaps in the threat model and focus their assessment.

Use static analysis. Using static analysis tools to detect security flaws in the codebase is essential for improving code security. These tools, such as Dylint and Semgrep for the Rust ecosystem, analyze the code without executing it, identifying vulnerabilities, coding errors, and compliance issues early in the development process. This proactive approach helps developers address potential security issues before they reach production, ensuring a more secure and reliable codebase.

Perform dynamic analysis. Developing fuzzing harnesses for consensus mechanisms, domain-specific areas, and other critical components is essential for identifying security vulnerabilities and business logic issues. By employing invariants, these fuzzing tests can effectively uncover subtle flaws that might otherwise go unnoticed. The Polkadot codebase exemplifies this approach, utilizing multiple fuzzing harnesses based on the substrate-runtime-fuzzer, demonstrating how comprehensive and targeted fuzz testing can significantly enhance the security and reliability of complex systems; see the *substrate-runtime-fuzzer* [6].

Create an incident response plan. Developing a comprehensive incident response plan to address potential security breaches is vital for maintaining code security and organizational resilience. This plan should include detailed procedures for responding to various scenarios, such as compromised developers or exploited blockchain vulnerabilities, ensuring quick and effective mitigation of threats. By having a well-defined response strategy, organizations can minimize the impact of security incidents, protect sensitive data, and maintain trust with users and stakeholders.

Launch a bounty program. Establishing a bounty program to encourage the external discovery and reporting of security vulnerabilities is crucial for enhancing code security. By offering incentives, such programs motivate individuals to responsibly report vulnerabilities to Autonomys rather than exploiting them. This approach not only broadens the pool of people actively searching for security flaws but also fosters a collaborative security environment, helping to identify and address issues more quickly and effectively.

5.4 Address currently open security issues

We recommend addressing already known security issues before going live with mainnet to prevent attackers from exploiting them – even if an open issue has a limited impact, an attacker might use it as part of their exploitation chain, which may have a more severe impact on Autonomys.

5.5 Further recommended best practices

Extensive test implementation. Currently, most pallets have unit tests, but they are overall not comprehensive and do not cover most situations. For example, the messenger pallet supports arbitrary messages between domains, and various factors such as weights, fees, and delivery status play a role in the correctness of this pallet. Although there are essential tests already implemented, they do not cover all different situations. We recommend extending unit tests in general with a stronger focus on security. This includes testing for edge cases, potential vulnerabilities, and common attack vectors.

Engage in an economic audit. Although Security Research Labs has some knowledge of economic attacks, our primary goal during engagements is to find logic vulnerabilities through code assurance. Therefore, an economic audit for the domains, rewards and staking features is recommended to ensure the safety of Autonomys' platform and users.

Documentation. The documentation occasionally lacked consistency with the code and sometimes contained contradictions. This made it challenging to understand the code without consulting the development team. Accurate documentation ensures that developers, auditors, and other stakeholders can comprehend the codebase without needing to frequently consult the development team. To achieve this, it is essential to establish a practice of updating the documentation concurrently with any code changes. Incorporating documentation verification into the code review process can help detect discrepancies early.

Regular updates. New releases of Substrate may contain fixes for critical security issues. Since Autonomys 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.

Avoid forking pallets. While it may not always be possible to contribute upstream to popular pallets, such as pallet-grandpa-finality-verifier, 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.

6 Static analysis assessment

Throughout our auditing process, we utilized static analysis in our workflow to identify rule-based vulnerabilities in the Autonomys codebase. Semgrep[1] and Dylint[2] were the primary tools employed, with custom rules tailored for Rust and Substrate specific chains. These rules are based on previous findings that can be detected via static analysis. Examples of such rules include missing extrinsic weights, the use of unsafe cryptographic functions, and runtime misconfigurations. Additionally, we ran cargo-audit, which yielded 11 results a shown in Table 6, including vulnerable library versions and unmaintained libraries. The Autonomys team is aware of these issues. Notably, they mentioned in a dedicated GitHub Pull Request [7] that the vulnerability in rustls version 0.20.9 does not affect their codebase.

Dependency	Version	Description	Type
curve25519-dalek	3.2.0, 4.1.2	Timing variability in curve25519-dalek's `Scalar29::sub/Scalar52::sub`	Vulnerability
rustls	0.20.9	`rustls::ConnectionCommon::complete_io` could fall into an infinite loop based on network input	Vulnerability
ansi_term	0.12.1	ansi_term is Unmaintained	Warning
mach	0.3.2	mach is unmaintained	Warning
parity-wasm	0.45.0	Crate `parity-wasm` deprecated by the author	Warning
libc	0.2.154	Yanked	Warning
pest	2.7.9	Yanked	Warning
pest_derive	2.7.9	Yanked	Warning
pest_generator	2.7.9	Yanked	Warning
pest_meta	2.7.9	Yanked	Warning

Table 6. Cargo-audit results

7 Dynamic analysis assessment

During the audit, we utilized fuzz testing to identify bugs in the runtime of Autonomys. By applying fuzz testing, we are aiming to uncover issues that may not have been detected through manual code review. For this purpose, we created two harnesses. One for the main

[1] https://semgrep.dev
[2] https://github.com/trailofbits/dylint

runtime (`crates/subspace-runtime`) and one for the EVM domain runtime (`domains/runtime/evm`). The fuzzing campaigns were continuously updated as new features were developed, ensuring that each new functionality was thoroughly tested for potential bugs and vulnerabilities. This approach yielded some findings in the previous rounds of audit that were addressed by Autonomys team. Continuous updates of the fuzzing campaign reassured us that common programming mistakes such as unsafe math operations did not introduce such issues again into the codebase.

Fuzz harness creation: We chose our in-house developed and opensource tool Ziggy [8] as our fuzzing orchestration tool. As for the harness, we used a customized harness for both runtimes which is like our Substrate runtime fuzzer harness that is also available on Github [6].

