Secure Code Review of Gas Station

Technical Report

Pagoda

22 July 2024
Version: 2.1

Kudelski Security – Nagravision Sàrl

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

For Public Release

DOCUMENT PROPERTIES

Version:	2.1
File Name:	Kudelski_Security_Gas_Station_Secure_Code_Review_v2.0.pdf
Publication Date:	22 July 2024
Confidentiality Level:	For Public Release
Document Recipients:	Gautham Ravi
Document Status:	Final

TABLE OF CONTENTS

EXECUTIVE SUMMARY

Pagoda (“the Client”) engaged Kudelski Security (“Kudelski”, “we”) to perform the Secure Code Review of Gas Station.

The assessment was conducted remotely by the Kudelski Security Team.

The review took place between 21 May 2024 and 28 June 2024, and focused on the following objectives:

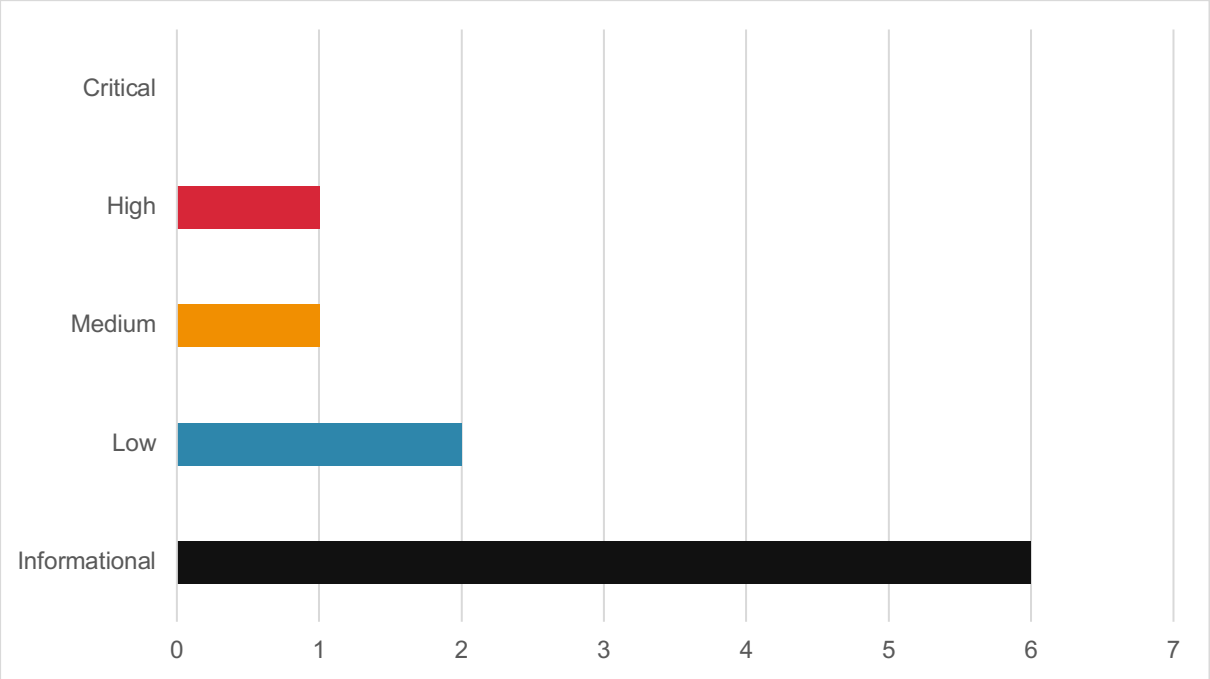
- Provide the customer with an assessment of their overall security posture and any risks that were discovered.
- To provide a professional opinion on the maturity, adequacy, and efficiency of the security measures that are in place.
- To identify potential issues and include improvement recommendations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

Key Findings

The following are the major themes and issues identified during the testing period:

- Hard-Coded Key Version
- Oracle Price Manipulation

These, along with other items within the findings section, should be prioritized for remediation to reduce to the risk they pose.

