Digital Asset Assessment

Findings and Recommendations Report Presented to:

Only1 Limited

November 1, 2021

Version 1.1.0 – Final for Public Release

Presented by:

Kudelski Security, Inc.
5090 North 40th Street, Suite 450
Phoenix, Arizona 85018

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Overview

Only1 Limited engaged Kudelski Security to perform a Digital Asset Assessment.

The assessment was conducted remotely by the Kudelski Security Team. Testing took place on October 04 - October 15, 2021, with a re-review on October 30, 2021 and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks that were discovered within the environment during the engagement.
- 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.

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

At the time of this final report, all findings have been resolved to our satisfaction in PR: https://github.com/only1nft/only1-programs/pull/24

Key Findings

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

- KS-ONLY1-01 – Claim influencer reward not authorized correctly
- KS-ONLY1-02 – Pool escrow is not authorized properly
- KS-ONLY1-03 – Stake state, pool global and staking state is not authorized correctly
- KS-ONLY1-04 – Staking pool creation not authorized correctly
- KS-ONLY1-05 – Staking pool global state can be created multiple times
- KS-ONLY1-06 – Genesis NFT mint creation not authorized correctly
- KS-ONLY1-07 – Genesis NFT global state can be created multiple times
- KS-ONLY1-08 – Marketplace global state can be created multiple times
- KS-ONLY1-09 – Last update field can be set to older time for pool state
- KS-ONLY1-10 – NFT escrow not authorized correctly in marketplace
- KS-ONLY1-11 – NFT order not authorized correctly in marketplace

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

- The team was very supportive and open to discuss the design choices made

Based on the account relationship graphs or reference graphs and the formal verification we can conclude that the reviewed code implements the documented functionality.

Scope and Rules of Engagement

Kudelski performed a Digital Asset Assessment. The following table documents the targets in scope for the engagement. No additional systems or resources were in scope for this assessment.

The source code was supplied through a private repository at https://github.com/only1nft/only1-programs with the commit hash 087052e3aa2b0229e8548211cbfc0708f6f61c10.

Files included in the code review
only1 ├── only1-genesis-nft/ │   ├── src/ │   │   └── lib.rs │   └── Cargo.toml ├── only1-marketplace/ │   ├── src/ │   │   └── lib.rs │   └── Cargo.toml ├── only1-sdk/ │   ├── src/ │   │   └── lib.rs │   └── Cargo.toml ├── only1-staking-pool/ │   ├── src/ │   │   └── lib.rs │   └── Cargo.toml └── Cargo.toml

Table 1: Scope

TECHNICAL ANALYSIS & FINDINGS

During the Digital Asset Assessment, we discovered:

- 3 findings with HIGH severity rating.
- 5 findings with MEDIUM severity rating.
- 3 findings with LOW severity rating.

The following chart displays the findings by severity.