Coverage analysis and optimization: Although modern fuzzers can achieve good coverage by utilizing various techniques, we manually generated some seeds to target specific functionalities that were not covered by the fuzzer after a certain period. This approach assisted the fuzzer and optimized the overall coverage, ensuring more comprehensive testing.

In our coverage analysis below, we only measure the Subspace codebase coverage that the harness is targeting.

7.1 Consensus runtime fuzzer

Component	Code path	Coverage achieved
Subspace	crates/	30.60%

The coverage for the consensus runtime stands at 30.60%, with most lines not covered being either inherent extrinsics or only root callable extrinsics for example `register_domain_runtime`. There remains room for improvement, particularly in testing XDM-related extrinsics, which require additional configuration and setup adjustments.

7.2 EVM domain runtime fuzzer

Component	Code path	Coverage achieved
Domains	domains/	12.62%

Our observed coverage for the EVM domain runtime is currently at 12.62%, with most uncovered lines being inherent extrinsics or root callable extrinsics for example `initiate_channel`. Improvement is needed, particularly in frontier specific configurations and XDM-related extrinsics. These necessitate significant configuration and setup adjustments but are planned for future enhancements.

8 Findings summary

We identified 29 issues during our analysis of the runtime modules in scope in the Autonomys codebase that enabled some of the attacks outlined above. In summary, 5 critical severity, 8 high severity, 8 medium severity, 6 low severity and 2 info level issues were found. This list excludes the issues that were not relevant anymore due to architectural changes. For example, issues related to the Domains v1.0 were omitted because this feature was removed from the codebase.

Please note that in our methodology, critical severity issues refer to high severity issues that could be exploited immediately by an attacker on already deployed infrastructure, including a parachain or a non-incentivized testnet.

Issue	Severity	References	Status
Oversized fraud proofs allow for crafting invalid domain bundle/execution receipt	Critical	[9]	Fix under review in PR#2801
Insufficient verification of `proof_of_election` when submitting bundle	Critical	[10]	Fixed in PR#2516
Missing verification of `final_state_root` in execution receipt	Critical	[11]	Fixed in PR#2446
Incorrect handling of out-of-bounds extrinsic index	Critical	[12]	Fixed in PR#2787
Incorrect handling of fraud proofs	Critical	[13]	Fixed in PR#2666
The unsigned extrinsic `submit_fraud_proof` allows an attacker to spam the chain	High	[14]	Fixed in PR#1942, PR#1691
An attacker may extend the domain head multiple times	High	[15]	Fixed in PR#2790
Attackers can waste chain resources by bloating up storage proofs in unsigned calls	High	[16]	Fixed in PR#2650
Missing fraud proof validation and slashing could enable malicious behavior	High	[17]	Fixed in PR#1942, PR#1691
Missing bundle equivocation checks allow malicious behavior by domain executors	High	[18]	Fix under review in PR#2775
Only part of plotted data required to guess whether a sector contains a valid solution	High	[19]	Fixed in PR#1604
Autonomys is not using exponential fee adjustments, making it vulnerable to DoS	High	[20]	Fixed in PR#2156
Malicious domain owner can exploit channel's fee model	High	[21]	Fix under review in PR#2841
Attackers can prevent new operators to register through `signing_key` frontrunning	Medium	[22]	Fixed in PR#2772
Execution transaction shuffling can lead to blockspace congestion	Medium	[23]	Risk accepted
Burst Denial of Service in Proof of Time	Medium	[24]	Fixed in PR#2320

A faster timekeeper will outrun its competitors	Medium	[25]	Risk accepted
`IdentityFee` implementation used in messenger and transaction-payment pallets	Medium	[26]	Open
Incorrect usage of `Pays::No` for unsigned calls	Medium	[27]	Fix under review in PR#2842
Incorrect usage of `Pays::No` for the transfer extrinsic	Medium	[28]	Fixed in PR#2421
Existing channels remain open when a domain is removed from the allowlist	Medium	[29]	Fixed in PR#2815
No reward for running timekeeper in the network leading may lead to centralization	Low	[30]	Risk accepted
Time compression attack	Low	[31]	Risk accepted
Nominators count per operator limit can lead to a DoS against nominator candidates	Low	[32]	Fixed in PR#2783
Fees are locked away forever if a channel to a destination is not successfully opened	Low	[33]	Fix under review in PR #2829
Delay in XDM message execution allows opportunistic attacks on src/dst chain	Low	[34]	Open
Dishonest domain owner or user can close channel to avoid processing messages	Low	[35]	Fix under review in PR #2829
XOR-based calculation of `evaluation_seed` may lead to seed reuse	Info	[36]	Fixed in PR#1607
Use full proofs-of-space when plotting	Info	[37]	Fixed in PR#1651

Table 7. Issue summary

9 Detailed findings

9.1 Oversized fraud proofs allow for crafting invalid domain bundle/execution receipt

Attack scenario	**Attackers may craft invalid domain bundle/execution receipt with excessive fraud proof size, making it impossible to submit a fraud proof on consensus chain**
Location	/domains/runtime
Tracking	https://github.com/subspace/subspace/issues/2425

Attack impact	Attackers may commit fraud without punishment
Severity	Critical
Status	Open (mitigation under review in PR #2801)

When a bundle or execution receipt with an invalid execution result is created, a fraud proof must be generated and submitted using the `submit_fraud_proof` call on the consensus chain. For invalid extrinsic execution results, the fraud proof will be an `InvalidStateTransitionProof`, which includes large storage proofs if the extrinsic accesses significant storage. These proofs can exceed the consensus chain's maximum block length of 5 MiB, preventing honest operators from submitting the fraud proof.

