CAIRO SECURITY CLAN

STARKNET ID

SECURITY ASSESSMENT REPORT

MAY 2024

Prepared for

STARKNET ID

Contents

1 About Cairo Security Clan

Cairo Security Clan is a leading force in the realm of blockchain security, dedicated to fortifying the foundations of the digital age. As pioneers in the field, we specialize in conducting meticulous smart contract security audits, ensuring the integrity and reliability of decentralized applications built on blockchain technology.

At Cairo Security Clan, we boast a multidisciplinary team of seasoned professionals proficient in blockchain security, cryptography, and software engineering. With a firm commitment to excellence, our experts delve into every aspect of the Web3 ecosystem, from foundational layer protocols to application-layer development. Our comprehensive suite of services encompasses smart contract audits, formal verification, and real-time monitoring, offering unparalleled protection against potential vulnerabilities.

Our team comprises industry veterans and scholars with extensive academic backgrounds and practical experience. Armed with advanced methodologies and cutting-edge tools, we scrutinize and analyze complex smart contracts with precision and rigor. Our track record speaks volumes, with a plethora of published research papers and citations, demonstrating our unwavering dedication to advancing the field of blockchain security.

At Cairo Security Clan, we prioritize collaboration and transparency, fostering meaningful partnerships with our clients. We believe in a customer-oriented approach, engaging stakeholders at every stage of the auditing process. By maintaining open lines of communication and soliciting client feedback, we ensure that our solutions are tailored to meet the unique needs and objectives of each project.

Beyond our core services, Cairo Security Clan is committed to driving innovation and shaping the future of blockchain technology. As active contributors to the ecosystem, we participate in the development of emerging technologies such as Starknet, leveraging our expertise to build robust infrastructure and tools. Through strategic guidance and support, we empower our partners to navigate the complexities of the blockchain landscape with confidence and clarity.

In summary, Cairo Security Clan stands at the forefront of blockchain security, blending technical prowess with a client-centric ethos to deliver unparalleled protection and peace of mind in an ever-evolving digital landscape. Join us in safeguarding the future of decentralized finance and digital assets with confidence and conviction.

2 Disclaimer

Disclaimer Limitations of this Audit:

This report is based solely on the materials and documentation provided by you to Cairo Security Clan for the specific purpose of conducting the security review outlined in the Summary of Audit and Scoped Files. The findings presented here may not be exhaustive and may not identify all potential vulnerabilities. Cairo Security Clan provides this review and report on an "as-is" and "as-available" basis. You acknowledge that your use of this report, including any associated services, products, protocols, platforms, content, and materials, occurs entirely at your own risk.

Inherent Risks of Blockchain Technology:

Blockchain technology remains in its developmental stage and is inherently susceptible to unknown risks and vulnerabilities. This review is specifically focused on the smart contract code and does not extend to the compiler layer, programming language elements beyond the reviewed code, or other potential security risks outside the code itself.

Report Purpose and Reliance:

This report should not be construed as an endorsement of any specific project or team, nor does it guarantee the absolute security of the audited smart contracts. No third party should rely on this report for any purpose, including making investment or purchasing decisions.

Liability Disclaimer:

To the fullest extent permitted by law, Cairo Security Clan disclaims all liability associated with this report, its contents, and any related services and products arising from your use. This includes, but is not limited to, implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Third-Party Products and Services:

Cairo Security Clan does not warrant, endorse, guarantee, or assume responsibility for any products or services advertised by third parties within this report, nor for any open-source or third-party software, code, libraries, materials, or information linked to, referenced by, or accessible through this report, its content, and related services and products. This includes any hyperlinked websites, websites or applications appearing on advertisements, and Cairo Security Clan will not be responsible for monitoring any transactions between you and third-party providers. It is recommended that you exercise due diligence and caution when considering any third-party products or services, just as you would with any purchase or service through any medium.

Disclaimer of Advice:

FOR THE AVOIDANCE OF DOUBT, THIS REPORT, ITS CONTENT, ACCESS, AND/OR USE, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHOULD NOT BE CONSIDERED OR RELIED UPON AS FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER PROFESSIONAL ADVICE.

3 Executive Summary

This document presents the security review performed by Cairo Security Clan on the Starknet ID protocol.