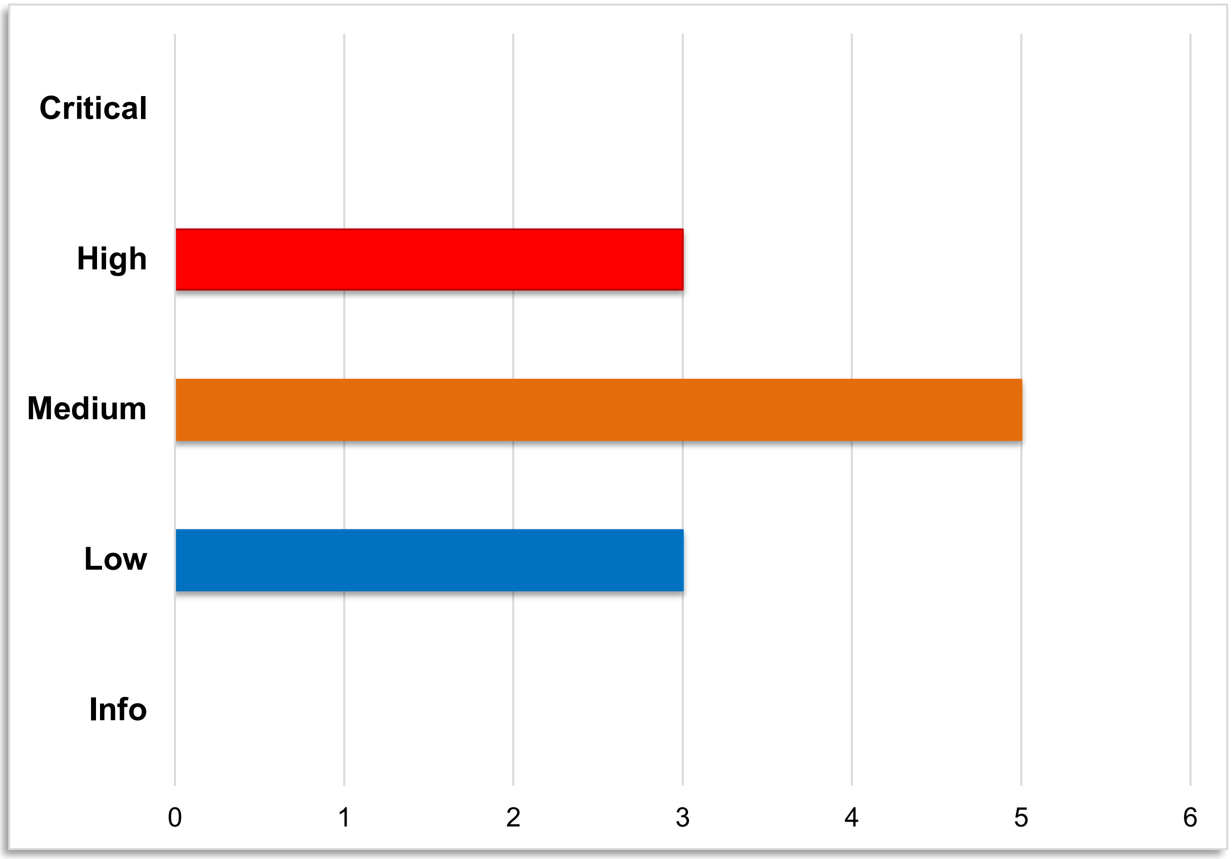

Figure 1: Findings by Severity

Findings

The *Findings* section 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	Description
KS-ONLY1-01	**High**	Claim influencer reward not authorized correctly
KS-ONLY1-02	**High**	Pool escrow is not authorized properly
KS-ONLY1-03	**High**	Stake state, pool global and staking state is not authorized correctly
KS-ONLY1-04	**Medium**	Staking pool creation not authorized correctly
KS-ONLY1-05	**Medium**	Staking pool global state can be created multiple times
KS-ONLY1-06	**Medium**	Genesis NFT mint creation not authorized correctly
KS-ONLY1-07	**Medium**	Genesis NFT global state can be created multiple times
KS-ONLY1-08	**Medium**	Marketplace global state can be created multiple times
KS-ONLY1-09	**Low**	Last update field can be set to older time for pool state
KS-ONLY1-10	**Low**	NFT escrow not authorized correctly in marketplace
KS-ONLY1-11	**Low**	NFT order not authorized correctly in marketplace

Table 2: Findings Overview

Technical analysis

Based on the source code the following account relationship graphs or reference graphs was made to verify the validity of the code as well as confirmation that the intended functionality was implemented correctly and to the extent that the state of the repository allowed.

Further investigations were made which concluded that they did not pose a risk to the application. They were:

- No potential panics were detected
- No potential errors regarding wraps/unwraps, expect and wildcards
- No internal unintentional unsafe references

Conclusion

Based on formal verification we can conclude that the code implements the documented functionality to the extent of the code reviewed.

Technical Findings

General Observations

The Only1 smart contracts are well written programs. Especially the test coverage is quite impressive.