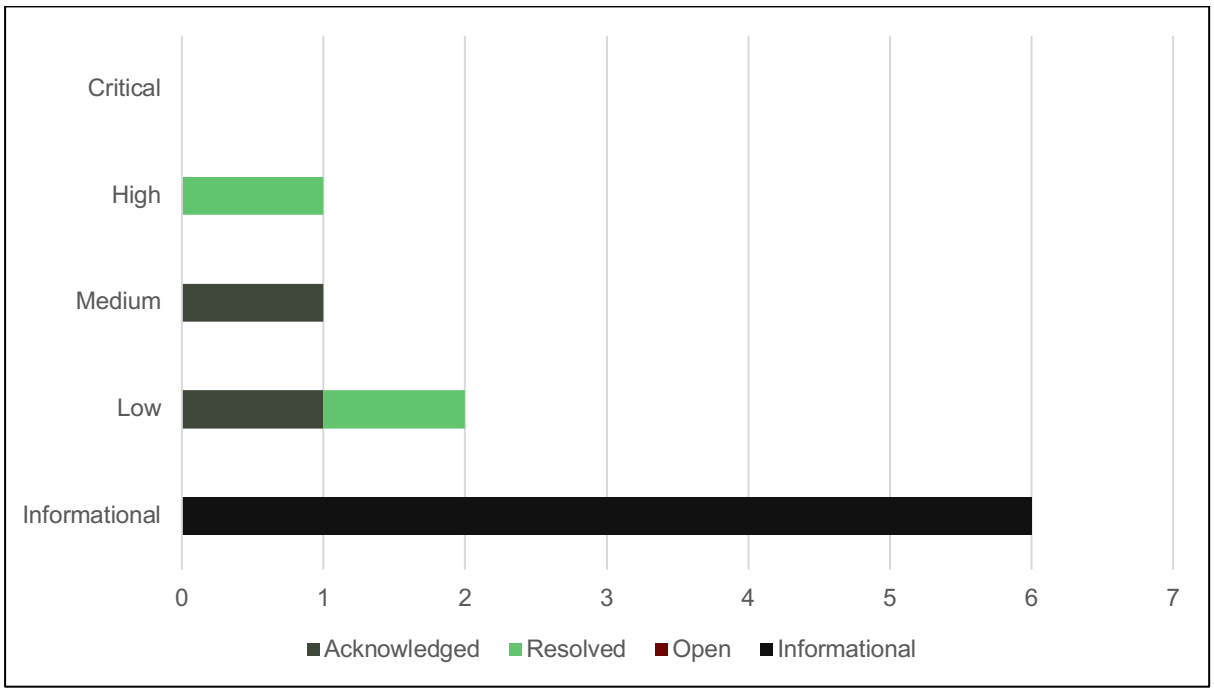


Findings ranked by severity.

Status of Findings after Re-review

The Client modified the source code based on the recommendations in this report. Subsequently, the Kudelski Security Team has evaluated these changes and updated the status of each finding:

- **Resolved** – the Client did modify the implementation, and we considered the fix to be correct.
- **Acknowledged** – the Client decided to accept the risk, or the mitigation is difficult or impossible to implement with the existing tools and resources.
- **Open** – nothing was implemented or communicated by the client between the two code reviews.



Overview of findings and their status.

1. PROJECT SUMMARY

This report summarizes the engagement and findings. The document 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 multichain gas station smart contract is a service that accepts payments in NEAR tokens in exchange for gas funding on foreign (i.e., non-NEAR) chains, in an effort to make NEAR an effortlessly cross-chain network.

1.2 Scope

The scope consisted in specific Rust files and folders located at:

- Commit hash : `c83bdb069bda7ebdbf44b87fd9a35205d55e6743`
- Source code repository: https://github.com/near/multichain-gas-station-contract/tree/wombat

The files and folders in scope are:

- `gas_station/src/`
- `lib/src/`
- `nft_key/src/`

The files and folders out of scope are:

- `gas_station/tests/`
- `mock/`
- `nft_key/tests/`

1.3 Remarks

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

- Excellent quality of code
- Good safety checks in public functions

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 5 of this document.

2. 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 findings.

#	SEVERITY	TITLE	STATUS
KS–PA–F–01	High	Hard-Coded Key Version	Resolved
KS–PA–F–02	Medium	Oracle Price Manipulation	Acknowledged
KS–PA–F–03	Low	`cargo audit` Reports Vulnerabilities	Acknowledged
KS–PA–F–04	Low	Unhandled Arithmetic Overflows	Resolved

Findings overview.

