Secure Code Review

Findings and Recommendations Report Presented to:

AZERO.ID

July 28, 2023
Version: 1.3.1

Presented by:

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

FOR PUBLIC RELEASE

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

EXECUTIVE SUMMARY

Overview

AZERO.ID engaged Kudelski Security to perform a secure code assessment.

The assessment was conducted remotely by the Kudelski Security Team.
Testing took place on April 26, 2023 - May 22, 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.

- Lack of input validation for admin,
- Overpaying for the user was possible.

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

- Smart contracts are managed by a centralized authority, which can be different for each smart contract. AZERO.ID's team were well aware of this and planned to use a multisig account to reduce the risk of having corrupted admins.

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

- The code was well written,
- Security was a part of AZERO.ID' reflection during the implementation, which is demonstrated by the fact we did not find any findings with a High severity level.
- Tests were also provided as part of the project.
- Finally, AZERO.ID' team were extremely responsive, and always available to have helpful technical discussions.

While our comprehensive smart contract audit has highlighted security vulnerabilities into AZERO.ID 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 AZERO.ID. 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/azero-id/contracts/commit/d3edd6f20c0388e572929243f2e0b4e1f6f42fb7
  - Subfolder:
    - `anzs_fee_calculator`
    - `anzs_registry`
    - `anzs_merkle_verifier`
    - `anzs_name_checker`
    - `anzs_router`
  - Written with ink! version 4.0.1

AZERO.ID
anzs_fee_calculator/ ├── lib.rs ├── Cargo.toml anzs_merkle_verifier/ ├── lib.rs ├── Cargo.toml anzs_name_checker/ ├── lib.rs ├── Cargo.toml anzs_registry/ ├── lib.rs ├── address_dict.rs ├── Cargo.toml anzs_router/ ├── lib.rs ├── Cargo.toml

Table 1: Scope

A further round of review was performed by Kudelski Security, May 31, 2023, on remediations with the code available at:

- https://github.com/azero-id/contracts/pull/98

TECHNICAL ANALYSIS & FINDINGS

During the Secure Code Review, we discovered 9 findings with low severity.

The following chart displays the findings by severity.