If a fraud proof cannot be submitted due to its size, an invalid bundle or execution receipt may be deemed final. This poses a risk if the storage root from the invalid bundle contains unauthorized ATC token transfers. These fraudulent transfers would be accepted by the consensus chain or sibling domains, allowing an attacker to mint unlimited ATC tokens.

To prevent this, it is recommended to set a limit on the `proof_size` using the 2D weight system, ensuring fraud proofs can always be submitted. Additionally, it's important to guarantee that no calls can generate proof sizes exceeding the predicted values from weight calculations, as this would prevent invalid execution receipts from being challenged with fraud proofs.

9.2 Insufficient verification of `proof_of_election` when submitting bundle

Attack scenario	**Operator manipulates `proof_of_election` values to unfairly win VRF outputs**
Location	pallet-domains
Tracking	https://github.com/subspace/subspace/issues/2451
Attack impact	Unfair block production rewards and unauthorized ATC token minting
Severity	Critical
Status	Fixed in PR #2516

Operators can submit a bundle only when their VRF-based proof of election is below a threshold, which depends on their stake. The VRF input comes from `global_randomness` and `slot_number` in the `ProofOfElection`. However, there is no check to ensure these values match the consensus chain's global randomness and `slot_number`. A malicious operator could exploit this by brute-forcing these values to get a winning VRF output, gaining an unfair share of bundle submissions.

This allows the attacker to unfairly increase their domain block production and operator rewards. Additionally, they could submit multiple invalid bundles in one block, potentially leading to unauthorized minting of ATC tokens.

To prevent this, it is recommended to verify that the `global_randomness` and `slot_number` values in the `ProofOfElection` match the consensus chain's values.

9.3 Missing verification of `final_state_root` in execution receipt

Attack scenario	**Malicious operator submits a bundle with a valid execution trace but an incorrect `final_state_root`, potentially manipulating domain storage**
Location	pallet-domains
Tracking	https://github.com/subspace/subspace/issues/2437
Attack impact	Manipulating the domain storage can undermines the integrity of a domain
Severity	Critical

Status	Fixed in PR #2446

When a domain operator submits an execution receipt, the `process_domain_block` function assigns the `final_state_root` and the last entry of the execution trace. However, there's no verification during the `submit_bundle` call to ensure that they match.

This oversight allows a malicious operator to submit a bundle with a valid execution trace but an incorrect `final_state_root`, potentially manipulating domain storage. Without a fraud proof type to challenge this, unauthorized actions like minting ATC tokens can go unchecked for up to 18 blocks.

We recommended implementing verification checks that ensure the `final_state_root` matches the end of the execution trace.

Autonomys addressed this issue by implementing verification checks in PR #2446.

9.4 Incorrect handling of out-of-bounds extrinsic index in fraud proofs verification

Attack scenario	**Malicious operator marks valid bundles as invalid, exploiting chain acceptance before honest operators can generate a complete fraud proof**
Location	sp-domain-fraud-proof, domains/client
Tracking	https://github.com/subspace/subspace/issues/2810
Attack impact	Allows unauthorized manipulation of bundle validity and potential creation of ATC tokens without proper authorization
Severity	Critical
Status	Fixed in PR #2787

A malicious domain operator can falsely label valid bundles as invalid by manipulating `ExecutionReceipt.inboxed_bundles`, initially fooling the chain into accepting them. Honest operators will later detect this discrepancy and attempt to generate a fraud proof, but they may fail if the invalidity type involves an out-of-bound extrinsic index, preventing a fraud proof from being generated. This loophole allows attackers to exclude valid bundles and introduce additional errors in the execution receipt, potentially leading to unauthorized minting of ATC tokens if the receipt is finalized without challenge.

To mitigate this, implement some form of protection against out-of-bounds extrinsic indices (which may be submitted in the different `InvalidBundleType` variants within `ExecutionReceipt.inboxed_bundles`).

Autonomys addressed this issue in PR #2787.

9.5 Incorrect handling of fraud proofs verification for bundles

Attack scenario	**A malicious domain executor spams the chain with equivocating transactions**
Location	pallet-domains
Tracking	https://github.com/subspace/subspace/issues/2660
Attack impact	Spamming the chain can undermines the availability and the integrity of the blockchain
Severity	Critical
Status	Fixed in PR #2666

The gossip message validator logic for both system and core domains lacks the `check_equivocation` function. In both `system_gossip_message_validator` and `core_gossip_message_validator`, this function is a placeholder that does not actually perform equivocation checks, leaving the system vulnerable to abuse.

The lack of equivocation checks allows malicious domain executors to spam the chain with equivocating transactions, undermining the integrity of the blockchain.

Having a None returned by `is_xdm_valid` either means that the call is non-decodable or that the call is not XDM-related. The first case is already covered via `InvalidBundleType::UndecodableTx` and for the second case the call should be considered valid since it is not XDM-related, so the call should not be rejected with `InvalidBundleType::InvalidXDM`.

9.6 The unsigned extrinsic `submit_fraud_proof` allows an attacker to spam the chain

Attack scenario	**Attackers can spam the chain by submitting multiple unsigned fraud proofs**
Location	pallet-domains
Tracking	https://github.com/subspace/subspace/issues/1465
Attack impact	An attacker can potentially perform a denial-of-service attack at no cost
Severity	High
Status	Fixed in PRs #1942 and #1691

The `submit_fraud_proof` extrinsic in the pallet-domains did not require a signed origin, allowing anyone to submit fraud proof without needing an account. However, this lack of signature checking permits attackers to exploit the extrinsic for spamming the chain, as there are no fees or tracking mechanisms in place. Without signatures, there is no way to slash or penalize entities that misuse this feature.

To mitigate this risk, we recommended to properly implement the `ValidateUnsigned` traits for this extrinsic to include some form of signature, enabling account tracking and disincentivizing spamming by imposing fees or deposits. Alternatively, converting it to a signed extrinsic can provide the necessary controls to prevent abuse.

