Algorand Name Service

Smart Contract Audit

coinspect

Algorand Name Service

Smart Contract Audit

V220602 Prepared for Algorand Name Service • May 2022

1. Executive Summary

In **April 2022, Algorand Name Service** engaged Coinspect to perform a source code review of the **registry smart contracts**. The objective of the project was to evaluate the security of the smart contracts.

The following issues were identified during the initial assessment:

High Risk	Medium Risk	Low Risk
1	**0**	**1**
Fixed	Fixed	Fixed
1	0	1

2. Assessment and Scope

The audit started on **April 20, 2022** and was conducted on the release tagged **v1.0** of the git repository at github.com/algorand-naming-service as of commit **3d6d9a07e0593075ea5ce025574bfc30648a8ec9** of **Mar 09, 2022**.
The audited files have the following sha256sum hashes:

```
80b99e7c028bb731ef17323a8cc93c69b71c2b4e56e3203332d58ddf2cae56f1  contracts/constants.py
1fa679c985c8025b49b68a4016fa8614015eda0aa7bc105100e88316af80b100  contracts/dot_algo_name_record.py
0b4cf2aa548824e2ff90bcbe313c19ecdaddf0cbbb33f58efa16e25b541348b0  contracts/dot_algo_registry.py
728a9d0dc3d55c81c87bbe2794875a839b942c6a8f1729a89bf55820718b5c27  unit-tests/ans_helper.py
c513c62bdda4a04c3e30ad9e3c43f4a6af113f35a93b256bf7b6eb1f981bc5ee
unit-tests/TestDotAlgoNameRegistry.py
```

“The Algorand Name Service implements a key-value store of name-address pairs using an Algorand stateful smart contract and smart signatures. Each smart contract is a name registry representing a domain suffix (such as .algo). A name record account opt's into the name registry smart contract and mentions the "*owner*" address and name's metadata in its local storage for the smart contract. The desired name is used as input to a standardized TEAL program and compiled, thereby converting the TEAL program into a Logic Signature.” - ANS documentation.

Coinspect did not find issues with the design, but the implementation lacks rigorous transaction fields validation that gives place to ANS-1.

Update

On **June 1, 2022** Coinspect verified that the changes applied on commit **5cdfb894ffbe458402909b88e12fa4d20f2f9d92** of the **main** branch address the issues raised in the initial assessment. Additionally, Coinspect verified that these changes were deployed on **mainnet** on transaction 7SRW4RA3XN2UHX573P5OCGBLWVWV7OVWKOGLAUETET3QXWT2F4VA.

3. Summary of Findings

Id	Title	Total Risk	Fixed
ANS-1	Attackers can steal any name record	High	✔
ANS-2	Lack of fee validation	Low	✔

4. Detailed Findings

ANS-1 Attackers can steal any name record

Total Risk	Impact	Location
High	High	contracts/dot_algo_name_record.py contracts/dot_algo_registry.py
Fixed ✔	Likelihood: High	

Description

Attackers can take control of a Logic Signature name record by rekeying it to another address. This gives them full control of the address of the Logic Signature and allows them to clear the state and register themselves as owners.

The attack can be performed with the following steps:

- Fund Logic Signature with enough algo to pay for the base fee of the chosen name
- Submit the following Group Transactions:
    1. Payment from Logic Signature to Registry Smart contract.
    2. Payment from Logic Signature to the attacker's address (can be 0).
    3. Opt-in from attacker address into Registry Smart contract.
    4. Call register_name from Logic Signature to Registry Smart contract with rekeyTo set to an attacker address.
- Submit ClearState of Logic Signature.

