Security Assessment

Airnode RRP

Oct 15th, 2021

Table of Contents

Summary

This report has been prepared for Api3 to discover issues and vulnerabilities in the source code of the Airnode RRP project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Manual Review and Static Analysis techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	Airnode RRP
Platform	Ethereum
Language	Solidity
Codebase	https://github.com/api3dao/airnode/tree/master/packages/protocol/contracts
Commit	f09cb54a0a6e66a4547551fd69d1b659c4e6dafd 8467c9522e0c86dc73d84a94a170616c766f2c8c

Audit Summary

Delivery Date	Oct 15, 2021
Audit Methodology	Manual Review, Static Analysis
Key Components	

Vulnerability Summary

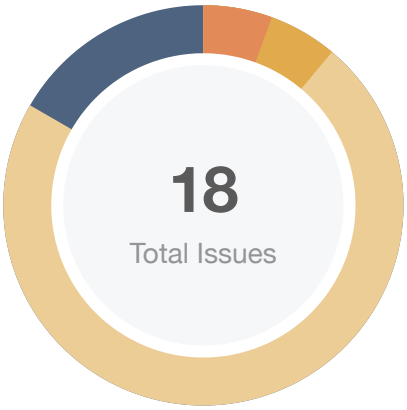
Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
Major	1	0	0	0	0	1
Medium	1	0	0	1	0	0
Minor	13	0	0	8	1	4
Informational	3	0	0	0	0	3
Discussion	0	0	0	0	0	0

Audit Scope

ID	File	SHA256 Checksum
IMA	admin/interfaces/IMetaAdminnable.sol	c10cb75ad39b0504ac46075a9802553b30e4a467d4c72210d68132b3c0d2eb31
IRA	admin/interfaces/IRankedAdminnable.sol	dc73d8a9aa6fcd94b7711c651111315ffb7750789b9b1fc902e7ce41a9e4add6
IWO	admin/interfaces/IWhitelister.sol	cb324e4d9ad3ad8d139f94f290a8f8335e1008ab6b3e015d7dc3171c5903393f
MAO	admin/MetaAdminnable.sol	76c14b74b594b742742082622f1c4c9713a1ed267816062ab2b1f9fd1840afd3
RAO	admin/RankedAdminnable.sol	bbd4145ff1abf960d059094d884bf2fc4335d729ca87d03540daccd3f552cca1
WHI	admin/Whitelister.sol	7b32f6a900033bd9526df843c21ce5cfb643dabc661aa685b203e017260fe21d
IAR	rrp/authorizers/interfaces/IApi3RequesterRrpAuthorizer.sol	a17e5874b80f9d9a23d763e9c8297616e2afee6a975930bed519b5019190a331
IRR	rrp/authorizers/interfaces/IRequesterRrpAuthorizer.sol	7d471cc283391a238cb3999915af07c609996c0115bec472a23feb3adbe4e19e
IRP	rrp/authorizers/interfaces/IRrpAuthorizer.sol	b43ae90e38e372f9a14baea11b19394dad7c76bc81625c2349c2c67f7f3e2aba
ISR	rrp/authorizers/interfaces/ISelfRequesterRrpAuthorizer.sol	98f33a9e8ed1d49708fa79fb3387e3796be0a1476d943bbdbcd4d21d9e530100
MRA	rrp/authorizers/mock/MockRrpAuthorizerAlwaysFalse.sol	afb65195eeddfb6dbff91467e2a6913e53be9a0309a870ce97b2df6747bedf02
MRT	rrp/authorizers/mock/MockRrpAuthorizerAlwaysTrue.sol	727344b9ed2e6376b5b1dc11567ef3dc166f02b3a6b85d4f80e81b29ed8a26bd
ARR	rrp/authorizers/Api3RequesterRrpAuthorizer.sol	7e032e5f0543460753e03629f86339a3c5497d859074d5cd7b58c729d81c6f5b
RRA	rrp/authorizers/RequesterRrpAuthorizer.sol	6c7d05dd6e8bb411625e4f28c12dbaa1b1274b40be40f55b7677a61e8d4c48f8
SRR	rrp/authorizers/SelfRequesterRrpAuthorizer.sol	5f088c0eb2e330561ee24901331abeb4357a097dc2e8473d18bcd8b23aa5d8e2