Whole system is divided into four parts:

- genesis NFT program
- staking pool program
- marketplace program
- only1 sdk

Each part is located in a single file. Each file is quite big, but at least common operations, especially authorization and account handling are implemented in the common sdk.

The code seems to be panic free, and all math operations are checked.

All the account addresses are Program Derived Addresses (PDA). Unfortunately, bump seeds are passed through instruction data and collisions are possible. It is especially painful for global states.

It is assumed that only one instance of global state may exist, which makes code very nice. However, if it is possible to create more instances, there are many ways to attack the system.

Before using the programs, it is important that bump seeds for global states are generated on the chain. Such solution has one big advantage that there is no need to do any changes in the client applications.

Claim influencer reward not authorized correctly

Finding ID: KS-ONLY1-01
Severity: High
Status: Resolved

Description

Attacker can claim influencer rewards from a fake staking pool. The instruction is protected by a NFT signature, but such a signature can be forged.

Proof of issue

File name: only1-staking-pool/src/lib.rs
Line number: 303

```
let nft_signature_seeds = &amp;amp;[
    b"only1_genesis_nft_signature",
    instruction.genesis_nft_mint_id.as_ref(),
    &amp;amp;[instruction.nft_signature_seed],
];
let nft_signature =
SerializedProgramAccount::<only1_genesis_nft::GenesisNftSignature>::from(
    next_account_info(it)?,
    state_data.genesis_nft_program_id,
    nft_signature_seeds,
)?;
let nft_signature_data = nft_signature.read();
if *user.key != nft_signature_data.influencer_id {
    return Err(ProgramError::MissingRequiredSignature);
}
```

The NFT signature address is derived from `state_data.genesis_nft_program_id`. Normally this is the genesis NFT program, but a malicious user can create a fake global state with any program as `state_data.genesis_nft_program_id` and then `nft_signature_data.influencer_id` can be used set to any value. It will let the attacker to invoke any instruction as influencer.

Recommendation

Only one instance of global staking state, controlled by only1 admin should be available to create to make NFT signature reliable.

Pool escrow is not authorized properly

Finding ID: KS-ONLY1-02
Severity: **High**
Status: **Resolved**

Description

Pool escrow is an account, which holds tokens transferred during stake instruction. In the code it is assumed, that only one staking pool can be created for each NFT mint, so it is not authorized at all.

Proof of issue

File name: only1-staking-pool/src/lib.rs
Line number: 561

```
token::Transfer {
    authority: user.clone(),
    authorities: &amp;amp;[],

    from: associated_token_account.clone(),
    to: pool_escrow.clone(),
    amount: instruction.num_like_tokens,

    token_program: token_program.clone(),
    signer_seeds: None,
}
.invoke()?;
```

For `stake` instruction `pool_escrow` from a fake staking pool can be used by an attacker. It will let the attacker change his staking balance without adding any tokens to the proper staking pool. For the `unstake` instruction, the proper account must be used, because transfer from escrow must be authorized by the proper global state account.

Recommendation

The pool escrow address should have the staking pool state address as one of its seeds.

Stake state, pool global and staking state is not authorized correctly

Finding ID: KS-ONLY1-03
Severity: **High**
Status: **Resolved**

Description

All the transactions using staking pool state assume that there is only one staking pool for each genesis NFT mint. As proved in finding KS-ONLY1-04 this is not true.

Proof of issue

File name: only1-staking-pool/src/lib.rs
Line number: 581

```
state.write(Entity::State(state_data))?;
pool_state.write(Entity::PoolState(pool_state_data))?;
stake.write(Entity::Stake(stake_data))?;
```

In most instructions, three accounts are mutated at the same instruction. As relationship between those accounts are not checked, there are many ways to do something unexpected for program, for example:

- use real `stake` with fake `state` and `pool_state`
- use real `pool_state` and `stake` with fake `state`

Recommendation

All mentioned accounts should be connected somehow with each other and such connections should be verified. It is described in finding KS-ONLY1-04, how safe setup could be achieved.