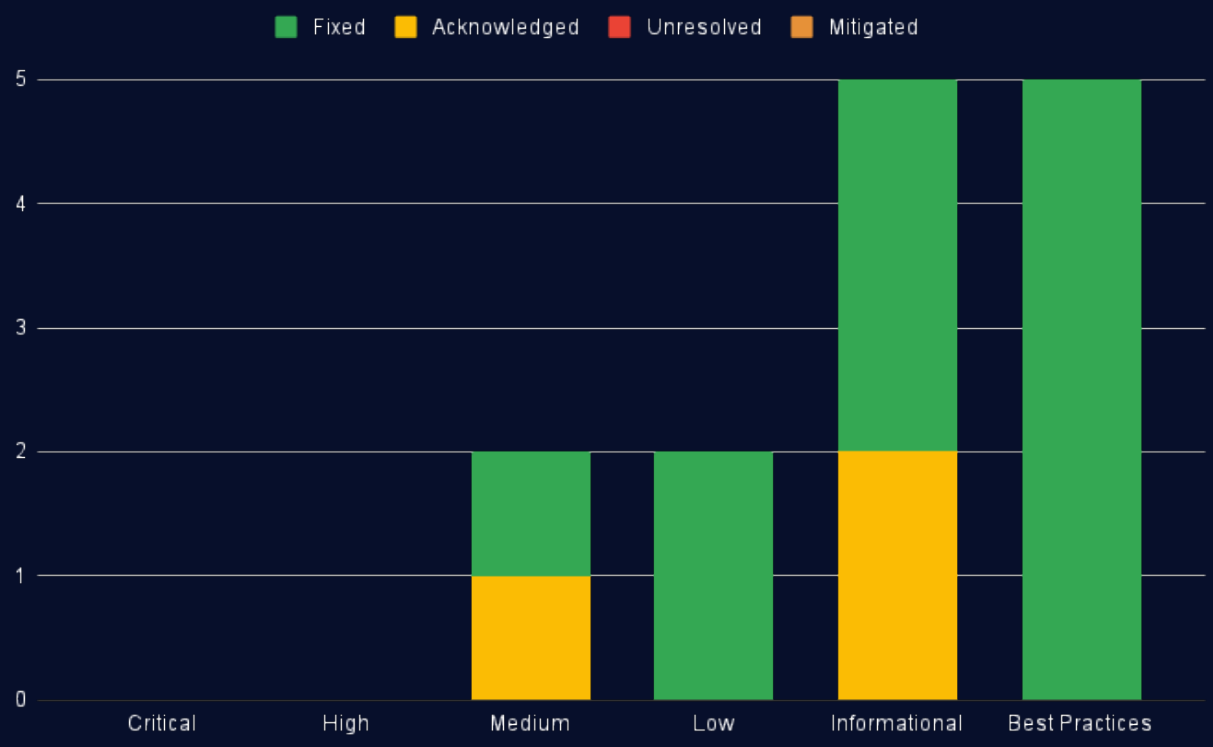
Starknet ID serves as a versatile passport for Starknet, facilitating seamless storage and sharing of user-specific data within the Starknet ID ecosystem. This robust identity protocol allows various Starknet app to access and utilize user information effortlessly, enhancing the overall user experience. When a user registers a stark name with Starknet ID, this information is accessible across all the Starknet Ecosystem. This means that every app will share the Starknet ID info and could use/display it instead of asking users to create their profile again.Learn more from docs.

The audit was performed using

- manual analysis of the codebase,
- automated analysis tools,
- simulation of the smart contract,
- analysis of edge test cases

14 points of attention, where 0 are classified as `Critical`, 0 are classified as `High`, 2 is classified as `Medium`, 2 are classified as `Low`, 5 are classified as `Info` and 5 are classified as `Best Practices`. The issues are summarized in Fig. 1.

This document is organized as follows. Section 1 About Cairo Security Clan. Section 2 Disclaimer. Section 3 Executive Summary. Section 4 Summary of Audit. Section 5 Risk Classification. Section 6 Issues by Severity Levels. Section 7 Test Evaluation.



(a) distribution of issues according to the severity and status

Fig 1: (a) Distribution of issues: Critical (0), **High** (0), **Medium** (2), **Low** (2), **Informational** (5), **Best Practices** (5). **(b) Distribution of status: Fixed** (11), **Acknowledged** (3), **Mitigated** (0), **Unresolved** (0)

4 Summary of Audit

Audit Type	Security Review
Cairo Version	2.6.3
Initial Report	06/05/2024
Response from Client	15/06/2024
Final Report	18/06/2024
Repository (identity)	starknet-id/identity
Commit Hash (identity)	c8734d2ac2893e162ad9a2e506a0ce61ac36d3f9
Repository (naming)	starknet-id/naming
Commit Hash (naming)	8abe7e0e76947b88f45e61016d5d416c94366233
Documentation	Website documentation
Test Suite Assessment	High

