Syncra Smart Contracts Secure Code Review

Technical Report

Syncra

04 September 2024
Version: 1.1

Kudelski Security – Nagravision Sàrl

Corporate Headquarters
Kudelski Security – Nagravision Sàrl
Route de Genève, 22-24
1033 Cheseaux sur Lausanne
Switzerland

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Syncra (“the Client”) engaged Kudelski Security (“Kudelski”, “We”) to perform the Syncra Smart Contracts Secure Code Review.

The assessment was conducted remotely by the Kudelski Security Team.

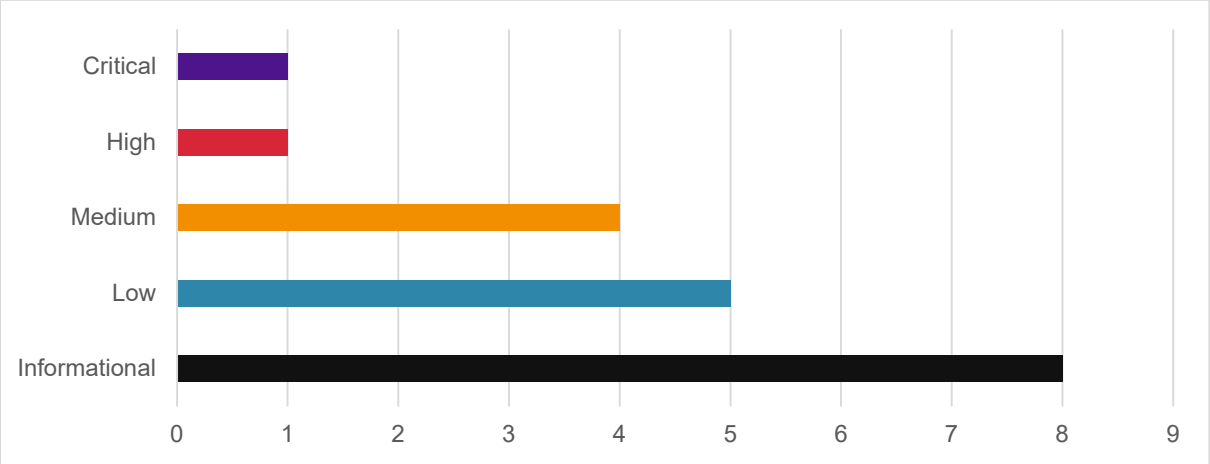
The review took place between 22nd July 2024 and 22nd August 2024, and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks that were discovered.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations based on the result of our tests.

Key Findings

The following are the major themes and issues identified during the audit period. These, along with other items within the findings section, should be prioritized for remediation to reduce to the risk they pose.

- Vote Counted Incorrectly
- Misuse of Voting Power in cast_vote Function
- Proposal Hash can be set Manually



Findings ranked by severity

1. PROJECT SUMMARY

This report summarizes the engagement, tests performed, and findings. It also contains detailed descriptions of the discovered vulnerabilities, steps the Kudelski Security Team took to identify and validate each issue, as well as any applicable recommendations for remediation.

1.1 Context

The `contracts.syncra.xyz` repository contains a set of rust traits and DAO smart contracts built using ink! with some features based on OpenBrush.

1.2 Scope

The scope consisted in specific ink! files and folders located at:

- Commit hash: `ebcf9e44aeba296079ce7bef19203bfa0f769515`
- Source code repo: https://github.com/SyncraDAO/contracts.syncra.xyz/

The folders in scope are:

```
contracts/
    ├── governor_psp22_lock
    ├── governor_whitelist
    ├── proposal_basic
    ├── proposal_poll
    ├── proposal_weighted
    ├── psp22
    ├── psp34
    ├── syncra_governor_factory
    ├── syncra_subscription
    ├── syncra_subscription_factory
    ├── syncra_subscription_manager
    ├── syncra_treasury
    └── syncra_treasury_factory
```

```
governance_traits/
    ├── lib.rs
    ├── access_control
    ├── governor
    ├── lock
    ├── psp22
    ├── psp34
    ├── treasury
    └── whitelist
```

The goal of the evaluation was to perform a security audit on the source code.