ID	File	SHA256 Checksum
IAI	rrp/interfaces/IAirnodeRrp.sol	5335d2ca548ff341162f0cda6e8b973a4457ae2369a0b55d0a2d7ec549b23d5e
IAU	rrp/interfaces/IAuthorizationUtils.sol	fa3cb1c4c080e7a3ded6f4c3f8a089a1f08273ad910853f3c3328e20831f5e00
ITU	rrp/interfaces/ITemplateUtils.sol	0b72be621dedc4af7f5bf364b38c97b0c2d6ad8612838f211b4b20afb43505ac
IWU	rrp/interfaces/IWithdrawalUtils.sol	0e454a39b4137fa32290afff4758ff9c552e559c11ace3de802940ab9a617eba
IRB	rrp/requesters/interfaces/IRrpBeaconServer.sol	a782e22c2c0dc7c21732ab48ccf7732ddb9a9b8409e0b2ec55de05783026021b
MRR	rrp/requesters/mock/MockRrpRequester.sol	2b2ce9eef1939cee736c2bd0321e344104ff3f6e9e22fcef956e21dfa45e974b
RBS	rrp/requesters/RrpBeaconServer.sol	d826f61748b63e3f3a6b478184e3b9915572aca59e42c455601b53b8b97c4fcb
RRO	rrp/requesters/RrpRequester.sol	97710beb922a16c0997703d82448d5e631a232b4c42d3a7ef9a2ee235d157c92
ARO	rrp/AirnodeRrp.sol	1510733f70e2963d97bc389c607cdee716613fec2e3de72e5ffebb13e4439472
AUO	rrp/AuthorizationUtils.sol	d6f029cb65c0f6737441b17339c110b8ed27e223f5f6181b47fa42da3d0aa1f4
TUO	rrp/TemplateUtils.sol	1b054b6dca4a8faaba5f59d2520b26f20dca54bb91bf1caa18b2052e2cb5c3e2
WUO	rrp/WithdrawalUtils.sol	6ed08fdea4eb30d6fea51221f871b011db87ed0dfe3aac44e800f66d15a817d5

Findings



Critical	**0** (0.00%)
Major	**1** (5.56%)
Medium	**1** (5.56%)
Minor	**13** (72.22%)
Informational	**3** (16.67%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
GLOBAL-01	Third Party Dependencies	Volatile Code	Minor	Acknowledged
ARO-01	"airnode" should be checked	Logical Issue	Minor	Resolved
ARO-02	"requestId" should include "airnode"	Logical Issue	Minor	Acknowledged
ARO-03	Dangerous low level "call()"	Language Specific	Major	Resolved
ARO-04	Possible DOS attack on "makeTemplateRequest()" and "makeFullRequest()"	Logical Issue	Minor	Acknowledged
ARR-01	Centralization Risk	**Centralization / Privilege**	**Medium**	Acknowledged
ARR-02	"setRank()" and "decreaseSelfRank()" should be overridden and enforce checking on parameter "adminnedId"	Logical Issue	Minor	Resolved
MAO-01	Built-in function should be used for integer type value range	Language Specific	Informational	Resolved
RBS-01	Built-in function should be used for integer type value range	Language Specific	Informational	Resolved
RBS-02	Uint32 may be too short for timestamp	Logical Issue	Minor	Acknowledged
RBS-03	Centralization Risk	**Centralization / Privilege**	**Minor**	Acknowledged