4.1 Scoped Files

	Naming Contracts
1	/src/interface.cairo
2	/src/lib.cairo
3	/src/naming.cairo
4	/src/pricing.cairo
5	/src/tests.cairo
6	/src/interface/naming.cairo
7	/src/interface/pricing.cairo
8	/src/interface/referral.cairo
9	/src/interface/resolver.cairo
10	/src/naming/asserts.cairo
11	/src/naming/internal.cairo
12	/src/naming/main.cairo
13	/src/naming/utils.cairo
14	/src/tests/naming.cairo
15	/src/tests/test_pricing.cairo
16	/src/tests/utils.cairo
17	/src/tests/naming/common.cairo
18	/src/tests/naming/test_abuses.cairo
19	/src/tests/naming/test_altcoin.cairo
20	/src/tests/naming/test_custom_resolver.cairo
21	/src/tests/naming/test_features.cairo
22	/src/tests/naming/test_usecases.cairo

	Identity Contracts
1	/src/identity.cairo
2	/src/interface.cairo
3	/src/lib.cairo
4	/src/tests.cairo
5	/src/identity/internal.cairo
6	/src/identity/main.cairo
7	/src/interface/identity.cairo
8	/src/tests/test_addresses_comp.cairo
9	/src/tests/test_extended_data.cairo
10	/src/tests/test_identity.cairo
11	/src/tests/utils.cairo

4.2 Issues

	Findings	Severity	Update
1	Anyone can transfer or purchase a domain for an arbitrary ID.	Medium	Acknowledged
2	Incorrect handling for reset subdomains.	Medium	Fixed
3	User could buy empty domain.	Low	Fixed
3	Owner transfer can cause loss of access.	Low	Fixed
4	SRC5 imported but no interface registered.	Informational	Acknowledged
5	Returning 0 as `hashed_domain` when resolving through a resolver.	Informational	Fixed
6	Function add_commission(...) should take the erc20 address as input parameter.	Informational	Fixed
7	Parent domain might not existed when a subdomain is created.	Informational	Acknowledged
8	Unchecked return value of ERC20 transfer() and transferFrom() in `pay_domain(...)` and `claim_balance(...)`.	Informational	Fixed
9	Unused storage variable.	Best Practices	Fixed
10	Unnecessary type casting to felt252.	Best Practices	Fixed
11	Caching get_caller_address to reduce redundancy.	Best Practices	Fixed
12	Incosistent checks for maximum purchase.	Best Practices	Fixed
13	Unused variable.	Best Practices	Fixed

5 Risk Classification

The risk rating methodology used by Cairo Security Clan follows the principles established by the CVSS risk rating methodology. The severity of each finding is determined by two factors: **Likelihood** and **Impact**.

Likelihood measures how likely an attacker will uncover and exploit the finding. This factor will be one of the following values:

a) **High**: The issue is trivial to exploit and has no specific conditions that need to be met;

b) **Medium**: The issue is moderately complex and may have some conditions that need to be met;

c) **Low**: The issue is very complex and requires very specific conditions to be met.

When defining the likelihood of a finding, other factors are also considered. These can include but are not limited to Motive, opportunity, exploit accessibility, ease of discovery, and ease of exploit.

Impact is a measure of the damage that may be caused if an attacker exploits the finding. This factor will be one of the following values:

a) **High**: The issue can cause significant damage such as loss of funds or the protocol entering an unrecoverable state;

b) **Medium**: The issue can cause moderate damage such as impacts that only affect a small group of users or only a particular part of the protocol;

c) **Low**: The issue can cause little to no damage such as bugs that are easily recoverable or cause unexpected interactions that cause minor inconveniences.

When defining the impact of a finding other factors are also considered. These can include but are not limited to Data/state integrity, loss of availability, financial loss, and reputation damage. After defining the likelihood and impact of an issue, the severity can be determined according to the table below.

		Likelihood		
		High	**Medium**	**Low**
Impact	**High**	Critical	High	Medium
	Medium	High	Medium	Low
	Low	Medium	Low	Info/Best Practices

To address issues that do not fit a High/Medium/Low severity, Cairo Security Clan also uses three more finding severities: **Informational**, **Best Practices** and **Gas**

a) **Informational** findings do not pose any risk to the application, but they carry some information that the audit team intends to formally pass to the client;

b) **Best Practice** findings are used when some piece of code does not conform with smart contract development best practices;

b) **Gas** findings are used when some piece of code uses more gas than it should be or have some functions that can be removed to save gas.

6 Issues by Severity Levels

6.1 Medium

6.1.1 Anyone can transfer or purchase a domain for an arbitrary ID.

File(s): /src/naming/main.cairo

Description: The functions `buy()` and `altcoin_buy()` permit anyone to purchase a domain for an arbitrary ID, regardless of whether the caller is the actual owner of that ID. This could potentially be exploited to assign an arbitrary ID with a domain that the ID owner did not intend to buy.