- No additional systems or resources were in scope for this assessment.
- The dependencies are out of scope of the review.
- Test codes are out of scope.
- Access check by DAO roles is out of scope since the Client plan to remove them.

Follow-Up

After the initial report (V1.0) was delivered, Syncra team addressed all findings and some of observations in the following commit:

- Merge pull request #13 from SyncraDAO/audit (commit 360b6ac0dd6c6ca75bac354a5a96e70a5847e175).

1.3 Remarks

During the code review, the following positive observations were noted regarding the scope of the engagement:

- The code is well structured.
- Quick and open communication via Telegram
- The developers have made a careful and in-depth analysis of their project.
- We had regular and enriching technical exchanges on various topics.

1.4 Additional Note

It is important to notice that, although we did our best in our analysis, no code audit assessment is per se guarantee of absence of vulnerabilities. Our effort was constrained by resource and time limits, along with the scope of the agreement.

In assessing the severity of some of the findings we identified, we kept in mind both the ease of exploitability and the potential damage caused by an exploit.

While assessing the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. This is a solid baseline for severity determination. Information about the severity ratings can be found in Chapter Vulnerability Scoring System of this document.

2. STATIC CODE ANALYSIS

2.1 Cargo audit

`Cargo audit` (v0.20.0) identified 3 vulnerabilities and 4 allowed warnings. They are reported in Chapter 3. The output of cargo audit is in the Appendix **Error! Reference source not found.**.

2.2 Cargo clippy

`Cargo clippy` (v0.1.77) identified multiple findings. Some of them, which are relevant to the security, are reported in Chapter 4. The output of `cargo clippy` is in the Appendix **Error! Reference source not found.**.

2.3 Cargo nextest

`Cargo nextest` (v0.9.53) didn't identify any finding.

2.4 Semgrep

`Semgrep` (v1.61.1) did not identify any finding.

3. TECHNICAL DETAILS OF SECURITY FINDINGS

This chapter provides detailed information on each of the findings, including methods of discovery, explanation of severity determination, recommendations, and applicable references.

The following table provides an overview of the security findings.

#	SEVERITY	TITLE	STATUS
KS-SYN-F-01	Critical	Vote Counted Incorrectly	Resolved
KS-SYN-F-02	High	Misuse of Voting Power in cast_vote Function	Resolved
KS-SYN-F-03	Medium	Proposal Hash can be set Manually	Resolved
KS-SYN-F-04	Medium	Proposer Threshold Not Validated	Resolved
KS-SYN-F-05	Medium	Quorum Calculation May Lead to Unexpected Voting Outcomes	Resolved
KS-SYN-F-06	Medium	Risk of Unwanted Proposals' Rules Deletion	Resolved
KS-SYN-F-07	Low	Emergency Shutdown/ Pause Not Implemented	Resolved
KS-SYN-F-08	Low	Number of Proposal Not Restricted	Acknowledged
KS-SYN-F-09	Low	Minimum Delay Period Not Enforced	Acknowledged
KS-SYN-F-10	Low	Minimum Total Votes Not Checked	Resolved
KS-SYN-F-11	Low	Voting End for Lock Not Set	Acknowledged

Findings overview.

3.1 KS-SYN-F-01 Vote Counted Incorrectly

Severity	Impact	Likelihood	Status
Critical	High	High	Resolved

Description

In the function `check_proposal_upvoted_quorum_reached`, the vote for abstain is not counted properly; instead of `VotingOption::Abstain`, it counts `VotingOption::No`. Hence, the vote for yes is not calculated correctly, leading to the misjudgment of `upvoted` and `quorum_reached` status.

