Secure Code Review

Findings and Recommendations Report Presented to:

Interlock Network

June 20, 2023
Version: 2.1

Presented by:

Kudelski Security, Inc.
Route de Genève 22-24
1033, Cheseaux-sur-Lausanne,
Switzerland

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Overview

Interlock Network engaged Kudelski Security to perform a secure code assessment of smart contracts powering its decentralized security platform that delivers Web3-centered solutions to combat cybercrime.

The assessment was conducted remotely by the Kudelski Security Team.
Testing took place on March 20, 2023 - April 11, 2023, and focused on the following objectives:

- Provide the customer with an assessment of their overall security posture and any risks that were discovered with the smart contracts.
- 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.

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 the risk they pose.

- Overflow/underflow
- Input validation
- Centralization:
  - Missing pause function

Important note regarding all smart contracts and the way they are managed:

- Smart contracts are managed by a centralized authority, who has considerable power and needs to be trusted.

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

- Extensive documentation was available.
- The code was well commented and well written.

- Tests were also provided as part of the project.
- Finally, Interlock-network's team were extremely responsive, and always available to have helpful technical discussions.

While our comprehensive smart contract audit has highlighted security vulnerabilities into the Interlock network smart contracts, it is important to recognize that this assessment does not guarantee the identification of all potential vulnerabilities, as the constantly evolving nature of the Blockchain security landscape requires ongoing vigilance and adaptation.

Scope and Rules of Engagement

Kudelski performed a Secure Code Review for Interlock Network. 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 with the commit hashes in private repositories at:

- https://github.com/interlock-network/interlock-smartcontracts/commit/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6
  - Subfolder:
    - `contract_application`
    - `contract_ilockmvp`
    - `contract_uanft`
  - Written with ink! version 4.0.1

A further round of review has been performed by Kudelski Security, June 1-2, 2023, on remediations with the commit hash: 38c72771b77484a2a78af8440b9808ff34af4ccd available at:

- https://github.com/interlock-network/interlock-smartcontracts/commit/38c72771b77484a2a78af8440b9808ff34af4ccd

Interlock-network

`contract_application/`
`├── lib.rs`
`contract_ilockmvp/`
`├── lib.rs`
`contract_uanft/`
`├── lib.rs`

Table 1: Scope

TECHNICAL ANALYSIS & FINDINGS

During the Secure Code Review, we discovered 1 finding of critical severity, 3 findings of high severity, and 9 findings of a medium severity.

The following chart displays the findings by severity.

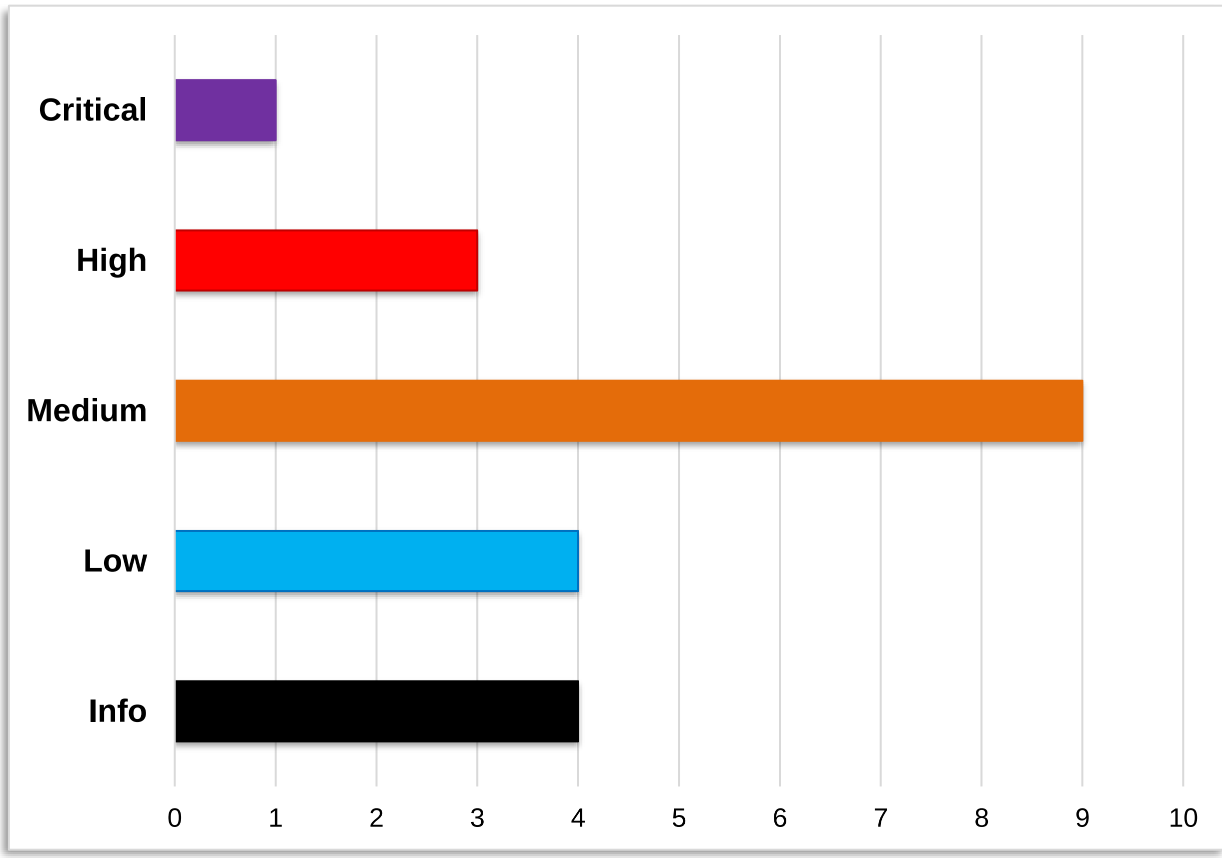


Figure 1: Findings by Severity

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	Status
KS-INT-01	Critical	Missing allowance guard	Resolved
KS-INT-02	High	Missing pause functionality	Resolved
KS-INT-03	High	Renounce ownership function can be called	Resolved
KS-INT-04	Low	Missing Zero Address Validation	Resolved
KS-INT-05	High	Password length is not sufficient	Resolved
KS-INT-06	Medium	Authentication tokens exposed	Resolved
KS-INT-07	Medium	Overwritten mapping: stakeholder data	Resolved
KS-INT-08	Medium	Single point of failure	Acknowledged
KS-INT-09	Medium	Insufficient password hashing primitive	Resolved
KS-INT-10	Medium	Unsafe math: integer overflow/underflow	Resolved
KS-INT-11	Medium	unwrap() can lead to panic	Resolved
KS-INT-12	Medium	Missing existing port validation	Resolved
KS-INT-13	Medium	Unbounded index	Resolved
KS-INT-14	Medium	Missing argument validation with Interlock registration	Resolved

KS-INT-15	Low	Division by zero	Resolved
KS-INT-16	Low	Incorrect inequality check	Resolved
KS-INT-17	Low	Token mint race attack possible	Resolved
KS-INT-18	**Informational**	Unnecessary memory allocation	Informational
KS-INT-19	**Informational**	Hardcoded values	Informational
KS-INT-20	**Informational**	Function defined for public and restricted access	Informational
KS-INT-21	**Informational**	Incomplete test coverage	Informational

Table 2: Findings Overview

KS-INT- 01 – Missing allowance guard

Severity	CRITICAL
Status	RESOLVED

Impact	Likelihood	Difficulty
High	High	Easy

Description

transfer and *transfer_from* may be called to transfer tokens arbitrarily. The *openbrush* transfer functions contain guards in the form of restrictions on *allowance*, preventing arbitrary token transfers. However, the versions implemented here do not have the same functionality.

