hexens

API3 SMART CONTRACT AND WEB RESOURCES AUDIT REPORT

05.09.2022

CONTENTS

SUMMARY

SEVERITY	NUMBER OF FINDINGS
CRITICAL	1
HIGH	3
MEDIUM	1
LOW	0
INFORMATIONAL	2

TOTAL: 7

SCOPE

The analyzed resources are located on:
https://github.com/api3dao/api3-dao/commit/da7a1754e5cccac0bc0e382225b5648c90ba0aef
https://github.com/api3dao/api3-dao-dashboard/commit/bc8356f28ced64f971f1e035e6a0a2e5add9c7cd
https://api3.org

WEAKNESSES

This section contains the list of discovered weaknesses.

1. INCONSISTENCY BETWEEN METADATA AND ENCODED EVM SCRIPT: INVISIBLE CONTRACT CALL

SEVERITY: Critical

REMEDIATION: decodeEvmScript function before the line 216 decode execution must check whether the callData variable contains redundant bytes or not

STATUS: fixed in the following PR

DESCRIPTION:

decodeEvmScript function on line 203 in the file **encoding.ts** lets a malicious actor create a proposal without using application's UI, but with RPC calls to bypass the UI validations. He can also attach other contract's function call inside the proposal's EVM script. This works as the AragonOS's **execScript** function on line 33 in the file **CallsScript.sol** can execute one or many functions.

```
function execScript(bytes _script, bytes, address[] _blacklist) external isInitialized
returns (bytes) {
    uint256 location = SCRIPT_START_LOCATION; // first 32 bits are spec id
    while (location < _script.length) {
      // Check there's at least address + calldataLength available
      require(_script.length - location >= 0x18, ERROR_INVALID_LENGTH);

...

      bool success;
      assembly {
        success := call(
          sub(gas, 5000),      // forward gas left - 5000
          contractAddress,     // address
          0,                   // no value
          calldataStart,       // calldata start
....
          )
        }
        default { }
      }
    }
```

decodeEvmScript function on line 216 slices second contract's function call, so **decodeEvmScript** works without any exception.

```
const evmScriptPayload = utils.hexDataSlice(script, 4);
const callData = utils.hexDataSlice(evmScriptPayload, 24);
// Decode the parameters of the "execute" function:
//
https://github.com/aragon/aragon-apps/blob/631048d54b9cc71058abb8bd7
c17f6738755d950/apps/agent/contracts/Agent.sol#L70
const executionParameters = utils.defaultAbiCoder.decode(
['address', 'uint256', 'bytes'],
utils.hexDataSlice(callData, 4)
);
```

2. INCONSISTENCY BETWEEN METADATA'S FUNCTION SIGNATURE AND ENCOED EVM SCRIPT FUNCTION

SEVERITY: High

REMEDIATION: decodeEvmScript function must do a validation whether metadata's target signature is the same as in the EVM script's function after decoding

STATUS: fixed in the following PR

DESCRIPTION:

decodeEvmScript function on line 203 in the file encoding.ts lets a malicious actor create a proposal without using application's UI, but with RPC calls to bypass the UI validations, and inside the proposal he can fake the metadata. Created proposal's metadata function **signature(targetSignature)** can differ from the actual function inside the EVM script. So a normal user will vote for a proposal, but actually after the voting another function we be called.

3. EXPOSED SENDGRID API KEY

SEVERITY: High

PATH: api3.org/sendgrid.env

REMEDIATION: make the .env file private

STATUS: fixed

DESCRIPTION:

Sengrid is a service that is used for e-mail delivery. Below presented is a snippent of the leaked API key that can be used to authenticate into **API3's Sendgrid** account.