3.2 KS-SYN-F-02 Misuse of Voting Power in cast_vote Function

Severity	Impact	Likelihood	Status
High	High	High	Resolved

Description

The `cast_vote` function in the `governor_whitelist/lib.rs` is used to cast a vote on an active proposal. It validates if the proposal is active and calcuates the voting power. It is observed that the function does not correctly handle the distribution of voting power across multiple voting options. It calculates the voter's total power and then assigns this total power to each voting option, effectively multiplying the voter's power by the number of options they vote for.

3.3 KS-SYN-F-03 Proposal Hash can be set Manually

Severity	Impact	Likelihood	Status
Medium	High	Low	Resolved

Description

When creating a proposal, the user calling the `propose` function needs to provide a `proposal_hash` that will be used as a proposal ID. This proposal ID is used to identify the proposal during voting and execution. The issue is that this `proposal_hash` can be chosen by the user creating the proposal and is not generated by any cryptographic hash function (e.g., SHA3).

3.4 KS-SYN-F-04 Proposer Threshold Not Validated

Severity	Impact	Likelihood	Status
Medium	High	Low	Resolved

Description

In the `contracts/governor_psp22_lock/lib.rs`, the proposer threshold is checked in the function `propose`, but no error is returned regardless of the locked amount. In comparison, in the `contracts/governor_whitelist/lib.rs`, the proposer threshold is checked out and an error is returned if the proposer votes are less than the threshold.

3.5 KS-SYN-F-05 Quorum Calculation May Lead to Unexpected Voting Outcomes

Severity	Impact	Likelihood	Status
Medium	High	Low	Resolved

Description

`check_proposal_upvoted_quorum_reached` function calculates the quorum based on the total number of members at the time the function is called, typically after the voting period has ended. This means that if members are added or removed during the voting period, it could potentially impact the quorum calculation.

3.6 KS-SYN-F-06 Risk of Unwanted Proposals' Rules Deletion

Severity	Impact	Likelihood	Status
Medium	High	Low	Resolved

Description

In syncra, a proposal are defined by a set of rules. This rules are stored into a mapping, `proposal_rules: Mapping<u8, ProposalRules>`. Any new proposal will take the last create rules which is identified with `proposal_rules_id`. This variable is incremented by one every time is a new rules is set by the central authority. As `proposal_rules_id` is as unsigned interger coded on 8 bits (type u8) and this means that its maximum value is 255.

The incrementation is performed by using `saturing_add`, which outputs 255 when the addition's result is bigger than 255. This means that after keeping incrementing `proposal_rules_id` 255 times, all new rules will share the same identification any new rules setting will erase the previous one setted.

3.7 KS-SYN-F-07 Emergency Shutdown/ Pause Not Implemented

Severity	Impact	Likelihood	Status
Low	High	Low	Resolved

Description

An emergency shutdown mechanism can halt all transactions and prevent additional damage temporarily. It could be useful in the event of a serious security issue. However, it seems this critical functionality is not implemented yet.

3.8 KS-SYN-F-08 Number of Proposal Not Restricted

Severity	Impact	Likelihood	Status
Low	Low	Low	Acknowledged

Description

The function `enable_proposal_contract` does not check how many proposals have been already enabled.

3.9 KS-SYN-F-09 Minimum Delay Period Not Enforced

Severity	Impact	Likelihood	Status
Low	High	Low	Acknowledged

Description

The function `set_proposal_rules` does not check whether the delay parameters satisfy the minimum requirements. A vote delay and execution delay between the proposal passing and its execution is impotant to reduce the risk of flash loan attack. This gives the community time to react and potentially challenge a malicious proposal.

3.10 KS-SYN-F-10 Minimum Total Votes Not Checked

Severity	Impact	Likelihood	Status
Medium	High	Low	Resolved

Description

The function `check_proposal_upvoted_quorum_reached` does not check the `total_votes` satisfies the minimum requirements. The low participation rates of (delegated) tokens can be seen as a considerable risk factor.

