

Autonomys Continuous Security Assurance Consensus Chain Report

Threat model and hacking assessment report

v1.0-FINAL, November 4, 2024

Prepared for:
Autonomys

Content

Disclaimer.....	2
Timeline	3
Integrity Notice.....	4
1 Executive summary.....	5
1.1 Engagement overview.....	5
1.2 Observations and risk.....	5
1.3 Recommendations	5
2 Evolution suggestions	6
2.1 Business logic improvement suggestions	6
2.2 Secure development improvement suggestions	6
2.3 Address currently open security issues.....	7
2.4 Further recommended best practices	7
3 Motivation and scope	8
4 Methodology.....	9
4.1 Threat modeling and attacks	9
4.2 Security design coverage check	11
4.3 Implementation check	11
4.4 Remediation support	12
5 Dynamic analysis assessment.....	13
5.1 Consensus runtime fuzzer.....	13
6 Findings summary.....	14
6.1 Risk profile.....	14
6.2 Issue summary	15
7 Detailed findings	16
7.1 Only part of plotted data required to guess whether a sector contains a valid solution .	16
7.2 Absence of exponential fee adjustments makes transactions vulnerable to DoS.....	17
7.3 Transaction filter bypass using batched transactions.....	18
7.4 Burst denial of service in proof of time	19
7.5 A faster timekeeper will outrun its competitors.....	20
7.6 Incorrect usage of Pays : : No for unsigned calls.....	21
7.7 No reward for running timekeeper in the network may lead to centralization	22
7.8 Time compression attack	23
7.9 XOR-based calculation of <code>evaluation_seed</code> may lead to seed reuse.....	24
7.10 Best practice deviation in consensus	25
8 Bibliography	26
Appendix A: Technical services	28

Disclaimer

This report describes the findings and core conclusions derived from the audit carried out by Security Research Labs within the timeframe and scope as detailed in Chapter 3.

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

Version:	v1.0-FINAL	
Prepared For:	Autonomys	
Date:	November 4, 2024	
Prepared By:	Marc Heuse	marc@srlabs.de
	Mostafa Sattari	mostafa@srlabs.de
	Daniel Schmidt	schmidt@srlabs.de
	Jakob Lell	jakob@srlabs.de

Timeline

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

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
November 4, 2024	Continuous audit report on Consensus chain delivered (this document)

Table 1: Audit timeline

Integrity Notice

This document contains proprietary information belonging to Security Research Labs and Autonomys. No part of this document may be reproduced or cited separately; only the document in its entirety may be reproduced. Any exceptions require prior written permission from Security Research Labs or Autonomys. Those granted permission must use the document solely for purposes consistent with the authorization. Any reproduction of this document must include this notice.

1 Executive summary

1.1 Engagement overview

This work describes the result of the continuous security assurance audit Security Research Labs performed for Autonomys covering the period from August 2023 to October 2024. Security Research Labs is a consulting firm that has been providing specialized blockchain audit services since 2019.

During this study, Autonomys development team provided access to relevant documentation and supported the research team effectively. The protocol design, concept documentation, and relevant available source code of Autonomys was verified to assure that the Autonomys protocol is resilient to hacking and abuse.

Security Research Labs conducted a comprehensive security assurance audit of the Autonomys blockchain's code (formerly called Subspace), in partnership with the organization, reviewing critical components from April 2022 to October 2024. This audit assessed the Autonomys codebase for resilience against hacking and abuse scenarios. Key areas of scrutiny included the Autonomys runtime and consensus protocol. Our testing approach combined static and dynamic analysis techniques, leveraging both automated tools and manual inspection. We prioritized reviewing critical functionalities and conducting thorough security tests to ensure robustness of the Autonomys platform. During the audit, Security Research Labs collaborated closely with the Autonomys team, utilizing full access to source code and documentation to perform a rigorous assessment.

This report focuses on our assessment results concerning the Autonomys consensus chain; assessment results for upcoming features like domains and XDM are therefore excluded.