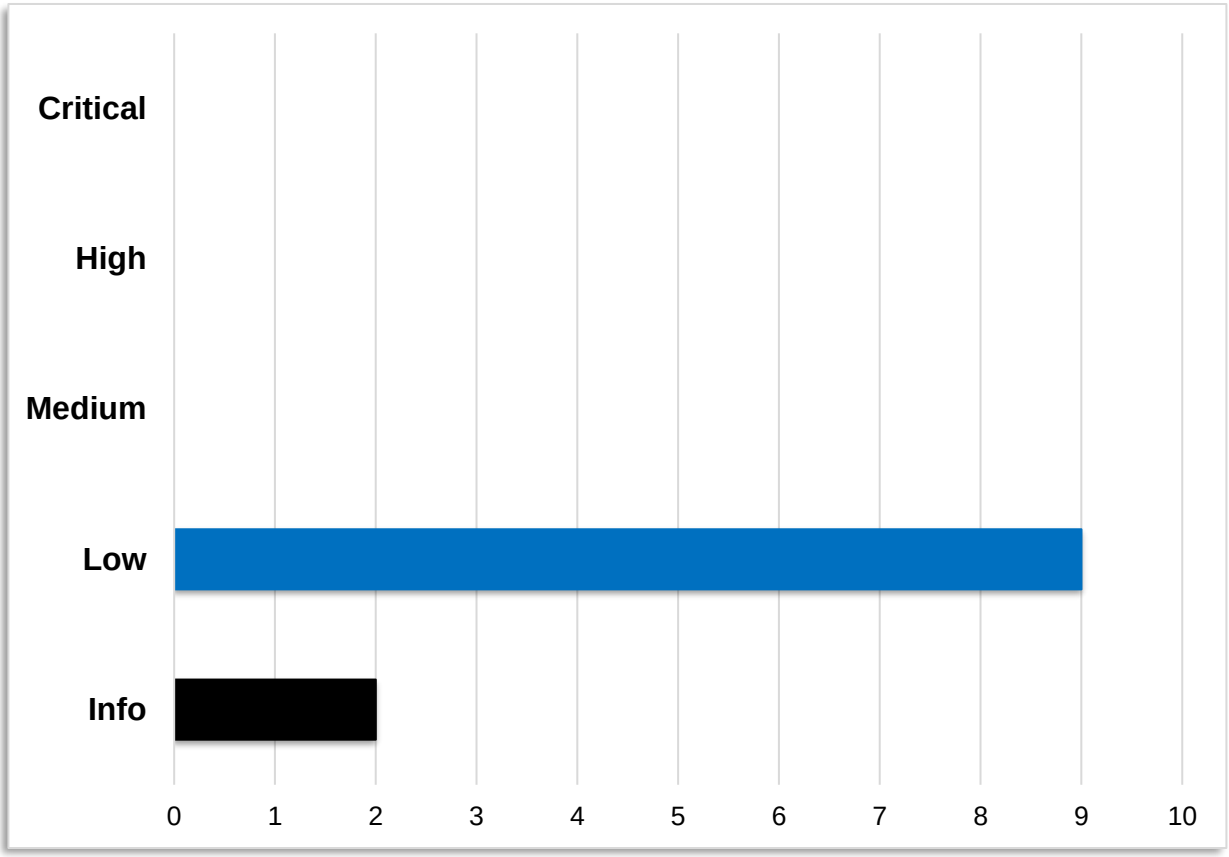


Figure 1: Findings by Severity

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-AZID-01	Low	Admin-led contract	Acknowledged
KS-AZID-02	Low	Zero Address Verification	Resolved
KS-AZID-03	Low	Absence of Pause Function	Acknowledged
KS-AZID-04	Low	Risk of overspending for the users	Resolved
KS-AZID-05	Low	Lack of inputs validation	Resolved
KS-AZID-06	Low	Invisible Unicode characters accepted	Resolved
KS-AZID-07	Low	Risk of phishing domains	Resolved
KS-AZID-08	Low	Risk of Overflow/Underflow	Resolved
KS-AZID-09	Low	Unwanted registration/Role possible	Resolved
KS-AZID-10	Informational	Modification of Storage before input validation	Informational
KS-AZID-11	Informational	Duplication of code	Resolved

Table 2: Findings Overview

KS-AZID-01 – Admin-led contract

Severity	LOW
Status	ACKNOWLEDGED

Impact	Likelihood	Difficulty
Low	Low	Difficult

Description
The account identified as `admin` in any of the AZERO.ID contracts reserves the power to withdraw tokens, delegate the admin function or update the contract logic entirely. These features increase the power of the admin which makes it crucial to protect the admin the correct way. Of course, compromising the admin account is difficult, but Kudelski Security team wants to highlight that there is also an internal threat factor. This factor can not necessarily be a malicious user but also errors during admin operations or compromised credentials from mismanaged key material. For example, in this case of the Axie Infinity hack (see Reference below) a compromised credential was used to gain admin control and steal funds.

However, it is **important** to highlight that the Kudelski Security team engaged in conversations about this risk with AZERO.ID team during the audit, and they confirmed that they will use multisig accounts when it comes to admin operations which greatly limit the risk of the above-mentioned scenario to happen.

Impact
A malicious admin can drain funds from the contract, modify the registry database and update the contract logic introducing malicious code.

Evidence

lib.rs

```
918         /// Upgrade contract code
919         #[ink(message)]
920         pub fn upgrade_contract(&mut self, code_hash: [u8; 32]) -> Result<()> {
921             self.ensure_admin()?;
922
923             ink::env::set_code_hash(&code_hash).unwrap_or_else(|err| {
924                 panic!(
925                     "Failed to `set_code_hash` to {:?} due to {:?}",
926                     code_hash, err
927                 )
928             });
929             ink::env::debug_println!("Switched code hash to {:?}.", code_hash);
930
931             Ok(())
932         }
```

Snipped

Figure 2:The upgrade_contract function allows an admin to completely substitute the contract code with new one.

Affected Resource

- `anzs_fee_calculator`
- `anzs_name_checker`
- `anzs_merkle_verifier`
- `anzs_registry`
- `anzs_router`

Recommendation

The Kudelski Security team recommend following best practices in setting up and utilizing a multi-signature or threshold signature account for the admin role. Control over this account should be distributed among trusted parties and none of the parties involved should have a majority of secret shares of the admin account at any time during creation, utilization and at rest.