3.11 KS-SYN-F-11 Voting End for Lock Not Set

Severity	Impact	Likelihood	Status
Low	Low	Low	Acknowledged

Description

The voting end is passed to the function `lock` as a parameter, however, it is not used to set `lock.until_timestamp` in the function `lock`.

4. OBSERVATIONS

This chapter contains additional observations that are not directly related to the security of the code, and as such have no severity rating or remediation status summary. These observations are either minor remarks regarding good practice or design choices or related to implementation and performance. These items do not need to be remediated for what concerns security, but where applicable we include recommendations.

#	SEVERITY	TITLE	STATUS
KS-SYN-O-01	Informational	Incomplete Compilation Script	Informational
KS-SYN-O-02	Informational	Lack of Documents and Comments	Informational
KS-SYN-O-03	Informational	Test Coverage Not Sufficient	Informational
KS-SYN-O-04	Informational	Outdated Dependencies	Informational
KS-SYN-O-05	Informational	Recommended Secure Coding Practices	Informational
KS-SYN-O-06	Informational	Proposal_core.executed Set Redundantly	Informational
KS-SYN-O-07	Informational	Some Error Codes Not Used	Informational
KS-SYN-O-08	Informational	Role Grant Duplication Not Checked	Informational
KS-SYN-O-09	Informational	Overflow Protection is Disabled	Informational

Observations overview.

4.1 KS-SYN-O-01 Incomplete Compilation Script

Description

The compilation script is not complete and the following script is missing in `scripts/compile-all-contracts.sh`.

Furthermore, it would be recommended to compile the code from the top directory by using Makefile.

4.2 KS-SYN-O-02 Lack of Documents and Comments

Description

Documents and in-line comments are not sufficiently provided. This should be clarified in the README.md file.

- It is not specified which version of `rustup` toolchain works for building the Syncra contracts.
- No procedure is given to test the contracts.

4.3 KS-SYN-O-03 Test Coverage Not Sufficient

Description

According to the cargo llvm-cov tool (v0.6.10), the overall test coverage of code in scope reaches far less than the full coverage.

4.4 KS-SYN-O-04 Outdated Dependencies

Description

The `cargo audit` tool identified that the following dependencies include vulnerabilities or are unmaintained:

- `curve25519-dalek`: RUSTSEC-2024-0344, Vulnerability, Upgrade to >=4.1.3
- `ed25519-dalek`: RUSTSEC-2022-0093, Vulnerability, Upgrade to >=2
- `ansi_term`: RUSTSEC-2021-0139, Unmaintained
- `mach`: RUSTSEC-2020-0168, Unmaintained
- `parity-wasm`: RUSTSEC-2022-0061, Unmaintained
- `term_size`: RUSTSEC-2020-0163, Unmaintained

4.5 KS-SYN-O-05 Recommended Secure Coding Practices

Description

The following coding suggestions might be useful to help readers understand the code better:

- In `contracts/governor_psp22_lock/lib.rs, line 360`, it would be clearer to use the function `is_enabled_proposal_contract()` instead.
- In `governance_traits/whitelist/types.rs, line 71`, the comment does not match the code.
- In `syncra_subscription_manager/lib.rs`, an identical code block (line 62-65) is repeated 6 times. This code block can be replaced by a sub-function for better visibility, similarly to what has been done in `contracts/syncra_treasury/lib.rs` using the function `validate_caller`.

4.6 KS-SYN-O-06 Proposal_core.executed Set Redundantly

Description

The flag `proposal_core.executed` is set to `true` in the success of the call function. However, after the loop, `proposal_core.executed` is always set to `true` even the proposal has an empty transaction. Hence, there is no reason for the flag to set to `true` inside the loop.

4.7 KS-SYN-O-07 Some Error Codes Not Used

Description

The error codes are defined but some of them are never used. It is not clear whether they are supposed to be used or they are forgotten. Also, some error code names are not clear.

