PUBLIC

Code Assessment

# of the Auto Renew Smart Contract

October 6, 2023

Produced for

by

CHAINSECURITY

Contents

1 Executive Summary

Dear all,

Thank you for trusting us to help StarknetID with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Auto Renew according to Scope to support you in forming an opinion on their security risks.

StarknetID has implemented a non-upgradable auto-renewal contract to streamline domain renewals for users. Users can seamlessly enable or disable spending flows, which, subject to certain conditions, are executed by a designated, whitelisted renewer. These conditions include annual execution and ensure the domain expires in less than a month. The contract is governed by an admin, with users being responsible for setting accurate allowances.

The most critical subjects covered in our audit are functional correctness and security of user funds. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	2
• Code Corrected	1
• Specification Changed	1

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Auto Renew repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

V	Date	Commit Hash	Note
1	21 September 2023	54bcb58cd5c840f58f9b7f206c93e9db0b826ffd	Initial Version
2	4 October 2023	9fce6d7ffc746107ab31216d96b04de698e33f8f	After Intermediate Report

For the cairo smart contract, the compiler version `2.2.0` was chosen. At the time of this review (September 2023) Starknet v0.12.2 was live on mainnet. This review cannot account for future changes and possible bugs in Starknet and it's libraries / token contracts.

The following file was in scope of this review:

- auto_renewal.cairo

2.1.1 Excluded from scope

Any file not listed above is excluded from the scope. Notably the naming and starknet identity contracts are not in the scope of this review.

2.2 System Overview

This system overview describes the initially received version (**Version 1**) of the contracts as defined in the Assessment Overview.

At the end of this report section we have added subsections for each of the changes accordingly to the versions.

StarknetID offers a non-upgradable auto renewal contract which facilitates the renewal of a user's domain.

2.2.1 Auto Renewal contract

Users can activate the automatic renewal by creating a spending flow for a domain and approve sufficient allowance for the contract. The spending flows will be consumed by the renewer to call `renew()` on the naming contract in a recurrent way. Users will keep the possibility of cutting it at any time and won't pay the transaction costs, and will only be charged the `limit_price` that is specified by the user.

The main entrypoints for the users are:

1. `enable_renewals()` - The function for a user to enable a spending flow. Each spending flow is uniquely identified by the triplet of (`user_address`, `domain`, `limit_price`). The `last_renewal` for the `user` of this `domain` will also be reset to 0 in this call.
2. `disable_renewals()` - A user can cut one of his spending flows anytime by calling this function with the same triplet of (`user_address`, `domain`, `limit_price`). Though there is no view function provided for this data, it can be retrieved from the emitted events in `enable_renewals()`.

The whitelisted renewer is a privileged role who can call `renew()` or `batch_renew()` to renew the domain for the users according to enabled spending flows. `batch_renew()` will invoke `_renew()` in a loop for the input span, and will revert if any of the individual `_renew()` fails.

The following checks are done in `_renew()`:

1. The spending flow is checked to be enabled.
2. The `last_renewal` timestamp is compared to the current block timestamp to ensure at least 364 days have elapsed since the last renewal. (Note a user can reset `last_renewal` to 0 by disable a renewal and enable it again)
3. The `expiry` of the domain is fetched from the naming contract. The renewal can only proceed if the domain will expire within one month. This also guarantees a domain will not be renewed simultaneously by multiple spending flows.

Afterwards, the `last_renewal` will be updated to the current block timestamp. Then it will spend allowance to transfer `limit_price` from the user to the auto renewal contract, transfer `tax` to the tax contract and finally call `renew()` on the naming contract to renew the domain. It's worth noting that the `renew()` call to the naming contract will revert if there is insufficient funds on the auto renewal contract instead of checking the `limit_price-tax_price` with the domain price. Besides, the user should not set a `limit_price` that is larger than the actual domain price, otherwise, the redundant amount will still be charged and remain with the auto renewal contract.

The following view functions are provided:

1. `is_renewing()` - Returns if a spending flow is enabled given a triplet of (`user_address`, `domain`, `limit_price`).
2. `get_contracts()` - Returns the naming, token, and tax contracts addresses.

The whole contract is governed by an `admin` who has the privilege to call:

1. `update_admin()` - To set the admin to another address.
2. `update_tax_contract()` - To set the tax contract.
3. `update_whitelisted_renewer()` - To set the whitelisted renewer.
4. `toggle_off()` - To disable the renewal irreversibly.
5. `claim()` - To withdraw excess ERC-20 tokens from this auto renewal contract.

2.2.2 Roles and Trust Model

The admin ultimately governs this contract. The `renewer` is another privileged role triggering the execution of the renewals. They are always trusted to never behave against the users and the system, otherwise:

1. The admin can withdraw the dust in the auto renewal contract and the tax.
2. The admin can disable the contract irreversibly by `toggle_off()`.
3. The renewer can only stop calling `renew()` or `batch_renew()` as a keeper, or not sending any tax to the tax contract.