Reference

- https://www.certik.com/resources/blog/What-is-centralization-risk
- https://wiki.polkadot.network/docs/learn-account-multisig
- https://blog.mollywhite.net/axie-hack/

KS-AZID-02 – Zero Address Verification

Severity	LOW
Status	RESOLVED

Impact	Likelihood	Difficulty
High	Low	Difficult

Description
AZERO.ID contracts do not perform the Zero Address Verification. The Aleph Zero "Zero Address" is an existing address on the network which has a publicly known secret seed. Errors could lead to having token assigned to the Zero Address by a user. The Zero Address could become the admin of smart contracts when the `set_admin` function has been called using the Zero Address as argument. In this case the issue would have high impact on the AZERO.ID project, compromising its core assets. There are two possibilities to have such event happen:

1. The current admin is making an error by using the Zero Address as argument.
2. The admin account is compromised. This would mean that an external attacker successfully took control of the secret key of the admin account.

Impact
Once the Zero Address receives any coins/token then those tokens can be considered lost, as they can be stolen by any users. In a worst-case scenario, the Zero Address could become an admin of a smart contract and update the contract to anything. As an external attacker needs to compromise the admin account to critically impact the AZERO.ID project, we consider this as a low likelihood and difficult to achieve attack. For these reasons the Kudelski Security's team has set the severity of the finding to LOW, despite having a high impact.

Evidence
To prove this finding, we used the cargo package manager to set the admin of one of the contracts to the Zero Address.
Command executed:

```
cargo contract call --suri "$SEEDKS3" --url "$URL" --contract "$CONTRACT_3" \\
--message set_admin \\
--args 0x0000000000000000000000000000000000000000000000000000000000000000 -x
```

It accepts the change admin to the Zero Address as the Figure below demonstrate it.

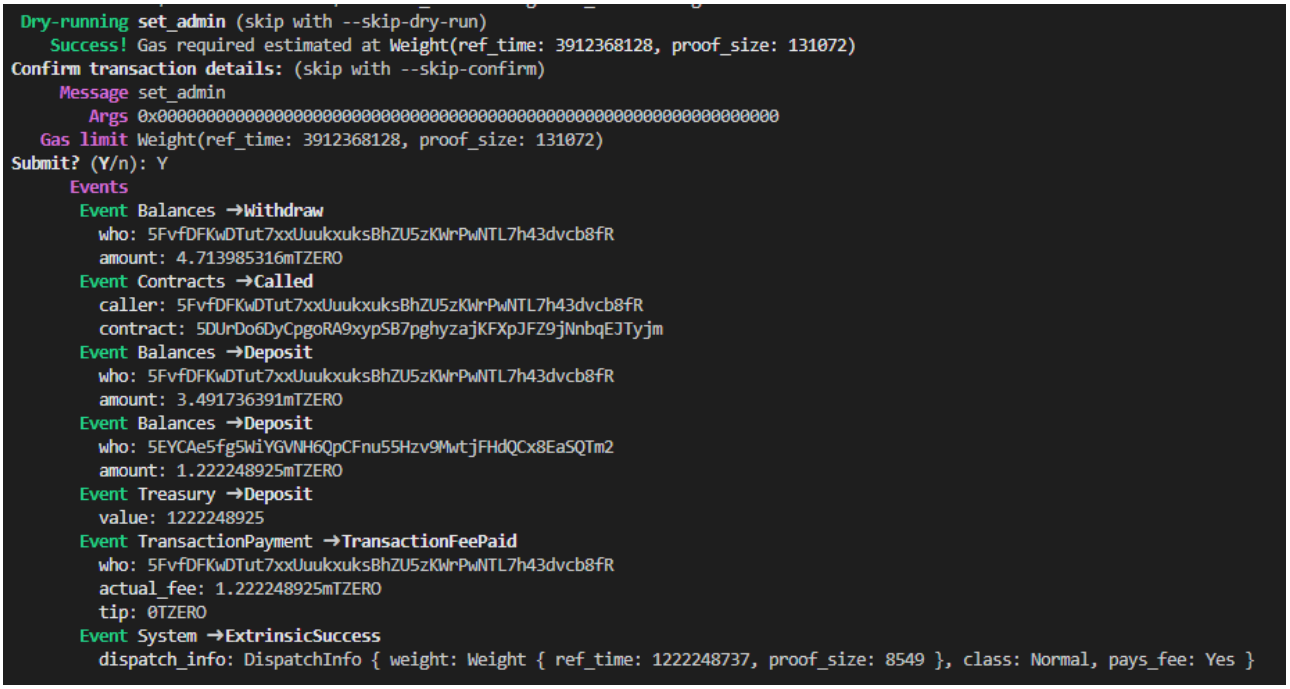


Figure 3: Output of after changing the Admin to the Zero Address

After the completion of this call, *CONTRACT_3*, which is in our case the `anzs_merkle_verifier` smart contract, the admin is set to the Zero Address which means that everyone could call admin-only function.