1.2 Observations and risk

The research team identified several issues ranging from high to informational level severity, many of which concerned the runtime and the consensus protocol. The Autonomys team, in cooperation with the auditors, remediated or accepted all identified issues.

1.3 Recommendations

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 will identify potential risks and ensure the integrity of the blockchain. Finally, improving documentation will facilitate better understanding and maintenance of the codebase, contributing to a more robust and secure network.

2 Evolution suggestions

Autonomys in collaboration with Security Research Labs has been continuously improving its security metrics and undertaken the security recommendations outlined in the previous audits to improve its 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.

2.1 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 stricter fee systems for transactions to disincentivize spamming.

Consider implementing an equivocation punishment mechanism. Although transfers are disabled for the launch of Autonomys, preventing double-spending attacks, the lack of equivocation checks still poses a risk. The network could remain in a persistent state of forks, leading to strain, potential denial-of-service attacks, and poor user experiences.

2.2 Secure development improvement suggestions

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

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. It also helps the audit team 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* [3].

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 exploit them. Not only does this approach broaden the pool of people actively searching for security flaws, but it also fosters a collaborative security environment, helping to identify and address issues more quickly and effectively.

2.3 Address currently open security issues

We recommend addressing already known security issues before going live with the mainnet to prevent adversaries 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.

2.4 Further recommended best practices

Extensive test implementation. Currently, most pallets have unit tests, but they are neither comprehensive nor 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.

Regular updates. New releases of Polkadot SDK may contain fixes for critical security issues. Since Autonomys is a product that heavily relies on Polkadot SDK, 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.

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 audits. 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 Chapter 6 onwards.

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 [4]. It should be noted that the protocol, repository, and codebase is still called Subspace.

Autonomys envisions to become a verticalized decentralized AI (deAI) stack encompassing high-throughput, permanent distributed storage, data availability, scalable distributed compute, and a modular execution layer. The deAI ecosystem should provide the components to build and deploy dApps (AI-powered dApps) and on-chain agents, equipping them with AI capabilities for dynamic and autonomous functionality. 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. Autonomys aims to pioneer a future where AI and humans coexist securely and autonomously, with a goal of launching this in 2025 onwards.

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 Proof-of-Archival-Storage (PoAS), 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.

The 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.

Notably, although the audit also covered Domains and Cross-Domain Messaging (XDM), these features are outside the scope of this report, which focuses solely on findings related to the consensus chain. Issues identified regarding Domains and XDM will be addressed in a future report.

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 [5] to guide efforts on exploring potential security flaws and realistic attack scenarios. Additionally, Autonomys online documentation and specification provided the auditors with a good runtime module design and implementation overview.

Repository	Priority	Component(s)	Reference
Subspace	High	Runtime	[6]
	Medium	Farmer	[6]

Table 2: In-scope Autonomys components with audit priority

4 Methodology

This report details the continuous security assurance results for the Autonomys network to create transparency in four steps: (1) threat modeling, (2) security design coverage checks, (3) implementation baseline check, and (4) remediation support. We applied the following four-step methodology when performing feature and PR reviews.

4.1 Threat modeling and attacks

The threat model framework aims to determine some specific areas of risk in the 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 adversary to successfully execute an attack. The hacking value is calculated as:

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

While *incentive* describes what an adversary might gain from performing an attack successfully, *effort* estimates the complexity of this 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, using expensive resources, or possessing 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.

Incentives and efforts are classified in 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, if 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.

The damage and hacking value are classified in Table 4.

Risk	Low hacking value	Medium hacking value	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 these three security promises that can be violated by a hacking attack: (1) confidentiality, (2) integrity, and (3) 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, for example, if an attacker can double-spend or mint tokens for themselves. Other threat scenarios do not yield an immediate monetary reward but could rather threaten to damage Autonomys' functionality and, in turn, its reputation.

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 the availability of blockchain systems include denial-of-service (DoS) attacks on the farmers, 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 the attacks that enable them is included in the threat model deliverable. This list can serve as a starting point for 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.