9.7 An attacker may extend the domain head multiple times

Attack scenario	**Attackers can extend the domain head multiple times in a single consensus block**
Location	pallet-domains
Tracking	https://github.com/subspace/subspace/issues/2442
Attack impact	Attackers may mint ATC tokens via invalid domain execution receipts before honest operators can submit a fraud proof
Severity	High
Status	Fixed in PR #2790

Once a bundle is submitted to the transaction pool of the consensus chain, since it is an unsigned call, it is validated before being imported to the pool. Inside the `validate_bundle` function, the execution receipt is validated by comparing the current `HeadReceiptNumber + 1` with the `domain_block_number` from the execution receipt. Since the `HeadReceiptNumber` is not updated in the state during the `validate_unsigned` check, it will only allow bundles carrying an execution receipt with `domain_block_number == HeadReceiptNumber + 1`. This validation process ensures that only the next sequential bundle is processed, theoretically preventing multiple consecutive bundles from being submitted within the same consensus block.

However, this protection is circumvented if an attacker is producing the consensus chain block, which is a permissionless operation requiring only a valid proof of archival storage submission. In such a scenario, the attacker can skip the transaction pool validation, and all other nodes will only run the validation via `pre_dispatch` during block execution. By this time, the `HeadReceiptNumber` may have already been updated by a previous `submit_bundle` call within the same consensus chain block.

We recommend adding an additional on-chain check to ensure that the `HeadReceiptNumber` can only be updated a single time per consensus block.

Autonomys addressed this issue in PR #2790.

9.8 Attackers can waste chain resources by bloating up storage proofs in unsigned calls

Attack scenario	**Attackers can submit bloated storage proofs at no cost, wasting chain resources**
Location	sp-domains
Tracking	https://github.com/subspace/subspace/issues/2646
Attack impact	Submitting bloated storage proofs can exhaust resources, potentially allowing invalid execution receipts to be confirmed
Severity	High
Status	Fixed PR #2650

Some parts of the Subspace domain implementation, such as message delivery, depend on storage proofs. A storage proof consists of trie nodes necessary for verifying a specific storage entry in the trie of another chain or state root. Subspace does not enforce the minimality of these storage proofs. As a result, an attacker can consume chain resources by submitting calls like `relay_message` with unnecessarily bloated storage proofs. Since it is an unsigned call, the attacker will not have to pay for this waste of resources and push out critical transactions, such as `submit_fraud_proof`, potentially allowing invalid execution receipts to be confirmed since nobody manages to submit a fraud proof in time.

To mitigate this Implement Parity's `ensure_no_unused_nodes` function to enforce minimal storage proofs, preventing inclusion of unnecessary trie nodes and preserving chain resources for essential operations.

Autonomys addressed this issue in PR #2650.

9.9 Missing fraud proof validation and slashing could enable malicious behavior

Attack scenario	**Misbehaving participants exploit the absence of slashing mechanisms to commit fraud without repercussions**
Location	pallet-domains
Tracking	https://github.com/subspace/subspace/issues/1338
Attack impact	Undetected misbehavior can undermine protocol security and integrity
Severity	High
Status	Fixed in PRs #1942 and #1691

In the Subspace codebase, slashing is not yet implemented for security-critical actions involving potentially untrusted participants. Effective slashing requires mechanisms to report and validate fraud committed by misbehaving parties.

Currently, functions like `validate_invalid_transaction_proof` and `validate_bundle_equivocation_proof` lack misbehavior checks, and the domain block processor does not validate fraud proofs in the `on_domain_block_processed` function.

To mitigate risks, it is crucial to implement and audit the necessary validation and slashing mechanisms. Additionally, slashing should also apply to those who submit false fraud proofs to deter malicious activity and maintain protocol integrity.

This issue was addressed by Autonomys through a series of PRs [38] [39] that implemented this feature.

9.10 Missing bundle equivocation checks allow malicious behavior by domain executors

Attack scenario	**Malicious actors spam the blockchain with equivocating transactions due to the absence of equivocation checks**
Location	sp-domains
Tracking	https://github.com/subspace/subspace/issues/1340
Attack impact	The integrity of the blockchain is compromised by unchecked equivocating transactions
Severity	High
Status	Fixed in PR #2775

Currently, the gossip message validator logic for both system and core domains lacks the `check_equivocation` function. In both `system_gossip_message_validator` and `core_gossip_message_validator`, this function is a placeholder that does not actually perform equivocation checks, leaving the system vulnerable to abuse.

The lack of equivocation checks allows malicious domain executors to spam the chain with equivocating transactions, undermining the integrity of the blockchain.

To mitigate this, it is recommended to conduct a security audit to uncover any logic bugs in the equivocation checks once the feature is implemented.

This issue was addressed by Autonomys by implementing the bundle equivocation checks.

9.11 Only part of plotted data required to guess whether a sector contains a valid solution

Attack scenario	**Farmer stores partial data to find valid solutions, reducing storage costs**
Location	subspace-verification
Tracking	https://github.com/subspace/subspace/issues/1478
Attack impact	Unfair farming advantages and compromised data availability in Proof-of-Archival-Storage consensus
Severity	High
Status	Fixed in PR#1604

During auditing, the function `calculate_solution_distance` XORs the plotted data (`audit_chunk`) with `sector_slot_challenge`. If the result falls within the solution range, it's deemed valid. However, if a farmer stores only the 16 most significant bits of the 64-bit `audit_chunk`, they can still rule out most invalid solutions, requiring minimal data to verify potential solutions, thereby discarding 75% of the plotted data.

This tactic allows an attacker to significantly reduce storage requirements while still identifying valid solutions. If the 16 bits suggest a possible solution, the data can be re-encoded on the fly.

This method balances time and memory by increasing false positives but saves on storage costs, with 16 out of 64 bits being a practical compromise for attackers.