2.1 KS–PA–F–01 Hard-Coded Key Version

Severity	Impact	Likelihood	Status
High	High	Medium	Resolved

Description

The NFT smart contract requests signatures using a hard-coded key version.

Impact

By assigning a fixed value to the key version, the smart contract will always request a signature from the MPC service with the same key (i.e., the key corresponding to version zero), regardless of the actual version of the key in use for the NFT token. Consequently, even if the key used to mint the next NFT tokens is updated (due to, e.g., compromise), the smart contract will continue to request signatures with the first key, despite no longer being in use.

Evidence

```
    fn ckt_sign_hash(
        &mut self,
        token_id: TokenId,
        path: Option<String>,
        payload: Vec<u8>,
        approval_id: Option<u32>,
    ) -> PromiseOrValue<String> {

        ...

        PromiseOrValue::Promise(
            ext_signer::ext(self.signer_contract_id.clone())
                .with_unused_gas_weight(10)
                .sign(
                    payload.try_into().unwrap(),
                    &format!("{token_id},{path}"),
                    0,
                )
                .then(
                    Self::ext(env::current_account_id())
                        .with_static_gas(near_sdk::Gas::from_tgas(3))
                        .with_unused_gas_weight(1)
                        .sign_callback(),
                ),
        )
```

nft_key/src/lib.rs.

Affected Resources

- nft_key/src/lib.rs line 153

Recommendation

Replace the hard-coded value of 0 in `nft_key/src/lib.rs` at line 153 with `key_data.key_version`.

2.2 KS–PA–F–02 Oracle Price Manipulation

Severity	Impact	Likelihood	Status
Medium	Medium	Low	Acknowledged

Description

The smart contract automatically accepts price data from the Pyth oracle without validation.

Impact

Market manipulation has the potential to influence the oracle's reported price for a token. This could result in users being overcharged or undercharged for native tokens required to cover the required gas fees, possibly causing an increased acquisition or depletion of foreign tokens in exchange for the current deposit.

Evidence

```
    fn try_create_transaction_callback(
        &mut self,
        sender: &AccountId,
        token_id: String,
        deposit: &AssetBalance,
        transaction_request: ValidTransactionRequest,
        local_asset_price_result: Result<pyth::Price, PromiseError>,
        foreign_asset_price_result: Result<pyth::Price, PromiseError>,
    ) -> Result<(u128, TransactionSequenceCreation),
TryCreateTransactionCallbackError> {
        let local_asset_price = local_asset_price_result.map_err(|_|
OracleQueryFailureError)?;
        let foreign_asset_price =
            foreign_asset_price_result.map_err(|_| OracleQueryFailureError)?;

        ...

        let local_asset_fee = foreign_chain.price_for_gas_tokens(
            gas_tokens_to_sponsor_transaction,
            &foreign_asset_price,
            &local_asset_price,
            accepted_local_asset.decimals,
        )?;
```

gas_station/src/lib.rs.

Affected Resources

- `gas_station/src/lib.rs` lines 338–339, 368–374

Recommendation

Detect significant price fluctuations in the oracle data and reject the result if the price has not remained stable for a long enough period of time.

References

https://www.halborn.com/blog/post/what-is-oracle-manipulation-a-comprehensive-guide

2.3 KS–PA–F–03 `cargo audit` Reports Vulnerabilities

Severity	Impact	Likelihood	Status
Low	Medium	Low	Acknowledged

Description

The command cargo audit reports 4 vulnerabilities and 4 warnings.

Impact

The reported vulnerabilities expose the code to potential abuse, however there is currently no known method to directly exploit them.

