Contracts vulnerabilities

Vulnerabilities list: #1

Involved contracts and level of the bugs

The present document aims to point out some vulnerabilities in the autonolas-registry contracts.

Vulnerabilities

1. `tokenURI` function

Severity: Low

The following function is implemented in the GenericRegistry contract:

```
function tokenURI(uint256 unitId) public view virtual override
returns (string memory)
```

This function is defined by the EIP-721 standard. The standard states that the function is supposed to throw if ***unitId*** is not a valid NFT. However, in our contract, the function does not revert if the ***unitId*** is out of bounds, but just returns the value of a string with the defined prefix and 64 zeros derived from a zero bytes32 value.

Therefore, we recommend checking the return value of this view function, and if the last 64 symbols are zero, consider it to be an invalid NFT. Also one might use the ***exists()*** function to preliminary check if the requested NFT Id exists.

2. `create` function

Severity: Low

The following function is implemented in the GnosisSafeMultisig contract:

```
function create(address[] memory owners, uint256 threshold, bytes
memory data) external returns (address multisig)
```

This function creates a Safe service multisig when the service is deployed. Since Autonolas protocol follows an optimistic design, none of the fields for the Safe multisig creation are restricted. This way, the service owner might pass the *payload* field as they feel fit for the purposes of the service multisig. That said, any possible malicious behavior can also be embedded in the *payload* value.

In the event of the intended malicious multisig creation, the Autonolas protocol is not affected, however, accounts interacting with the corresponding service might bear eventual consequences of such a setup.

We strongly recommend not abusing the *payload* field of the service multisig when deploying the service to perform any malicious actions. If the payload field affects a service in any way, an eventual service ranking implemented in tokenomics, is going to signal about the intended service misbehavior.