SG.x7*gc4uGNeebuQ_TiA**8HH-RyBPU**

Our team authenticated into the account using this token and listed the functionality that is available. Here's a snippet:

```
{“scopes”:[“alerts.create”,“alerts.read”,“alerts.update”,“alerts.delete”,“asm.groups.create”,“asm.groups.read”,“asm.groups.update”,“asm.groups.delete”,“asm.groups.suppressions.create”,“asm.groups.suppressions.read”,“asm.groups.suppressions.update”,“asm.groups.suppressions.delete”,“asm.suppressions.global.create”,
```

As a result, a malicious actor that got the token could, for example, send e-mails on behalf of **API3**.

4. QUORUM MAY GET INCREASINGLY HARDER TO REACH, UNTIL TOTAL GOVERNANCE DEADLOCK IN CASE OF MASSIVE UNSTAKE

SEVERITY: High

REMEDIATION: keep track of how many shares are scheduled for unstake in total, and subtract it from total voting power when creating the proposal snapshot

STATUS: acknowledged, see commentary

DESCRIPTION:

The user shares are immediately revoked, as well as any vote delegation, when the user schedules an unstake by calling **scheduleUnstake** function on line 73 in the file **StakeUtils.sol**, but the pool shares checkpoint is not updated. It gets updated when the user unstakes and triggers **updateCheckpointArray** on line 149, one week or more later. This design choice is duly motivated in the project documentation and understandably so in the context of rewards and fairness. The remaining users don't wish to share rewards nor give a voice to "programmed quitters", but the funds are still in the pool. This however poses a crucial problem in a bankrun scenario, which, however unlikely, might deal a fatal blow if not properly addressed.

Imagine that a very large portion of the user base decides to schedule their unstake at once. All those votes are lost during a week, while the total voting power is still the same. If a vote is started after that mass departure signal, it will be harder to reach quorum.

This gets increasingly bad the more users leave until it gets critical at **1 - min_acceptance_quorum** % of the users, at which point the proposal can never pass quorum, because there are simply not enough voters left.

Furthermore, users who scheduled their unstake could delay their unstake transaction indefinitely to continue blocking the governance. As long as they are willing to immobilize their stake, they can collectively keep any proposal from passing.

Commentary from client:
"Stakers being able to abstain to block proposals from passing is intended behavior. The staking rewards become withdrawable after 1 year, which is intended to discourage the stakers from mis-governing in general. As a separate note, being able to pass time-critical proposals is not considered to be a requirement due to the scale of the DAO (as in, it is too decentralized to hope to achieve this). Therefore, not being able to pass any proposals during a potential bank-run scenario is not a requirement, and not being able to do so is not an issue."

5. POSSIBLE EXECUTION BYPASS DURING VOTING PHASE

SEVERITY: Medium

REMEDIATION: require supportRequiredPct >= PCT_BASE / 2

STATUS: acknowledged, see commentary

DESCRIPTION:

The function **_canExecute** on lines 352, 107 and 88 systematically gates the only way to execute an EVMScript, **_unsafeExecuteVote**, on lines 327 and 334. In turn, on line 347, **_unsafeExecuteVote** calls runScript with the passed vote's EVMScript as argument, and without any additional input, nor any blacklisted address, meaning the script can call any contract, including the app itself, giving it dangerous power.

```
 bytes memory input = new bytes(0); // TODO: Consider input for voting scripts
        runScript(vote_.executionScript, input, new address[](0));
```

Scripts can be executed either after the voting phase if they avoid all the exit conditions, or during the vote, but only if they satisfy the "bypassing condition" of `_canExecute`, namely:

```
line 360 paraphrased:
IF yes_votes_now / total_shares_at_snapshot > supportRequired
ALLOW_EXECUTION
equivalent to :
IF current_quorum > supportRequired ALLOW_EXECUTION
```

```
function _canExecute(uint256 _voteId) internal view returns (bool) {
    Vote storage vote_ = votes[_voteId];

    if (vote_.executed) {
        return false;
    }

    // Voting is already decided
    if (_isValuePct(vote_.yea, vote_.votingPower, vote_.supportRequiredPct)) {
        return true;
    }

    // Vote ended?
    if (_isVoteOpen(vote_)) {
        return false;
    }
    // Has enough support?
    uint256 totalVotes = vote_.yea.add(vote_.nay);
    if (!_isValuePct(vote_.yea, totalVotes, vote_.supportRequiredPct)) {
        return false;
    }
    // Has min quorum?
    if (!_isValuePct(vote_.yea, vote_.votingPower, vote_.minAcceptQuorumPct)) {
        return false;
    }

    return true;
}
```

