Nodle

Substrate Pallets Security Audit

Prepared by: **Halborn**

Date of Engagement: **October 29th, 2021 - Feb 14th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	01/25/2022	Timur Guvenkaya
0.2	Document Edits	02/09/2022	Timur Guvenkaya
1.0	Remediation Plan	04/01/2022	Timur Guvenkaya
1.1	Remediation Plan Review	04/01/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Timur Guvenkaya	Halborn	Timur.Guvenkaya@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

Nodle engaged Halborn to conduct a security assessment on their main Substrate pallets on October 29th, 2021 and ending February 14th, 2022. Nodle is a crowdsourced decentralized IoT network.

1.2 AUDIT SUMMARY

The team at Halborn was provided 10 weeks for the engagement and assigned one full-time security engineer to audit the security of the assets in scope. The engineer is a blockchain and smart contract security expert with advanced penetration testing, smart-contract hacking, and in-depth knowledge of multiple blockchain protocols.

The purpose of this audit is to achieve the following:

- Identify potential security issues within the main Nodle pallets.

In summary, Halborn identified few security risks that should be addressed by the Nodle team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy regarding the scope of the Bridge Substrate pallet. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose.

- Substrate Pallets manual code review and walkthrough
- Mapping out possible attack vectors
- On chain testing of core functions.
- Fuzzing of core functions through cargo-fuzz
- - Fuzzing of core functions through honggfuzz
- Finding security vulnerabilities through cargo_audit
- Finding usage of unsafe Rust within the project through cargo-geiger
- Testnet deployment (polkadot.js)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.

2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

The review was scoped to the pallets directory in the audit-halborn branch in NodleCode/chain repository.

- Pallets
  - Allocations
  - Amendments
  - Emergency-Shutdown
  - Grants
  - Reserve
  - Root Of Trust
  - TCR
  - Staking

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	3	3	5

LIKELIHOOD

IMPACT

		(HAL-01) (HAL-03)		
	(HAL-06)	(HAL-04) (HAL-05)	(HAL-02)	
(HAL-07) (HAL-08) (HAL-09) (HAL-11)	(HAL-10)			

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
HAL-01 TCR VOTING DESIGN SHOULD BE IMPROVED	Medium	NOT APPLICABLE
HAL-02 DENOMINATION LOGIC SHOULD BE IMPROVED	Medium	SOLVED - 02/14/2022
HAL-03 EMERGENCY SHUTDOWN NOT USED IN MANY CRITICAL FUNCTIONS	Medium	FUTURE RELEASE
HAL-04 MISSING SANITY CHECKS	Medium	SOLVED - 02/14/2022
HAL-05 VESTING TO YOURSELF IS ALLOWED	Low	SOLVED - 02/14/2022
HAL-06 MISSING ZERO VALUE CHECK	Low	SOLVED - 02/14/2022
HAL-07 VESTING SCHEDULES LESS THAN A CURRENT BLOCK CAN BE CREATED	Informational	NOT APPLICABLE
HAL-08 REDUNDANT CHECK	Informational	SOLVED - 02/14/2022
HAL-09 REDUNDANT VARIABLE	Informational	SOLVED - 02/14/2022
HAL-10 USAGE OF VULNERABLE CRATES	Informational	ACKNOWLEDGED
HAL-11 OUTDATED RUST EDITION	Informational	SOLVED - 02/14/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) HAL-01 TCR VOTING DESIGN SHOULD BE IMPROVED - MEDIUM

Description:

It was observed that it is possible to:

- Vote with 0 amount
- Challenge yourself
- Counter yourself
- Vote for yourself

By combining these properties, some scenarios might be possible:

- A whale can influence any challenge/counter decision by voting for itself.
- A whale can also farm additional tokens upon success by countering any application and then voting to itself.
- By countering your application and voting with 0 amount, it is possible to fill up the storage since the values are pushed into vector
- To remove yourself from members in the root of trust.