Evidence

```
$ cargo audit
    Fetching advisory database from `https://github.com/RustSec/advisory-db.git`
      Loaded 630 security advisories (from /home/user/.cargo/advisory-db)
    Updating crates.io index
    Scanning Cargo.lock for vulnerabilities (508 crate dependencies)
Crate:     curve25519-dalek
Version:   4.1.1
Title:     Timing variability in `curve25519-dalek`'s
           `Scalar29::sub`/`Scalar52::sub`
Date:      2024-06-18
ID:        RUSTSEC-2024-0344
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0344
Solution:  Upgrade to >=4.1.3

Crate:     h2
Version:   0.3.24
Title:     Degradation of service in h2 servers with CONTINUATION Flood
Date:      2024-04-03
ID:        RUSTSEC-2024-0332
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0332
Solution:  Upgrade to ^0.3.26 OR >=0.4.4

Crate:     mio
Version:   0.8.10
Title:     Tokens for named pipes may be delivered after deregistration
Date:      2024-03-04
ID:        RUSTSEC-2024-0019
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0019
Solution:  Upgrade to >=0.8.11

Crate:     rustls
Version:   0.21.10
Title:     `rustls::ConnectionCommon::complete_io` could fall into an infinite
           loop based on network input
```

```
Date:      2024-04-19
ID:        RUSTSEC-2024-0336
URL:       https://rustsec.org/advisories/RUSTSEC-2024-0336
Severity:  7.5 (high)
Solution:  Upgrade to >=0.23.5 OR >=0.22.4, <0.23.0 OR >=0.21.11, <0.22.0

Crate:     wee_alloc
Version:   0.4.5
Warning:   unmaintained
Title:     wee_alloc is Unmaintained
Date:      2022-05-11
ID:        RUSTSEC-2022-0054
URL:       https://rustsec.org/advisories/RUSTSEC-2022-0054

Crate:     atty
Version:   0.2.14
Warning:   unsound
Title:     Potential unaligned read
Date:      2021-07-04
ID:        RUSTSEC-2021-0145
URL:       https://rustsec.org/advisories/RUSTSEC-2021-0145

Crate:     borsh
Version:   0.9.3
Warning:   unsound
Title:     Parsing borsh messages with ZST which are not-copy/clone is unsound
Date:      2023-04-12
ID:        RUSTSEC-2023-0033
URL:       https://rustsec.org/advisories/RUSTSEC-2023-0033

Crate:     iana-time-zone
Version:   0.1.59
Warning:   yanked

error: 4 vulnerabilities found!
warning: 4 allowed warnings found
```

Truncated log of cargo audit.

Affected Resources

- gas_station/
- nft_key/

Recommendations

1. Upgrade the following crates to their respective newest versions:
    - curve25519-dalek (>=4.1.3)
    - h2 (^0.3.26 OR >=0.4.4)
    - mio (>=0.8.11)
    - rustls (>=0.23.5 OR >=0.22.4 OR >=0.21.11)
    - borsh (>=1.0.0-alpha.1)
    - iana-time-zone (>=0.1.60)
2. Replace the following crates with alternatives that are currently maintained:
    - wee_alloc
    - atty

References

- https://rustsec.org/advisories/RUSTSEC-2024-0344
- https://rustsec.org/advisories/RUSTSEC-2024-0332
- https://rustsec.org/advisories/RUSTSEC-2024-0019
- https://rustsec.org/advisories/RUSTSEC-2024-0336
- https://rustsec.org/advisories/RUSTSEC-2022-0054
- https://rustsec.org/advisories/RUSTSEC-2021-0145
- https://rustsec.org/advisories/RUSTSEC-2023-0033

2.4 KS–PA–F–04 Unhandled Arithmetic Overflows

Severity	Impact	Likelihood	Status
Low	Low	Low	Resolved

Description

The code performs arithmetic operations without gracefully handling the potential overflows.

Impact

In Rust, with the `overflow-checks` profile enabled, an unchecked arithmetic overflow will cause the smart contract to panic at runtime.

⚠ However, given the large range of the variables handled, the occurrence of an overflow is unlikely. Still, sometimes, the operands are controlled by the user and might be injected on purpose to make the program crash.