ID	Title	Category	Severity	Status
RBS-04	"templateId" should be checked	Logical Issue	Minor	Resolved
RBS-05	Beacon should be checked	Logical Issue	Minor	Acknowledged
RBS-06	Wrong sponsorship checking	Logical Issue	Minor	Resolved
SRR-01	Built-in function should be used for integer type value range	Language Specific	Informational	Resolved
TUO-01	Function parameters should be checked	Logical Issue	Minor	Partially Resolved
WUO-01	Dangerous low level "call()"	Language Specific	Minor	Acknowledged
WUO-02	Possible DOS attack on "requestWithdrawal()"	Logical Issue	Minor	Acknowledged

GLOBAL-01 | Third Party Dependencies

Category	Severity	Location	Status
Volatile Code	Minor	Global	Acknowledged

Description

The contracts in scope is part of a complex system. It needs to interact with on-chain components such as `requester`, `sponsor`, `sponsorWallet`, and off-chain component `airnode`. In the real world, harmless data in part A might trigger an exploit on part B. The scope of the audit treats the connected entities as black boxes and assumes their functional correctness. For example, we assume the `airnode` correctly performs the authorization check and verifies received data before processing them, so that request data is not verified in the smart contract.

Recommendation

We understand that the business logic of the system requires the contract interaction with components out of the audit scope. We encourage the team to develop a plan to ensure the rest of the systems are securely build and adequately tested.

Alleviation

[API3 core tech team]: We acknowledge that the security of the protocol depends on

1. The accompanying oracle node software to be implemented correctly
2. The operator to use the node in the intended way

To mitigate potential vulnerabilities that can be caused by (1), we develop the node software in an open source way, following the best software development practices. In addition, we are planning to have its implementation audited.

(2) is a common issue for all oracle implementations, that is, one cannot trustlessly verify the integrity of an oracle. We mitigate this issue at the operational-level (i.e., this is outside the scope of this audit): We utilize first-party oracles (i.e., oracles operated by API providers), based on the fact that the requester would have to trust the API provider to operate the API honestly, and thus also having the API provider operate the oracle is the optimally trust-minimized configuration. Where applicable, the requester is recommended to use multiple of these first-party oracles and a consensus mechanism to improve the trustlessness further.

ARO-01 | "airnode" should be checked

Category	Severity	Location	Status
Logical Issue	Minor	rrp/AirnodeRrp.sol: 136	Resolved

Description

If "airnode" is zero address, it means "templateId" does not exist.

Recommendation

We advise the client to add check "airnode == address(0)", if it is true the code should revert. This can help reduce the amount of unqualified request.

Alleviation

[API3 core tech team]: We will implement this improvement, as attempting to use non-existent templates will probably be a common mistake and erroring earlier is preferable.

Addressed here:
https://github.com/api3dao/airnode/compare/3dfd88a627e058fb42883258ad755c1828d500d8..8467c9522e0c86dc73d84a94a170616c766f2c8c#diff-5f696e2f84f9cff7b62eecced80fe4a2a64c32098b68780f670ce2520374c27fR135

We also prevent templates with zero Airnode address being created here:
https://github.com/api3dao/airnode/compare/3dfd88a627e058fb42883258ad755c1828d500d8..8467c9522e0c86dc73d84a94a170616c766f2c8c#diff-1dddbfee34cd9053a76f0782d261bef93f6724d4a51260517405e316b9259da8R36

ARO-02 | "requestId" should include "airnode"

Category	Severity	Location	Status
Logical Issue	Minor	rrp/AirnodeRrp.sol: 191~195	Acknowledged

Description

When calculating "requestId", just like "templateId", "airnode" should be used to identify the airnode where the request should be sent.

Recommendation

We advise the client to use "airnode" when calculating "requestId".

Alleviation

[API3 core tech team]:

`airnode` was left out of the `requestId` calculation deliberately. This is because it is already used to calculate the respective `requestIdToFulfillmentParameters`, and its integrity is checked by the `onlyCorrectFulfillmentParameters()` modifier. (If `airnode` in the logs was tampered with, the respective `fulfill()` call would revert.) Therefore, we will not address this issue.