Attackers gain an unfair advantage in farming, driving honest farmers out and compromising Proof-of-Archival-Storage consensus by encouraging users to store multiple copies of the same sector.

To mitigate this, use hashing instead of XOR, such as HMAC with `sector_slot_challenge` and plotted data, ensuring all bits are relevant. Larger data chunks (e.g., 32 bytes) can reduce hash calculations for performance.

This issue was addressed by Autonomys as part of PR#1604.

9.12 Autonomys is not using exponential fee adjustments, making it vulnerable to DoS

Attack scenario	**Attacker spams transactions with regular fees, blocking zero-tip transactions**
Location	subspace-runtime
Tracking	https://github.com/subspace/subspace/issues/1284
Attack impact	DoS against legitimate transactions, disrupting automated trading bots and causing economic losses
Severity	High
Status	Fixed in PR#2156

In all Autonomys runtime configurations (system, core-payments, and subspace-runtime), the `FeeMultiplierUpdate` is not set to a `TargetedFeeAdjustment`. Without `TargetedFeeAdjustment`, fees do not automatically adjust when the chain is spammed with transactions, allowing attackers to execute a DoS attack against zero-tip transactions by paying regular transaction fees. This can block legitimate transactions for an extended period without significantly increasing costs.

Attackers can block legitimate transactions that do not include a sufficient tip, disrupting automated tools like trading or arbitraging bots, giving attackers an economic advantage.

To mitigate this, set `FeeMultiplierUpdate` to a `TargetedFeeAdjustment` in the runtime configuration. Although fees are currently not enabled in Autonomys, it is important to consider this adjustment when enabling fees in the future.

This issue was addressed by Autonomys by setting the proper values for the runtime configuration.

9.13 Malicious domain owner can exploit channel's fee model

Attack scenario	**Malicious user sets extraordinarily high fees by replacing existing channels**
Location	sp-domains
Tracking	https://github.com/subspace/subspace/issues/2808
Attack impact	Financial losses for users and damage to Autonomys reputation
Severity	High
Status	Fix under review in PR#2841

In the current fee model, the caller sets the fee for opening a channel through the channel initialization parameters, allowing for zero or excessively high fees. If multiple channels exist between two domains, the latest one is used. A malicious user can exploit this by replacing an existing channel with a new one that has extremely high fees, forcing users to pay exorbitant fees for transactions.

Users may face excessively high fees, leading to financial losses and unfair costs, and this attack can be repeatedly executed. This not only impacts users financially but also harms Autonomys' reputation.

To mitigate this, adjust XDM fees only through permissioned roles such as governance.

9.14 Attackers can prevent new operators to register through `signing_key` frontrunning

Attack scenario	**A malicious operator can front-run the registration process by copying the signing keys of honest operators**
Location	pallet-domains
Tracking	https://github.com/subspace/subspace/issues/2554
Attack impact	Honest operators are prevented from registering
Severity	Medium
Status	Mitigated (https://github.com/subspace/subspace/pull/2772)

The `register_operator` extrinsic accepts a `config.signing_key` as a parameter. Since there is no proof that the operator registering possesses the private key for the corresponding `config.signing_key`, a malicious operator could front-run the registration of other operators and copy the honest operators' `config.signing_keys` into their configuration. This leads to the honest operator's registration extrinsic failing because of the duplicate operator signing key check, which will yield the `DuplicateOperatorSigningKey` error.

To mitigate this risk, we recommended implementing a proof of ownership for the `config.signing_key` to ensure that only the legitimate owner can register with the key.

9.15 Execution transaction shuffling can lead to blockspace congestion

Attack scenario	**MEV searchers spam transactions to position near targeted transactions**
Location	sp-domains
Tracking	https://github.com/subspace/subspace/issues/2265
Attack impact	Blockspace congestion, higher fees, and discouraged network participation
Severity	Medium
Status	Risk accepted

Shuffling domain transactions to mitigate MEV can cause blockspace congestion due to MEV searchers spamming transactions. This occurs as searchers submit numerous transactions to increase their chances of positioning near a targeted transaction containing MEV opportunities. For example, in backrunning, searchers aim to have their transaction positioned immediately after a targeted transaction. Transactions not optimally positioned are reverted, resulting in many transactions being reverted on the blockchain.

This causes blockspace congestion, leading to longer inclusion times and higher execution fees for users, which could discourage participation in the Autonomys network. Similar issues have been observed in the Ethereum network during Priority-Gas-Auctions.

Implementing Proposer-Builder-Separation in domains can mitigate this issue. This model separates block building and proposes conducting auctions outside blockspace. Only the winning searcher's transaction is included in the bundle, reducing reverted transactions and resulting in faster inclusion times and lower transaction fees.

9.16 Burst Denial of Service in Proof of Time

Attack scenario	**Attacker spams faulty PoT messages to delay verification and disrupt node operations**
Location	subspace-proof-of-time
Tracking	https://github.com/subspace/subspace/issues/2142
Attack impact	Node misses block creation opportunities, leading to network disruptions and slowdowns
Severity	Medium
Status	Fixed in PR#2320

An attacker can exploit the high cost associated with verifying Proof of Time (PoT). According to the AES Latency Report, an ASIC can generate PoTs 2.4 times faster than COTS hardware. If an ASIC produces PoTs every second, COTS hardware takes 2.4 seconds to generate a PoT independently and 300ms to verify one, assuming it can verify 8 checkpoints in parallel.

In the first attack scenario, if up to seven PoT messages are queued for verification, the system checks each message sequentially. If the valid PoT is the 6th message, the node spends 1.5 seconds verifying faulty messages. Given that AES iterations can be 50% above the current iteration number, this becomes 2.25 seconds.

In the second scenario, if eight or more PoT messages are received, the node generates the PoT itself, this costs 2.4 seconds. Though nodes sending faulty PoTs are banned, attackers can easily spawn new instances with fresh IP addresses, especially using IPv6.