Evidence

```
    pub fn with_request_nonce<R>(
        &mut self,
        deduct_amount: U256,
        f: impl FnOnce(&Self, &PaymasterConfiguration) -> R,
    ) -> Result<R, RequestNonceError> {

        ...

        paymaster_config.nonce += 1;
        paymaster_config.minimum_available_balance = new_minimum_balance.0;
        self.paymasters.insert(&paymaster_key, &paymaster_config);
        self.next_paymaster = paymaster_key_after;

        Ok(r)
    }
```

gas_station/src/chain_configuration.rs.

```
    pub fn calculate_gas_tokens_to_sponsor_transaction(
        &self,
        transaction: &ValidTransactionRequest,
    ) -> U256 {
        (transaction.gas() + U256(self.transfer_gas)) *
transaction.max_fee_per_gas()
    }
```

gas_station/src/chain_configuration.rs.

```
    pub fn price_for_gas_tokens(
        &self,
        quantity_to_convert: U256,
        this_asset_price_in_usd: &pyth::Price,
        into_asset_price_in_usd: &pyth::Price,
        into_asset_decimals: u8,
    ) -> Result<u128, PriceDataError> {

        ...

        // Apply conversion rate to quantity in two steps: multiply, then divide.
        let a = quantity_to_convert * U256::from(conversion_rate.0) *
U256::from(self.fee_rate.0);
        let (b, rem) = a.div_mod(U256::from(conversion_rate.1) *
U256::from(self.fee_rate.1));

        // Round up. Again, pessimistic pricing.
        Ok(if rem.is_zero() { b } else { b + 1 }.as_u128())
    }
```

gas_station/src/chain_configuration.rs.

```
    pub fn increase_paymaster_balance(&mut self, chain_id: U64, token_id: String,
balance: U128) {
        #[cfg(not(feature = "debug"))]
        <Self as Rbac>::require_role(&Role::MarketMaker);

        self.with_mut_chain(chain_id.0, |chain_config| {
            let mut paymaster =
chain_config.paymasters.get(&token_id).unwrap_or_reject();
            paymaster.minimum_available_balance =
                (U256(paymaster.minimum_available_balance) +
U256::from(balance.0)).0;
            chain_config.paymasters.insert(&token_id, &paymaster);
        });
    }
```

gas_station/src/impl_management.rs.

```
    pub fn sign_next(&mut self, id: U64) -> Promise {

        ...

        // ensure not expired
        require!(
            env::block_height()
                <= self.expire_sequence_after_blocks +
transaction.created_at_block_height.0,
            "Transaction is expired",
        );
```

gas_station/src/lib.rs.

```
    #[private]
    pub fn sign_next_callback(
        &mut self,
        id: U64,
        index: u32,
        #[callback_result] result: Result<String, PromiseError>,
    ) -> String {

        ...

        // Remove escrow from record.
        // This is important to ensuring that refund logic works correctly.
        if let Some(escrow) = pending_transaction_sequence.escrow.take() {
            let mut collected_fees =
self.collected_fees.get(&escrow.asset_id).unwrap_or(U128(0));
            collected_fees.0 += escrow.amount.0;
            self.collected_fees
                .insert(&escrow.asset_id, &collected_fees);
        }
```

gas_station/src/lib.rs.

```
    fn generate_id(&mut self) -> u32 {
        let id = self.next_id;
        self.next_id += 1;
        id
    }
```

nft_key/src/lib.rs.

Affected Resources

- `gas_station/src/chain_configuration.rs` lines 110, 130, 196–197, 200
- `gas_station/src/impl_management.rs` lines 267
- `gas_station/src/lib.rs` lines 485, 572
- `nft_key/src/lib.rs` lines 67

Recommendation

Replace the arithmetic operators with secure alternatives, such as `checked_add()`, and handle the error properly.

3. 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–PA–O–01	Informational	TODO Comments	Informational
KS–PA–O–02	Informational	Code Discrepancy	Informational
KS–PA–O–03	Informational	Magic Numbers	Informational
KS–PA–O–04	Informational	Possible Variable Name Confusion	Informational
KS–PA–O–05	Informational	Mark Initialization Functions Private	Informational
KS–PA–O–06	Good Practice	Effective Checks in Public Functions	Informational

Observations overview.

3.1 KS–PA–O–01 TODO Comments

Description

The code base presents TODO comments in certain sections, suggesting that there are areas that require further development.

Affected Resources

- `gas_station/src/impl_management.rs` line 359
- `gas_station/src/impl_nep141_receiver.rs` line 17
- `gas_station/src/lib.rs` lines 185, 555

Recommendation

Either implement the missing features or open a GitHub issue with a detailed description of the task to address.