This issue brought to our attention that `sponsor` should be added to `requestId` calculation and validated at the node. Otherwise, the node has to trust the blockchain provider for the integrity of its value, which is slightly problematic because the `sponsor` value is in turn used to validate `sponsorWallet`:

https://github.com/api3dao/airnode/compare/3dfd88a627e058fb42883258ad755c1828d500d8..8467c9522e0c86dc73d84a94a170616c766f2c8c#diff-5f696e2f84f9cff7b62eecced80fe4a2a64c32098b68780f670ce2520374c27fR150
https://github.com/api3dao/airnode/compare/3dfd88a627e058fb42883258ad755c1828d500d8..8467c9522e0c86dc73d84a94a170616c766f2c8c#diff-5f696e2f84f9cff7b62eecced80fe4a2a64c32098b68780f670ce2520374c27fR217

ARO-03 | Dangerous low level "call()"

Category	Severity	Location	Status
Language Specific	Major	rrp/AirnodeRrp.sol: 258~266	Resolved

Description

Low-level "call()" will always succeed if the destination address does not exist or is an EOA. The "call()" in the code is used to call a user-specified address, which is very dangerous because it can call into privileged system special contracts like "AirnodeRrp" and authorizers to perform privileged operations.

Recommendation

We advise the client to use openzeppelin "Address" library for low-level calls and use a blacklist to prevent low-level calls from calling into privileged system special contracts like "AirnodeRrp" and authorizers, etc.

Alleviation

[API3 core tech team]: We disallowed `fulfillAddress` from being the contract address

https://github.com/api3dao/airnode/compare/3dfd88a627e058fb42883258ad755c1828d500d8..8467c9522e0c86dc73d84a94a170616c766f2c8c#diff-5f696e2f84f9cff7b62eecced80fe4a2a64c32098b68780f670ce2520374c27fR136
https://github.com/api3dao/airnode/compare/3dfd88a627e058fb42883258ad755c1828d500d8..8467c9522e0c86dc73d84a94a170616c766f2c8c#diff-5f696e2f84f9cff7b62eecced80fe4a2a64c32098b68780f670ce2520374c27fR203

We documented that the request fulfillment will succeed if `fulfillAddress` does not point to a contract
https://github.com/api3dao/airnode/compare/3dfd88a627e058fb42883258ad755c1828d500d8..8467c9522e0c86dc73d84a94a170616c766f2c8c#diff-5f696e2f84f9cff7b62eecced80fe4a2a64c32098b68780f670ce2520374c27fR251-R253

ARO-04 | Possible DOS attack on "makeTemplateRequest()" and "makeFullRequest()"

Category	Severity	Location	Status
Logical Issue	Minor	rrp/AirnodeRrp.sol: 112, 173	Acknowledged

Description

Anyone can create fake requests. On blockchain where the gas price is cheap, DOS attacks can generate excessive fake requests which airnode has to process.

Recommendation

We advise the client to add airnode-sponsor relationship functionality to on-chain such that smart contract can verify whether a sponsor is registered with an airnode.

Alleviation

[API3 core tech team]: The node makes lightweight off-chain computation and static calls to validate requests. The node is expected to be implemented and configured in a way to be able to do this even under an on-chain DOS attempt.

We advise the client to add airnode-sponsor relationship functionality to on-chain

We interpreted the above as having the authorizer checks as a step in the call that makes the request. We do not see this as a solution mainly because a whitelisted requester is not trusted not to attempt a DOS attack, but only to use the oracle services (e.g., because they made a small payment). Only being able to whitelist fully-trusted requesters would be impractical.