Staking pool creation not authorized correctly

Finding ID: KS-ONLY1-04
Severity: **Medium**
Status: **Resolved**

Description

Only1 smart contracts assume that only one user, the only1 admin, can create genesis NFT mint. The mint instruction requires the transaction to be signed by the admin. However, it is possible to create multiple instances of the global state and then create a valid genesis NFT mint without any reference to the global state.

Proof of issue

File name: only1-staking-pool/src/lib.rs
Line number: 303

```
let pool_state_seeds = &amp;amp;[
    b"only1_staking_pool".as_ref(),
    instruction.genesis_nft_mint_id.as_ref(),
    &amp;amp;[instruction.pool_state_seed],
];
let pool_state =
SerializedProgramAccount::<Entity>::from_mut_init(next_account_info(it)?,
program_id, pool_state_seeds)?;
pool_state.write(Entity::PoolState(PoolState {
    is_initialized: true,
    genesis_nft_mint_id: instruction.genesis_nft_mint_id.clone(),

    started_at: clock.unix_timestamp,
    num_followers: instruction.num_followers,

    total_num_like_token_rewards: "0".to_string(),
    num_unclaimed_owner_like_token_rewards: 0,
    num_unclaimed_influencer_like_token_rewards: 0,

    pool_last_updated_at: clock.unix_timestamp,
    last_computed_num_like_token_rewards_per_second: 0,
    last_computed_num_owner_like_token_rewards_per_second: 0,
    last_computed_num_influencer_like_token_rewards_per_second: 0,
}))?;
```

The pool state address does not depend on the global state address and the bump seed is passed through the instruction data. This makes it is possible to create approximately 128 valid staking pools by anyone. Neither does the pool state payload contain any reference to the global state.

Recommendation

The staking pool state address should have the global state address as one of its seeds.

Staking pool global state can be created multiple times

Finding ID: KS-ONLY1-05
Severity: **Medium**
Status: **Resolved**

Description

Only1 smart contracts assume that only one instance of the staking pool global state is created. Such an assumption allows to simplify authorization. However, it is possible to create multiple instances of the global state giving the attacker a dangerous weapon.

Proof of issue

File name: only1-staking-pool/src/lib.rs
Line number: 257

```
let state_seeds = &amp;amp;[b"only1_staking_pool".as_ref(),
&amp;amp;[instruction.state_seed]];
let state =
SerializedProgramAccount::<Entity>::from_mut_init(next_account_info(it)?,
program_id, state_seeds)?;
```

The program expects that a seed is provided through the instruction data to pass to the `find_program_address` function for calculating a PDA. However, any value can be passed, and approximately 128 (50%) of the possible values will generate valid addresses.

Recommendation

To have only one instance of specific type, `find_program_address` should be used on-chain to generate a bump seed. The other way is to hardcode the global state seed, the account address, or the admin key.

Genesis NFT mint creation not authorized correctly

Finding ID: KS-ONLY1-06
Severity: **Medium**
Status: **Resolved**

Description

Only1 smart contracts assume, that only one user, only1 admin, can create genesis NFT mint. Mint instruction requires transaction to be signed by admin. However, it is possible to create multiple instances of global state and then create valid genesis NFT mint without any reference to global state.

Proof of issue

File name: only1-genesis-nft/src/lib.rs
Line number: 150

```
let nft_signature_seeds = &amp;amp;[
    b"only1_genesis_nft_signature",
    nft_mint.key.as_ref(),
    &amp;amp;[instruction.nft_signature_seed],
];
let nft_signature =

SerializedProgramAccount::<GenesisNftSignature>::from_mut_init(next_account_i
nfo(it)?, program_id, nft_signature_seeds)?;

nft_signature.write(GenesisNftSignature {
    is_initialized: true,
    influencer_id: *influencer.key,
    mint_id: *nft_mint.key,
})?;
```

NFT signature address does not depend on global state address and its payload does not have any reference to global state. It indicates, that everyone is allowed to create valid NFT mint.