Users are untrusted. They are assumed to set `limit_price` correctly (guided by the official front end) and give sufficient allowance.

The ERC-20 token is expected to be the StarkGate: ETH Token. Most importantly the ERC-20 token used by the auto renew contract must revert in case of a failed `transfer()` or `transferFrom()`.

2.2.3 Changes in Version 2

- Instead of having to spend the allowance specified in the flow, the automation bot can now spend a less amount. When enabling a spending flow for a domain (`enable_renewals()`) the user now sets an allowance. `is_renewing()` has been replaced by `get_renewing_allowance()`. This function returns the allowance a renewer has given for a domain. In `renew()`: `limit_price` has been replaced by `domain_price`. To disable renewals (`disable_renewals()`) the caller has to specify only the domain.
- Updating the admin address has been changed to a two step process: The current admin initiates the the change using `start_admin_update()`, the new admin finalizes the update by calling `confirm_admin_update()`. Until the new admin has taken over, the current admin can always stop or change the update.
- Admin functions changing states, except `start_admin_update()` now emit events.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	**Impact**		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

- Design: Architectural shortcomings and design inefficiencies
- Correctness: Mismatches between specification and implementation

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	2

- Inconsistent Sanity Checks of Span Length Code Corrected
- Incorrect Comments Specification Changed

6.1 Inconsistent Sanity Checks of Span Length

Design Low Version 1 Code Corrected

CS-STKIDAUTO-005

In `batch_renew()` there are sanity checks to validate that the `domain`, `renewer`, and `limit_price` spans have the same length. However, similar length verifications are missing for the `tax_price` and `metadata` spans, which is inconsistent.

Discrepancies in length would ultimately lead to a failure with an `expect('pop_front error');` error, nevertheless sanity checks may be implemented consistently.

Code corrected:

The missing sanity checks have been added.

6.2 Incorrect Comments

Correctness Low Version 1 Specification Changed

CS-STKIDAUTO-004

In the constructor, the auto renewal contract approves max u256 (`integer::BoundedInt::max()`) allowance to the naming contract. Whereas the comment says the allowance is set to `2^251-1`.

Specification changed:

The comment has been corrected to: `allowing naming 2^256-1`.

6.3 Admin Functions Do Not Emit Events

Informational Version 1 Code Corrected

CS-STKIDAUTO-006

The admin functions to update admin address, tax contract, whitelisted renewer, toggle flag, and claim tokens do not emit events. Users are not able to observe these state updates by events and are expected to query these up-to-date states onchain.

Code corrected:

Admin functions changing states, except `start_admin_update()` now emit events.

6.4 Unused Imports

Informational **Version 1** **Code Corrected**

CS-STKIDAUTO-003

The following imports are not used and could be removed.

```
use traits::{TryInto, Into};
use option::OptionTrait;
use integer::u64_try_from_felt252;
use debug::PrintTrait;
```

Code corrected:

The unused imports have been removed.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Disable Renewals Do Not Check the Flag

Informational | Version 1 | Risk Accepted

CS-STKIDAUTO-001

`disable_renewals()` will set the flag of the corresponding spending channel to false regardless of the current flag. As a result, a user can call `disable_renewals` on an already disabled channel and emit the `DisabledRenewal` event, which might be misleading to the observers of this event.

7.2 `batch_renew()` Reverts if One Action Fails

Informational | Version 1 | Risk Accepted

CS-STKIDAUTO-002

Inherent to the design of `batch_renew()`, a single unsuccessful renew action within the loop reverts the entire execution. It is the responsibility of the whitelisted renewer to ensure the batch is valid.

Technically, interference with the operation of the Auto Renew Bot is possible by front running its transaction, for instance by removing token transfer approval, renewing a domain (which is permissionless), disabling the flow. Given the centralized sequencer, such disruptions are not expected. However, potential future decentralization of the sequencer could elevate the likelihood of such disruptions occurring throughout the contract's lifetime (years).

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 A Domain With No Owner Can Still Be Renewed

Note Version 1

There is no ownership check if a domain has an owner when it is renewed. In case a user set a renewal spending flow for a domain that nobody owns, the renewal will succeed and the domain's expiry will be set to one year. The user who buys the domain later will take the benefit from this renewal:

- Alice renews a domain D that nobody owns, D will have one year of expiry since then.
- Later, Bob buys domain D for one year.
- Now Bob owns domain D which has two years expiry.

8.2 The Admin Can Pause Auto Renew

Note Version 1

Though there is no specific functionality to pause calls to `renew()` and `batch_renew()`, the admin can still achieve this. The admin can set the tax contract address to 0x0. In the present ETH ERC-20 contract implementation, transactions transferring to the 0x0 address revert.