ARR-01 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	Medium	rrp/authorizers/Api3RequesterRrpAuthorizer.sol: 25	Acknowledged

Description

In the contract `Api3RequesterRrpAuthorizer`, the role `metaAdmin` has the authority over the following function:

- setRank()
- extendWhitelistExpiration()
- setWhitelistExpiration()
- setWhitelistStatusPastExpiration()

Any compromise to the `metaAdmin` account may allow the hacker to take advantage of this and change the whitelist to allow arbitrary accounts to call oracle.

Recommendation

We advise the client to carefully manage the `metaAdmin` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[API3 core tech team]: Renamed the Api3RequesterRrpAuthorizer contract as DaoRequesterRrpAuthorizer to emphasize that the `metaAdmin` of this contract must be a DAO.

ARR-02 | "setRank()" and "decreaseSelfRank()" should be overridden and enforce checking on parameter "adminnedId"

Category	Severity	Location	Status
Logical Issue	Minor	rrp/authorizers/Api3RequesterRrpAuthorizer.sol: 35~38	Resolved

Description

To make the code more robust, it should not rely on caller behavior to function properly. So "setRank()" and "decreaseSelfRank()" should also be overridden and checking on parameter "adminnedId" should be enforced in "getRank()", "setRank()", "decreaseSelfRank()" instead of ignoring it.

Recommendation

We advise the client to override functions "setRank()", "decreaseSelfRank()" and add check in "getRank()", "setRank()", "decreaseSelfRank()" to make sure "adminnedId == bytes32(0)".

Alleviation

[API3 core tech team]: This issue arose from trying to reuse a data structure in two contracts with very different mechanics. To address this, we refactored the underlying “adminnable” contracts. As a result, the interfaces of DaoRequesterRrpAuthorizer and AirnodeRequesterRrpAuthorizer are no longer weird.

MAO-01 | Built-in function should be used for integer type value range

Category	Severity	Location	Status
Language Specific	Informational	admin/MetaAdminnable.sol: 13	Resolved

Description

In solidity, system built-in "type(integerType).max" and "type(integerType).min" are used to get integer type value range. It is easy to get wrong when you calculate them by yourself.

Recommendation

We advise the client to use "type(uint256).max" as uint256 max value.

Alleviation

[API3 core tech team]: This contract no longer assigns the maximum unsigned integer value to `metaAdmin`, but has it pass `onlyWithRank` checks automatically https://github.com/api3dao/airnode/compare/3dfd88a627e058fb42883258ad755c1828d500d8..8467c9522e0c86dc73d84a94a170616c766f2c8c#diff-6435b6da1770cf73f43fe61910453dcdcdfa453dddda0989bdcfa2fcc9bf047cR20 As such, this issue is no longer relevant.

RBS-01 | Built-in function should be used for integer type value range

Category	Severity	Location	Status
Language Specific	Informational	rrp/requesters/RrpBeaconServer.sol: 31~33	Resolved

Description

In solidity, system built-in "type(integerType).max" and "type(integerType).min" are used to get integer type value range. It is easy to get wrong when you calculate them by yourself.

Recommendation

We advise the client to use "type(int224).max", "type(int224).min", "type(uint32).max"

Alleviation

[API3 core tech team]: Addressed here: https://github.com/api3dao/airnode/compare/3dfd88a627e058fb42883258ad755c1828d500d8..8467c9522e0c86dc73d84a94a170616c766f2c8c#diff-59f5b84df1692cd60a4f14f049b10e9b2c99049be76f345e415b464557b53e36R133-R134

RBS-02 | Uint32 may be too short for timestamp

Category	Severity	Location	Status
Logical Issue	Minor	rrp/requesters/RrpBeaconServer.sol: 27	Acknowledged

Description

According to Linux doc(https://man7.org/linux/man-pages/man2/time.2.html), "Applications intended to run after 2038 should use ABIs with time_t wider than 32 bits."

Recommendation

We advise the client to use at least "uint64" for timestamp.

Alleviation

[API3 core tech team]: We do prefer 64 bit-types for timestamps, but there is a tradeoff to be made here, as we also want a high overhead for `value` so that it does not overflow (or rather we do not even need to think about if it will overflow). The choice was deliberate here, so we will keep it.

