SBP Automated Security Assessment for *peaq*

Security Research Labs

2022-12-13

Introduction and Scope

As part of the Substrate Builders Program, the blockchain security team of SRLabs has been tasked to provide a semi-automated security assessment for peaq.

This assessment was conducted on the week of the 19th of September 2022, targeting specifically the `peaq-node-runtime` runtime on branch `SLabs_Test_1.1`.

The automatic analysis is capable of identifying the following security vulnerabilities:

- Reachable runtime panic conditions
- Integer overflows
- Underweighted extrinsics
- Unbounded allocations
- Unbenchmarked extrinsics
- Exponential complexity in weights
- Use of twox_64_concat hasher for user provided keys
- Unsafe code
- Lack of storage deposits
- Wrongly configured ExistentialDeposit

Logic bugs are still the most widespread and impactful security bug category identified during blockchain security assurance projects provided by SRLabs. Finding a logic bug is a complex process that requires the audit team to have detailed knowledge of the involved protocols and business logic. Please note that this cannot be covered via the semi-automated tooling used for this assessment.

Findings

We have found a total of 1 vulnerabilities during this review.

[High] Insecure randomness used

Summary

The source of randomness in the runtime and in `pallet_contracts` is configured to use the collective flip random generator implemented in Substrate. The output of collective flip is highly predictable as it is based on the last 81 blocks and should not be used as a true source of randomness.

Issue details

In the definition of the configuration for the runtime in `runtime/src/lib.rs`:

```
// Create the runtime by composing the FRAME pallets that were previously configured.
construct_runtime!(
        pub enum Runtime where
                Block = Block,
                NodeBlock = opaque::Block,
                UncheckedExtrinsic = UncheckedExtrinsic
        {
                System: frame_system::{Pallet, Call, Config, Storage, Event<T>},
                RandomnessCollectiveFlip: pallet_randomness_collective_flip::{Pallet, Storage},
                ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                Timestamp: pallet_timestamp::{Pallet, Call, Storage, Inherent},
                Aura: pallet_aura::{Pallet, Config<T>},
                Grandpa: pallet_grandpa::{Pallet, Call, Storage, Config, Event},
                Balances: pallet_balances::{Pallet, Call, Storage, Config<T>, Event<T>},
                TransactionPayment: pallet_transaction_payment::{Pallet, Storage},
                Sudo: pallet_sudo::{Pallet, Call, Config<T>, Storage, Event<T>},
                Contracts: pallet_contracts::{Pallet, Call, Storage, Event<T>},
                // Include the custom pallets
                PeaqDid: peaq_pallet_did::{Pallet, Call, Storage, Event<T>},
                PeaqStorage: peaq_pallet_storage::{Pallet, Call, Storage, Event<T>},
                Transaction: peaq_pallet_transaction::{Pallet, Call, Storage, Event<T>},
                PeaqRbac: peaq_pallet_rbac::{Pallet, Call, Storage, Event<T>},
                MultiSig: pallet_multisig::{Pallet, Call, Storage, Event<T>},
                Utility: pallet_utility::{Pallet, Call, Event},

                // // EVM
                Ethereum: pallet_ethereum::{Pallet, Call, Storage, Event, Config, Origin},
                EVM: pallet_evm::{Pallet, Config, Call, Storage, Event<T>},
                DynamicFee: pallet_dynamic_fee::{Pallet, Call, Storage, Config, Inherent},
                BaseFee: pallet_base_fee::{Pallet, Call, Storage, Config<T>, Event},
        }
 );
```

as well as in the configuration of the `pallet_contract`, the `pallet_randomness_collective_flip` is used.

```
impl pallet_contracts::Config for Runtime {
        type Time = Timestamp;
        type Randomness = RandomnessCollectiveFlip;
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
        type Currency = Balances;
        type Event = Event;
        type Call = Call;
        /// The safest default is to allow no calls at all.
        ///
        /// Runtimes should whitelist dispatchables that are allowed to be called from contracts
        /// and make sure they are stable. Dispatchables exposed to contracts are not allowed to
        /// change because that would break already deployed contracts. The `Call` structure itself
        /// is not allowed to change the indices of existing pallets, too.
        type CallFilter = Nothing;
        type DepositPerItem = DepositPerItem;
        type DepositPerByte = DepositPerByte;
        type WeightPrice = pallet_transaction_payment::Pallet<Self>;
        type WeightInfo = pallet_contracts::weights::SubstrateWeight<Self>;
        type ChainExtension = ();
        type Schedule = Schedule;
        type CallStack = [pallet_contracts::Frame<Self>; 31];
        type DeletionQueueDepth = DeletionQueueDepth;
        type DeletionWeightLimit = DeletionWeightLimit;
        type AddressGenerator = pallet_contracts::DefaultAddressGenerator;
}
```

Risk

In the case of Peaq, this seems to be used not for collators (as you seem to use Aura), but for the `contracts_pallet`. We do not know the impact of this specifically for the contracts, but the collective flip random generator is inherently insecure and should not be used.

Mitigation

Using a secure randomness for the contracts pallet, either with the usage of an oracle of a project like `drand` or a secure library. Parity is also working on a secure implementation for randomness which could be used in the future.