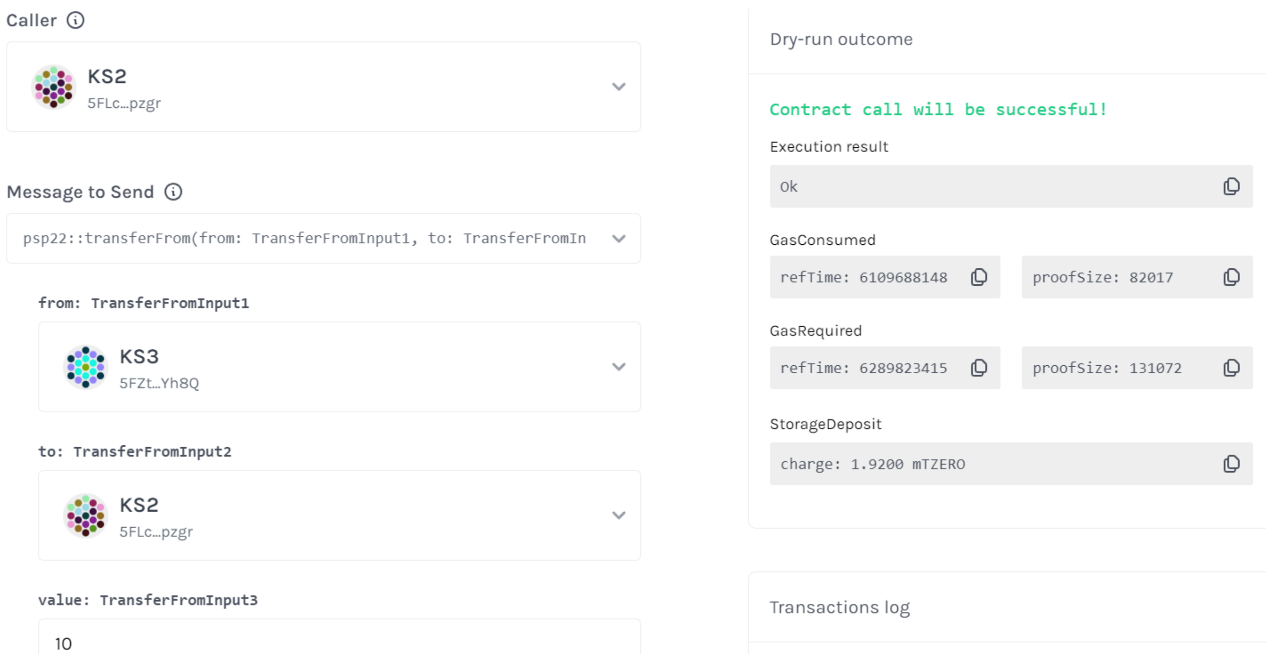
Impact

Anyone can transfer tokens to and from anyone else.

Evidence

```
fn transfer_from(
    &mut self,
    from: AccountId,
    to: AccountId,
    value: Balance,
    data: Vec<u8>,
) -> PSP22Result<()> {

    let caller = self.env().caller();
    let allowance = self._allowance(&from, &caller);
    let _ = self._approve_from_to(from, caller, allowance - value)?;
    let _ = self._transfer_from_to(from, to, value, data)?;

    // if sender is owner, then tokens are entering circulation
    if from == self.ownable.owner {

        match self.balances[CIRCULATING as usize].checked_add(value) {
            Some(sum) => self.balances[CIRCULATING as usize] = sum,
            None => return Err(OtherError::Overflow.into()),
        };
    }

    // if recipient is owner, then tokens are being returned or added to rewards pool
    if to == self.ownable.owner {

        match self.balances[REWARDS as usize].checked_add(value) {
            Some(sum) => self.balances[REWARDS as usize] = sum,
            None => return Err(OtherError::Overflow.into()),
        };
        match self.balances[CIRCULATING as usize].checked_sub(value) {
            Some(difference) => self.balances[CIRCULATING as usize] = difference,
            None => return Err(OtherError::Underflow.into()),
        };
    }
```

transfer_from function in contract_ilockmvp/lib.rs



Call of the transfer_from function done by KS2 which transfer himself token from KS3

Affected Resource

- `contract_ilockmvp/lib.rs` (lines 554, 590-623)

Recommendation

Simply provide a wrapper for the Openbrush functions rather than reimplement them, as these contain appropriate guards.

Reference

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_ilockmvp/lib.rs#L590-L623

https://openbrush.io/

KS-INT-02 – Missing pause functionality

Severity	HIGH
Status	RESOLVED

Impact	Likelihood	Difficulty
High	Medium	Moderate

Description

The Interlock-network smart contracts do not have a pause function even though a central authority is supervising their code.

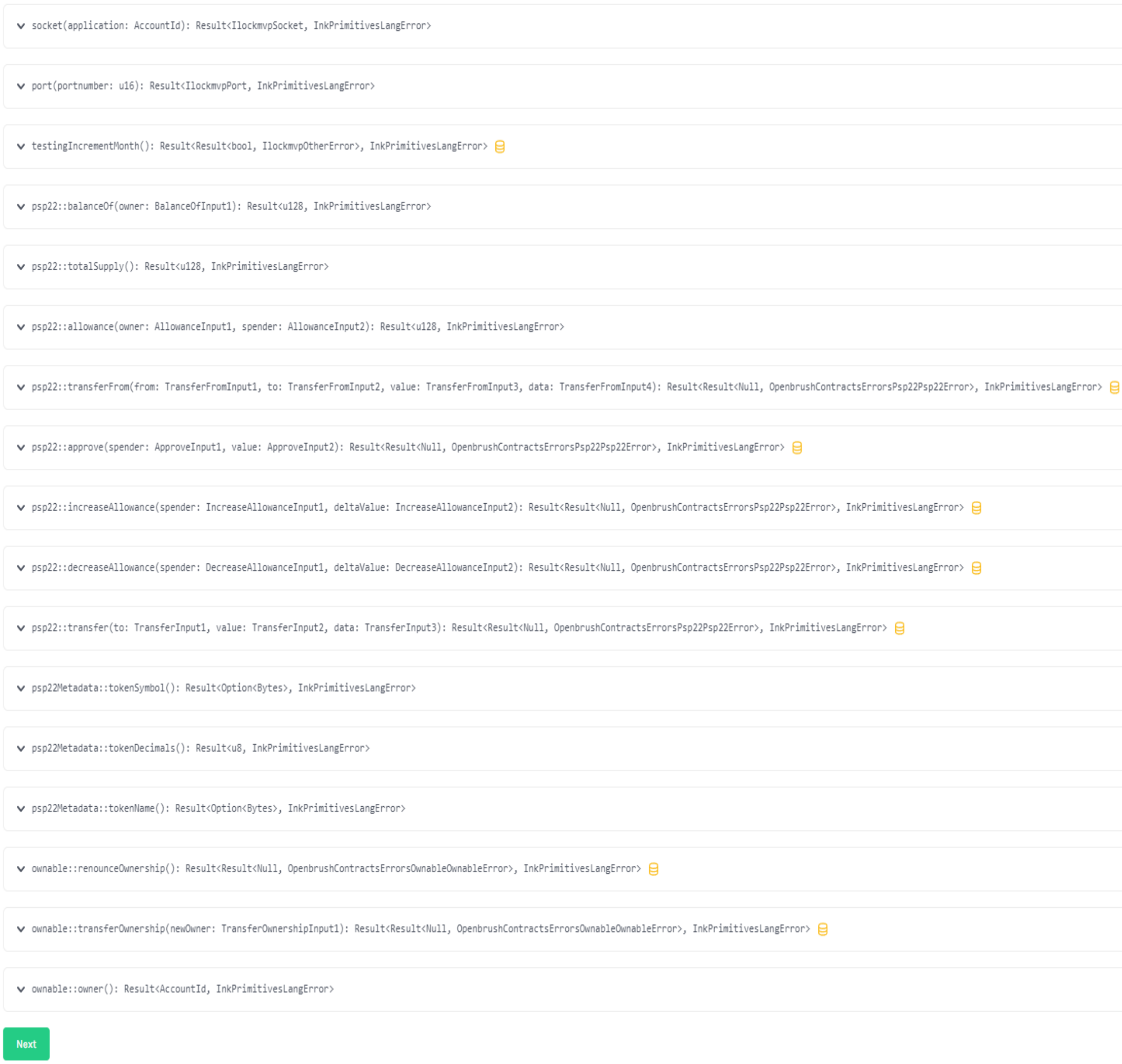
Impact

In the case where an attack occurs and resources are compromised, there will be no way to pause the smart contract to limit the damage of an ongoing attack or to prevent further attacks.

Evidence

```
newToken()
checkTime(): Result<Result<Null, OpenbrushContractsErrorsPsp22Psp22Error>, InkPrimitivesLangError>
remainingTime(): Result<u64, InkPrimitivesLangError>
registerStakeholder(stakeholder: AccountId, share: Balance, pool: u8): Result<Result<Null, OpenbrushContractsErrorsPsp22Psp22Error>, InkPrimitivesLangError>
stakeholderData(stakeholder: AccountId): Result<(IlockmvpStakeholderData,u128,u128,Text), InkPrimitivesLangError>
distributeTokens(stakeholder: AccountId): Result<Result<Null, IlockmvpOtherError>, InkPrimitivesLangError>
payoutTokens(stakeholder: AccountId, amount: Balance, pool: String): Result<Result<Null, IlockmvpOtherError>, InkPrimitivesLangError>
poolData(poolnumber: u8): Result<(Text,Text,Text,Text), InkPrimitivesLangError>
poolBalance(poolnumber: u8): Result<(Text,u128), InkPrimitivesLangError>
rewardInterlocker(reward: Balance, interlocker: AccountId): Result<Result<u128, IlockmvpOtherError>, InkPrimitivesLangError>
rewardedInterlockerTotal(interlocker: AccountId): Result<u128, InkPrimitivesLangError>
rewardedTotal(): Result<u128, InkPrimitivesLangError>
monthsPassed(): Result<u16, InkPrimitivesLangError>
cap(): Result<u128, InkPrimitivesLangError>
updateContract(codeHash: [u8;32]): Result<Result<Null, OpenbrushContractsErrorsPsp22Psp22Error>, InkPrimitivesLangError>
createPort(codehash: Hash, tax: Balance, cap: Balance, locked: bool, number: u16, owner: AccountId): Result<Result<Null, OpenbrushContractsErrorsPsp22Psp22Error>, InkPrimitivesLangError>
createSocket(operator: AccountId, portnumber: u16): Result<Result<Null, IlockmvpOtherError>, InkPrimitivesLangError>
callSocket(address: AccountId, amount: Balance, data: Vec): Result<Result<Null, IlockmvpOtherError>, InkPrimitivesLangError>
socket(application: AccountId): Result<IlockmvpSocket, InkPrimitivesLangError>
port(portnumber: u16): Result<IlockmvpPort, InkPrimitivesLangError>
```

Figure below and above demonstrate the methods available when deploying `contract_ilockmvp`, and there is no pause function available.