Security promise	Hacking value	Example threat scenarios	Hacking effort	Example attack ideas
Confidentiality	Medium	- Compromise a user's private key	High	- Targeted attacks to compromise a user's private key
Integrity	High	- Gain farming advantage over other farmers - Manipulate the blockchains history	Medium	- Slash other farmers by sending false offence reports - Equivocation attack
Availability	High	- Stall block production - Spam the blockchain with bogus transactions	Low	- Cheaply bloat blockchain storage - Make farmers crash

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:

- Coverage.** Is each potential security vulnerability sufficiently covered?
- Underlying assumptions.** Which assumptions must hold for the design to effectively reach the desired security goal?

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, the auditors ensured that sufficient protection measures against specific attacks were present.
4. Immediately reported any discovered vulnerability to the development team along with suggestions how to mitigate the flaw.

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

While performing static, fuzz, and dynamic tests establishes a baseline assurance, this audit focused on a manual code review of the Autonomys codebase to identify logic bugs, design flaws, and best practice deviations. We reviewed the Autonomys repository which contains Autonomys implementation up to commit `788d42e` from the *16th of October 2024*. We traced the intended functionality of the runtime modules in scope and assessed 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 an adversary 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.

Fuzz testing takes some valid input for a method under test, applies a semi-random mutation to it, and then invokes the method again with this semi-valid input. Through repeating this process, fuzz testing can unearth inputs that would cause a crash or other instances of undefined behavior (e.g., integer overflows) in the method under analysis. 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 auditors verify the fix to ensure that it mitigates the issue and does not introduce other bugs.

During the audit, findings were shared via the GitHub repository [6]. We also used a private Slack channel for asynchronous communication and status updates. In addition, biweekly joint fix meetings were held to provide detailed updates and address open questions.

5 Dynamic analysis assessment

During the audit, we utilized fuzz testing to identify bugs in the Autonomys runtime. By applying fuzz testing, we aim to uncover issues that may not have been detected through manual code review. For this purpose, we used consensus runtime (crates/subspace-runtime). 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 the Autonomys team. Continuous updates of the fuzzing campaign reassured us that common programming mistakes, such as unsafe math operations, were not introduced again into the codebase.

Fuzz harness creation: We chose our in-house developed and open-source tool Ziggy [7] as our fuzzing orchestration tool. As for the harness, we used a customized harness for the runtime which is comparable to our substrate runtime fuzzing harness. It is also available on Github [3].

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 Autonomys codebase coverage that the harness is targeting.


5.1 Consensus runtime fuzzer

Component	Code path	Coverage achieved
Consensus Runtime	crates/	30.60%

The coverage for the consensus runtime stands at 30.60%, with most uncovered lines being either inherent 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 but are out of scope for this report on the consensus chain.