Recommendation

The NFT signature address should have the global state address as one of its seeds.

Genesis NFT global state can be created multiple times

Finding ID: KS-ONLY1-07
Severity: **Medium**
Status: **Resolved**

Description

Only1 smart contracts assume that only one instance of the genesis NFT state is created. Such an assumption allows to simplify authorization. However, it is possible to create multiple instances of the global state giving the attacker a dangerous weapon.

Proof of issue

File name: only1-genesis-nft/src/lib.rs
Line number: 119

```
let genesis_nft_program_state_seeds =
&amp;amp;[b"only1_genesis_nft".as_ref(),
&amp;amp;[instruction.genesis_nft_program_state_seed]];
let genesis_nft_program_state =
SerializedProgramAccount::<GenesisNft>::from_mut_init(
    next_account_info(it)?,
    program_id,
    genesis_nft_program_state_seeds,
)?;
```

The program expects that a seed is provided through the instruction data to pass to the `find_program_address` function for calculating a PDA. However, any value can be passed, and approximately 128 (50%) of the possible values will generate valid addresses.

Recommendation

To have only one instance of specific type, `find_program_address` should be used on-chain to generate a bump seed. The other way is to hardcode the global state seed, the account address, or the admin key.

Marketplace global state can be created multiple times

Finding ID: KS-ONLY1-08
Severity: **Medium**
Status: **Resolved**

Description

Only1 smart contracts assume that only one instance of marketplace state is created. Such assumption allows to simplify authorization. However, it is possible to create multiple instances of the global state giving the attacker a dangerous weapon.

Proof of issue

File name: only1-marketplace/src/lib.rs
Line number: 149

```
let marketplace_state_seeds = &amp;amp;[
    b"only1_marketplace",
    program_id.as_ref(),
    &amp;amp;[instruction.marketplace_state_seed],
];
let marketplace_state =

SerializedProgramAccount::<Marketplace>::from_mut_init(next_account_info(it)?
, program_id, marketplace_state_seeds)?;
```

The program expects that a seed is provided through the instruction data to pass to the `find_program_address` function for calculating a PDA. However, any value can be passed, and approximately 128 (50%) of the possible values will generate valid addresses.

Recommendation

To have only one instance of specific type, `find_program_address` should be used on-chain to generate a bump seed. The other way is to hardcode the global state seed, the account address, or the admin key.

Last update field can be set to older time for pool state

Finding ID: KS-ONLY1-09
Severity: **Low**
Status: **Resolved**

Description

After pool reward update it is required to set the last update time in order to properly perform calculations in the next iteration. However, it is possible to set a timestamp from the past.

Proof of issue

File name: only1-staking-pool/src/lib.rs
Line number: 1334

```
let seconds_elapsed_since_pool_last_updated = current_time
    .checked_sub(pool_state_data.pool_last_updated_at)
    .and_then(|amount| amount.to_u64())
    .unwrap_or(0);
pool_state_data.pool_last_updated_at = current_time;
```

If the `last_updated_at` field is greater than the current time, the function will not fail. It would allow setting an earlier time as last updated and then collect the same reward multiple times.

However, the timestamp is taken from clock sysvar, so it should at least be equal to the previous one. Anyway, in the documentation there are no guarantees that the clock is steady.

Recommendation

Instead of setting `current_time` as `last_updated_at`, `last_update_at` could be increased by `seconds_elapsed_since_pool_last_updated`.

References

- Solana Documentation: Sysvar Cluster Data

NFT escrow not authorized correctly in marketplace

Finding ID: KS-ONLY1-10
Severity: **Low**
Status: **Resolved**

Description

The pool escrow is an account, which holds NFT assets transferred during list instruction. In the code it is assumed that only one staking pool can be created for each NFT mint. So it is not authorized at all.