Affected Resource

- `contract_uanft/lib.rs`
- `contract_ilockmvp/lib.rs`

Recommendation

We recommend implementing a pause function callable only by the contract owner.

Reference

https://brushfam.github.io/openbrush-contracts/smart-contracts/pausable/

KS-INT-03 – Renounce ownership function can be called

Severity	HIGH
Status	RESOLVED

Impact	Likelihood	Difficulty
High	Medium	Easy

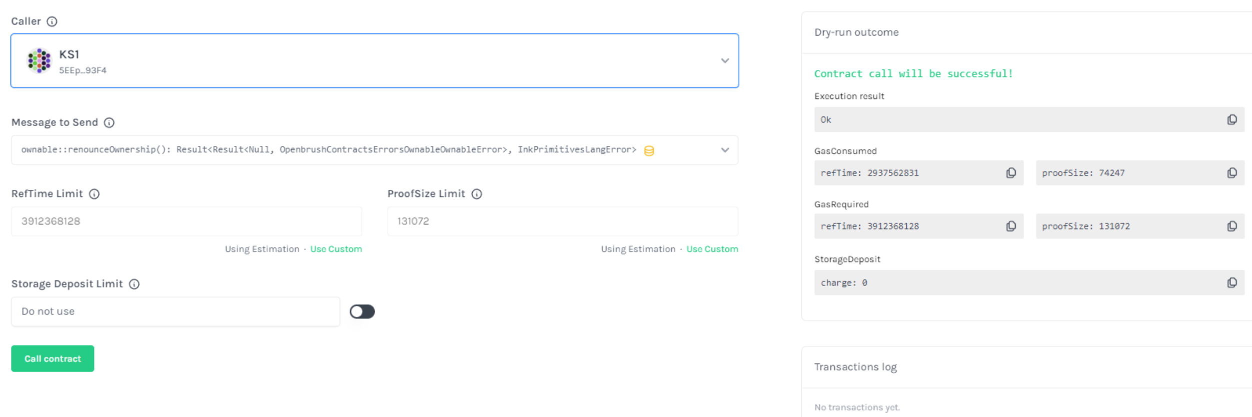
Description

Interlock network uses the Openbrush library, which includes a function *ownable::renounceOwnership()*. Once invoked, ownership is removed from the owner account.

Impact

If the owner of the Interlock smart contract mistakenly renounces ownership, they lose access to vital functions such as *payout_token*, *reward_interlocker*, and *register_stakeholder*. This would result in a loss of all Interlock network contract tokens. Once the ownership has been renounced by the Interlock network accounts, the Zero Address owns the smart contract. The Zero Address is an address which has a public seed.

Evidence



KS1 which is the contract owner can renounce to its ownership by calling the ownable::renounceOnwership() function.

Affected Resource

- `contract_application/lib.rs`
- `contract_uanft/lib.rs`
- `contract_ilockmvp/lib.rs`

Recommendation

Do not import this function. or disable the renounce functionality, as it is unnecessary and might cause irreversible damage to the Interlock project.

Reference
https://openbrush.io/

KS-INT-04 – Missing Zero Address Validation

Severity	LOW
Status	RESOLVED

Impact	Likelihood	Difficulty
High	Low	Difficult

Description

The Aleph Zero “Zero Address” is an existing address on the network which has a publicly known secret seed. We noticed that input validation on addresses were not checked against the Zero Address. For example, the “Zero Address” address could become a stakeholder as demonstrated in the Evidence below.

An important note:

Impact

If Interlock tokens are allocated to the Zero Address, those tokens can be stolen by any users.

Evidence

```
        #[ink(message)]
        fn transfer_ownership(
            &mut self,
            newowner: AccountId
        ) -> Result<(), OwnableError> {

            let oldowner = self.ownable.owner;

            // only-owner modifier not present in this scope
            if oldowner != self.env().caller() {

                return Err(OwnableError::CallerIsNotOwner);
            }
            let oldbalance: Balance = self.balance_of(oldowner);

            // transfer all remaining owner tokens (pools) to new owner
            let mut newbalance: Balance = self.psp22.balance_of(newowner);
            match newbalance.checked_add(oldbalance) {
                Some(sum) => newbalance = sum,
                None => (), // case not possible
            };
            self.psp22.balances.insert(&newowner, &newbalance);

            // deduct tokens from owners account
            self.psp22.balances.insert(&oldowner, &0);

            self.ownable.owner = newowner;

            Ok(())
        }
    }
```

The function transfer_ownership allows the transfer to the Zero Address

A user can transfer ownership of his token to the Zero Address as demonstrate in the figure below where the KS1 call this function passing the Zero Address as argument and the call is executed successfully.

```
maxime.buser@macC17CW1NJML7H contract_ilockmvp % cargo contract call --suri "$SEED" --url "$URL" --contract "$CONTRACT" --mess
age Ownable::transfer_ownership --args 0x0000000000000000000000000000000000000000000000000000000000000000 -x
 Dry-running Ownable::transfer_ownership (skip with --skip-dry-run)
    Success! Gas required estimated at Weight(ref_time: 3912368128, proof_size: 131072)
Confirm transaction details: (skip with --skip-confirm)
     Message Ownable::transfer_ownership
        Args 0x0000000000000000000000000000000000000000000000000000000000000000
   Gas limit Weight(ref_time: 3912368128, proof_size: 131072)
Submit? (Y/n): y
      Events
       Event Balances → Withdraw
         who: 5EqReoNMRGofmN3VBLWnTR4BGczkSk7umwVbCbJ3rPuwPS7m
         amount: 4.713985315mTZERO
       Event Contracts → Called
         caller: 5EqReoNMRGofmN3VBLWnTR4BGczkSk7umwVbCbJ3rPuwPS7m
         contract: 5Ee4SqcmDFw6s7jWHfhB8W2cDgxDM1VPEGN1cxsrUj8vPaDS
       Event Balances → Transfer
         from: 5EqReoNMRGofmN3VBLWnTR4BGczkSk7umwVbCbJ3rPuwPS7m
         to: 5Ee4SqcmDFw6s7jWHfhB8W2cDgxDM1VPEGN1cxsrUj8vPaDS
         amount: 1.92mTZERO
       Event Balances → Reserved
         who: 5Ee4SqcmDFw6s7jWHfhB8W2cDgxDM1VPEGN1cxsrUj8vPaDS
         amount: 1.92mTZERO
       Event Balances → Deposit
         who: 5EqReoNMRGofmN3VBLWnTR4BGczkSk7umwVbCbJ3rPuwPS7m
         amount: 577.902888µTZERO
       Event Balances → Deposit
         who: 5EYCAe5fg5WiYGVNH6QpCFnu55Hzv9MwtjFHdQCx8EaSQTm2
         amount: 4.136082427mTZERO
       Event Treasury → Deposit
         value: 4136082427
       Event TransactionPayment → TransactionFeePaid
         who: 5EqReoNMRGofmN3VBLWnTR4BGczkSk7umwVbCbJ3rPuwPS7m
         actual_fee: 4.136082427mTZERO
         tip: 0TZERO
       Event System → ExtrinsicSuccess
         dispatch_info: DispatchInfo { weight: Weight { ref_time: 4136082240, proof_size: 78507 }, class: Normal, pays_fee: Ye
s }
```

transfer_ownership to the Zero Address can be successfully executed

```rust
#[ink(message)]
#[openbrush::modifiers(only_owner)]
pub fn register_stakeholder(
    &mut self,
    stakeholder: AccountId,
    share: Balance,
    pool: u8,
) -> PSP22Result<()> {

    // make sure share is > 0
    if share == 0 {
        return Err(OtherError::ShareTooSmall.into());
    }

    // create stakeholder struct
    let this_stakeholder = StakeholderData {
        paid: 0,
        share: share,
        pool: pool,
    };

    // insert stakeholder struct into mapping
    self.vest.stakeholder.insert(stakeholder, &this_stakeholder);

    Ok(())
}
```