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

Consider improving the design by not letting the same account to:

- Vote to itself
- Counter itself

- Challenge itself
- Vote with 0 deposit

Remediation Plan:

NOT APPLICABLE: The issue is marked as **not applicable** by the Nodle team as the **tcr** and **root of trust** pallets will be removed.

3.2 (HAL-02) HAL-02 DENOMINATION LOGIC SHOULD BE IMPROVED - MEDIUM

Description:

It was observed that if a nominator has a single validator, it is not possible to remove a validator through nominator_denominate since it has a check for <StakingMinNominatorTotalBond<T>>
In that case, nominator_denominate_all has to be used, which bypasses that check, which is not intentional.

Code Location:

Listing 1: pallets/staking/src/lib.rs

```
if !do_force {
                ensure!(
                    remaining >= <StakingMinNominatorTotalBond<T
↳ >>::get(),
                    <Error<T>>::NominatorBondBelowMin
                );
            }
```

Risk Level:

Likelihood - 4
Impact - 2

Recommendation:

Consider having a conditional statement in nominator_denominate that allows to force remove of validator if nominator has only one validator.

Remediation Plan:

SOLVED: The issue was solved by the Nodle team.

- Fix Commit

3.3 (HAL-03) HAL-03 EMERGENCY SHUTDOWN NOT USED IN MANY CRITICAL FUNCTIONS - MEDIUM

Description:

It was observed that the emergency shutdown pallet is used only in the allocate function in the allocations pallet. However, there are more public functions across different pallets that might be problematic if, at any point in time, there is a bug (security/non-security) discovered within them. There should be a functionality to shutdown them down before new fixes are pushed

Code Location:

These functions should have a shutdown functionality:

Grants pallet

- add_vesting_schedule

Staking pallet

- validator_join_pool
- validator_exit_pool
- validator_bond_more
- validator_bond_less
- nominator_nominate
- nominator_denominate
- nominator_bond_more
- nominator_bond_less
- nominator_move_nomination
- unbond_frozen
- withdraw_unbonded
- withdraw_staking_rewards

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

Consider enabling shutdown functionality in critical public functions.

Example Code:

Listing 2

```
ensure!(
        !pallet_emergency_shutdown::Pallet::<T>::shutdown(),
        Error::<T>::UnderShutdown
        );
```

Remediation Plan:

PENDING: In a future release, the Nodle team will modify the emergency shutdown pallet to better generalize.

3.4 (HAL-04) HAL-04 MISSING SANITY CHECKS - LOW

Description:

It was observed that the set_staking_limits privileged function is missing sanity checks on provided values. Even though it is a protected function, it is still advised to have some sanity checks to avoid any human error.

Code Location:

Listing 3: pallets/staking/src/lib.rs

```
201 pub fn set_staking_limits(
202             origin: OriginFor<T>,
203             max_stake_validators: u32,
204             min_stake_session_selection: BalanceOf<T>,
205             min_validator_bond: BalanceOf<T>,
206             min_nominator_total_bond: BalanceOf<T>,
207             min_nominator_chill_threshold: BalanceOf<T>,
208         ) -> DispatchResultWithPostInfo {
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

It is recommended to add sanity checks to ensure:

- max_stake_validators != 0
- min_stake_session_selection != 0
- min_validator_bond != 0
- min_nominator_total_bond != 0

Remediation Plan:

SOLVED: The issue was solved by the Nodle team.

- Fix Commit

3.5 (HAL-05) HAL-05 VESTING TO YOURSELF IS ALLOWED - LOW

Description:

It was observed that you can create a vesting schedule to yourself.

Code Location:

Listing 4: pallets/grants/src/lib.rs

```
157       pub fn add_vesting_schedule(
158             origin: OriginFor<T>,
159             dest: <T::Lookup as StaticLookup>::Source,
160             schedule: VestingScheduleOf<T>,
161         ) -> DispatchResultWithPostInfo {
162             let from = ensure_signed(origin)?;
163             let to = T::Lookup::lookup(dest)?;
164             Self::do_add_vesting_schedule(&from, &to, schedule.
 ↳ clone())?;
165
166             Self::deposit_event(Event::VestingScheduleAdded(from,
 ↳ to, schedule));
167             Ok(().into())
168         }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

Please add a check that ensures that the `from != to` in the `fn add_vesting_schedule`.