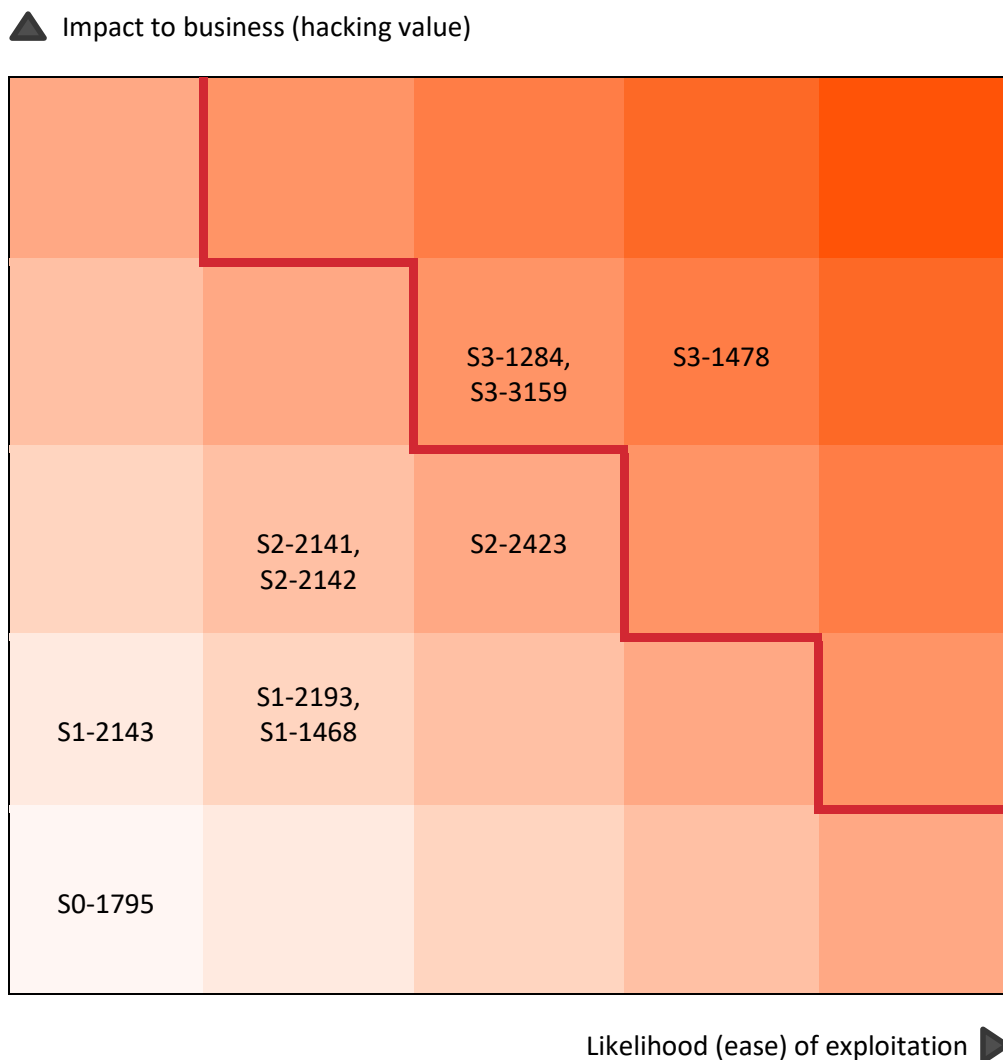
6 Findings summary

We identified 10 issues during our analysis of the runtime modules in scope in the Autonomys codebase that enabled some of the attacks outlined above. In summary, 3 high-severity, 3 medium-severity, 3 low-severity, and 1 information-level issues were found. An overview of all findings can be found in Table 6.

High	3	
Medium	3	
Low	3	
Informational	1	
Total Issues	10	

6.1 Risk profile

The chart below summarizes vulnerabilities according to business impact and likelihood of exploitation, increasing to the top right.



6.2 Issue summary

ID	Issue	Severity	Status
S3-1478 [8]	Only part of plotted data required to guess whether a sector contains a valid solution	High	Mitigated [9]
S3-1284 [10]	Absence of exponential fee adjustments, making it vulnerable to DoS	High	Mitigated [11]
S3-3159 [12]	Transaction filter bypass using batched transactions	High	Mitigated [13]
S2-2142 [14]	Burst Denial of Service in Proof of Time	Medium	Mitigated [15]
S2-2141 [16]	A faster timekeeper will outrun its competitors	Medium	Risk accepted
S2-2423 [17]	Incorrect usage of Pays : :No for unsigned calls	Medium	Mitigated [18]
S1-2143 [19]	No reward for running timekeeper in the network leading may lead to centralization	Low	Risk accepted
S1-2193 [20]	Time compression attack	Low	Risk accepted
S1-1468 [21]	XOR-based calculation of evaluation_seed may lead to seed reuse	Low	Mitigated [22]
S0-1795 [23]	Best practice deviation in consensus	Info	Mitigated [24]

Table 6: Findings overview

7 Detailed findings

7.1 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
Component	subspace-verification
Tracking	https://github.com/autonomys/subspace/issues/1478
Attack impact	Unfair farming advantages and compromised data availability in Proof-of-Archival-Storage consensus
Severity	High
Status	Mitigated [9]

Context

In the Proof of Space consensus, participants allocate data on their hard drives, in a process known as plotting. They then generate proofs to demonstrate to the network that they possess a specific set of data, which means that they solved the current block production challenge. The more space a participant dedicates, the higher their chance of solving the next block production challenge.