The function register_stakeholder allows the registration the Zero Address.

```
● maxime.buser@macC17CW1NJML7H contract_ilockmvp % cargo contract call --suri "$SEED" --url "$URL" --contract "$CONTRACT" --mess
age register_stakeholder --args 0x0000000000000000000000000000000000000000000000000000000000000000 10 2 -x
 Dry-running register_stakeholder (skip with --skip-dry-run)
    Success! Gas required estimated at Weight(ref_time: 3912368128, proof_size: 131072)
Confirm transaction details: (skip with --skip-confirm)
     Message register_stakeholder
        Args 0x0000000000000000000000000000000000000000000000000000000000000000 10 2
   Gas limit Weight(ref_time: 3912368128, proof_size: 131072)
Submit? (Y/n): y
      Events
       Event Balances → Withdraw
         who: 5EqReoNMRGofmN3VBLWnTR4BGczkSk7umwVbCbJ3rPuwPS7m
         amount: 4.713985332mTZERO
       Event Contracts → Called
         caller: 5EqReoNMRGofmN3VBLWnTR4BGczkSk7umwVbCbJ3rPuwPS7m
         contract: 5Ee4SqcmDFw6s7jWHfhB8W2cDgxDM1VPEGN1cxsrUj8vPaDS
       Event Balances → Transfer
         from: 5EqReoNMRGofmN3VBLWnTR4BGczkSk7umwVbCbJ3rPuwPS7m
         to: 5Ee4SqcmDFw6s7jWHfhB8W2cDgxDM1VPEGN1cxsrUj8vPaDS
         amount: 2.60mTZERO
       Event Balances → Reserved
         who: 5Ee4SqcmDFw6s7jWHfhB8W2cDgxDM1VPEGN1cxsrUj8vPaDS
         amount: 2.60mTZERO
       Event Balances → Deposit
         who: 5EqReoNMRGofmN3VBLWnTR4BGczkSk7umwVbCbJ3rPuwPS7m
         amount: 799.487798µTZERO
       Event Balances → Deposit
         who: 5EYCAe5fg5WiYGVNH6QpCFnu55Hzv9MwtjFHdQCx8EaSQTm2
         amount: 3.914497534mTZERO
       Event Treasury → Deposit
         value: 3914497534
       Event TransactionPayment → TransactionFeePaid
         who: 5EqReoNMRGofmN3VBLWnTR4BGczkSk7umwVbCbJ3rPuwPS7m
         actual_fee: 3.914497534mTZERO
         tip: 0TZERO
       Event System → ExtrinsicSuccess
         dispatch_info: DispatchInfo { weight: Weight { ref_time: 3914497330, proof_size: 75403 }, class: Normal, pays_fee: Ye
s }
```

Example of lack address verification, with the Zero Address becoming a stakeholder of the Interlock project.

Affected Resource

- `contract_ilockmvp/lib.rs` (lines 589-628, 640- 670, 814-860, 875-962, 972-1019)

Recommendation

Validate all input arguments by checking them against the Zero Address, and for the functions implemented from the Openbrush, ensure that their protection has been used.

Reference

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_ilockmvp/lib.rs#L812-L837

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_ilockmvp/lib.rs#L640-L670

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_ilockmvp/lib.rs#L589-L628

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_ilockmvp/lib.rs#L589-L628
https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_ilockmvp/lib.rs#L875-962

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_ilockmvp/lib.rs#L589-L628
https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_ilockmvp/lib.rs#L972-1019

KS-INT-05 – Password length is not sufficient

Severity	HIGH
Status	RESOLVED

Impact	Likelihood	Difficulty
High	High	Moderate

Description

Length of a password is the primary factor in password strength, which limits brute force attacks. We found a 8-character minimum imposed on the password length of a user's Interlock account password. Although this ensures some degree of password strength, it is below the recommendation for strong passwords.

Impact

An attacker may crack a user's password using brute force methods. This may allow an attacker to access to user resources that vary by application.

Evidence

Affected Resource

- `contract_ilockmvp`
- `contract_uanft`
- `contract_application`

Recommendation

Follow the NIST guideline for password requirements. Although there is a strict 8 character minimum requirement in the NIST standard, we recommend 12 characters for a strong password with no complexity requirements. Moreover, the password should be updateable.

Reference

https://auth0.com/blog/dont-pass-on-the-new-nist-password-guidelines/

KS-INT-06 – Authentication tokens exposed

Severity	MEDIUM
Status	RESOLVED

Impact	Likelihood	Difficulty
Low	High	Easy

Description

The smart contract `contract_uanft/lib.rs` (UANFT) stores a mapping in the *credential* field of the public struct *AccessData*. This maps username hashes onto a tuple of password hashes and UANFT contract instances, indicated by id:

usernameHash → (passwordHash, uanftId).

This is used to identify a user on a specific UANFT contract instance. According to the their architecture design, the user sends their username and password to the Interlock server over HTTPS. The server authenticates the user by hashing the username and password separately and then comparing these to the publicly available ones on the blockchain. The server does not accept the hashes directly from the user to authenticate. However, since the hashes are publicly stored on chain, these will be available for any attacker to read and use in a brute force attack.

Impact

An attacker may read the public credentials and attempt to determine the original username or password from their hashes. A brute force attack using dictionaries is more likely to succeed considering limited password lengths. Moreover, if the publicly available username hash can be correlated with identifying information such as user address or phone number, then an attacker may use this in a brute force attack as well. Once the username and password are acquired, an attacker would gain access to any resource the original user had. This may include access to private areas that could potentially be chained into further attacks.

Evidence

```
155             /// credentials:          username hash -> (password hash, uanft ID)
156             pub credentials: Mapping<Hash, (Hash, Id)>,
```

The hash of the username is publicly stored with the password hash in `contract_uanft/lib.rs`*.*

Affected Resource

- `contract_uanft/lib.rs` (line 156)

Recommendation

Remove this credential scheme. It is generally better to use existing cryptographic libraries which have been tested over time. However, if they must be used, store and transmit them securely through private channels.

You may use a public/private keypair strategy for identification and authentication. Otherwise, you may use temporarily generated authentication tokens combined with OTP multifactor authentication.

Reference

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_uanft/lib.rs#L156

KS-INT-07 – Overwritten mapping: stakeholder data

Severity	MEDIUM
Status	RESOLVED

Impact	Likelihood	Difficulty
High	Medium	Difficult

Description

The data structure *StakeholderData* allows only one pool. If the stakeholder mapping is overwritten and a different pool is assigned, then the original pool may not be referenced with regard to that stakeholder. Similarly, the amount paid and share is overwritten. This data is lost and can no longer be referenced.

Impact

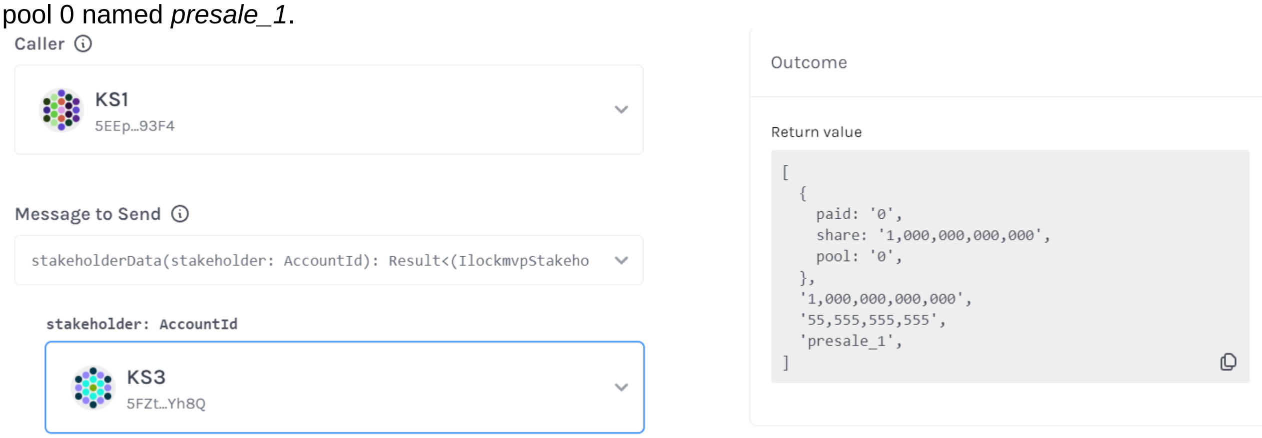
Payouts associated with “lost pool” may not be given as his shares have been overwritten, so the stakeholder receives less payout.