Proof of issue

File name: only1-marketplace-pool/src/lib.rs
Line number: 561

```
let nft_escrow_seeds = &amp;amp;[
    b"only1_marketplace_escrow",
    program_id.as_ref(),
    nft_mint.key.as_ref(),
    &amp;amp;[instruction.nft_escrow_seed],
];

let nft_escrow = PackedProgramAccount::<token::Account>::from_mut_init(
    next_account_info(it)?,
    AccountOwner::Key(program_id),
    nft_escrow_seeds,
)?;
// seller transfers genesis nft from their associated account to the escrow
account
token::Transfer {
    authority: seller.clone(),
    authorities: &amp;amp;[],

    from: seller_associated_nft_account.clone(),
    to: nft_escrow.clone(),
    amount: 1,

    token_program: token_program.clone(),
    signer_seeds: None,
}
.invoke()?;
```

For the `list` instruction `nft_pool_escrow` a fake marketplace can be used. It will let attacker list his item, but nobody can buy them. There is only one copy of the NFT item and the `list` instruction modifies only the order escrow account, so nothing can be broken using this bug.

Recommendation

The NFT pool escrow address should have the marketplace state address as one of its seeds.

NFT order not authorized correctly in marketplace

Finding ID: KS-ONLY1-11
Severity: **Low**
Status: **Resolved**

Description

The NFT order is an account that keeps information about order. It is cleared when items are sold or unlisted. It seems that there could exist only one order for each mint, but because of seed collision, it is possible to create many.

Proof of issue

File name: only1-marketplace-pool/src/lib.rs
Line number: 241

```
let nft_order_seeds = &amp;amp;[
    b"only1_marketplace_order",
    program_id.as_ref(),
    nft_mint.key.as_ref(),
    &amp;amp;[instruction.nft_order_seed],
];
let nft_order = SerializedProgramAccount::<Order>::from_mut_or_init(
    next_account_info(it)?,
    AccountOwner::Key(program_id),
    nft_order_seeds,
)?;
```

As seen in the code snippet, there are approximately 128 possible accounts for compatible orders. However, there is only one item for each NFT mint and the order is cleared, so there is no way to exploit program using fake order.

Recommendation

The NFT order should have the global marketplace state as one of its seeds.

METHODOLOGY

Kudelski Security uses the following high-level methodology when approaching engagements. They are broken up into the following phases.

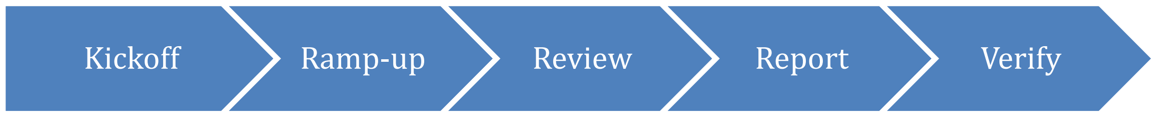

Figure 2: Methodology Flow

Kickoff

The project is kicked all of the sales process has concluded. We typically set up a kickoff meeting where project stakeholders are gathered to discuss the project as well as the responsibilities of participants. During this meeting we verify the scope of the engagement and discuss the project activities. It's an opportunity for both sides to ask questions and get to know each other. By the end of the kickoff there is an understanding of the following:

- Designated points of contact
- Communication methods and frequency
- Shared documentation
- Code and/or any other artifacts necessary for project success
- Follow-up meeting schedule, such as a technical walkthrough
- Understanding of timeline and duration

Ramp-up

Ramp-up consists of the activities necessary to gain proficiency on the particular project. This can include the steps needed for familiarity with the codebase or technological innovation utilized. This may include, but is not limited to:

- Reviewing previous work in the area including academic papers
- Reviewing programming language constructs for specific languages
- Researching common flaws and recent technological advancements

Review

The review phase is where most of the work on the engagement is completed. This is the phase where we analyze the project for flaws and issues that impact the security posture. Depending on the project this may include an analysis of the architecture, a review of the code, and a specification matching to match the architecture to the implemented code.