Affected Resource

- `anzs_fee_calculator`
- `anzs_name_checker`
- `anzs_merkle_verifier`
- `anzs_registry`
- `anzs_router`

Recommendation

The Kudelski Security team recommend the implementation of the Zero Address checks in the code particularly for critical function such as `set_admin` (all smart contracts) or withdraw (`anzs_registry`). Additionally, Kudelski Security team recommends a process for updating and deploying the smart contracts that can check and ensure the zero address is not set prior to committing the code to the network.

Reference

- https://wiki.polkadot.network/docs/learn-account-multisig

KS-AZID-03 – Absence of Pause Function

Severity	LOW
Status	ACKOWNLEDGED

Impact	Likelihood	Difficulty
Low	Low	Difficult

Description
All five smart contracts are controlled by a central authority called admin The admin has different responsibilities such has defining prices, setting Unicode code range, or updating smart contracts. This is a non-exhaustive list of responsibility. Although of acceptable, there is a level of centralization to AZERO.ID' smart contract. There is currently no admin-controlled pause function in any of the five smart contracts.

Impact
The absence of a pause function prevents the admin to limit the damage in case of attack or vulnerabilities discovery. For example, the attack on the Nomad Bridge (see Reference below) where the attack was replayed multiple times to drain all the founds. A pause function, once the attack has been discovered, would allow to stop, and limit the damage inflicted by this attack.

Evidence
N/A

Affected Resource

- `anzs_fee_calculator`
- `anzs_name_checker`
- `anzs_merkle_verifier`
- `anzs_registry`
- `anzs_router`

Recommendation
The Kudelski Security team recommends the implementation of a pause function which can be only called by the admin. We particularly recommend it for the smart contract `anzs_registry` which the central element of the AZERO.ID project.

Reference

- https://www.halborn.com/blog/post/the-nomad-bridge-hack-a-deeper-dive
- https://sm4rty.medium.com/nomad-bridges-200-million-exploit-postmortem-9d1cd83db1f7

KS-AZID-04 – Risk of overspending for the users

Severity	LOW
Status	RESOLVED

Impact	Likelihood	Difficulty
Low	Low	Easy

Description
When a user registers a user a domain, they need to pay a fee which must be at least of a certain price. However, there is no protection for user of overpaying a domain name. In particular, the function `register_on_behalf_of` in `azns_registry` checks only if `transferred < price`, so any value equal but also greater than the domain price will be accepted.

Impact
There is a risk of overspending for the user, meaning that without additional checks the user might end up paying more that the value of a domain.