A similar issue exists within the `transfer_domain()` function. Anyone can transfer a spam domain to an arbitrary `target_id`, which prevents the purchase of a new domain.

```
fn buy(
    ref self: ContractState,
    id: u128,
    domain: felt252,
    days: u16,
    resolver: ContractAddress,
    sponsor: ContractAddress,
    discount_id: felt252,
    metadata: felt252,
) { // @audit anyone can buy domain for arbitrary ID
    let (hashed_domain, now, expiry) = self.assert_purchase_is_possible(id, domain, days);
    // we need a u256 to be able to perform safe divisions
    let domain_len = self.get_chars_len(domain.into());
    // find domain cost
    let (erc20, price) = IPricingDispatcher {
        contract_address: self._pricing_contract.read()
    }
        .compute_buy_price(domain_len, days);
    self.pay_domain(domain_len, erc20, price, now, days, domain, sponsor, discount_id);
    self.emit(Event::SaleMetadata(SaleMetadata { domain, metadata }));
    self.mint_domain(expiry, resolver, hashed_domain, id, domain);
}
```

Recommendation(s): Consider adding check to ensure the owner of the identity has approved to receive the domain.

Status: Acknowledged

Update from client: We take notice of that issue. It could currently be used to target owners of virgin identities and give them a spam domain (for example an insulting subdomain). This would be relatively expensive to do at scale and moderately problematic because users would be able to remove the spam domain and most users wouldn't be impacted.

To completely fix the problem a solution would be to set up an approval system for attaching a domain to an identity. We think this is not worth it for the moment as it would damage the whole experience (slightly more expensive transactions, more error prone, less user friendly multicall). Instead in the eventuality that this "attack" takes place we suggest adapting the frontend to allow the domain name associated with an identity to be overwritten without requiring an additional click in order to minimize user inconvenience. In terms of communication we would present it as the fact that a wallet can receive ERC20 without giving its consent.

6.1.2 Incorrect handling for reset subdomains.

File(s): /src/naming/internal.cairo /src/naming/main.cairo

Description: The Naming contract's `DomainData` struct uses the `key` and `parent_key` fields to manage subdomains. The `key` is referred by subdomains of the current domain, while the `parent_key` points to the current domain's parent.

Given this design, a domain's admin/owner can remove all subdomains under the current domain simply by changing the `key`. Consequently, all previously created subdomains will have an invalid `parent_key`. This feature is implemented in the `reset_subdomains(...)` function shown in the code snippet below.

```
fn reset_subdomains(ref self: ContractState, domain: Span<felt252>) {
    self.assert_control_domain(domain, get_caller_address());
    let hashed_domain = self.hash_domain(domain);
    let current_domain_data = self._domain_data.read(hashed_domain);
    let new_domain_data = DomainData {
        owner: current_domain_data.owner,
        resolver: current_domain_data.resolver,
        address: current_domain_data.address,
        expiry: current_domain_data.expiry,
        key: current_domain_data.key + 1,
        parent_key: current_domain_data.parent_key,
    };
    self._domain_data.write(hashed_domain, new_domain_data);
    self.emit(Event::SubdomainsReset(SubdomainsReset { domain: domain, }));
}
```

When processing or reading a domain in the `domain_to_address(...)`, `domain_to_id(...)` and `resolve_util(...)` functions, they check whether the current domain has been reset by its parent domain. However, these functions only check the direct parent of the current domain, neglecting to verify if the parent's parent domain has been reset. For example, consider the domain `aa.bb.cc.stark`. If the admin resets the subdomain for cc.stark, this should invalidate the domain `aa.bb.cc.stark`. However, since the check only validates the direct parent domain (`bb.cc.stark`, which has not had its `key` updated), the domain `aa.bb.cc.stark` remains valid.

```
let data = self._domain_data.read(hashed_domain);
if data.address.into() != 0 {
    if domain.len() != 1 {
        let parent_key = self
            ._domain_data
            .read(self.hash_domain(domain.slice(1, domain.len() - 1)))
            .key;
        if parent_key == data.parent_key {
            return data.address; // @audit parent of parent could be reset
        };
    };
    return data.address;
};
```

Recommendation(s): Consider checking for the parent's parent domain when handling reset subdomains.

Status: Fixed

Update from client: Fixed in this commit.