Remediation Plan:

SOLVED: The issue was solved by the Nodle team.

- Fix Commit

3.6 (HAL-06) HAL-06 MISSING ZERO VALUE CHECK - LOW

Description:

It was observed that the allocate function that should have a zero value check on the amount argument.

Code Location:

Listing 5: pallets/grants/src/lib.rs (Line 88)

```
85   pub fn allocate(
86               origin: OriginFor<T>,
87               to: T::AccountId,
88               amount: BalanceOf<T>,
89               proof: Vec<u8>,
90           ) -> DispatchResultWithPostInfo {
91               Self::ensure_oracle(origin)?;
92               ...
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Consider adding zero value checks to those functions to avoid performing redundant operations if a zero value is received.

Remediation Plan:

SOLVED: The issue was solved by the Nodle team.

- Fix Commit

3.7 (HAL-07) HAL-07 VESTING SCHEDULES LESS THAN A CURRENT BLOCK CAN BE CREATED - INFORMATIONAL

Description:

It was observed that the pallet allows the creation of vesting schedules that are less than the current block number. Those vesting schedules are not more than the regular transfers with extra steps. Therefore, those are redundant

Example:

Listing 6

```
Current Block: 100

Vesting Schedule Start: 1st Block

Period: 10 Blocks

Period_count: 2

Per Period: 1knodl

==================

Vesting Duration: 10 * 2 + 1 = 21 Blocks
Initial Transfer sent: 2knodl

Next Claims: 0 since Vesting Duration < Current Block
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider adding a check that ensures that the:

`(period * period_count)+ start > current_block_number`

Remediation Plan:

NOT APPLICABLE: The issue was marked as **not applicable** by the Nodle team saying:

This can be useful to keep it as it is. In fact, we may have to create retroactive awards that may have been partially vested.

3.8 (HAL-08) HAL-08 REDUNDANT CHECK - INFORMATIONAL

Description:

It was observed that the **grants** pallet contains a redundant check.

Code Location:

There is no need for a second new_lock.is_zero() since it was already checked prior. Removing of the vestingSchedule can be performed within the first check.

Listing 7: pallets/grants/src/lib.rs (Line 253)

```
247 if new_lock.is_zero() {
248                 T::Currency::remove_lock(VESTING_LOCK_ID, &target)
 ↳ ;
249             } else {
250                 T::Currency::set_lock(VESTING_LOCK_ID, &target,
 ↳ new_lock, WithdrawReasons::all());
251             }
252
253             if new_lock.is_zero() {
254                 // No more claimable, clear
255                 VestingSchedules::<T>::remove(target.clone());
256             } else {
257                 T::Currency::set_lock(VESTING_LOCK_ID, &target,
 ↳ new_lock, WithdrawReasons::all());
258             }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Please remove the second `new_lock.is_zero()` check and remove the `vestingSchedule` within the first check.

Listing 8: pallets/grants/src/lib.rs

```
247 if new_lock.is_zero() {
248                 T::Currency::remove_lock(VESTING_LOCK_ID, &target)
 ↳ ;
249                 VestingSchedules::<T>::remove(target.clone());
250             } else {
251                 T::Currency::set_lock(VESTING_LOCK_ID, &target,
 ↳ new_lock, WithdrawReasons::all());
252             }
```

Remediation Plan:

SOLVED: The issue was solved by the Nodle team.

- Fix Commit

3.9 (HAL-09) HAL-09 REDUNDANT VARIABLE - INFORMATIONAL

Description:

It was observed that the old1 variable in on_finalize function in tcr pallet is redundant. Tuple returned from commit_applications is Ok((new_members, Vec::new())). Therefore, old1 is always going to be an empty vector. Hence, extending it with old2 does not make any difference. In this scenario, we only care about new_1

Code Location:

Listing 9: pallets/tcr/src/lib.rs