A node busy with verification can miss its opportunity to create a block or perform other actions on the chain, disrupting its function and slowing down the network.

To mitigate this, after discussion with the Autonomys team, it was decided for PoT to always default to verification, but sort peers by reputation and pick smaller number (or just one) of those that have the highest reputation first.

Autonomys implemented the proposed mitigation in PR#2320.

9.17 A faster timekeeper will outrun its competitors

Attack scenario	**Faster timekeepers render slower ones obsolete, leading to centralization**
Location	subspace-proof-of-time
Tracking	https://github.com/subspace/subspace/issues/2141
Attack impact	Chain disruption if few timekeepers are present and one fails
Severity	Medium
Status	Risk accepted

Due to the design of the PoT mechanism, significantly faster timekeepers will cause slower ones to be less favored, resulting in a centralized set of timekeepers. PoT creation and distribution time is the chain's heartbeat, and slower timekeepers produce outdated PoT messages that nodes ignore. If new CPUs or ASICs are much faster, all slower timekeepers will be ignored, leaving only the faster one(s) active.

This results in older timekeepers becoming obsolete, reducing motivation to run them, especially without incentives. If only a few timekeepers are present and one goes offline, the chain is disrupted as nodes can only produce PoT very slowly, potentially causing failure.

An attacker running the fastest timekeeper can abuse this by trying to presolve PoS and operator slots, and if one is found, prevent collisions with competitors by announcing multiple following PoTs, therefore preventing others to also providing a solution in time. Additionally, an attacker can disrupt block production when a high enough number of PoT packets have been calculated until the next epoch.

The risk was accepted by the Autonomys team. We still highly recommend that Autonomys implements a watch service that identifies which servers produce the fastest PoT, and detect when an attacker sends out PoTs much faster than seen usually before.

9.18 `IdentityFee` implementation used in messenger and transaction-payment pallets

Attack scenario	**Malicious users exploit low fees to spam the network**
Location	subspace-runtime
Tracking	https://github.com/subspace/subspace/issues/2439
Attack impact	Increased network congestion and potential service disruption
Severity	Medium
Status	Open

In the Autonomys codebase, the fee calculation for messages or transactions in the pallets messenger and transaction-payment relies on the `WeightToFee` implementation, configured via the runtime's configuration. The current implementation, `IdentityFee`, converts weight (computation time) directly into currency without considering the network's economic conditions or congestion times.

This simplistic fee calculation can underestimate fees, allowing malicious participants to spam the chain without facing significant costs.

To mitigate this, implement a `WeightToFee` mechanism that dynamically adjusts fees based on the network's economic conditions and congestion levels.

9.19 Incorrect usage of `Pays::No` for unsigned calls

Attack scenario	**Malicious users submit signed transactions for unsigned calls to avoid fees**
Location	pallet-messenger
Tracking	https://github.com/subspace/subspace/issues/2423
Attack impact	Wasted chain resources and potential network degradation
Severity	Medium
Status	Fixed under review in PR #2842

Various calls, such as `relay_message`, which are dispatched as unsigned calls, have a `Pays::No` annotation in their weight annotation. This is only relevant for signed transactions, as unsigned calls do not have a signer to be charged. However, if these calls are mistakenly submitted as signed transactions, they will bypass the `ValidateUnsigned` checks, fail due to the `ensure_none` check, and not incur transaction fees due to the `Pays::No` annotation.

This allows attackers to exploit this to waste chain resources without paying fees.

To mitigate this, Remove the `Pays::No` annotation from all unsigned calls, ensuring that resources are protected. This change will not affect regular usage since transaction payments are only required for signed extrinsics.

9.20 Incorrect usage of `Pays::No` for the transfer extrinsic

Attack scenario	**users submit failing transfer calls to consume resources without cost**
Location	pallet-messenger
Tracking	https://github.com/subspace/subspace/issues/2420
Attack impact	Wasted chain resources and potential network congestion
Severity	Medium
Status	Fixed in PR #2421

The transfer extrinsic in the transporter pallet is annotated with `Pays::No` in its weight declaration. This allows users to submit the transfer call, knowing it will fail (e.g., due to `NoOpenChannel`), but it will still be included in the domain block without charging transaction fees because of the `Pays::No` annotation. This means attackers can exploit this to waste chain resources without paying fees.

To mitigate this, remove the `Pays::No` annotation from the `transfer` call. And instead, charge a transaction fee at the beginning. If the call is legitimate and successful, it can return a `PostDispatchInfo` with `Pays::No` to refund the fee.

This issue was addressed by Autonomys by removing the `Pays::No` annotation.

9.21 Existing channels remain open when a domain is removed from the allowlist

Attack scenario	**An attacker exploits open channels to conduct unauthorized transactions.**
Location	pallet-messenger
Tracking	https://github.com/subspace/subspace/issues/2805
Attack impact	Unauthorized activities and potential compromise of network integrity
Severity	Medium
Status	Fixed in PR #2815

When a domain is removed from the allowlist, the existing communication channels associated with that domain remain open. This poses a risk because these channels can be exploited for unauthorized communication or malicious activities, despite the domain no longer being authorized to use XDM. This situation could potentially lead to security threats, especially considering there may have been reasons for removing the domain from the allowlist.

To mitigate this risk, all open channels to a domain that is removed from the allowlist should be closed automatically. This ensures that unauthorized communication cannot occur through these channels, enhancing overall security.

This issue was addressed by Autonomys by closing the channel when it is removed from the allowlist.