6.2 Low

6.2.1 User could buy empty domain.

File(s): /src/naming/main.cairo

Description: In many places in the codebase, the value 0 is reserved as a special value. For example, when reading the domain from `domain_hash`, the read stops when it encounters a 0 value, which is the default value of a storage slot.

```
fn unhash_domain(self: @Naming::ContractState, domain_hash: felt252) -> Span<felt252> {
    let mut i = 0;
    let mut domain = ArrayTrait::new();
    loop {
        let domain_part = self._hash_to_domain.read((domain_hash, i));
        if domain_part == 0 {
            break;
        };
        domain.append(domain_part);
        i += 1;
    };
    domain.span()
}
```

However, there is no logic in place to check that the data does not contain any 0 values when storing them. For instance, the function `buy(...)` does not verify that the domain being purchased is not 0. As a result, user could buy an empty domain, which leads to unexpected behavior.

This issue exists in:

- `get_unbounded(...)` in identity/internal.cairo
- `unhash_domain(...)` in naming/utils.cairo
- `read_address_to_domain(...)` in naming/internal.cairo

Recommendation(s): Add checks to ensure that the data does not contain any 0 values when storing it.

Status: Fixed

Update from client: Fixed in this commit.

6.2.2 Owner transfer can cause loss of access.

File(s): /src/naming/main.cairo

Description: Function `set_admin(...)` updates admin address of the contract. However, it won't check if the new address is a real account. This may cause a mistaken loss of access to the contract, and there is no way to take that access back.

```
fn set_admin(ref self: ContractState, new_admin: ContractAddress) {
    assert(get_caller_address() == self._admin_address.read(), 'you are not admin');
    self._admin_address.write(new_admin);
}
```

Recommendation(s): Consider changing the admin address after the new address approval.

Status: Fixed

Update from client: Fixed in this commit.

6.3 Informationals

6.3.1 SRC5 imported but no interface registered.

File(s): /src/identity/main.cairo

Description: The SRC5 introspection contract, by default, only supports the value below as a supported interface.

```
const ISRC5_ID: felt252 = 0x3f918d17e5ee77373b56385708f855659a07f75997f365cf87748628532a055;
```

To support any other interfaces, the `register_interface(...)` method has to be called in the constructor with `interface_id`. Other imported components from Openzeppelin already registering these interfaces. There is no need to register them again.

Recommendation(s): Consider using SRC5 with your own interface ID; if it is not necessary, remove the SRC5 component from the contract.

Status: Acknowledged

Update from client: We use the SRC5 component because it is required by the ERC721 component which our identity contract implements. Our contract should support ISRC5 and IERC721_ID and IERC721_METADATA_ID interface ids (the later being registered by ERC721 component).

6.3.2 Returning 0 as `hashed_domain` when resolving through a resolver.

File(s): /src/naming/internal.cairo

Description: In the `resolve_util(...)` function, if the resolver contract exists, it returns 0 for the `hashed_domain`. This isn't a recommended practice.

```
fn resolve_util(
    self: @Naming::ContractState, domain: Span<felt252>, field: felt252, hint: Span<felt252>
) -> (felt252, felt252) {
    let (resolver, parent_start) = self.domain_to_resolver(domain, 1);
    if (resolver != ContractAddressZeroable::zero()) {
        (   0,
            IResolverDispatcher { contract_address: resolver }
                .resolve(domain.slice(0, parent_start), field, hint)
        )
```

If the resolver fails to resolve a domain (returns 0), the `domain_to_address(...)` function will still use the `hashed_domain` to proceed with the normal process. Returning 0 for `hashed_domain` could affect the logic and yield incorrect results.

```
fn domain_to_address(
    self: @ContractState, domain: Span<felt252>, hint: Span<felt252>
) -> ContractAddress {
    let (hashed_domain, value) = self.resolve_util(domain, 'starknet', hint);
    if value != 0 {
        let addr: Option<ContractAddress> = value.try_into();
        return addr.unwrap();
    };
    let data = self._domain_data.read(hashed_domain);
    if data.address.into() != 0 {
        if domain.len() != 1 {
            let parent_key = self
                ._domain_data
                .read(self.hash_domain(domain.slice(1, domain.len() - 1)))
                .key;
            if parent_key == data.parent_key {
                return data.address;
            };
        };
        return data.address;
    };
    IIdentityDispatcher { contract_address: self.starknetid_contract.read() }
        .owner_from_id(self.domain_to_id(domain))
}
```

Recommendation(s): Always calculate and return the hashed domain, even when resolving through a resolver.

Status: Fixed

Update from client: Fixed in this commit.