RBS-03 | Centralization Risk

Category	Severity	Location	Status
Centralization / Privilege	**Minor**	rrp/requesters/RrpBeaconServer.sol: 42	Acknowledged

Description

In the contract `RrpBeaconServer`, the role `metaAdmin` has the authority over the following function:

- setRank()
- extendWhitelistExpiration()
- setWhitelistExpiration()
- setWhitelistStatusPastExpiration()

Any compromise to the `metaAdmin` account may allow the hacker to take advantage of this and change the whitelist to read beacon data.

Recommendation

We advise the client to carefully manage the `metaAdmin` account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol to be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., Multisignature wallets.

Indicatively, here is some feasible suggestions that would also mitigate the potential risk at the different level in term of short-term and long-term:

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key;
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.

Alleviation

[API3 core tech team]: The `metaAdmin` role is planned to be given to the API3 DAO.

RBS-04 | "templateId" should be checked

Category	Severity	Location	Status
Logical Issue	Minor	rrp/requesters/RrpBeaconServer.sol: 101	Resolved

Description

If "templateId == bytes32(0)", it means "requestId" is wrong or does not exist.

Recommendation

We advise the client to add a check "templateId == bytes32(0)", if it is true the code should revert.

Alleviation

[API3 core tech team]: Addressed here: https://github.com/api3dao/airnode/compare/3dfd88a627e058fb42883258ad755c1828d500d8..8467c9522e0c86dc73d84a94a170616c766f2c8c#diff-59f5b84df1692cd60a4f14f049b10e9b2c99049be76f345e415b464557b53e36R128

RBS-05 | Beacon should be checked

Category	Severity	Location	Status
Logical Issue	Minor	rrp/requesters/RrpBeaconServer.sol: 140	Acknowledged

Description

If "beacon.timestamp" is zero, it means either "templateId" is wrong or the beacon has never been updated.

Recommendation

We advise the client to add a check "beacon.timestamp == 0", if it is true the code should revert.

Alleviation

[API3 core tech team]: The beacon server returns the timestamp because it expects the client to use it. If we revert automatically, we would be disabling the client from handling it as an error case, e.g., “If the timestamp of templateId1 is 0, use templateId2 instead.” Therefore, we will not be implementing the recommendation (but we will document this as being the case).

Acknowledged, behavior documented here: https://github.com/api3dao/airnode/compare/3dfd88a627e058fb42883258ad755c1828d500d8..8467c9522e0c86dc73d84a94a170616c766f2c8c#diff-59f5b84df1692cd60a4f14f049b10e9b2c99049be76f345e415b464557b53e36R237-R240

RBS-06 | Wrong sponsorship checking

Category	Severity	Location	Status
Logical Issue	Minor	rrp/requesters/RrpBeaconServer.sol: 65~71	Resolved

Description

Sponsorship checking is done by AirnodeRrp and should not be done in the requester contract. It will result in rejecting all valid requests. (https://docs.api3.org/pre-alpha/protocols/request-response/endorsement.html)

Recommendation

We advise the client to remove the wrong sponsorship checking.

Alleviation

[API3 core tech team]: We re-implemented the beacon update permission logic, and replaced the check here https://github.com/api3dao/airnode/compare/3dfd88a627e058fb42883258ad755c1828d500d8..8467c9522e0c86dc73d84a94a170616c766f2c8c#diff-59f5b84df1692cd60a4f14f049b10e9b2c99049be76f345e415b464557b53e36R94-R97

SRR-01 | Built-in function should be used for integer type value range

Category	Severity	Location	Status
Language Specific	Informational	rrp/authorizers/SelfRequesterRrpAuthorizer.sol: 17	Resolved

Description

In solidity, system built-in "type(integerType).max" and "type(integerType).min" are used to get integer type value range. It is easy to get wrong when you calculate them by yourself.