- In `governance_traits/governor/types.rs, line 110,` the error code `TransactionsHash`, `CalleeNotSpecified`, and `NotEnoughPower` are never used.
- In `governance_traits/access_control/types.rs, line 82,` the error code `RoleAlreadyExists` is never used.
- In `governance_traits/psp22/types.rs, line 15,` the error code `ZeroRecipientAddress`, `ZeroSenderAddress`, and `SafeTransferCheckFailed` are never used.
- In `syncra_subscription/lib.rs, line 35,` the error code `NotPlatformAdmin` is never used.
- In `syncra_subscription_factory/lib.rs, line 23,` the error code `InstantiationFailed` is never used.
- In `governance_traits/governor/types.rs, line 110`, the error code name `VotingPeriod` and `ProposalThreshold` are not clear. It would be better to name them, e.g. `VotingPeriodNotMatched` and `ProposalThresholdNotReached`.

4.8 KS-SYN-O-08 Role Grant Duplication Not Checked

Description

The function `grant_role` does not check whether `role_id` has been already granted to `to_account`. Although a duplicated role seems no harm, it could give a hint that there is a logical error or an undetected attack from a malicious hacker.

In comparison, `governance_traits/whitelist/types.rs, line 177`, the member account is checked before added whether it is duplicated.

4.9 KS-SYN-O-09 Overflow Protection is Disabled

Description

The Syncra disabled the overflow protection in most of the smart contracts, does not use the function `checked_add` and `checked_sub` to perform addition and substraction.

We estimated the risk of having overflow or underflow is extremely low and therefore we kept it as an observation instead of a security finding.

5. METHODOLOGY

For this engagement, Kudelski Security used a methodology that is described at a high level in this chapter. This is broken up into the following phases.

5.1 Kickoff

The Kudelski Security Team set up a kickoff meeting where project stakeholders were gathered to discuss the project as well as the responsibilities of participants. During this meeting, we verified the scope of the engagement and discussed the project activities.

5.2 Ramp-up

Ramp-up consisted of the activities necessary to gain proficiency on the particular project. This included the steps required for gaining familiarity with the codebase and technological innovations utilized.

5.3 Review

The review phase is where most of the work on the engagement was performed. In this phase we have analyzed the project for flaws and issues that could impact the security posture. The review for this project was performed using manual methods and utilizing the experience of the reviewer. No dynamic testing was performed, only the use of custom-built scripts and tools was used to assist the reviewer during the testing. We discuss our methodology in more detail in the following subsections.

Code Review

Kudelski Security Team reviewed the code within the project utilizing an appropriate IDE. During every review, the team spends considerable time working with the client to determine correct and expected functionality, business logic, and content, to ensure that findings incorporate this business logic into each description and impact. Following this discovery phase, the team works through the following categories:

- authentication (*e.g.* A07:2021, CWE-306)
- authorization and access control (*e.g.* A01:2021, CWE-862)
- auditing and logging (*e.g.* A09:2021)
- injection and tampering (*e.g.* A03:2021, CWE-20)
- configuration issues (*e.g.* A05:2021, CWE-798)
- logic flaws (*e.g.* A04:2021, CWE-190)
- cryptography (*e.g.* A02:2021)

These categories incorporate common weaknesses and vulnerabilities such as the OWASP Top 10 and MITRE Top 25.

5.4 Smart Contracts

We reviewed the smart contracts, checking for additional specific issues that can arise such as:

- assessment of smart contract admin centralization
- reentrancy attacks and external contracts interactions
- verification of compliance with existing standards such as ERC20 or PSP34
- unsafe arithmetic operations such as overflow and underflow
- verification dependance on timestamp
- access control verification to ensure that only authorized users can call sensitive functions.

5.5 Reporting

Kudelski Security delivered to the Client a preliminary report in PDF format that contained an executive summary, technical details, and observations about the project.

In the report we not only point out security issues identified but also observations for improvement. The findings are categorized into several buckets, according to their overall severity: **Critical**, **High**, **Medium**, **Low**.