Following this logic, the current vote's `support >= quorum` systematically, since both depend on the number of yeas, and casted votes (divisor of support) can never exceed total voting power. E.g. for a total voting power of 1000, where 400 voted yea and 200 nay:

- support = 400 / 600 = 2/3 = 0.666

- quorum = 400 / 1000 = 2/5 = 0.4

For reasons described below, **supportRequiredPct** should never be below 50%. Let's assume for now it is exactly 50%, keeping us in the simple majority case.

When we take a closer look at the beginning of a vote, we notice that if it starts with a bunch of yea votes, support will keep being 1, while quorum slowly creeps towards **supportRequiredPct**. After that, each nay vote lowers the support, has no impact on quorum, and may suddenly block the proposal once nays surpass the level of yeas. During all this time the proposal cannot be executed though, except if `**yeas/total_vp > supportRequiredPct**`, meaning if ***quorum*** surpasses `**supportRequiredPct**`, because as seen above, this means automatically that *support* is also larger than `**supportRequiredPct**`, and implies that there is not enough voting power left to change that, so there's no need to count “nays” at all.

In the case of **supportRequiredPct** = 50%, the logic works. If there are more than 50% of yeas over the whole voting power, there are not enough votes left to counter them.

If **supportRequiredPct** < 50%, this governance becomes very dangerous, as it allows supporters to try and rush a proposal through before opposers vote, and bypass the proper support and quorum check.

If **supportRequiredPct** > 50%, then the bypass gets harder to reach, because a smaller number of nays is sufficient to reject the proposal.

Note that `**1 - support_required**` % of the total voting power is sufficient to reject a proposal (make it impossible to execute). For example, if the target support is 80%, as soon as 20% of the total voting power votes nay, the proposal is gone. In conclusion, raising `**supportRequiredPct**` does not just make proposals harder/longer to pass, it makes them, more importantly, easier/faster to reject.

Commentary from client:

"*On both of our voting apps, required support is initialized at 50% and has not been changed since. These values can be updated by a primary proposal (which requires a 50% quorum at the moment), and we will inform governance participants about the potential risks of decreasing this to a value below 50%.*

As a note, the described implementation belongs to the original Aragon Voting app, and supportRequiredPct is enforced to be larger than or equal to minAcceptQuorumPct (and not 50%). We suspect that this is because requiring the majority to be in favor of a proposal for it to pass is deemed to be too opinionated for a DAO framework, so the case described in this finding is probably intended behavior."

6. EXTERNALLINK COMPONENT DOESN'T VALIDATE URLS

SEVERITY: Informational

REMEDIATION: use validation for the URL's protocol

STATUS: fixed in the following PR

DESCRIPTION:

ExternalLink component in the file **external-link.tsx** contains an anchor tag, where the href's value can be externally controlled. A malicious actor can use "javascript:" protocol (javascript:alert(1) for instance) to trigger an **XSS**.

7. VULNERABLE DEPENDENCIES

SEVERITY: Informational

REMEDIATION: update the necessary dependencies

STATUS: fixed in the following PR

DESCRIPTION:

walletconnect/web3-provider dependency brings the following transitive dependencies: **ansi-regex, async, json-schema, path-parse**. In the yarn.lock file these dependencies are mentioned with vulnerable versions. It is recommended to update them to the following versions:

- **ansi-regex** to version 3.0.1, 4.1.1, 5.0.1, 6.0.1 or higher;
- **async** to version 2.6.4, 3.2.2 or higher;
- **json-schema** to version 0.4.0 or higher;
- **path-parse** to version 1.0.7 or higher.