Evidence

```
// create stakeholder struct
let this_stakeholder = StakeholderData {
    paid: 0,
    share: share,
    pool: pool,
};

// insert stakeholder struct into mapping
self.vest.stakeholder.insert(stakeholder, &this_stakeholder);
```

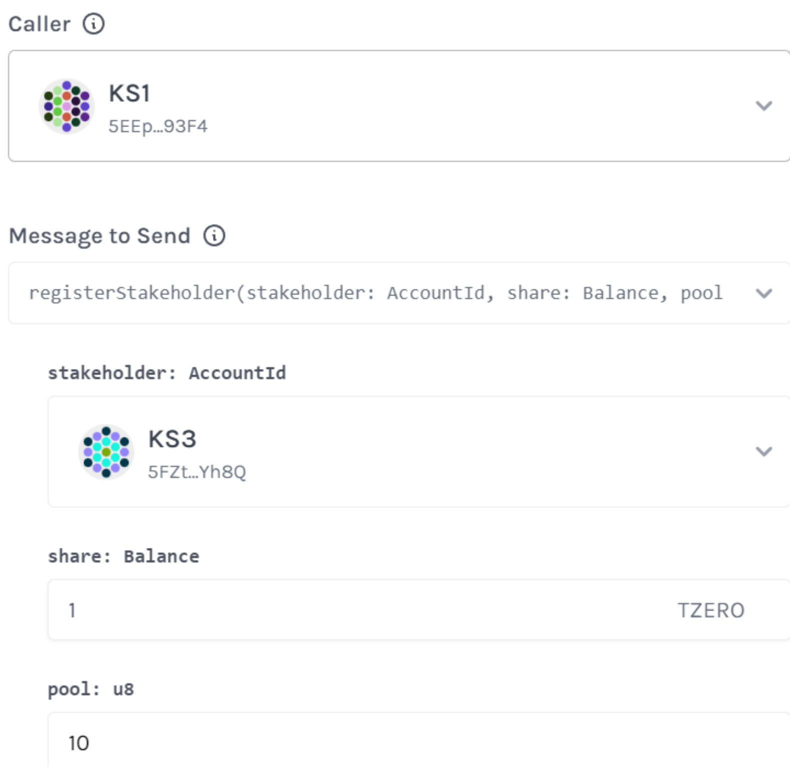
`contract_ilockmvp/lib.rs` *stakeholder data structure illustration.*

The three figures below illustrate the loss of share from the stakeholder KS3. KS3 has currently shares in the pool 0 named *presale_1*.



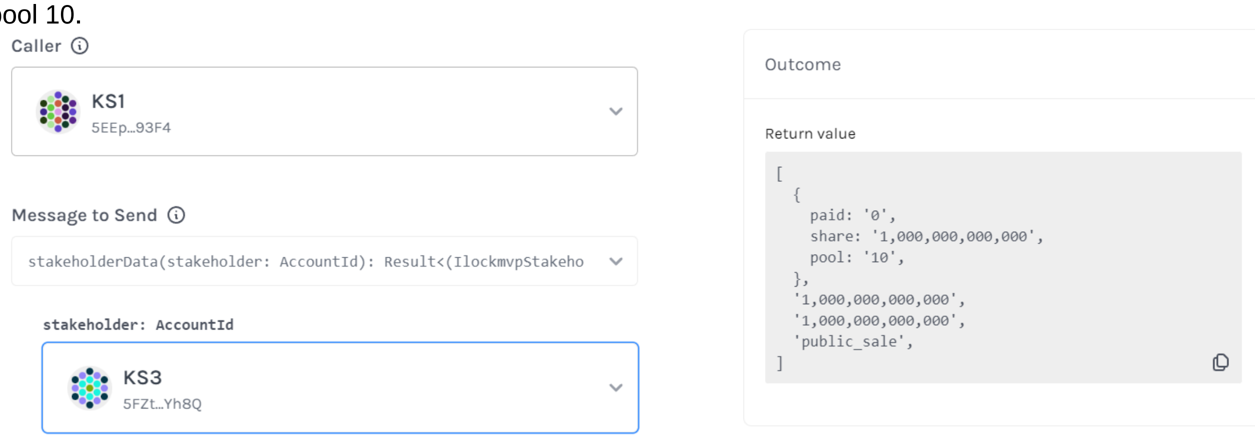
stakeholder_data output indicates that KS3 has shares associated with the pool 0.

Then, KS3 is registration is performing a registration for the pool 10 as illustrated in the figure below



KS3 registration in the pool 10.

After the successful registration of KS3 into the pool 10, KS3 lost his share belonging to the pool 0 even though it did not sell it or did anything. The figure below demonstrates that KS3 owns only share from the pool 10.



stakeholder_data output indicate that KS3 has only shares associcated with the pool 10.

Affected Resource

- `contract_ilockmvp/lib.rs` (lines 811-836)

Recommendation

Use an array of pools or other appropriate data structure to assign to a stakeholder. Then include that in the calculation of payout. Or limit to one registration per public key

Reference

N/A

KS-INT-08 – Single point of failure

Severity	MEDIUM
Status	ACKNOWLEDGED

Impact	Likelihood	Difficulty
High	Medium	Moderate

Description

Many functions are restricted to access by a single account using the *only_owner* modifier. This means that the contract's owner plays a crucial role in the functioning of the contract. The owner is exclusively responsible for the stakeholder registration, distribution of the token, and reward payouts.

Impact

If the contract owner is compromised, the consequence is severe because of the power of the owner over the Interlock protocol.

Evidence

N/A

Affected Resource

- `contract_ilockmvp`
- `contract_uanft`
- `contract_application`

Recommendation

Multi-signature accounts for each of the contract owners. A multisig account needs the signature of multiple "sub-accounts" in order to call a function in the Interlock network smart contract. This means an attacker needs to corrupt multiple sub-accounts instead of just one.

Reference

https://wiki.polkadot.network/docs/learn-account-multisig
https://polkadot.js.org/docs/util-crypto/examples/create-multisig/

KS-INT-09 – Insufficient password hashing primitive

Severity	MEDIUM
Status	RESOLVED

Impact	Likelihood	Difficulty
Medium	Medium	Moderate

Description

Interlock uses the hash function SHA256 to perform password hashing which is not suitable for password hashing. Additionally, a salt should be used while performing password hashing.

Impact

SHA256 is optimized when it comes to computations and memory complexity. This sounds interesting in terms of efficiency of an application; however, this also generates drawback from a security point of view as it decreases attacker computations and memory requirement for brute force attacks.

Evidence

```
/// - Credentials contains a SHA256 (or other) hashed secret and uanft ID for said
/// secret, one pair per user identifing (eg username) SHA256 hash.
/// - This is important because it provides a means of verifying possession
/// of secret, and for which uanft this owner has access to for those
/// given credentials.
///
/// credentials:         username hash -> (password hash, uanft ID)
pub credentials: Mapping<Hash, (Hash, Id)>,
```

SHA256 is used for password hashing without the use of any salt.

Affected Resource

- `contract_uanft/lib.rs` line 165

Recommendation

We suggest including a salt for password hashing. Also, use a hash function designed for password hashing with higher memory and computation complexity such as scrypt, argon2, PBKDF2. This will limit brute force attacks.

Reference

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_uanft/lib.rs#L156
https://www.ietf.org/rfc/rfc2898.txt

https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

KS-INT-10 – Unsafe math: integer overflow/underflow

Severity	MEDIUM
Status	RESOLVED

Impact	Likelihood	Difficulty
High	Low	Difficult

Description

The protection against overflow has been turned-off in Cargo.toml, by setting over-flow checks to false in the *profile.release* section. This makes some parts of the Interlock project vulnerable to overflows.

Impact

Integer values such as *last_token_id* may wrap around. This increases the risk of two tokens having the same ID. Two users sharing the same access token could result in the mixing of their shares or payouts.

Evidence

The risk of overflow is present at the following lines in the file `contract_ilockmvp/lib.rs`:

- line 484,517, 549, 606.

Interlock > interlock-smartcontracts > contract_uanft > Cargo.toml

```
[profile.release]
overflow-checks = false
```

Interlock > interlock-smartcontracts > contract_ilockmvp > Cargo.toml

```
[profile.release]
overflow-checks = false
```

Cargo.toml files for both smart contracts have turned the overflow protection off.

```
484¦ self.last_token_id += 1;
  :¦-----------------------------------------
517¦ self.last_token_id += 1;
  :¦-----------------------------------------
549¦ self.last_token_id += 1;
  :¦-----------------------------------------
606¦ self.last_token_id += 1;
  :¦-----------------------------------------
```

Lines in contract_uanft/lib.rs which can result in an overflow.

Affected Resource

- `contract_uanft/lib.rs` (lines 484, 517, 549, 606)
- `contract_ilockmvp/lib.rs` (lines 47-48)
- `contract_uanft/Cargo.toml` (line 47-48)
- `contract_ilockmvp/Cargo.toml` (line 15-16)

Recommendation

Enable overflow checks in profile.release of Cargo.toml. If this is disallowed by the current version of ink, then use safe math functions instead.