3.2 KS–PA–O–02 Code Discrepancy

Description

There are instances where a same task was implemented in different ways at different locations of the code base. This inconsistency can lead to the introduction of bugs during code maintenance.

Affected Resources

- `gas_station/src/lib.rs` lines 305–324 vs. lines 406–423.
- `gas_station/src/lib.rs` lines 264–271 vs. lines 348–355.

Recommendation

Either modularize the code by converting similar code segments into functions or, at least, make sure to have consistent code for similar tasks.

3.3 KS–PA–O–03 Magic Numbers

Description

Hard-coding numerical values in the code is a bad coding practice, as it hides the meaning behind the number, leading to a harder code maintenance.

Affected Resources

- `gas_station/src/lib.rs` line 515
- `nft_key/src/lib.rs` line 149, 157, 201

Recommendation

Define constants or enums to represent each quantity.

3.4 KS–PA–O–04 Possible Variable Name Confusion

Description

In some cases, variables have names that are very similar, which can cause confusion and make code maintenance harder.

Affected Resources

- `gas_station/src/lib.rs` line 146 vs. line 66
  `PendingTransactionSequence` vs. `PendingTransactionSequences`
- `gas_station/src/lib.rs` lines 264, 269, 348 vs. line 181
  `user_chain_keys` vs. `user_chain_key`
- `gas_station/src/lib.rs` line 343 vs. 177
  `accepted_local_assets` vs. `accepted_local_asset`
- `gas_station/src/lib.rs` line 359 vs. 179
  `foreign_chains` vs. `foreign_chain`
- `nft_key/src/lib.rs` lines 132, 248, 267, 366 vs. line 34
  `key_data` vs. `self.key_data`

Recommendation

Make sure that the variable names are sufficiently distinct to avoid confusion between them.

3.5 KS–PA–O–05 Mark Initialization Function Private

Description

According to the NEAR documentation[1], it is good practice to mark initialization functions as private.

Affected Resources

- `gas_station/src/lib.rs` line 192
- `nft_key/src/lib.rs` line 43

Recommendation

Decorate the initialization functions with the `#[private]` macro.

[1] https://docs.near.org/build/smart-contracts/anatomy/storage

3.6 KS–PA–O–06 Effective Checks in Public Function

Description

The checks-effects-interactions pattern aims to reduce the attack surface for a malicious user attempting to hijack the control flow of a contract. More specifically, a public function should first perform checks to ensure the consistency of the state as well as the validity of the inputs.

We have verified the correct implementation of the checks-effects-interactions pattern, and in particular, confirm the presence of sound checks in every public function.

Affected Resources

- `gas_station/src/lib.rs` lines 338–382, lines 534–560
- `nft_key/src/lib.rs` lines 132–145

Recommendation

Keep up the good work.

4. 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.

4.1 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.

4.2 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.

4.3 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.

4.4 Smart Contract

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.

4.5 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.

4.6 Verify

After the preliminary findings have been delivered, we verify the fixes applied by the Client. 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.

5. 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.

6. CONCLUSION

The objective of this Secure Code Review was to evaluate whether there were any vulnerabilities that would put Pagoda or its customers at risk.

The Kudelski Security Team identified 4 security issues: 1 high risk, 1 medium risk, and 2 lower risks. On average, the effort needed to mitigate these risks is estimated as **low**.

During the remediation phase, Pagoda acknowledged and resolved all the issues that were reported, which was further verified by the Kudelski Security team during re-review.

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

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

KUDELSKI SECURITY CONTACTS

NAME	POSITION	CONTACT INFORMATION
Amy Fleischer	Project Manager/ Operations Coordinator	amy.fleischer@kudelskisecurity.com
Jean-Sebastien Nahon	Application and Blockchain Security Practice Manager	jean-sebastien.nahon@kudelskisecurity.com

DOCUMENT HISTORY

VERSION	DATE	STATUS/ COMMENTS
V0.1	1 July 2024	First draft.
V0.2	5 July 2024	Second draft.
V1.0	8 July 2024	First version delivered to client.
V1.1	11 July 2024	Added KS–PA–F–01.
V2.0	19 July 2024	Re-review.
V2.1	22 July 2024	Minor changes.