6.3.3 Function add_commission(...) should take the erc20 address as input parameter.

File(s): /src/naming/internal.cairo

Description: The `add_commission(...)` function is invoked when paying for a domain to add a sponsor commission if eligible. The `pay_domain(...)` function accepts various tokens for domain payment. The `discounted_price` value signifies the amount of payment token that will be paid by the caller. However, the `add_commission(...)` function does not consider the token used for domain payment. If the sponsor commission is dependent on the `discounted_price`, users who use a low-priced token for payment will have a higher `discounted_price` than those who use a higher-priced token. Even though both payments are of the same value, the sponsor commission will vary.

```
IERC20CamelDispatcher { contract_address: erc20 }
    .transferFrom(get_caller_address(), get_contract_address(), discounted_price);
if sponsor.into() != 0 {
    IReferralDispatcher { contract_address: self._referral_contract.read() }
        .add_commission(discounted_price, sponsor, sponsored_addr: get_caller_address());
}
```

Recommendation(s): Consider the payment token when adding a sponsor commission.

Status: Fixed

Update from client: Fixed in this commit.

6.3.4 Parent domain might not existed when a subdomain is created.

File(s): /src/naming/main.cairo

Description: When a domain is transferred through the function `transfer_domain(...)`, the `parent_key` is set by fetching the ID of the parent domain. The function `transfer_domain(...)` is also used to create a new subdomain by transferring a subdomain from nothing to a target ID.

However, it does not verify whether the parent domain exists before transferring a subdomain. In that case, the parent_key will be set to 0, making it appear like a top domain (no parent domain) even though it is a subdomain.

```
let new_domain_data = DomainData {
    owner: target_id,
    resolver: current_domain_data.resolver,
    address: current_domain_data.address,
    expiry: current_domain_data.expiry,
    key: current_domain_data.key,
    parent_key: if domain.len() == 1 {
        current_domain_data.parent_key
    } else {
        let hashed_parent_domain = self.hash_domain(domain.slice(1, domain.len() - 1));
        let next_domain_data = self._domain_data.read(hashed_parent_domain);
        next_domain_data.key
    }
};
```

Recommendation(s): Consider adding a check to ensure the parent domain exists before creating a new subdomain. Additionally, consider adding a function to create a subdomain instead of relying on the function `transfer_domain(...)` to create a subdomain.

Status: Acknowledged

Update from client: I am not sure to understand the bug, what would be the issue of using a parent domain of zero? The point is that if the parent domain is claimed, the subdomains would be reset.

Update from CSC: We reviewed the contracts under the assumption that a subdomain can only be created after the parent domain exists. It is like in a tree structure, a root node must exist before creating child nodes. However, if the design is intended to permit this behavior, then there is no issue. Ultimately, this is just an Info finding.

6.3.5 Unchecked return value of ERC20 transfer() and transferFrom() in `pay_domain(...)` and `claim_balance(...)`.

File(s): /src/naming/main.cairo , /src/naming/internal.cairo

Description: `pay_domain(...)` and `claim_balance(...)` uses ERC20 transferFrom() and transfer() functions respectively. These functions return a boolean value indicating whether the transfer was successful. However, the current implementation doesn't check the returned value, potentially resulting in unexpected behavior, particularly for tokens that do not revert on failure.

```
fn pay_domain() {
    ...
    IERC20CamelDispatcher { contract_address: erc20 }
    .transferFrom(get_caller_address(), get_contract_address(), discounted_price);
}

fn claim_balance(){
    ...
    IERC20CamelDispatcher { contract_address: erc20 }
    .transfer(get_caller_address(), balance);
}
```

Recommendation(s): Consider implementing a check for boolean return value of above functions.

Status: Fixed

Update from client: Fixed in this commit.

6.4 Best Practices

6.4.1 Unused storage variable.

File(s): /src/identity/main.cairo

Description: The state variable Proxy_admin remains unused. Although likely introduced for ownership management, it duplicates the functionality already provided by the OpenZeppelin Ownable component utilized by the identity contract. The value stored in this variable is identical to the value stored within the Ownable component's owner variable

```
    #[storage]
    struct Storage {
        user_data: LegacyMap<(u128, felt252), felt252>,
        verifier_data: LegacyMap<(u128, felt252, ContractAddress), felt252>,
        main_id_by_addr: LegacyMap<ContractAddress, u128>,
        // legacy owner
        Proxy_admin: felt252,
        #[substorage(v0)]
        storage_read: storage_read_component::Storage,
        #[substorage(v0)]
        src5: SRC5Component::Storage,
        #[substorage(v0)]
        erc721: ERC721Component::Storage,
        #[substorage(v0)]
        ownable: OwnableComponent::Storage,
        #[substorage(v0)]
        upgradeable: UpgradeableComponent::Storage
    }
```

Recommendation(s): Consider removing this storage variable.