Reference

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_uanft/Cargo.toml#L47-L48
https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_ilockmvp/Cargo.toml#L15-L16

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_uanft/lib.rs#L484
https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_uanft/lib.rs#L517

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_uanft/lib.rs#L549

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_uanft/lib.rs#L606

KS-INT-11 – unwrap() can lead to panic

Severity	MEDIUM
Status	RESOLVED

Impact	Likelihood	Difficulty
Low	High	Easy

Description

The function *stakeholder_data* in `contract_ilockmvp/lib.rs` uses unwrap() to check if the stakeholder exists. If the argument AcccountId is not a valid account, then this function will panic and stop. This issue also appears in `contract_uanft` in the function *contract_hash.*

Impact

Panics can break the state of the contract, while recoverable errors enable continued execution. Panics may enable DDOS attacks, which may be chained into other attacks (arbitrage, consensus, etc.)

Evidence

```
#[ink(message)]
pub fn stakeholder_data(
    &self,
    stakeholder: AccountId,
) -> (StakeholderData, Balance, Balance, String) {

    // get pool and stakeholder data structs first
    let this_stakeholder = self.vest.stakeholder.get(stakeholder).unwrap();
    let pool = &POOLS[this_stakeholder.pool as usize];

```

stakeholder_data in contract_ilockmvp calls the unwrap() function and could make the contract panic

```
#[ink(message)]
pub fn contract_hash(
    &self,
    application: AccountId,
) -> Hash {

    self.env().code_hash(&application).unwrap()
}
```

contract_hash in contract_uanft calls the unwrap() function and could make the contract panic

Affected Resource

- `contract_ilockmvp/lib.rs` (line 847)
- `contract_uanft/lib.rs` (line 851)

Recommendation

As a best practice, one should assign an error and error handler for recoverable errors.

Reference

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_uanft/lib.rs#L865-L872

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_ilockmvp/lib.rs#L847

KS-INT-12 – Missing existing port validation

Severity	MEDIUM
Status	RESOLVED

Impact	Likelihood	Difficulty
Low	High	Easy

Description

The *create_port* function in `contract_ilockmvp/lib.rs` can overwrite an existing port with new information. There is no check that a port already exists before updating the Port struct. Moreover, there is no validation of the function arguments.

Impact

One may inadvertently overwrite existing port information. Additionally, the lack of input verification may lead to the creation of a faulty port with parameters that would block the execution of other functions.

Evidence

```
1221        #[ink(message)]
1222        #[openbrush::modifiers(only_owner)]
1223        pub fn create_port(
1224            &mut self,
1225            codehash: Hash,
1226            tax: Balance,
1227            cap: Balance,
1228            locked: bool,
1229            number: u16,
1230            owner: AccountId,
1231        ) -> PSP22Result<()> {
1232
1233            let port = Port {
1234                application: codehash,     // <--! a port defines an external staking/reward contract plus any
1235                tax: tax,                  //      custom logic preceding the tax_and_reward() function
1236                cap: cap,
1237                locked: locked,
1238                paid: 0,
1239                collected: 0,
1240                owner: owner,
1241            };
1242            self.app.ports.insert(number, &port);
1243
1244            Ok(())
1245        }
```

The create_port function does not perform any verifications about the parameters of the port

Affected Resource

- `contract_ilockmvp/lib.rs` (lines 1223-1245)

Recommendation

A force flag or separate *update_port* function would reduce the risk of inadvertently overwriting a port. Additionally, we suggest implementing a validation check for every single argument of this function.

Reference

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_ilockmvp/lib.rs#L1221-L1245

KS-INT-13 – Unbounded index

Severity	MEDIUM
Status	RESOLVED

Impact	Likelihood	Difficulty
Low	Low	Easy

Description

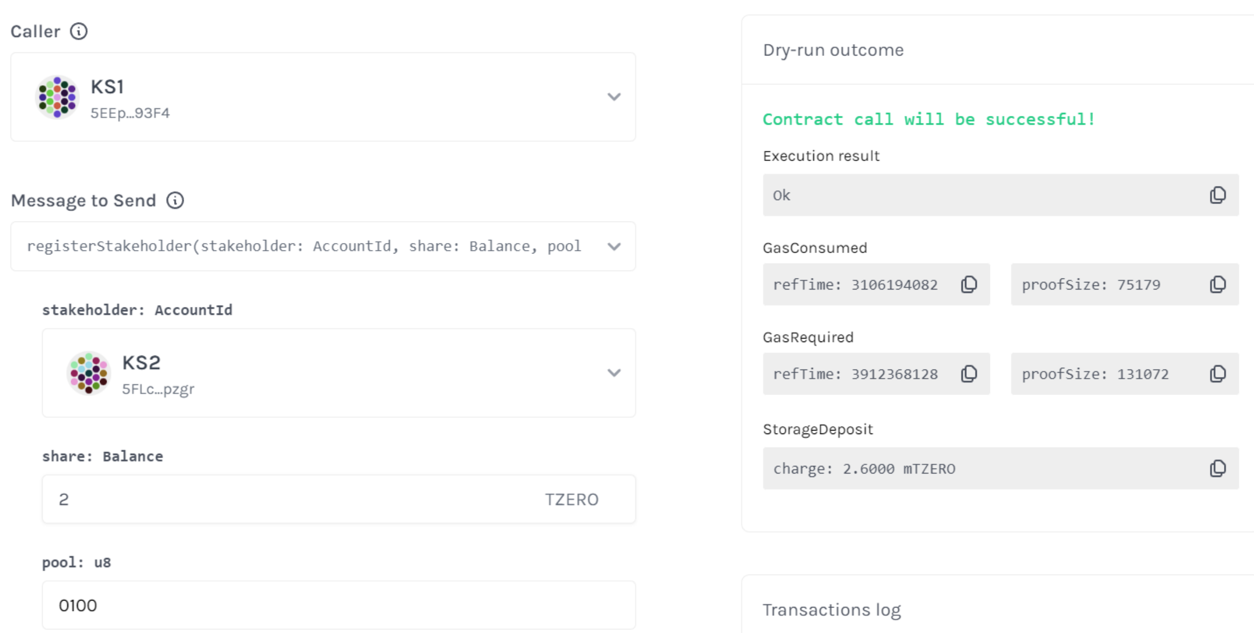
In `contract_ilocmvp`, a stakeholder can be registered to a non-existing pool. For example, if the function *register_stakeholder* is called with a *pool* argument greater than 12.

Impact

Indexing into *pool* with an out of bound index will make the contract panic. These panics may be used in a DDOS attack and possibly chain into other attacks.

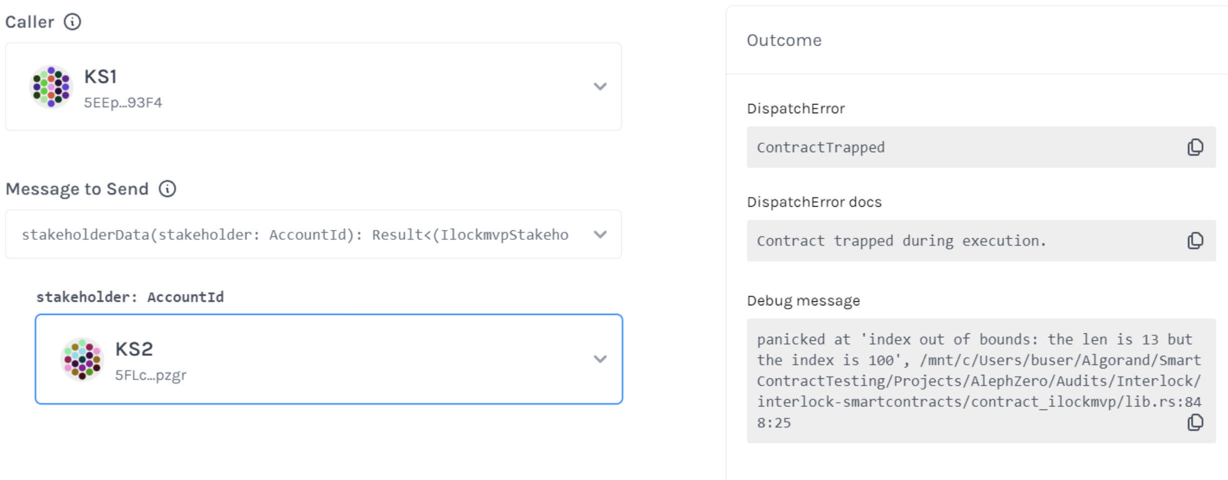
Evidence

The figure below demonstrates the registration of KS2 to the pool 100 which does not exist. This should not be possible.



KS2 registration to the nonexistent pool 100 is successful.

Then if a user wants to have information about KS2 shares and calls the function *stakeholder_data*, the contract will be trapped because it panicked for an out of bound exception as illustrated in the figure below.