Evidence

Figure 4: Code snippet showing check on price in azns_registry/lib.rs

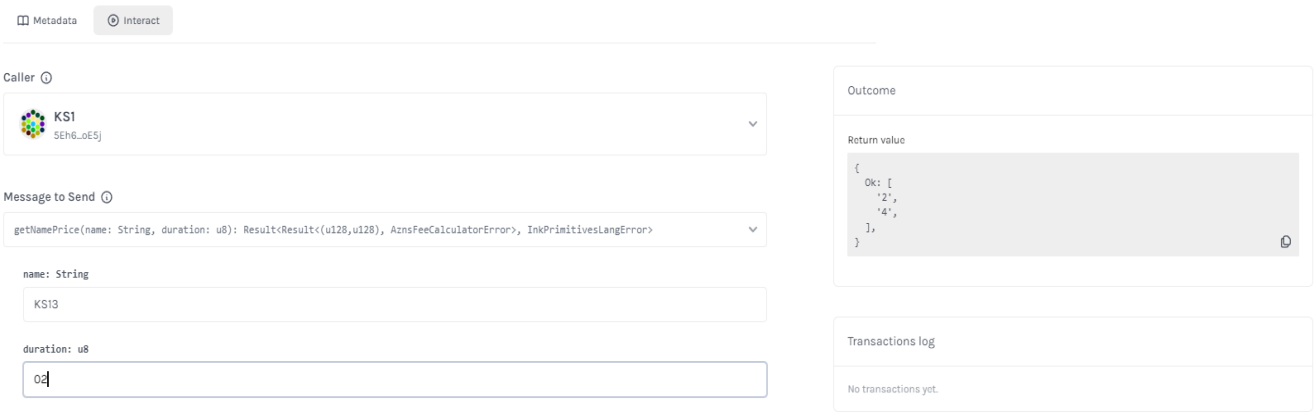


Figure 5: Proof of overspending on the aleph zero testnet

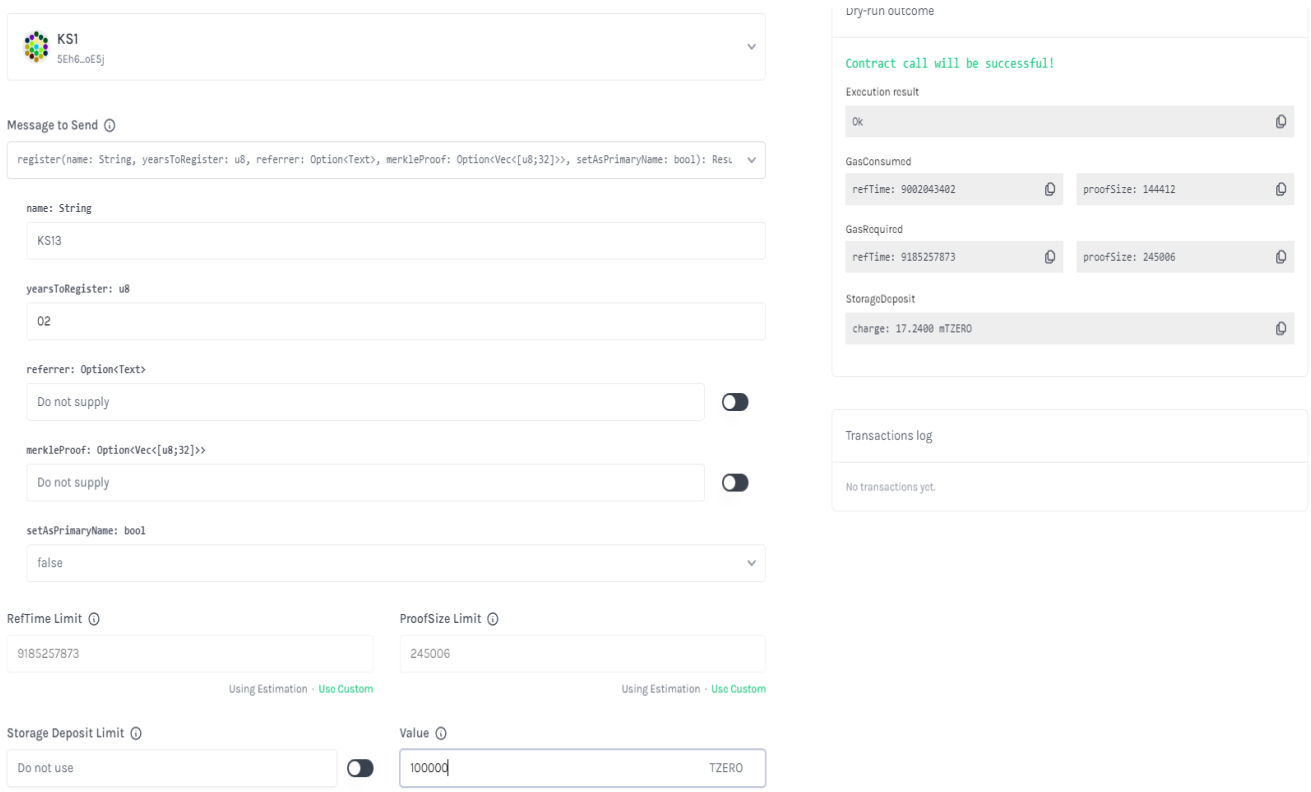


Figure 6: Proof of overspending with large amount of TZERO on aleph zero testnet

Affected Resource

- `anzs_registry/lib.rs` line 360

Recommendation
The Kudelski Security team recommends verifying the correctness of the price paid and to set a fixed maximum value that can be transferred to the contract when registering a new domain to protect the user from overspending.

Reference
N/A

KS-AZID-05 – Lack of inputs validation

Severity	LOW
Status	RESOLVED

Impact	Likelihood	Difficulty
Low	Low	High

Description
AZERO.ID smart contracts lack input verification particularly from admin-only function. This means that errors made by the contract's admin could results into contracts' logic break or even contract failure.
For example, in the smart contract `anzs_name_checker`, the Unicode range accepts all `u32` without any checks, this means that the upper bound of this range could be set to a smaller number than the lower bound.

Impact
The impact of this finding is that it could break the logic of the contract. In the example mentioned above, for example when setting up the Unicode range in `anzs_name_checker` with an upper bound smaller than the lower bound, users will not be able to register new domain names. This would prevent the AZERO.ID project from any earnings.