Status: Fixed

Update from client: Fixed in this commit.

6.4.2 Unnecessary type casting to felt252.

File(s): /src/identity/main.cairo

Description: The function `get_extended_verifier_data(...)` includes a basic inefficiency. It accepts a parameter "length" of type felt252. However, when calling the internal function `get_extended(...)`, the "length" parameter is explicitly cast to `felt252` using the `into()` method. This type casting is unnecessary because "length" is already of the expected type felt252

```
fn get_extended_verifier_data(
    self: @ContractState,
    id: u128,
    field: felt252,
    length: felt252,
    verifier: ContractAddress,
    domain: u32
) -> Span<felt252> {
    self
        .get_extended(
        VERIFIER_DATA_ADDR,
        array![id.into(), field, verifier.into()].span(),
        length.into(),
        domain,
        )
}
```

Recommendation(s): Consider removing the unnecessary type casting by simply passing "length" directly to the `get_extended(...)` function call.

Status: Fixed

Update from client: Fixed in this commit.

6.4.3 Caching `get_caller_address()` to reduce redundancy.

File(s): /src/identity/main.cairo

Description: The `reset_main_id(...)` function calls `get_caller_address()` twice. This is unnecessary as the caller's address stays the same within the function

```
fn reset_main_id(ref self: ContractState) {
    let id = self.main_id_by_addr.read(get_caller_address());
    self.main_id_by_addr.write(get_caller_address(), 0);
    self
        .emit(
        Event::MainIdUpdate(MainIdUpdate { id, owner: ContractAddressZeroable::zero() })
        );
}
```

Recommendation(s): Consider caching `get_caller_address()` in a local variable upon retrieval and use that variable throughout the function.

Status: Fixed

Update from client: Fixed in this commit.

6.4.4 Incosistent checks for maximum purchase.

File(s): /src/naming/asserts.cairo

Description: The checks for the 25-year maximum purchase are inconsistent. Modifying these could improve the code quality.

```
// buy
assert(days < 365 * 25, 'max purchase of 25 years');

// renew
assert(new_expiry <= now + 86400 * 9125, 'purchase too long');
```

Recommendation(s): Alter the comparison operator from < to <=.

Status: Fixed

Update from client: Fixed in this commit.

6.4.5 Unused variable.

File(s): /src/naming/main.cairo

Description: In other functions like buy or renew, the `metadata` is used to emit event but the `metadata` in the function `set_expiry(...)` is unused.

```
fn set_expiry(
    ref self: ContractState, root_domain: felt252, expiry: u64, metadata: felt252 // @audit unused metadata
) {
    assert(get_caller_address() == self._admin_address.read(), 'you are not admin');
    let hashed_domain = self.hash_domain(array![root_domain].span());
    let domain_data = self._domain_data.read(hashed_domain);
    let data = DomainData {
        owner: domain_data.owner,
        resolver: domain_data.resolver,
        address: domain_data.address,
        expiry: expiry,
        key: domain_data.key,
        parent_key: 0,
    };
    self._domain_data.write(hashed_domain, data);
    self
        .emit(
            Event::DomainRenewal(DomainRenewal { domain: root_domain, new_expiry: expiry })
        );
}
```

Recommendation(s): Removing unused variables.

Status: Fixed

Update from client: Fixed in this commit.

7 Test Evaluation

7.1 Compilation Output

7.1.1 Identity

```
scarb build
   Compiling lib(identity) identity v0.1.0 (...\006-STARKNET-ID\identity\Scarb.toml)
   Compiling starknet-contract(identity) identity v0.1.0 (...\006-STARKNET-ID\identity\Scarb.toml)
    Finished release target(s) in 20 seconds
```

7.1.2 Naming

```
scarb build
   Compiling lib(naming) naming v0.1.0 (...\006-STARKNET-ID\naming\Scarb.toml)
   Compiling starknet-contract(naming) naming v0.1.0 (...\006-STARKNET-ID\naming\Scarb.toml)
    Finished release target(s) in 21 seconds
```

