API3

API3 Airnode Protocol

Software and architectural audit by

Version: **1**

Authors:
Christian Veselinov
CTO, Co-Founder & Blockchain Solutions Architect @ LimeChain

Date Created: **January 25, 2021**
Last Edit: **February 5, 2021**

Table of contents

Table of contents

Overview
- Methodology
- Scope

Findings
- #1 Not needed payable modifier and funds forwarding in createProvider (Low Impact, Medium Probability)
- #2 Smart Contracts Layout (Low Impact, High Probability)
- #3 Only one admin address (Low Impact, Low Probability)
- #4 Authorizers per request (Medium Impact, Medium Probability)

Recommendations
- Helpers into the unit tests
- Newer solidity version

Overview

The Software and Security Audit was performed on API3's airnode request-response protocol smart contracts. The first protocol implemented for Airnode is request–response. An Airnode serving the request–response protocol listens for requests, makes the API call specified by the request, and fulfills the request as soon as possible.
The code is very well documented, commented and easy to read.
The existing documentation in github is nice and clear, allowing everyone to quickly understand what is the purpose of each component.
Functions are grouped together according to their functionality.

Methodology

The used methodology for the security audit consists of:

1. Understanding the purpose and specification of the project.
2. Make sure that all tests and coverage report are passing without errors or unexpected behaviour
3. Manual analysis on API3's airnode request-response protocol, covering the functional and security aspects of the smart contracts.
4. Producing Audit Report containing all foundings, potential problems and providing recommendations on how to resolve them.

Scope

GitHub Repository - package: https://github.com/api3dao/airnode/tree/master/packages/protocol
Git Commit: 128305033da1e2d0dd2e9ebe9a6c98dec7092e63

File Name	SHA-1
README.md	8453f16ff4ac30e6c46ce1186909734a3878ad93
contracts/Airnode.sol	30521b109911a1b466cd1330d22e6f939e799eb3
contracts/AirnodeClient.sol	d6f0c0584795eba55b0d06589178dc6801815fbc
contracts/Convenience.sol	9f154581e81681af155a9754dd21d21687d7137d
contracts/EndpointStore.sol	96b33f4b9d94f8b8394ffb91186f223b2a412a70
contracts/ProviderStore.sol	d1e323ed628e46e6873eea41ac95b64d7b1dc8db
contracts/RequesterStore.sol	30de7b3bc9b2e6416d37a8aa8b0323122c810d29
contracts/TemplateStore.sol	6daf42a3197365b15fe8dfe2654954f810902f5a
contracts/authorizers/MinBalanceAuthorizer.sol	6245681abb00c3167ee58c4a9f34311583148842
contracts/authorizers/interfaces/IAuthorizer.sol	4dc4eb0754f1681205644075032794fdd5db1a78
contracts/authorizers/interfaces/IMinBalanceAuthorizer.sol	d8e9f271ba8ffb126ee83232a800653755ca5d52
contracts/interfaces/IAirnode.sol	fd48727d5f7624e2947ae6bcf4230ab4e3b8a1d3
contracts/interfaces/IAirnodeClient.sol	b21952f1a5c05dd5040f3d9ca99c5bd50f7e2c37
contracts/interfaces/IConvenience.sol	b9051e6440396b57627cb5d38d57a095694e5cec
contracts/interfaces/IEndpointStore.sol	f95a9f004d82438be206ad036d4dd95deae889c8
contracts/interfaces/IProviderStore.sol	a92596cad7d818956a42f1fc8cbf6b83448a8644
contracts/interfaces/IRequesterStore.sol	10a120b69a75cfb4925a4586159d9e8be2a49b2a
contracts/interfaces/ITemplateStore.sol	b47df7ce44cd7767201038339b65433343adfd2b
contracts/mock/MockAirnodeClient.sol	e35cfe41b893f4f53b176e0e034b8c32f6426b33
credentials.example.json	ef4b61c8c583266bd984380046bcdcd67330b719
deploy/1_deploy.js	089c1044df81dd0a1398eec2971ff8a3904765c7