Contract trapped after the call of stakeholder_data.

Affected Resource

- `contract_ilockmvp/lib.rs` (line 886, 1036, 1055)

Recommendation

We recommend checking that the index is always within the set of valid indices of the *pool* array. For example, this validation can be done in the *register_stakeholder* function on line 812. Alternatively, this can be implemented as an enum, as the number of pools is small and constant.

Reference

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_ilockmvp/lib.rs#L1030-L1044

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_ilockmvp/lib.rs#L1048-L1058

KS-INT- 14 – Missing argument validation with Interlock registration

Severity	MEDIUM
Status	RESOLVED

Impact	Likelihood	Difficulty
Low	High	Easy

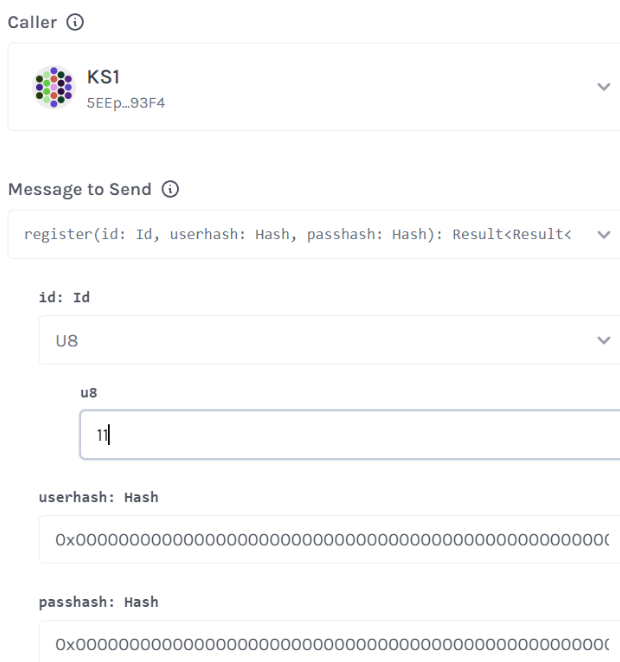
Description

Users can call the *register* function with arbitrary hashes as arguments. There is no additional validation other than the parameter type.

Impact

Registration of a fake user account can result in space used in place of a possible honest user. This may also be used in a DOS attack.

Evidence



KS1 can register his userhash and passhash himself and there is not verification about the validity of those information.

Affected Resource

- `contract_uanft/lib.rs` (line 666-728)

Recommendation

We recommend computing the hash directly from user id instead of let the user enter is own hash.

Reference

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_uanft/lib.rs#L666-L728

KS-INT-15 – Division by zero

Severity	LOW
Status	RESOLVED

Impact	Likelihood	Difficulty
Low	Low	Moderate

Description

The function *distribute_tokens* in `contract_ilockmvp` is vulnerable to a division by zero. If shareholder share is smaller than the *pool.vest*, then *payout* will be rounded down to 0. This leads to a division by zero when calculating payments two lines after.

Impact

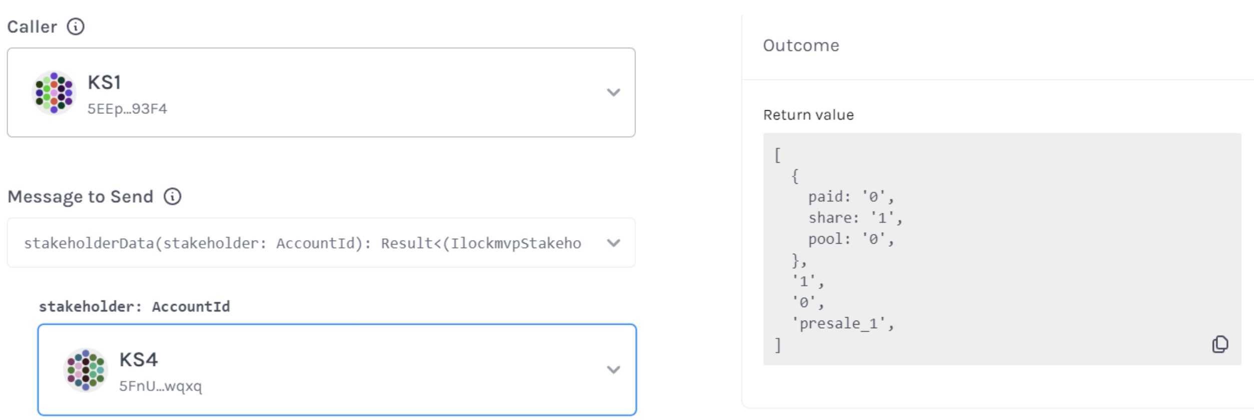
A division by zero in the function *distribute_tokens* will make the smart contract fail. The contract will panic and the transaction will revert.

Evidence

```
                // calculate the payout owed
                // ! no checked_div needed; pool.vests guaranteed to be nonzero
                let mut payout: Balance = this_stakeholder.share / pool.vests as Balance;

                // require that payout isn't repeatable for this month
                // ! no checked_div needed; this_stakeholder.share guaranteed to be nonzero
                let payments = this_stakeholder.paid / payout;
```

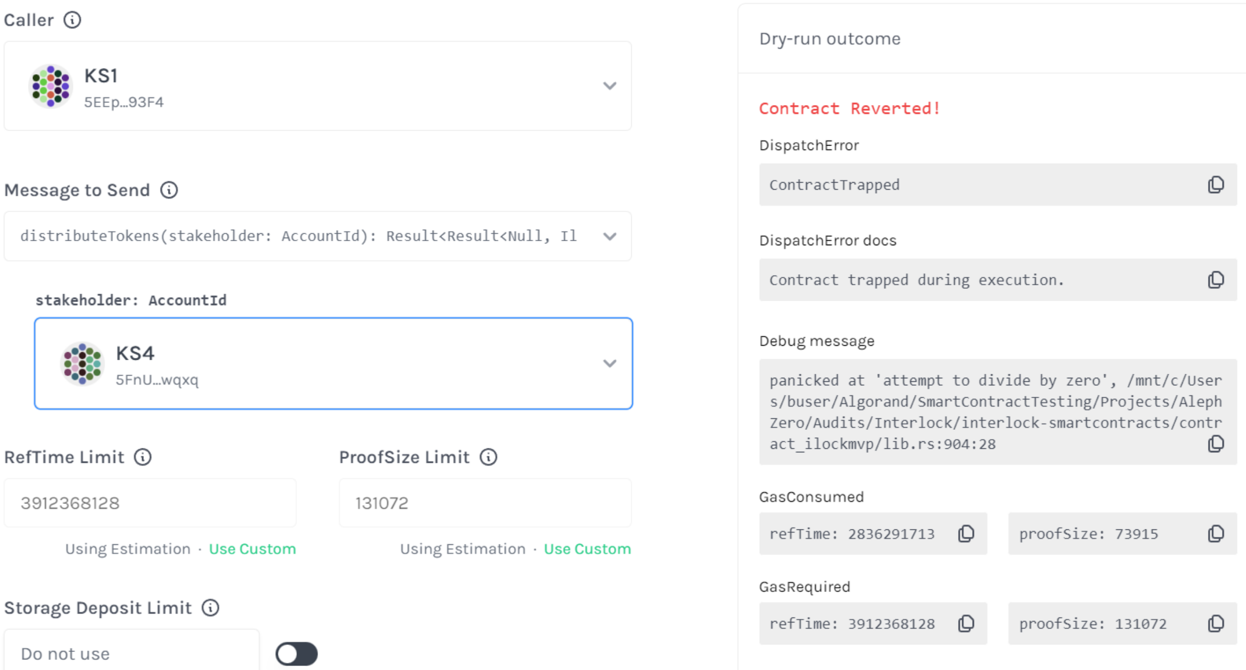
The variable payout will be equal to 0 if this_stakeholder.share<pool.vests and no check are performed before the next

The figure below demonstrates that KS4 owns only the minimum share in the pool 0.



KS4 share is equal to 1

Then when the contract owner KS1 wants to distribute the token to KS4, the contract will panic because of KS4's share being smaller than the pool vesting. This error is demonstrated in the figure below.



Contract panicking because of a division by 0.

Affected Resource

- `contract_ilockmvp/lib.rs` (line 889-904)

Recommendation

One may account for rounding error in the calculation of *payout* by imposing a nonzero minimum value on *payout*. We also suggest verifying before each division that divisor is not equal to zero.

Reference

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_ilockmvp/lib.rs#L898-L904

KS-INT-16 – Incorrect inequality check

Severity	LOW
Status	RESOLVED

Impact	Likelihood	Difficulty
Low	Low	Difficult

Description

If the rewards being allocated exceed the value in balances, it should return an error. However, the inequality does not cover the edge case where the balance is equal to the reward.

Impact

This means that Interlock users could see their reward refused even though it is a valid reward. This has only a limited impact on the security because the probability of happening is low.