Evidence

lib.rs

```
133         #[ink(message)]
134         pub fn set_allowed_unicode_ranges(&mut self, new_ranges: Vec<UnicodeRange>) -> Result<()> {
135             self.ensure_admin()?;
136             self.allowed_unicode_ranges = new_ranges;
137             Ok(())
138         }
```

Snipped

Figure 7: Example of lack of input verification

Affected Resource

- `anzs_fee_calculator`
- `anzs_name_checker`
- `anzs_merkle_verifier`
- `anzs_registry`
- `anzs_router`

Recommendation
Addition of inputs verification for admin function, to protect against mistake done by the admin

Reference
N/A

KS-AZID-06 – Invisible Unicode characters accepted

Severity	LOW
Status	RESOLVED

Impact	Likelihood	Difficulty
Low	Medium	High

Description

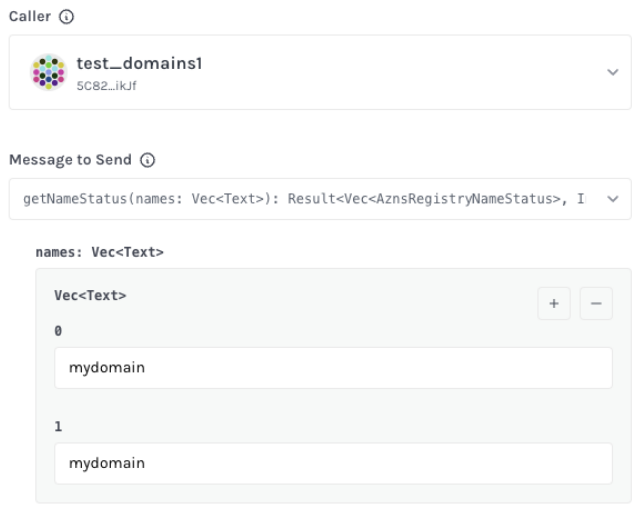
The Unicode Standard includes many control characters that have not visual representation, but instead serve to control the interpretation and display of other characters. For example, the zero-width space (`U+200B`) or the zero-width non-joiner (`U+200C`).
The contract `azns_name_checker` does not check if the individual characters in a name are correct, printable UTF-8 characters and this allows non-printable characters to be included in the domain name. As these characters are invisible, it would be impossible for a user to recognize when a domain name includes these characters.

As an important note, the Kudelski Security team was made aware that AZERO.ID intends to exclude these characters from the *allowed Unicode ranges* in their deployment pipeline, which remains out of scope of this audit.

Impact

A malicious user has the capability to fabricate a domain name that closely resembles a legitimate user's domain name, incorporating a variety of concealed characters within it. This could potentially deceive unsuspecting victims into erroneously associating the forged domain name with the authentic one.