9.22 No reward for running timekeeper in the network leading may lead to centralization

Attack scenario	**Malicious actors exploit the absence of incentives to discourage timekeeper operation**
Location	subspace-proof-of-time
Tracking	https://github.com/subspace/subspace/issues/2143
Attack impact	Centralization of timekeepers weakens network security and resilience
Severity	Low
Status	Risk accepted

Running a timekeeper in the network consumes power and requires substantial CPU resources or costly ASIC hardware. Currently, there are no incentives for individuals or entities to operate as timekeepers, resulting in a lack of participation beyond the Autonomys organization.

Without sufficient incentives, the network may end up with very few timekeepers, potentially controlled by a single entity. This centralization increases the risk of the chain's operations being dominated by a single party. Furthermore, the concentration of timekeepers from one source heightens vulnerability to denial-of-service attacks targeting these critical network components.

To mitigate this issue, incentives should be introduced to encourage broader participation as timekeepers. These incentives could include rewards or compensation mechanisms for the operational costs incurred by running a timekeeper. Such measures would promote decentralization and enhance the network's resilience against potential attacks.

9.23 Time compression attack

Attack scenario	**Rapid flooding of PoT messages ensures sole acceptance of PoS solutions**
Location	subspace-proof-of-time
Tracking	https://github.com/subspace/subspace/issues/2193
Attack impact	Dominance in blockchain consensus undermines fairness and decentralization
Severity	Low
Status	Risk accepted

An attacker with a privately held fastest timekeeper can exploit a time compression attack to ensure their Proof-of-Stake (PoS) solutions prevail over others on the blockchain. By leveraging their faster timekeeper, which generates PoT (Proof-of-Time) messages more rapidly than the public timekeepers, the attacker's nodes can identify and prepare PoS solutions for future slots earlier than other nodes. When the solution is due, the attacker floods the network with backlogged PoT messages at an accelerated rate, making it impossible for other nodes to verify and submit their own solutions in time. This results in the attacker's PoS solution being the only viable one on the blockchain.

This allows the attacker to dominate PoS solution submission, potentially controlling the blockchain's progression.

To mitigate this, ensure that the fastest timekeepers are public and transparently accessible. Implement a watchdog to monitor and analyze the speed of gossiped PoT messages; if a burst of future PoT slot messages is detected, it indicates a potential attack. Governmental measures or development of faster timekeeper ASICs could be initiated in response.

9.24 Nominators count per operator limit can lead to a DoS against nominator candidates

Attack scenario	**Exploiting the MaxNominators limit to flood operator candidates with minimal stake nominations**
Location	pallet-domains
Tracking	https://github.com/subspace/subspace/issues/2546
Attack impact	Potential domination of operator candidacy by well-funded attackers, leading to reduced decentralization and possible censorship of other operators
Severity	Low
Status	Fixed in PR #2783

In pallet-domains, the limit on the number of nominators per operator (`MaxNominators = 256`) can be exploited by a well-funded attacker to perform a DoS attack against operator candidates.

This allows attackers to dominate operator candidacy by using multiple accounts to nominate victim operators with minimal stakes, thereby preventing legitimate nominators from participating effectively.

To mitigate this, adjust the nomination process such that when the limit of 256 nominators is reached, any new nomination must exceed the stake of the least staked existing nominator for an operator. Implementing the phragmen algorithm for operator selection off-chain can also enhance fairness and resilience against such attacks.

This issue was addressed by Autonomys by removing the `MaxNominators` limit.

9.25 Fees are locked away forever if a channel to a destination is not successfully opened

Attack scenario	**Exploiting the indefinite locking of fees from failed channel openings**
Location	pallet-domains, pallet-messenger
Tracking	https://github.com/subspace/subspace/issues/2804
Attack impact	Users suffer financial losses and distrust due to irrecoverable fees, undermining platform reliability
Severity	Low
Status	Fixed in PR #2829

In the current implementation, fees associated with failed attempts to open a channel to a destination are locked indefinitely due to the lack of a mechanism for refund or fee reclamation. This results in permanent loss of funds for users, impacting financial security and platform trust.

This will cause users financial harm as they lose fees without recourse when channel openings fail, potentially reducing confidence in the platform.

To mitigate this issue, enable the closure of channels in the `ChannelState::Initiated` state, allowing users to recover funds associated with unsuccessful channel opening attempts.

Autonomys addressed this by allowing closure of channels in init state but taking a portion of reserve fee.

9.26 Delay in XDM message execution allows opportunistic attacks on src/dst chain

Attack scenario	**Malicious actors exploit the delayed execution of XDM messages to manipulate transactions or perform front-running attacks**
Location	pallet-messenger
Tracking	https://github.com/subspace/subspace/issues/2806
Attack impact	Increases vulnerability to opportunistic attacks, potentially compromising transaction fairness and integrity in the ecosystem
Severity	Low
Status	Open

An XDM message is not executed on the destination chain (`dst_chain`) until the domain block of the source chain (`src_chain_id`) containing the originating extrinsic is out of the challenge period or has reached archiving depth. This delay allows a malicious actor to exploit the pending status of the XDM message to leverage different types of attacks.

As we do not see an immediate solution for these issues, we recommend raising awareness about the presence of such threats in the ecosystem to better protect users.

9.27 Dishonest domain owner or user can close channel to avoid processing messages

Attack scenario	**Immediately close a channel to selectively block incoming messages**
Location	pallet-messenger
Tracking	https://github.com/subspace/subspace/issues/2807
Attack impact	Potential data loss or incomplete processing of messages undermines system reliability and user confidence in the network
Severity	Low
Status	Risk accepted

A dishonest domain owner or user controlling the newest channel can immediately close it to avoid handling incoming messages, enabling selective blocking or discarding of undesired messages. This manipulation can result in data loss or incomplete processing of legitimate messages, damaging Autonomys' reputation and eroding user trust over time.

To mitigate this, introduce a delay mechanism for channel closure requests. This ensures pending messages are processed before the channel is fully closed, thereby preventing immediate blocking and preserving message integrity.