Issue description

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 adversaries.

Risk

Attackers can gain an unfair farming advantage, pushing out honest farmers and threatening the availability guarantees of the Proof-of-Archival-Storage by duplicating data.

Mitigation

To mitigate this, use hashing instead of applying the XOR operation. For example, using HMAC with `sector_slot_challenge` and plotted data ensures that all bits are relevant. Larger data chunks (e.g., 32 bytes) can reduce hash calculations for performance.

7.2 Absence of exponential fee adjustments makes transactions vulnerable to DoS

Attack scenario	Attacker spams transactions with regular fees, blocking zero-tip transactions
Component	subspace-runtime
Tracking	https://github.com/autonomys/subspace/issues/1284
Attack impact	DoS against legitimate transactions, disrupting automated trading bots and causing economic losses
Severity	High
Status	Mitigated [11]

Context

The fee adjustment mechanism is responsible for updating fees based on the utilization of block resources. This is important for ensuring that the chain adapts effectively to changes in supply and demand.

Issue description

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 adversaries 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.

Risk

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

Mitigation

To mitigate this issue, 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.

7.3 Transaction filter bypass using batched transactions

Attack scenario	An attacker bypasses transfer restrictions to move funds illicitly
Component	subspace-runtime
Tracking	https://github.com/autonomys/subspace/issues/3159
Attack impact	Compromises system integrity by disrupting tokenomics and evasion of restrictions
Severity	High
Status	Mitigated [13]

Context

Balance transfers allow accounts to move assets between one another. Furthermore pallet-utility, with its batching functionality, can bundle many calls, such as transfers, into one, allowing them to be executed atomically. Balance transfers are decided to be disabled on the initial launch of Autonomys blockchain.

Issue description

To prevent balance transfers on Autonomys blockchain, a filter has been implemented in the form of a singed extension. However, this filter can be bypassed by using the batching functionality of the pallet-utility which is enabled in the runtime.

Risk

An attacker could exploit this loophole to circumvent the transfer restrictions and move assets to another account, compromising the system's integrity. This could cause unintended side effects such as disrupting tokenomics, or allowing malicious actors to evade sanctions or restrictions meant to control specific assets.

Mitigation

We recommend filtering the batch calls from the utility pallet to prevent nesting of the calls in this manner. Additionally, we recommend adding some tests to prevent reintroduction of the same issue in the future.

7.4 Burst denial of service in proof of time

Attack scenario	Attacker spams faulty PoT messages to delay verification and disrupt node operations
Component	subspace-proof-of-time
Tracking	https://github.com/autonomys/subspace/issues/2142
Attack impact	Node misses block creation opportunities, leading to network disruptions and slowdowns
Severity	Medium
Status	Mitigated [15]

Context

Proof of Time is a protocol that proves time has passed to prevent long-range attacks in the Proof of Space consensus protocol. The proofs of time are generated by timekeepers, who propagate these proofs through the network with the goal of reaching the farmer. When the proof reaches the farmer, they will verify it and include it in their block, given the farmer has a valid Proof of Space solution.

Issue description

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, which costs 2.4 seconds. Though nodes sending faulty PoTs are banned, attackers can easily spawn new instances with fresh IP addresses, especially using IPv6.

Risk

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.

Mitigation

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

7.5 A faster timekeeper will outrun its competitors

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

Context

Proof of Time is a protocol designed to demonstrate the passage of time and prevent long-range attacks in the Proof of Space consensus mechanism. Timekeepers generate these proofs and distribute them across the network, aiming to deliver them to the farmer. If the farmers have a valid Proof of Space solution, they will incorporate the Proof of Time into their block.

Issue description

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.

As a result, older timekeepers become obsolete, which reduces 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.

Risk

An attacker running the fastest timekeeper can abuse this by trying to pre-solve PoS and operator slots, and if one is found, prevent collisions with competitors by announcing multiple following PoTs. This prevents others from 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.