hardhat.config.js	a0a29cd74b565df17c1a410192e1761ef2afec0a
index.js	7b4f070a52e2208979ece5aa2e6e408042379849
package.json	e9ac9f6b1fbaa7385062b3d81c00e2a89db0ce76
package.publish.json	8e53155e1ecfb2e053fe048aec3d947d44a638e4
publish.sh	59e6c706c6e5183568c5947de4aaa633b847d6dd
test/Airnode.sol.js	3e4f9f7770b5653dc94c77f393ced25398bf8af4
test/Convenience.sol.js	06bed740c7e4cd98f0ad9e80db0247fa785f79e8
test/EndpointStore.sol.js	c465466dc4088b3dd49870f4ea3620cd123a9651
test/ProviderStore.sol.js	5e5f8ef0b11eb5edd7a70630fa56a4b315d99ae6
test/RequesterStore.sol.js	ea32dff9c09d3d66604713d2bf1a64c1a28acf84
test/TemplateStore.sol.js	b8260a5feb2c1d499f8d052ac610d6a88d122c24
test/UserFlow.js	c4ed25441cb24cb5b0e6b50a9ca9f0176b0d410d
test/helpers/provider.js	f240f1ee09b2ff62afe558119827e8b4db7f2cfe
test/helpers/request.js	eaad2313d9049c8e7fc90b93d281554083d55589
test/helpers/requester.js	39fddf6152b543de52c16890477e4888f3284f22
test/helpers/template.js	aef36a845d6fc574283b675078a40a3b83d91a38
test/util.js	7032e5661bf128dd058769953a09317f31aaedaf

Findings

#1 Not needed payable modifier and funds forwarding in `createProvider` (Low Impact, Medium Probability)

The function `createProvider` in `ProviderStore.sol` file should just create a new provider. Having the option to also send ETH and use `msg.value` brings not needed complexity in a function whose purpose is to create a new provider.

Suggestion:
Consider removing the `if statement` on `line 50.` And if the provider wants to transfer funds to the admin wallet, broadcast a second transaction only for the transfer.

#2 Smart Contracts Layout (Low Impact, High Probability)

Consider following the Solidity Style guide(https://docs.soliditylang.org/en/v0.8.1/style-guide.html#) for the order and formating of the smart contracts

Suggestion:
In `ProviderStore.sol` modifier on `line 172`, move the declaration of this modifier before other function declarations and after the state variables;
In `Airnode.sol` modifier on `line 314`, move the declaration of this modifier before other function declarations and after the state variables;

#3 Only one admin address (Low Impact, Low Probability)

Having only one address as `Provider admin` could potentially be a bottleneck in the future when a single provider could be working with a large number of requests, authorizers and in general provider administration that requires admin. At the moment the provider admin is not used for a lot of transactions, however if for example in the future there is a more complex authorizer which requires regular admin updates the admin could become a bottleneck.
Potentially the same approach could be used for the `Requester Admin`.

Suggestion:
Change `address admin;` on `line 11` in `ProviderStore.sol` to be a mapping (`address => bool)` and use the new mapping for checking if it is one of the admins calling the contracts. In future the provider would have the flexibility to add more than one admin address.

#4 Authorizers per request (Medium Impact, Medium Probability)

The current architecture allows a list of authorizers per `providerId and endpointId.`
Potentially this can lead to unpredicted and unwanted behaviour if there are currently requests which are going to be fulfilled but the provider updates the authorizers list at the same time.
Note: Following the suggestion below, would solve the potential issue described above but would introduce new complexities and gas costs.

Suggestion:
Consider having a `authorizersSets` each set could have the same logic behind it as in the current `authorizers` . When recording a new request in `Airnode.sol` the ID of the current active set could be used for generating the hash recorded in
`requestIdToFulfillmentParameters[requestId]`
If this approach is chosen there won't be an update function for authorizers, but just adding a new set and make it the active one.

Recommendations

Helpers into the unit tests

Unit tests helper functions are doing actually the main work in some tests. Would be good to keep the helpers only for some smaller pieces of logic. For example, the `ProviderStore.sol.js` is checked if the parameters are recorded correctly, but the `expect` statements about the value sent are into the helpers. Try to cover all expect statements into the test file itself.

Newer solidity version

Consider using the latest official solidity version - **0.8.1.** Upgrade guidance for each of the breaking changes could be found here:

- To 0.7.0 - https://docs.soliditylang.org/en/v0.8.1/070-breaking-changes.html#how-to-update-your-code
- To 0.8.0 - https://docs.soliditylang.org/en/v0.8.1/080-breaking-changes.html#how-to-update-your-code