Evidence



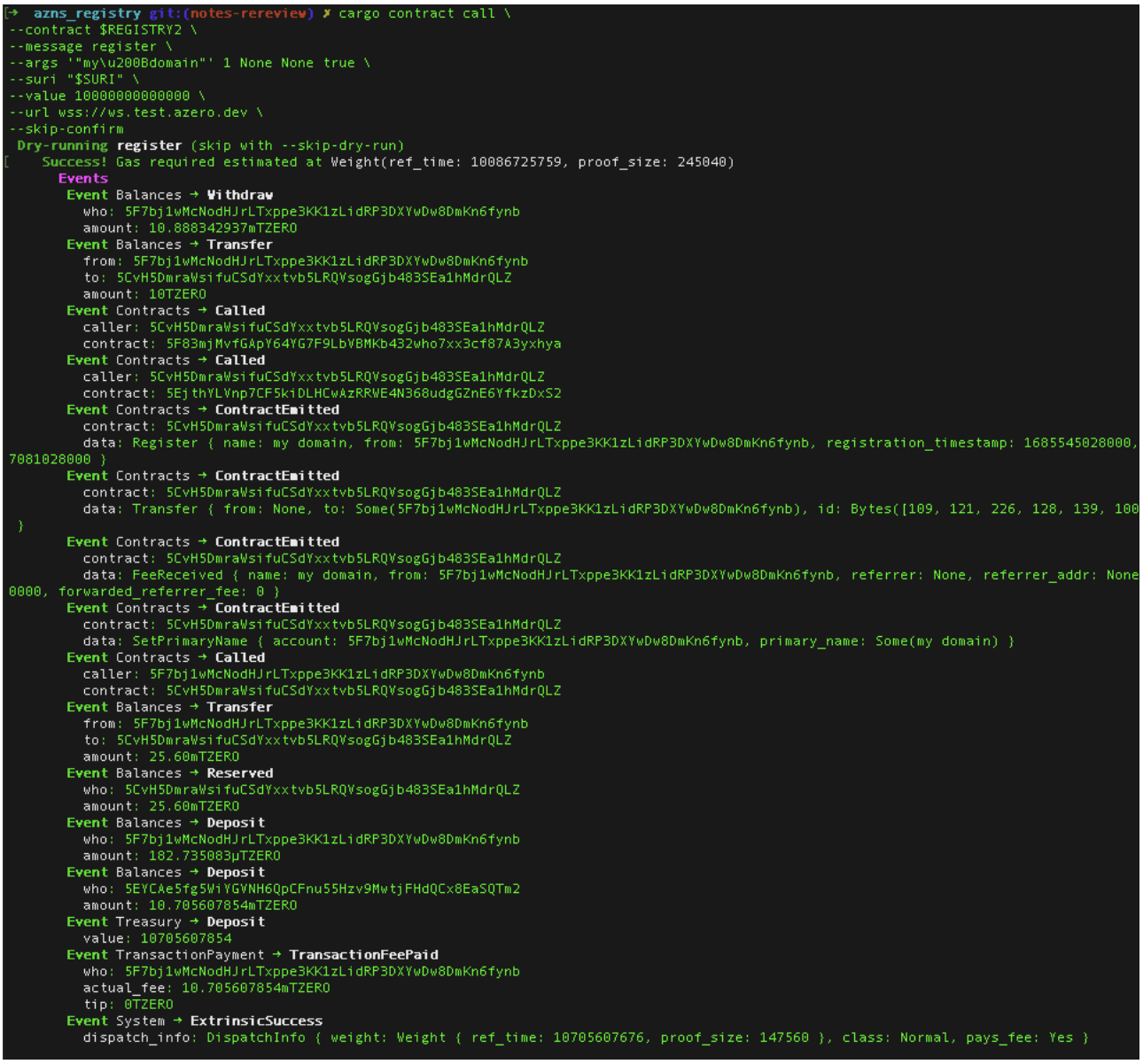


Figure 8:Two seemingly identical domains (one containing a zero-width character) registered to the same registry

Affected Resource

- `azns_name_checker/lib.rs` line 65

Recommendation

The Kudelski Security team recommends to thoroughly inspect the domain name to remove any non-printable characters from the user input, preventing the exploitation of such deceptive practices.

Reference

- https://en.wikipedia.org/wiki/Unicode_control_characters
- https://www.unicode.org/versions/Unicode15.0.0/ch05.pdf

KS-AZID-07 – Risk of phishing domains

Severity	LOW
Status	RESOLVED

Impact	Likelihood	Difficulty
Medium	Low	Moderate

Description
When allowing multiple alphabets, the user is exposed to the risk of confusing domain names that use similar characters. For example, the domain example.azero uses both Latin (blue) and Cyrillic (red) characters.
The Kudelsksi Security's team has discussed this issue with AZERO.ID, and they are already aware of this. In the code it is already present a functionality to allow subsets of Unicode in terms of "Unicode ranges" to exclude illegal characters.

Impact
Given Unicode ranges large enough, malicious actors can register ambiguous domain names with the goal to perform phishing attacks.

Evidence

Figure 9: The function to check allowed characters in the NameChecker contract allows for any Unicode range to be valid for each character

Affected Resource

- `azns_name_checker/lib.rs` line 98

Recommendation
The Kudelski Security team recommends following best practices in dealing with domain names over the Unicode space and align with the DNS rules, in particular:

- allow only one alphabet at the time for any domain name
- disallow Unicode control characters and non-printable characters (See KS-AZID-05).

Reference

- https://www.xudongz.com/blog/2017/idn-phishing
- https://www.icann.org/resources/pages/idn-guidelines-2011-09-02-en

KS-AZID-08 – Risk of Overflow/Underflow

Severity	LOW
Status	RESOLVED

Impact	Likelihood	Difficulty
High	Low	Difficult

Description
The protection against overflow/underflow has been turned off for the `anzs_registry` smart contract which is the central element of the AZERO.ID project. A user who registers a domain needs to pay a fee which correspond to a base price to which is added a premium. An attacker could register its domain for a period such that *base_price + premium = u128::MAX+1.* This would allow the attacker to register its domain for free.
This issue can also occur when the admin set a price to high.

Impact
It allows attacker to register domain for free. However, because the price and the premium are given in *u128*, the attacker would require for the attacker having an unrealistic amount of AZEROs to be able to perform the attack. Therefore, this attack is unlikely to happen.