Both validation functions below allow for the malicious group transactions.

```
[dot_algo_name_record.py]
33      is_valid_txn = Seq([
34
35          Assert(Len(Bytes(name)) <= Int(64)),
36          For(i.store(Int(0)), i.load() < Len(Bytes(name)), i.store(i.load() + Int(1))).Do(
37              Assert(
38                  Or(
39                      And(
40                          GetByte(Bytes(name), i.load()) >= Int(constants.ASCII_LOWER_CASE_A),
41                          GetByte(Bytes(name), i.load()) <= Int(constants.ASCII_LOWER_CASE_Z)
42                      ),
43                      And(
44                          GetByte(Bytes(name), i.load()) >= Int(constants.ASCII_DIGIT_0),
```

```
45                          GetByte(Bytes(name), i.load()) <= Int(constants.ASCII_DIGIT_9)
46                      )
47                  )
48              )
49          ),
50
51          Assert(
52              Or(
53                  Global.group_size() == Int(2),
54                  Global.group_size() == Int(4)
55              )
56          ),
57
58          Assert(Gtxn[0].sender() == Gtxn[1].sender()),
59          Assert(Gtxn[0].receiver() == Addr(DOT_ALGO_ESCROW_ADDRESS)),
60
61          If(Global.group_size() == Int(2))
62          .Then(
63              Assert(
64                  And(
65
66                      Gtxn[1].application_id() == Int(DOT_ALGO_APP_ID),
67                      Gtxn[1].application_args[0] == Bytes("register_name"),
68                      Gtxn[1].application_args[1] == Bytes(name)
69                  )
70              )
71          ).ElseIf(Global.group_size() == Int(4))
72          .Then(
73              Assert(
74                  And(
75
76                      Gtxn[1].receiver() == Gtxn[2].sender(),
77                      Gtxn[2].application_id() == Int(DOT_ALGO_APP_ID),
78                      Gtxn[2].on_completion() == OnComplete.OptIn,
79                      Gtxn[3].application_id() == Int(DOT_ALGO_APP_ID),
80                      Gtxn[3].sender() == Gtxn[0].sender(),
81                      Gtxn[3].application_args[0] == Bytes("register_name"),
82                      Gtxn[3].application_args[1] == Bytes(name)
83                  )
84              )
85          ).Else(
86              Return(Int(0))
87          ),
88
89          Int(1)
90      ])
```

```
[dot_algo_registry.py]
33      is_valid_txn = Seq([
34
35          Assert(
36              Or(
37                  Global.group_size() == Int(2),
38                  Global.group_size() == Int(4)
39              )
40          ),
41
42          Assert(Gtxn[0].sender() == Gtxn[1].sender()),
43          Assert(Gtxn[0].receiver() == Global.current_application_address()),
44          Assert(Gtxn[0].rekey_to() == Global.zero_address()),
45          Assert(Gtxn[0].close_remainder_to() == Global.zero_address()),
46
47          If(Global.group_size() == Int(2))
48          .Then(
49              Assert(
50                  And(
51
52                      Gtxn[1].application_id() == Global.current_application_id(),
53                      Gtxn[1].sender() == Gtxn[0].sender(),
54                      Gtxn[1].application_args[0] == Bytes("register_name")
55                  )
56              )
57          ).ElseIf(Global.group_size() == Int(4))
```

```
58          .Then(
59              Assert(
60                  And(
61
62                      Gtxn[1].receiver() == Gtxn[2].sender(),
63                      Gtxn[2].application_id() == Global.current_application_id(),
64                      Gtxn[2].on_completion() == OnComplete.OptIn,
65                      Gtxn[3].application_id() == Global.current_application_id(),
66                      Gtxn[3].sender() == Gtxn[0].sender(),
67                      Gtxn[3].application_args[0] == Bytes("register_name")
68                  )
69              )
70          ).Else(
71              Return(Int(0))
72          ),
73
74          Int(1)
75      ])
```

If the name was not registered yet, attackers can use the OptIn call to rekey the Logic Signature.

Alternatively, a similar attack can be performed by setting `close_remainder_to`. The close operation will clean the address storage, allowing an attacker to register themselves as owners.

Recommendation

Validate the `rekey_to` and `close_remainder_to` fields in all the protocol transactions.

Status

Issue addressed by following recommendation.

ANS-2	Lack of fee validation		

Total Risk
Low
	Impact
	Low
		Location
		`contracts/dot_algo_name_record.py`
		`contracts/dot_algo_registry.py`	

Fixed ✔	Likelihood Low		

Description

Lack of fee validation can lead to drainage of contract accounts.

It is recommended in the Algorand Guideline.

Recommendation

Validate fees to be less than a reasonable value.

Status

Issue addressed by following recommendation.

5. Disclaimer

The information presented in this document is provided "as is" and without warranty. The present security audit does not cover any off-chain systems or frontends that communicate with the contracts, nor the general operational security of the organization that developed the code.