7.2 Tests Output

7.2.1 Identity

```
scarb test
     Running cairo-test identity
   Compiling test(identity_unittest) identity v0.1.0 (...\006-STARKNET-ID\identity\Scarb.toml)
    Finished release target(s) in 10 seconds
testing identity ...
running 13 tests
test identity::tests::test_addresses_comp::test_compute_address_single_param ... ok (gas usage est.: 36440)
test identity::tests::test_addresses_comp::test_compute_address_empty_param ... ok (gas usage est.: 11370)
test identity::tests::test_addresses_comp::test_compute_address_multiple_params ... ok (gas usage est.: 33270)
test identity::tests::test_extended_data::test_set_user_data_not_owner ... ok (gas usage est.: 799470)
test identity::tests::test_identity::test_user_data ... ok (gas usage est.: 911520)
test identity::tests::test_extended_data::test_set_extended_verifier_data_len_1 ... ok (gas usage est.: 949010)
test identity::tests::test_extended_data::test_extended_verifier_data ... ok (gas usage est.: 1058720)
test identity::tests::test_extended_data::test_unbounded_verifier_data ... ok (gas usage est.: 1111460)
test identity::tests::test_identity::test_verifier_data ... ok (gas usage est.: 894810)
test identity::tests::test_extended_data::test_extended_user_data ... ok (gas usage est.: 1119530)
test identity::tests::test_extended_data::test_get_extended_verifier_data_len_1 ... ok (gas usage est.: 955280)
test identity::tests::test_extended_data::test_unbounded_user_data ... ok (gas usage est.: 1122930)
test identity::tests::test_extended_data::test_verifier_data ... ok (gas usage est.: 894810)
test result: ok. 13 passed; 0 failed; 0 ignored; 0 filtered out;
```

7.2.2 Naming

```
scarb test
     Running cairo-test naming
   Compiling test(naming_unittest) naming v0.1.0 (...\006-STARKNET-ID\naming\Scarb.toml)
    Finished release target(s) in 13 seconds
testing naming ...
running 28 tests
test naming::tests::naming::test_abuses::test_non_admin_cannot_set_admin ... ok (gas usage est.: 1188770)
test naming::tests::test_pricing::test_buy_price ... ok (gas usage est.: 1052520)
test naming::tests::naming::test_abuses::test_non_admin_cannot_claim_balance ... ok (gas usage est.: 1229540)
test naming::tests::naming::test_altcoin::test_buy_domain_altcoin_quote_expired ... ok (gas usage est.: 3271238)
test naming::tests::naming::test_usecases::test_discounts ... ok (gas usage est.: 3592860)
test naming::tests::naming::test_usecases::test_non_owner_can_renew_domain ... ok (gas usage est.: 5042520)
test naming::tests::naming::test_altcoin::test_buy_domain_with_strk ... ok (gas usage est.: 5055568)
test naming::tests::naming::test_custom_resolver::test_custom_resolver ... ok (gas usage est.: 6893790)
test naming::tests::naming::test_abuses::test_not_enough_eth ... ok (gas usage est.: 3069290)
test naming::tests::naming::test_altcoin::test_buy_domain_altcoin_wrong_quote ... ok (gas usage est.: 3275638)
test naming::tests::naming::test_altcoin::test_renew_domain_with_strk ... ok (gas usage est.: 7043566)
test naming::tests::naming::test_altcoin::test_hash_matches ... ok (gas usage est.: 14690)
test naming::tests::naming::test_features::test_subdomains ... ok (gas usage est.: 9035980)
test naming::tests::naming::test_altcoin::test_subscription_with_strk ... ok (gas usage est.: 7258328)
test naming::tests::naming::test_features::test_get_chars_len ... ok (gas usage est.: 962960)
test naming::tests::naming::test_altcoin::test_convert_quote_to_eth ... ok (gas usage est.: 24240)
test naming::tests::naming::test_usecases::test_basic_usage ... ok (gas usage est.: 8309940)
test naming::tests::naming::test_altcoin::test_subscription_not_whitelisted ... ok (gas usage est.: 2429880)
test naming::tests::naming::test_usecases::test_set_domain_to_resolver ... ok (gas usage est.: 4359720)
test naming::tests::naming::test_abuses::test_buying_twice_on_same_id ... ok (gas usage est.: 4152390)
test naming::tests::naming::test_abuses::test_non_owner_cannot_transfer_domain ... ok (gas usage est.: 2518640)
test naming::tests::naming::test_usecases::test_renewal ... ok (gas usage est.: 5476690)
test naming::tests::naming::test_abuses::test_buying_domain_twice ... ok (gas usage est.: 4430620)
test naming::tests::naming::test_abuses::test_renewal_period_too_short ... ok (gas usage est.: 4735280)
test naming::tests::naming::test_usecases::test_set_address_to_domain ... ok (gas usage est.: 13463710)
test naming::tests::naming::test_abuses::test_use_reset_subdomains ... ok (gas usage est.: 10043930)
test naming::tests::naming::test_features::test_claim_balance ... ok (gas usage est.: 4177200)
test naming::tests::naming::test_abuses::test_renewal_period_too_long ... ok (gas usage est.: 4734040)
test result: ok. 28 passed; 0 failed; 0 ignored; 0 filtered out;
```