Evidence

lib.rs

```
354             let (base_price, premium, discount, referrer_addr) =
355                 self.get_name_price(name.clone(), recipient, years_to_register, referrer.clone())?;
356             let price = base_price + premium - discount;
```

Snipped

Figure 10: Overflow Risk

Affected Resource

- `anzs_registry` line 355

Recommendation
We recommend enabling the overflow check in `anzs_registry/Cargo.tolm`, in order to avoid the risk of overflow.

Reference
N/A

KS-AZID-09 – Unwanted registration/role possible

Severity	LOW
Status	RESOLVED

Impact	Likelihood	Difficulty
Low	Low	Difficult

Description
In AZERO.ID smart contracts, it is possible to transfer the ownership/role to any users. For example, an attacker who owns domains could transfer the controlling and resolving addresses to a user who does not want to be the controller. It is important to highlight that an attacker could also directly use the `register_on_behalf_of` function to assign a domain to honest user.

Impact
Honest users own/control domains that they do not want to. This is the only impact as it does not prevent an honest user to register domain on their owns. Indeed, there is **no** risk of DoS for the honest users which justify the severity level of this findings.

Evidence

lib.rs

```
544         #[ink(message)]
545         pub fn set_controller(&mut self, name: String, new_controller: AccountId) -> Result<()> {
546             /* Ensure caller is either controller or owner */
547             let caller = Self::env().caller();
548             self.ensure_controller(&caller, &name)?;
549
550             let mut address_dict = self.get_address_dict_ref(&name)?;
551             let old_controller = address_dict.controller;
552             address_dict.set_controller(new_controller);
553             self.name_to_address_dict.insert(&name, &address_dict);
554
555             /* Remove the name from the old controller */
556             self.remove_name_from_controller(&caller, &name);
557
558             /* Add the name to the new controller */
559             self.add_name_to_controller(&new_controller, &name);
560
561             self.env().emit_event(SetController {
562                 name,
563                 from: caller,
564                 old_controller: Some(old_controller),
565                 new_controller,
566             });
567
568             Ok(())
569         }
```

Snipped

Figure 11: Set Controller can be assigned to anyone

Affected Resource

- `anzs_registry` lines 317-385, 545-570, 519-542

Recommendation

The Kudelski Security team suggests adding a possibility for users to rejects any domains they have been assigned too.

Reference

N/A

KS-AZID-10 – Modification of Storage before input validation

Severity	INFORMATIONAL

Description
The function `update_records` in the `anzs_registry` smart contract is modifying the storage before verifying the validity of the input. A user can modify the records for his domains and records have a maximum size. When a user is adding new elements to the records, the function first modifies the storage allocated to the records and only after verifies that the records do not exceed the maximum size allowed.

Impact
An adversary could attempt to register large records in an attempt to break the storage logic of the contract.

Evidence

lib.rs

```
600             let updated_records: Vec<(String, String)> = data.into_iter().collect();
601             self.records.insert(&name, &updated_records);
602
603             self.ensure_records_under_limit(&name)?;
604
605             self.env().emit_event(RecordsUpdated { name, from: caller });
606             Ok(())
607         }
```

Snipped

Figure 12: Snippet of the function update records modifying the storage before checking the validity of the input.

Affected Resource

- `anzs_registry` (line 601)

Recommendation
The Kudelski Security team suggests of checking the size of new updates such that they will increase the records size above the maximum limit. Another option could be a maximum size of input to limit the risk of arming the storage correctness of the contract.

Reference
N/A

KS-AZID-11 – Duplication of code

Severity	INFORMATIONAL

Description
Function `ensure_admin` has been duplicated in all five smart contracts.

Impact
N/A

Evidence

lib.rs

```
941         fn ensure_admin(&self) -> Result<()> {
942             if self.admin != self.env().caller() {
943                 Err(Error::NotAdmin)
944             } else {
945                 Ok(())
946             }
947         }
```

Snipped

lib.rs

```
82          fn ensure_admin(&self) -> Result<(), Error> {
83              match self.env().caller() == self.admin {
84                  true => Ok(()),
85                  false => Err(Error::NotAdmin),
86              }
87          }
```

Snipped

Figure 13: ensure_admin function in two different smart contracts

Affected Resource

- `anzs_fee_calculator`
- `anzs_name_checker`
- `anzs_merkle_verifier`
- `anzs_registry`
- `anzs_router`

Recommendation
We recommend avoiding the duplication of code.

Reference
N/A

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