Mitigation

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

7.6 Incorrect usage of Pays::No for unsigned calls

Attack scenario	Malicious users submit signed transactions for unsigned calls to avoid fees
Component	pallet-subspace
Tracking	https://github.com/autonomys/subspace/issues/2423
Attack impact	Wasted chain resources and potential network degradation
Severity	Medium
Status	Mitigated [18]

Context

In blockchains, transactions are processed on nodes. These are machines owned by independent entities that provide computational resources. To prevent users from exploiting these resources for free, each transaction must incur a cost.

Issue description

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 won't incur transaction fees due to the `Pays::No` annotation.

Risk

This allows attackers to waste chain resources without paying fees.

Mitigation

To mitigate this issue, 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.

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

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

Context

Proof of Time is a protocol intended to verify the passage of time and protect against long-range attacks in the Proof of Space consensus mechanism. Timekeepers, who can be separate entities from farmers, generate these proofs and propagate them throughout the network to reach the farmers. If the farmers possess a valid Proof of Space solution, they will include the proof in their block.

Issue description

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.

Risk

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.

Mitigation

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.

7.8 Time compression attack

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

Context

Proof of Time is a protocol intended to verify the passage of time and protect against long-range attacks in the Proof of Space consensus mechanism. Timekeepers, who can be separate entities from farmers, generate proofs based on chained AES-128 calculations, which take approximately 1 second to complete. These proofs are essential for farmers to include in their blocks for them to be considered valid submissions.

Issue description

An adversary 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 solutions in time. This results in the attacker's PoS solution being the only viable one on the blockchain.

Risk

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

Mitigation

To mitigate this issue, 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.

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

Attack scenario	Brute-forcing possible values of <code>sector_id</code> due to the limited variability introduced by XORing a u16 variable with a 256-bit hash
Component	subspace-proof-of-space
Tracking	https://github.com/autonomys/subspace/issues/1468
Attack impact	Potential exploitation in the consensus process
Severity	Low
Status	Mitigated [22]

Context

In the Proof of Space consensus, participants allocate data on their hard drives through a process called plotting. They generate proofs to demonstrate to the network that they hold a specific set of data, which serves as proof of winning the current block production slot. The more space a participant dedicates, the greater their chance of winning the next block production slot.

Issue description

The consensus protocol uses a u16 variable, such as `sector_index`, to derive `sector_id` by applying the XOR operation to 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.

Risk

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

Mitigation

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

7.10 Best practice deviation in consensus

Attack scenario	A farmer stores partial data to reduce storage costs
Component	subspace-proof-of-space
Tracking	https://github.com/autonomys/subspace/issues/1795
Attack impact	Unfair farming advantages
Severity	Info
Status	Mitigated [24]

Context

In the Proof of Space consensus mechanism, participants allocate storage space on their hard drives in a process called plotting. They generate proofs to show the network that they hold a specific set of data. This serves as evidence that they have solved the current block production challenge. The more storage space a participant dedicates, the greater their chances of solving the next block production challenge are.

Issue description

The consensus protocol uses small variables in some XOR operations. For example, `sector_index` is a `u16` and is used to derive `sector_id`. This is done by XORing this value with the hash of the public key (see also documentation of the verification). As `sector_index` is a `u16` and the hash of the public key is 256 bits, 240 bits will stay the same. We were unable to find a way to abuse this in the verification phase of the consensus as the derived values (here `sector_slot_challenge`) are later hashed in `keyed_hash(sector_slot_challenge, audit_chunk)`. Therefore, the severity of the issue is low.

Risk

XOR applies only to a small number of bits and does not change the whole value. This could be abused in the following manner in the various steps of the consensus:

- `u16` is a range that can be brute-forced, which can be used to precalculate all the possible values of a certain step.
- If the XOR only possibly changes a small part of the resulting variable, the other bits are fixed and known.

Mitigation

We recommend hashing small values that are used for applying the XOR operation in the consensus protocol.