Recommendation

We advise the client to use "type(uint256).max"

Alleviation

[API3 core tech team]: This contract no longer assigns the maximum unsigned integer value to `airnode`, but has it pass `onlyWithRank` checks automatically

https://github.com/api3dao/airnode/compare/3dfd88a627e058fb42883258ad755c1828d500d8..8467c9522e0c86dc73d84a94a170616c766f2c8c#diff-5f11089e0b8d2ff66ad8618ca1794247725f42b4eb0308b020bb3aa6e94993a2R22

As such, this issue is no longer relevant.

TUO-01 | Function parameters should be checked

Category	Severity	Location	Status
Logical Issue	Minor	rrp/TemplateUtils.sol: 32~33	Partially Resolved

Description

We assume that "airnode" must not be zero address and "endpointId" must not be zero bytes.

Recommendation

We advise the client to add checks to make sure "airnode" is not zero address and "endpointId" is not all zero bytes.

Alleviation

[API3 core tech team]: Since the solution to ARO-01 implies that zero `airnode` values are invalid, we will have to check for `airnode`. However, we would rather not check if `endpointId` is `bytes32(0)`, as we intend to use that value to signal that we want the Airnode to call `fulfill()` directly, without making an API call. We will document that `endpointId` not being checked is on purpose.

We disallowed `airnode` from being zero in TemplateUtils https://github.com/api3dao/airnode/compare/3dfd88a627e058fb42883258ad755c1828d500d8..8467c9522e0c86dc73d84a94a170616c766f2c8c#diff-1dddbfee34cd9053a76f0782d261bef93f6724d4a51260517405e316b9259da8R36 and all other contracts such as https://github.com/api3dao/airnode/compare/3dfd88a627e058fb42883258ad755c1828d500d8..8467c9522e0c86dc73d84a94a170616c766f2c8c#diff-8775d9a79085ac4ccb0d13e2107603c7ad7b0f79ee87915b6ff4916f753014b6R68

We documented that `endpointId` can be zero https://github.com/api3dao/airnode/compare/3dfd88a627e058fb42883258ad755c1828d500d8..8467c9522e0c86dc73d84a94a170616c766f2c8c#diff-5f696e2f84f9cff7b62eecced80fe4a2a64c32098b68780f670ce2520374c27fR184

WUO-01 | Dangerous low level "call()"

Category	Severity	Location	Status
Language Specific	Minor	rrp/WithdrawalUtils.sol: 78~79	Acknowledged

Description

Low level "call()" is used to call a user-specified address, which is very dangerous because it can call into privileged system special contracts like "AirnodeRrp" and authorizers to perform privileged operations.

Recommendation

We advise the client to use openzeppelin "Address" library for low-level calls and use a blacklist to prevent low-level calls from calling into privileged system special contracts like "AirnodeRrp" and authorizers, etc.

Alleviation

[API3 core tech team]: See ARO-03. We already use the same implementation from OpenZeppelin's Address.sol with the exception of the nice revert string, and we do not want it as a dependency unless absolutely required.

WUO-02 | Possible DOS attack on "requestWithdrawal()"

Category	Severity	Location	Status
Logical Issue	Minor	rrp/WithdrawalUtils.sol: 25	Acknowledged

Description

Anyone can create fake withdrawal requests. On blockchain where the gas fee is cheap, DOS attacks can generate excessive fake withdrawal requests which airnode has to process.

Recommendation

We advise the client to add airnode-sponsor relationship functionality to on-chain such that smart contract can verify whether a sponsor is registered with an airnode.

Alleviation

[API3 core tech team]: See ARO-04. In this case, only off-chain computation is required to detect an invalid withdrawal request, so it is less of a risk.

Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of private or delete.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK's prior written consent in each instance.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED "AS IS" AND "AS

AVAILABLE" AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER'S OR ANY OTHER PERSON'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING

MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.