9.28 XOR-based calculation of `evaluation_seed` may lead to seed reuse

Attack scenario	**Brute-forcing possible values of `sector_id` due to the limited variability introduced by XORing a u16 variable with a 256-bit hash**
Location	subspace-proof-of-space
Tracking	https://github.com/subspace/subspace/issues/1468
Attack impact	Potential exploitation in the consensus process
Severity	Info
Status	Fixed in PR #1607

The consensus protocol uses a u16 variable, such as `sector_index`, to derive `sector_id` by XORing it with a hash of the public key. Due to the u16 size (16 bits) compared to the 256-bit hash, only a small portion (16 bits) of `sector_id` varies, while the rest remains unchanged. The limited variability due to XORing with a small value (u16) can potentially allow attackers to brute-force possible values or exploit predictable bits in subsequent steps of the consensus process.

To mitigate this issue, hash small values used for XOR operations in the consensus protocol to increase variability and prevent predictable outcomes.

Autonomys implemented the fix for this issue in PR #1607.

9.29 Use full proofs-of-space when plotting

Attack scenario	**Storing raw qualities instead of encoded chunks to exploit space-saving and encode data cheaply on demand**
Location	subspace-proof-of-space
Tracking	https://github.com/subspace/subspace/issues/1625
Attack impact	Attackers gain a space-saving advantage, potentially undermining the security model of data storage and retrieval
Severity	Info
Status	Fixed in PR #1651

In the current implementation, we use the hash of quality to encode record chunks while plotting. This was an optimization to improve speed, but it allows an attacker with a single copy of history to store raw qualities (only ~5 bytes) and encode cheaply on demand.

An attacker can exploit this by storing raw qualities instead of encoded chunks, gaining a space-saving advantage and potentially undermining the security model.

To mitigate this, revert to using full proofs of space to encode chunks. This eliminates any space-saving advantage for attackers and ensures they cannot store raw qualities cheaply. Although this change is minor in terms of code, it will significantly impact plotting and proving time.

Autonomys addressed this issue by an implementation that uses the full PoS proof in PR #1651.

10 Bibliography

[1] [Online]. Available: https://securityresearchlabs.sharepoint.com/:b:/s/SubspaceNetwork/EUwNweQhTX1GnY0oCbpTs5MBfrmshDYNdBIC3ZZ5_dRQkg.

[2] [Online]. Available: https://securityresearchlabs.sharepoint.com/:b:/s/SubspaceNetwork/EYWtgwb2nidNizLQYmy56OcB99paH0XJu7A-W6GZ9hTfkg.

[3] [Online]. Available: https://blog.subspace.network/becoming-autonomys-new-vision-new-ceo-new-mainnet-launch-date-baa8accc1a76.

[4] [Online]. Available: https://securityresearchlabs.sharepoint.com/:x:/s/SubspaceNetwork/Ed1iEa767FdJo8jazVAzWtoBIzLmnyoKeXeJej0LKXYmog.

[5] [Online]. Available: https://github.com/subspace/subspace/.

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

[7] [Online]. Available: https://github.com/subspace/subspace/pull/2694.

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

[9] [Online]. Available: https://github.com/subspace/subspace/issues/2425.

[10] [Online]. Available: https://github.com/subspace/subspace/issues/2451.

[11] [Online]. Available: https://github.com/subspace/subspace/issues/2437.

[12] [Online]. Available: https://github.com/subspace/subspace/issues/2810.

[13] [Online]. Available: https://github.com/subspace/subspace/issues/2660.

[14] [Online]. Available: https://github.com/subspace/subspace/issues/1465.

[15] [Online]. Available: https://github.com/subspace/subspace/issues/2442.

[16] [Online]. Available: https://github.com/subspace/subspace/issues/2646.

[17] [Online]. Available: https://github.com/subspace/subspace/issues/1338.

[18] [Online]. Available: https://github.com/subspace/subspace/issues/1340.

[19] [Online]. Available: https://github.com/subspace/subspace/issues/1478.

[20] [Online]. Available: https://github.com/subspace/subspace/issues/1284.

[21] [Online]. Available: https://github.com/subspace/subspace/issues/2808.

[22] [Online]. Available: https://github.com/subspace/subspace/issues/2554.

[23] [Online]. Available: https://github.com/subspace/subspace/issues/2265.

[24] [Online]. Available: https://github.com/subspace/subspace/issues/2142.

[25] [Online]. Available: https://github.com/subspace/subspace/issues/2141.

[26] [Online]. Available: https://github.com/subspace/subspace/issues/2439.

[27] [Online]. Available: https://github.com/subspace/subspace/issues/2423.

[28] [Online]. Available: https://github.com/subspace/subspace/issues/2420.

[29] [Online]. Available: https://github.com/subspace/subspace/issues/2805.

[30] [Online]. Available: https://github.com/subspace/subspace/issues/2143.

[31] [Online]. Available: https://github.com/subspace/subspace/issues/2193.

[32] [Online]. Available: https://github.com/subspace/subspace/issues/2546.

[33] [Online]. Available: https://github.com/subspace/subspace/issues/2804.

[34] [Online]. Available: https://github.com/subspace/subspace/issues/2806.

[35] [Online]. Available: https://github.com/subspace/subspace/issues/2807.

[36] [Online]. Available: https://github.com/subspace/subspace/issues/1468.

[37] [Online]. Available: https://github.com/subspace/subspace/issues/1625.

[38] [Online]. Available: https://github.com/subspace/subspace/pull/1942.

[39] [Online]. Available: https://github.com/subspace/subspace/pull/1691.

[40] [Online]. Available: https://github.com/subspace/subspace/.

[41] [Online]. Available: https://github.com/subspace/subspace/issues/1337.

[42] [Online]. Available: https://github.com/subspace/subspace/issues/1339.

[43] [Online]. Available: https://github.com/subspace/subspace/issues/2140.