8 Bibliography

- [1] [Online]. Available: https://securityresearchlabs.sharepoint.com/:b:/s/SubspaceNetwork/EUwNweQhTX1GnY0oCb pTs5MBfrmshDYndBIC3ZZ5_dRQkg.
- [2] [Online]. Available: <https://securityresearchlabs.sharepoint.com/:b:/s/SubspaceNetwork/EYWtgwb2nidNizLQYmy 56OcB99paH0XJu7A-W6GZ9hTfkg>.
- [3] [Online]. Available: <https://github.com/srlabs/substrate-runtime-fuzzer>.
- [4] [Online]. Available: <https://medium.com/subspace-network/becoming-autonomys-new-vision-new-ceo-new-mainnet-launch-date-baa8accc1a76>.
- [5] [Online]. Available: <https://securityresearchlabs.sharepoint.com/:x:/s/SubspaceNetwork/Ed1iEa767FdJo8jazVAzW toBlzLmnyoKeXeJ0LKXYmog>.
- [6] [Online]. Available: <https://github.com/autonomys/subspace>.
- [7] [Online]. Available: <https://github.com/srlabs/ziggy>.
- [8] [Online]. Available: <https://github.com/autonomys/subspace/issues/1478>.
- [9] [Online]. Available: <https://github.com/autonomys/subspace/pull/1604>.
- [10] [Online]. Available: <https://github.com/autonomys/subspace/issues/1284>.
- [11] [Online]. Available: <https://github.com/autonomys/subspace/pull/2156>.
- [12] [Online]. Available: <https://github.com/autonomys/subspace/issues/3159>.
- [13] [Online]. Available: <https://github.com/autonomys/subspace/pull/3134>.
- [14] [Online]. Available: <https://github.com/autonomys/subspace/issues/2142>.
- [15] [Online]. Available: <https://github.com/autonomys/subspace/pull/2320>.
- [16] [Online]. Available: <https://github.com/autonomys/subspace/issues/2141>.
- [17] [Online]. Available: <https://github.com/autonomys/subspace/issues/2423>.
- [18] [Online]. Available: <https://github.com/autonomys/subspace/pull/2842>.

- [19] [Online]. Available: <https://github.com/autonomys/subspace/issues/2143>.
- [20] [Online]. Available: <https://github.com/autonomys/subspace/issues/2193>.
- [21] [Online]. Available: <https://github.com/autonomys/subspace/issues/1468>.
- [22] [Online]. Available: <https://github.com/autonomys/subspace/pull/1607>.
- [23] [Online]. Available: <https://github.com/autonomys/subspace/issues/1625>.
- [24] [Online]. Available: <https://github.com/autonomys/subspace/pull/1818>.
- [25] [Online]. Available: <https://github.com/autonomys/subspace/issues/2439>.
- [26] [Online]. Available: <https://github.com/autonomys/subspace/issues/1468>.
- [27] [Online]. Available: <https://github.com/autonomys/subspace/issues/1625>.

Appendix A: Technical services

Security Research Labs delivers extensive technical expertise to meet your security needs. Our comprehensive services include software and hardware evaluation, penetration testing, red team testing, incident response, and reverse engineering. We aim to equip your organization with the security knowledge essential for achieving your objectives.

SOFTWARE EVALUATION We provide assessments of application, system, and mobile code, drawing on our employees' decades of experience in developing and securing a wide variety of applications. Our work includes design and architecture reviews, data flow and threat modelling, and code analysis with targeted fuzzing to find exploitable issues.

BLOCKCHAIN SECURITY ASSESSMENTS We offer specialized security assessments for blockchain technologies, focusing on the unique challenges posed by decentralized systems. Our services include smart contract audits, consensus mechanism evaluations, and vulnerability assessments specific to blockchain infrastructure. Leveraging our deep understanding of blockchain technology, we ensure your decentralized applications and networks are secure and robust.

POLKADOT ECOSYSTEM SECURITY We provide comprehensive security services tailored to the Polkadot ecosystem, including parachains, relay chains, and cross-chain communication protocols. Our expertise covers runtime misconfiguration detection, benchmarking validation, cryptographic implementation reviews, and XCM exploitation prevention. Our goal is to help you maintain a secure and resilient Polkadot environment, safeguarding your network against potential threats.