Observations are considered to be **Informational**. Observations can also consist of code review, issues identified during the code review that are not security related, but are general best practices and steps, that can be taken to lower the attack surface of the project.

The technical details are aimed more at developers, describing the issues, the severity ranking and recommendations for mitigation.

5.6 Verify

After the preliminary findings have been delivered, we verify the fixes applied by Syncra. After these fixes were verified, we updated the status of the finding in the report.

The output of this phase is the final report with any mitigated findings noted.

6. VULNERABILITY SCORING SYSTEM

Kudelski Security utilizes a custom approach when computing the vulnerability score, based primarily on the **Impact** of the vulnerability and **Likelihood** of an attack.

Each metric is assigned a ranking of either low, medium or high, based on the criteria defined below. The overall severity score is then computed as described in the next section.

Severity

Severity is the overall score of the finding, weakness or vulnerability as computed from Impact and Likelihood. Other factors, such as availability of tools and exploits, number of instances of the vulnerability and ease of exploitation might also be taken into account when computing the final severity score.

IMPACT / LIKELIHOOD	LOW	MEDIUM	HIGH
HIGH	MEDIUM	HIGH	HIGH
MEDIUM	LOW	MEDIUM	HIGH
LOW	LOW	LOW	MEDIUM

Compute overall severity from Impact and Likelihood. The final severity factor might vary depending on a project's specific context and risk factors.

- **Critical** The identified issue may be immediately exploitable, causing a strong and major negative impact system-wide. They should be urgently remediated or mitigated.
- **High** The identified issue may be directly exploitable causing an immediate negative impact on the users, data, and availability of the system for multiple users.
- **Medium** The identified issue is not directly exploitable but combined with other vulnerabilities may allow for exploitation of the system or exploitation may affect singular users. These findings may also increase in severity in the future as techniques evolve.
- **Low** The identified issue is not directly exploitable but raises the attack surface of the system. This may be through leaking information that an attacker can use to increase the accuracy of their attacks.
- **Informational** findings are best practice steps that can be used to harden the application and improve processes. Informational findings are not assigned a severity score and are classified as Informational instead.

Impact

The overall effect of the vulnerability against the system or organization based on the areas of concern or affected components discussed with the client during the scoping of the engagement.

- **High** The vulnerability has a severe effect on the company and systems or has an affect within one of the primary areas of concern noted by the client.
- **Medium** It is reasonable to assume that the vulnerability would have a measurable effect on the company and systems that may cause minor financial or reputational damage.
- **Low** There is little to no affect from the vulnerability being compromised. These vulnerabilities could lead to complex attacks or create footholds used in more severe attacks.

Likelihood

The likelihood of an attacker discovering a vulnerability, exploiting it, and obtaining a foothold varies based on a variety of factors including compensating controls, location of the application, availability of commonly used exploits, difficulty of exploitation and institutional knowledge.

- **High** It is extremely likely that this vulnerability will be discovered and abused.
- **Medium** It is likely that this vulnerability will be discovered and abused by a skilled attacker.
- **Low** It is unlikely that this vulnerability will be discovered or abused when discovered.

7. REFERENCES

- https://docs.syncra.xyz/
- Typical governance vulnerabilities: from DAO building to DAO smart contract audit
- SoK: Attacks on DAOs

8. CONCLUSION

The objective of this code review was to evaluate the overall security of the code base and identify any vulnerabilities that would put the product at risk.

The Kudelski Security Team identified 11 security issues: 1 critical risk, 1 high risk, 4 medium risks, and 5 low risks. On average, the effort needed to mitigate these risks is estimated as `medium`.

In order to mitigate the risks posed by this engagement's findings, the Kudelski Security Team recommends applying the following best practices:

- Test the vote counting
- Use proper cryptographic hash function to compute the proposal IDs
- Implement protection against overflow/underflow risk.

Kudelski Security remains at your disposal should you have any questions or need further assistance.

Kudelski Security would like to thank Syncra for their trust, help and support over the course of this engagement and is looking forward to cooperating in the future.