In this code audit, we performed the following tasks:

1. Security analysis and architecture review of the original protocol
2. Review of the code written for the project
3. Compliance of the code with the provided technical documentation

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 were used to assist the reviewer during the testing. We discuss our methodology in more detail in the following sections.

Code Safety

We analyzed the provided code, checking for issues related to the following categories:

- General code safety and susceptibility to known issues
- Poor coding practices and unsafe behavior
- Leakage of secrets or other sensitive data through memory mismanagement
- Susceptibility to misuse and system errors
- Error management and logging

This list is general list and not comprehensive, meant only to give an understanding of the issues we are looking for.

Technical Specification Matching

We analyzed the provided documentation and checked that the code matches the specification. We checked for things such as:

- Proper implementation of the documented protocol phases
- Proper error handling
- Adherence to the protocol logical description

Reporting

Kudelski Security delivers a preliminary report in PDF format that contains an executive summary, technical details, and observations about the project.

The executive summary contains an overview of the engagement including the number of findings as well as a statement about our general risk assessment of the project as a whole. We may conclude that the overall risk is low, but depending on what was assessed we may conclude that more scrutiny of the project is needed.

We not only report security issues identified but also informational findings for improvement categorized into several buckets:

- Critical
- High
- Medium
- Low
- Informational

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

As we perform the audit, we may identify issues that aren't security related, but are general best practices and steps, that can be taken to lower the attack surface of the project. We will call those out as we encounter them and as time permits.

As an optional step, we can agree on the creation of a public report that can be shared and distributed with a larger audience.

Verify

After the preliminary findings have been delivered, this could be in the form of the approved communication channel or delivery of the draft report, we will verify any fixes withing a window of time specified in the project. After the fixes have been verified, we will change the status of the finding in the report from open to remediated.

The output of this phase will be a final report with any mitigated findings noted.

Additional Note

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

While assessment the severity of the findings, we considered the impact, ease of exploitability, and the probability of attack. These is a solid baseline for severity determination.

The Classification of identified problems and vulnerabilities

There are four severity levels of an identified security vulnerability.

Critical – vulnerability that will lead to loss of protected assets

- This is a vulnerability that would lead to immediate loss of protected assets
- The complexity to exploit is low
- The probablillty of exploit is high

High - A vulnerability that can lead to loss of protected assets

- All discrepancies found where there is a security claim made in the documentation that can not be found in the code
- All mismatches from the stated and actual functionality
- Unprotected key material
- Weak encryption of keys
- Badly generated key materials
- Tx signatures not verified
- Spending of funds through logic errors
- Calculation errors overflows and underflows

Medium - a vulnerability that hampers the uptime of the system or can lead to other problems

- Insecure calls to third party libraries
- Use of untested or nonstandard or non-peer-revied crypto functions
- Program crashes leaves core dumps or write sensitive data to log files

Low - Problems that have a security impact but does not directly impact the protected assets

- Overly complex functions
- Unchecked return values from 3rd party libraries that could alter the execution flow

Informational

- General recommendations

Tools

The following tools were used during this portion of the test. A link for more information about the tool is provided as well.

Tools used during the code review and assessment

- Rust – cargo tools
- IDE modules for Rust and analysis of source code
- Cargo audit which uses https://rustsec.org/advisories/ to find vulnerabilities cargo.

RustSec.org

About RustSec

The RustSec Advisory Database is a repository of security advisories filed against Rust crates published and maintained by the Rust Secure Code Working Group.

The RustSec Tool-set used in projects and CI/CD pipelines

‘cargo-audit’ - audit Cargo.lock files for crates with security vulnerabilities.

‘cargo-deny’ - audit `Cargo.lock` files for crates with security vulnerabilities, limit the usage of particular dependencies, their licenses, sources to download from, detect multiple versions of same packages in the dependency tree and more.

KUDELSKI SECURITY CONTACTS

NAME	POSITION	CONTACT INFORMATION
Scott Carlson	Head of Blockchain Center of Excellence	Scottj.carlson@kudelskisecurity.com