Evidence

```
1074            // make sure reward not too large
1075            if self.balances[REWARDS as usize] < reward {
1076 +              return Err(OtherError::PaymentTooLarge)
1077            }
```

Edge case in function reward_interlocker not correctly performed

Affected Resource

- `contract_ilockmvp/lib.rs` (line 1075)

Recommendation

We recommend changing “<” to “<=” at line 1075 in `contract_ilockmvp/lib.rs`.

Reference

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_ilockmvp/lib.rs#L1075-L1077

KS-INT-17 – Token mint race attack possible

Severity	LOW
Status	RESOLVED

Impact	Likelihood	Difficulty
Medium	Low	Difficult

Description

The interlock smart contract could be vulnerable to a token mint race attack. This could happen when a user is calling the *self_mint* function and then the contract owners change the token price by calling the function *set_token_*price in order to make the user pay more than what they wanted.

Impact

The security impact is quite limited thanks to the speed of the Aleph Zero consensus and the fact that it would only make the user call fail.

Evidence

```
564                 if self.access.nft_psp22price > price {
565                     return Err(Error::Custom(
566                         format!("Current NFT price greater than agreed sale price of {:?}.", price)))
567                 }
568 
569                 // make sure mint recipient can afford the PSP22 token price
570                 let recipient_balance: Balance = self.app.token_instance.balance_of(minter);
571                 if recipient_balance < price {
572                     return Err(Error::Custom(
573                         format!("Minter cannot affort NFT at current price of {:?}.", price)))
574                 }
```

Snippet of the “self_mint” function

Affected Resource

- contract_uanft/lib.rs line 543-593

Recommendation

One solution may be to limit the amount of tokens transferred contingent on price thresholds (see slippage management in exchanges).

Reference

SWC-114 · Overview (swcregistry.io)
not-so-smart-contracts/race_condition at master · crytic/not-so-smart-contracts (github.com)
https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_uanft/lib.rs#L564-L574

KS-INT-18 – Unnecessary memory allocation

Severity	**INFORMATIONAL**
Status	INFORMATIONAL

Description

The function *stakeholder_data* returns a clone of a StakeholderData struct which requires additional heap memory. Although this is moved into the scope of the calling function and the original struct is dropped at the end of the function, unnecessary cloning should be avoided. Without cloning, the parameter cannot occur before the POOLS parameter which references it, because it would be moved out of scope before then.

Impact

Additional heap memory is allocated for a short period of time between lines 860 and 863. This function may be called repeatedly without a gas cost, so returned values may fill memory leading to a memory leak. This can lead to a denial of service (DOS), meaning the node cannot operate for a period of time.

The risk of a memory leak is unlikely in rust due to superior memory management, except for reference-counted pointer types such as *Rc* and *Arc*. However, unnecessary cloning should be avoided as a best practice.

Evidence

```
859            return (
860                this_stakeholder.clone(),
861                payremaining,
862                payamount,
863                POOLS[this_stakeholder.pool as usize].name.to_string(),
864            )
```

Snippets of the stakeholder_data function which could be subject to memory overflow

Affected Resource

- contract_ilockmvp/lib.rs line 860

Recommendation

Change the order of the return parameters so that *this_stakeholder* is after *POOLS* and do not clone it. Thus, *this_stakeholder* will be moved into the calling function after all references to it have occurred.

Reference

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_ilockmvp/lib.rs#L860

KS-INT-19 – Hardcoded values

Severity	INFORMATIONAL

Description

Some values, such as the pool numbers are hardcoded in `contract_ilockmvp/lib.rs`.

Impact

There is no direct security impact. If the values yield a runtime error, or can be exploited in any contract, it is impossible to change the value to recover.

Evidence

```
981             let poolnumber: u8 = match pool.as_str() {
982                 "PARTNERS"      => 8,
983                 "COMMUNITY"     => 9,
984                 "PUBLIC"        => 10,
985                 "PROCEEDS"      => 11,
986                 _ => return Err(OtherError::InvalidPool)
987             };
```

Pool numbers are hardcoded.

Affected Resource

- contract_ilockmvp/lib.rs line 981-987

Recommendation

We recommend avoiding the use of hardcoded values in the code. In this case an enum can be used instead.

Reference

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_ilockmvp/lib.rs#L980-L987

KS-INT-20 – Function defined for public and restricted access

Severity	INFORMATIONAL

Description

The function *set_multiple_attributes* and *set_base_uri* in *contract_uanft/lib.rs* has two definitions, one being publicly callable, and the other being reserved to the contract owner.

Impact

There is no security impact.

Evidence

```
#[ink(message)]
#[modifiers(only_owner)]
fn set_multiple_attributes(
    &mut self,
    token_id: Id,
    metadata: Vec<(String, String)>,
) -> Result<(), Error> {

    if token_id == Id::U64(0){
        return Err(Error::InvalidInput)
    }
    if self.is_locked_nft(token_id.clone()) {
        return Err(Error::Custom(
                String::from("Token is locked")));
    }
    for (attribute, value) in &metadata {
        self.add_attribute_name(&attribute.clone().into_bytes());
        self._set_attribute(token_id.clone(), attribute.clone().into_bytes(), value.clone().into_bytes());
    }

    Ok(())
}
```

The figure above demonstrates the definition of the function set_multiple_attribute, which is restricted to the contract owner only. The figure below demonstrates the same function, but publicly callable.

```
#[ink(message)]
fn set_multiple_attributes(&mut self, token_id: Id, metadata: Vec<(String, String)>) -> Result<(), Error>;
```

```
966    #[ink(message)]
967    #[modifiers(only_owner)]
968    fn set_base_uri(
969        &mut self,
970        uri: String
971    ) -> Result<(), Error> {
972
973        self._set_attribute(
974            Id::U8(0),
975            String::from("baseURI").into_bytes(),
976            uri.into_bytes(),
977        );
978        Ok(())
979    }
```

Affected Resource

- `contract_uanft/lib.rs` line 385-401, line 984-1005,

Recommendation

We suggest keeping to give same access to both definition of these functions or remove the publicly callable one from the smart contract.

Reference

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_uanft/lib.rs#L385-L401

https://github.com/interlock-network/interlock-smartcontracts/blob/b217e89acca3af1011e7ba84b5ec8a4a5768eeb6/contract_uanft/lib.rs#L966-L979

KS-INT-21 – Incomplete Test Coverage

Severity	INFORMATIONAL

Description

We examined test coverage in three categories: unit tests, integration tests, and end-to-end tests. It is important to implement sad paths in each case to validate access control and security measures remain valid. Unit tests help to validate function arguments, integration tests may prevent attacks which require multiple function calls, such as reentrancy. End-to-end tests help to ensure that some attacks will be prevented.

However, we did not find sufficient test coverage, which is a recommended best practice for security. This is necessary to ensure that functions maintain security invariants across the codebase. This is especially important after code updates, to ensure that any changes can pass the same security tests.

Impact

A series of vulnerabilities may present itself over the course of updates. Specifically, security-related tests related to access control and input validation may lapse in future updates which can result in major exploits.

Affected Resource

The entire directory `interlock-network/interlock-smartcontracts` is affected.

Recommendation

Improve test coverage with additional unit tests, integration tests, and end-to-end tests.

METHODOLOGY

During this source code review, the Kudelski Security Services team reviewed code within the project within 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
- Authorization and Access Control
- Auditing and Logging
- Injection and Tampering
- Configuration Issues
- Logic Flaws
- Cryptography

These categories incorporate common vulnerabilities such as the OWASP Top 10

Tools

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

- Aleph Zero testnet
- Substrate
- Cargo contract package manager
- Semgrep

Vulnerability Scoring Systems

Kudelski Security utilizes a vulnerability scoring system based on impact of the vulnerability, likelihood of an attack against the vulnerability, and the difficulty of executing an attack against the vulnerability based on a high, medium, and low rating system

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 affect 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, 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.

Difficulty

Difficulty is measured according to the ease of exploit by an attacker based on availability of readily available exploits, knowledge of the system, and complexity of attack. It should be noted that a LOW difficulty results in a HIGHER severity.

Easy:
The vulnerability is easy to exploit or has readily available techniques for exploit

Moderate:

The vulnerability is partially defended against, difficult to exploit, or requires a skilled attacker to exploit.

Difficult:
The vulnerability is difficult to exploit and requires advanced knowledge from a skilled attacker to write an exploit

Severity
Severity is the overall score of the weakness or vulnerability as it is measured from Impact, Likelihood, and Difficulty

KUDELSKI SECURITY CONTACTS

NAME	POSITION	CONTACT INFORMATION
Luca Dolfi	Blockchain engineer	luca.dolfi@kudelskisecurity.com
Rez Khan	Blockchain security engineer	rez.khan@kudelskisecurity.com
Maxime Buser	Security engineer/ Project Manager	Maxime.buser@kudelskisecurity.com