```
138 fn on_finalize(block: T::BlockNumber) {
139             let (mut new_1, mut old_1) =
140                 Self::commit_applications(block).unwrap_or((Vec::
 ↳ new(), Vec::new()));
141             let (new_2, old_2) =
142                 Self::resolve_challenges(block).unwrap_or((Vec::
 ↳ new(), Vec::new()));
143
144             // Should never be the same, so should not need some
 ↳ uniq checks
145             new_1.extend(new_2);
146             old_1.extend(old_2);
147
148             new_1.sort();
149             old_1.sort();
150
151             Self::notify_members_change(new_1, old_1);
152         }
```

Listing 10: pallets/tcr/src/lib.rs (Line 478)

```
460 fn commit_applications(block: T::BlockNumber) ->
 ↳ FinalizeHelperResultFrom<T> {
461         let new_members = <Applications<T, I>>::iter()
462             .filter(|(_account_id, application)| {
463                 block
464                     .checked_sub(&application.clone().
 ↳ created_block)
465                     .expect("created_block should always be
 ↳ smaller than block; qed")
466                     >= T::FinalizeApplicationPeriod::get()
467             })
468             .map(|(account_id, application)| {
469                 <Applications<T, I>>::remove(account_id.clone());
470                 <Members<T, I>>::insert(account_id.clone(),
 ↳ application.clone());
471                 Self::unreserve_for(account_id.clone(),
 ↳ application.candidate_deposit);
472                 Self::deposit_event(Event::ApplicationPassed(
 ↳ account_id.clone()));
473
474                 account_id
475             })
476             .collect::<Vec<T::AccountId>>();
477
478         Ok((new_members, Vec::new())) //=== HERE ===
479     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider omitting old1 and remove all actions performed on it.

Listing 11: pallets/tcr/src/lib.rs

```
247 fn on_finalize(block: T::BlockNumber) {
248             let (mut new_1, _) =
249                 Self::commit_applications(block).unwrap_or((Vec::
 ↳ new(), Vec::new()));
250             let (new_2, mut old) =
251                 Self::resolve_challenges(block).unwrap_or((Vec::
 ↳ new(), Vec::new()));
252
253             // Should never be the same, so should not need some
 ↳ uniq checks
254             new_1.extend(new_2);
255
256
257             new_1.sort();
258             old.sort();
259
260             Self::notify_members_change(new_1, old);
261         }
```

Remediation Plan:

SOLVED: The issue was solved by the Nodle team.

- Fix Commit

3.10 (HAL-10) HAL-10 USAGE OF VULNERABLE CRATES - INFORMATIONAL

Description:

It was observed that the project uses crates with known vulnerabilities

Code Location:

ID	package	Short Description
RUSTSEC-2020-0159	chrono	Potential segfault in ‘localtime_r‘ invocations
RUSTSEC-2020-0071	time	Potential segfault in the time crate
RUSTSEC-2021-0130	lru	Use after free in lru crate
RUSTSEC-2021-0067	cranelift-codegen	Memory access due to code generation flaw in Cranelift module
RUSTSEC-2021-009	crossbeam-deque	Data race in crossbeam-deque
RUSTSEC-2021-0079	hyper	Integer overflow in hyper’s parsing of the Transfer-Encoding header leads to data loss
RUSTSEC-2021-0078	hyper	Lenient hyper header parsing of Content-Length could allow request smuggling
RUSTSEC-2021-0076	libsecp256k1	libsecp256k1 allows overflowing signatures
RUSTSEC-2021-0070	nalgebra	VecStorage Deserialize Allows Violation of Length Invariant
RUSTSEC-2021-0073	prost-types	Conversion from prost_types::Timestamp to SystemTime can cause an overflow and panic
RUSTSEC-2021-0013	raw-cpuid	Soundness issues in raw-cpuid
RUSTSEC-2021-0089	raw-cpuid	Optional Deserialize implementations lacking validation
RUSTSEC-2021-0124	tokio	Data race when sending and receiving after closing a oneshot channel
RUSTSEC-2021-0110	wasmtime	Multiple Vulnerabilities in Wasmtime
RUSTSEC-2021-0115	zeroize_derive	#[zeroize(drop)] doesn’t implement Drop for enums