TELCO SECURITY We deliver specialized security assessments for telecommunications networks, addressing the unique challenges of securing large-scale and critical communication infrastructures. Our services encompass vulnerability assessments, secure network architecture reviews, and protocol analysis. With a deep understanding of telco environments, we ensure robust protection against cyberthreats, helping maintain the integrity and availability of your telecommunications services.

DEVICE TESTING Our comprehensive device testing services cover a wide range of hardware, from IoT devices and embedded systems to consumer electronics and industrial controls. We perform rigorous security evaluations, including firmware analysis, penetration testing, and hardware-level assessments, to identify vulnerabilities and ensure your devices meet the highest security standards. Our goal is to safeguard your hardware against potential attacks and operational failures.

CODE AUDITING We provide in-depth code auditing services to identify and mitigate security vulnerabilities within your software. Our approach includes thorough manual reviews, automated static analysis, and targeted fuzzing to uncover critical issues such as logic flaws, insecure coding practices, and exploitable vulnerabilities. By leveraging our expertise in secure software development, we help you enhance the security and reliability of your codebase, ensuring robust protection against potential threats.

PENETRATION & RED TEAM TESTING We perform high-end penetration tests that mimic the work of sophisticated attackers. We follow a formal penetration testing methodology that emphasizes repeatable, actionable results that give your team a sense of the overall security posture of your organization.

SOURCE CODE-ASSISTED SECURITY EVALUATIONS We conduct security evaluations and penetration tests based on our code-assisted methodology, allowing us to find deeper vulnerabilities, logic flaws, and fuzzing targets than a black-box test would reveal. This gives your team a stronger assurance that the significant security-impacting flaws have been found and corrected.

SECURITY DEVELOPMENT LIFECYCLE CONSULTING We guide organizations through the Security Development Lifecycle to integrate security at every phase of software development. Our services include secure coding training, threat modelling, security design reviews, and automated security testing implementation. By embedding security practices into your development processes, we help you proactively identify and mitigate vulnerabilities, ensuring robust and secure software delivery from inception to deployment.

REVERSE ENGINEERING We assist clients with reverse engineering efforts not associated with malware or incident response. We also provide expertise in investigations and litigation by acting as experts in cases of suspected intellectual property theft.

HARDWARE EVALUATION We evaluate new hardware devices ranging from novel microprocessor designs, embedded systems, mobile devices, and consumer-facing end products to core networking equipment that powers Internet backbones.

VULNERABILITY PRIORITIZATION We streamline vulnerability information processing by consolidating data from compliance checks, audit findings, penetration tests, and red team insights. Our prioritization and automation strategies ensure that the most critical vulnerabilities are addressed promptly, enhancing your organization's security posture. By systematically categorizing and prioritizing risks, we help you focus on the most impactful threats, ensuring efficient and effective remediation efforts.

SECURITY MATURITY REVIEW We conduct comprehensive security maturity reviews to evaluate your organization's current security practices and identify areas for improvement. Our assessments cover a wide range of criteria, including policy development, risk management, incident response, and security awareness. By benchmarking against industry standards and best practices, we provide actionable insights and recommendations to enhance your overall security posture and guide your organization toward achieving higher levels of security maturity.

SECURITY TEAM INCUBATION We provide comprehensive support for building security teams for new, large-scale IT ventures. From Day 1, our ramp-up program offers essential security advisory and assurance, helping you establish a robust security foundation. With our proven track record in securing billion-dollar investments and launching secure telco networks globally, we ensure your new enterprise is protected against cyberthreats from the start.

HACKING INCIDENT SUPPORT We offer immediate and comprehensive support in the event of a hacking incident, providing expert analysis, containment, and remediation. Our services include detailed forensics, malware analysis, and root cause determination, along with actionable recommendations to prevent future incidents. With our rapid response and deep expertise, we help you mitigate damage, recover swiftly, and strengthen your defenses against potential threats.