Risk Level:

Likelihood - 2
Impact - 1

Recommendation:

Even if those vulnerable crates cannot impact the underlying application, it is advised to be aware of them and attempt to update them to no-vulnerable version. Furthermore, it is necessary to set up dependency monitoring to always be alerted when a new vulnerability is disclosed in one of the project's crates.

Remediation Plan:

ACKNOWLEDGED: The issue was **acknowledged** by the Nodle team and will be fixed later.

3.11 (HAL-11) HAL-11 OUTDATED RUST EDITION - INFORMATIONAL

Description:

It was observed that the project is using outdated rust edition(2018). Recently, 2021 rust edition came out, which includes a lot of stability improvements and new features that might make the code more readable.

Code Location:

- Cargo.toml

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider updating the Rust to the latest edition to use the latest features and stability improvements.

Reference:

Rust 2021 Edition Guide

Remediation Plan:

SOLVED: The issue was solved by the Nodle team.

- Fix Commit

AUTOMATED TESTING

4.1 AUTOMATED ANALYSIS

Description:

Halborn used automated security scanners to assist with detection of well-known security issues and vulnerabilities. Among the tools used was cargo audit, a security scanner for vulnerabilities reported to the RustSec Advisory Database. All vulnerabilities published in https://crates.io are stored in a repository named The RustSec Advisory Database. cargo audit is a human-readable version of the advisory database which performs a scanning on Cargo.lock. Security Detections are only in scope. All vulnerabilities shown here were already disclosed in the above report. However, to better assist the developers maintaining this code, the auditors are including the output with the dependencies tree, and this is included in the cargo audit output to better know the dependencies affected by unmaintained and vulnerable crates.

Results:

ID	package	Short Description
RUSTSEC-2020-0159	chrono	Potential segfault in ‘localtime_r‘ invocations
RUSTSEC-2020-0071	time	Potential segfault in the time crate
RUSTSEC-2021-0130	lru	Use after free in lru crate
RUSTSEC-2021-0067	cranelift-codegen	Memory access due to code generation flaw in Cranelift module
RUSTSEC-2021-009	crossbeam-deque	Data race in crossbeam-deque
RUSTSEC-2021-0079	hyper	Integer overflow in hyper’s parsing of the Transfer-Encoding header leads to data loss
RUSTSEC-2021-0078	hyper	Lenient hyper header parsing of Content-Length could allow request smuggling
RUSTSEC-2021-0076	libsecp256k1	libsecp256k1 allows overflowing signatures
RUSTSEC-2021-0070	nalgebra	VecStorage Deserialize Allows Violation of Length Invariant
RUSTSEC-2021-0073	prost-types	Conversion from prost_types::Timestamp to SystemTime can cause an overflow and panic
RUSTSEC-2021-0013	raw-cpuid	Soundness issues in raw-cpuid
RUSTSEC-2021-0089	raw-cpuid	Optional Deserialize implementations lacking validation
RUSTSEC-2021-0124	tokio	Data race when sending and receiving after closing a oneshot channel
RUSTSEC-2021-0110	wasmtime	Multiple Vulnerabilities in Wasmtime
RUSTSEC-2021-0115	zeroize_derive	#[zeroize(drop)] doesn’t implement Drop for enums

THANK YOU FOR CHOOSING

// HALBORN