Report for:

Aventus Network Services

February 2020

Version: 2.0

Prepared By:	Extropy.IO
Email:	info@extropy.io
Telephone:	+44 1865261424

Executive Summary

This report presents the findings of the Aventus validator registration contract security assessment conducted on behalf of Aventus Network Services.

The assessment was conducted between 03/02/2021 and 05/02/2021 and was authorised by Aventus Network Services.

RETESTED 05/02/2021

This report has been updated as all the issues were retested following the client's fixes.

Overview

The audit only had Low or Informational issues to report, code quality was generally high, following best practices and appropriate to its purpose.

There was clearly a testing framework in place, the code supplied passed all those tests.

This audit should be viewed in the context of the previous audit since the contract audited here was derived from a previously audited contract.

RETESTED 05/02/2021

The recommendations were followed or work arounds decided upon

The following table breaks down the issues which were identified by phase and severity of risk.

Phase	Description	Critical	High	Medium	Low	Info	Total
1	Smart Contracts Audit	0	0	0	1	3	4
2	Retest	0	0	0	0	0	0
Total		0	0	0	0	0	0

Assessment Summary

All issues reported here are deemed to pose a low risk or are reported for information only. Nevertheless, it is recommended that these are reviewed and addressed so as to bring the Smart Contracts systems within scope into line with security best practice.

It is important to recognise that even low risk issues can be exploited in combination with other issues as part of a wider attack which seeks to compromise an environment or application. In addition, resolving lower risk issues can have the dual benefit of reducing the attractiveness of systems to opportunistic attackers as well as enhancing the overall security posture.

More detailed information on each of the issues which were identified is included in Section 2.

Table of Contents

Using This Report

To facilitate the dissemination of the information within this report throughout your organisation, this document has been divided into the following clearly marked and separable sections.

Document Breakdown		
0	Executive Summary	Management level, strategic overview of the assessment and the risks posed to the business
1	Technical Summary	An overview of the assessment from a more technical perspective, including a defined scope and any caveats which may apply
2	Technical Details	Detailed discussion (including evidence and recommendations) for each individual security issue which was identified
3	Supplemental Data	Any additional evidence which was too lengthy to include in Section 2
4	Appendices	This section usually includes the security tools which were used, outlines the assessment methodologies and lists the assessment team members

Document Control

Client Confidentiality

This document contains Client Confidential information and may not be copied without written permission.

Proprietary Information

The content of this document should be considered proprietary information and should not be disclosed outside of Aventus Network Services.
Extropy gives permission to copy this report for the purposes of disseminating information within your organisation or any regulatory agency.

Document Version Control			
Data Classification	Client Confidential		
Client Name	Aventus Network Services		
Document Title	Aventus Protocol Smart Contracts Audit		
Author	Extropy Audit Team		

Document History			
Issue No.	**Issue Date**	**Issued By**	**Change Description**
1.0	05/02/2021	Laurence Kirk	Released to client
2.0	05/02/2021	Laurence Kirk	Released to client

Document Distribution List	
Sam Newman	Project Manager, Aventus Network Services
Alan Vey	Aventus Network Services
Laurence Kirk	CEO, Extropy

1 Technical Summary

Extropy was contracted by Aventus Network Services to conduct a code review and smart contracts vulnerability assessment in order to identify security issues that could negatively affect Aventus Network Services' business or reputation if they led to the compromise or abuse of systems.

1.1 Scope

The scope of the audit was defined as the Solidity code in the `contracts` folder, the contents of which are shown below, the main contracts are highlighted in bold, but all of the dependencies were looked at as well:

```
contracts
├── IERC20.sol
├── IValidatorRegistration.sol
├── Owned.sol
├── ValidatorRegistration2.sol
```

The Audit was conducted from the commit f99b03399be0aa4020ad4f510b90623206bd2d13 from the aventus-validator-registration repository.

1.2 Design

The deployed ValidatorRegistration.sol is the updated version of YieldFarm.sol that we audited in Dec 2020.

The main changes/updates in ValidatorRegistration.sol:

- Updated solc v0.7.4 to v0.7.5
- Renamed the contract
- New functions:
    - function getNodeStake(uint8 node)
    - function getStakerBalance(uint8 node, address staker)
    - function removeStaker(address staker, uint8 _node)

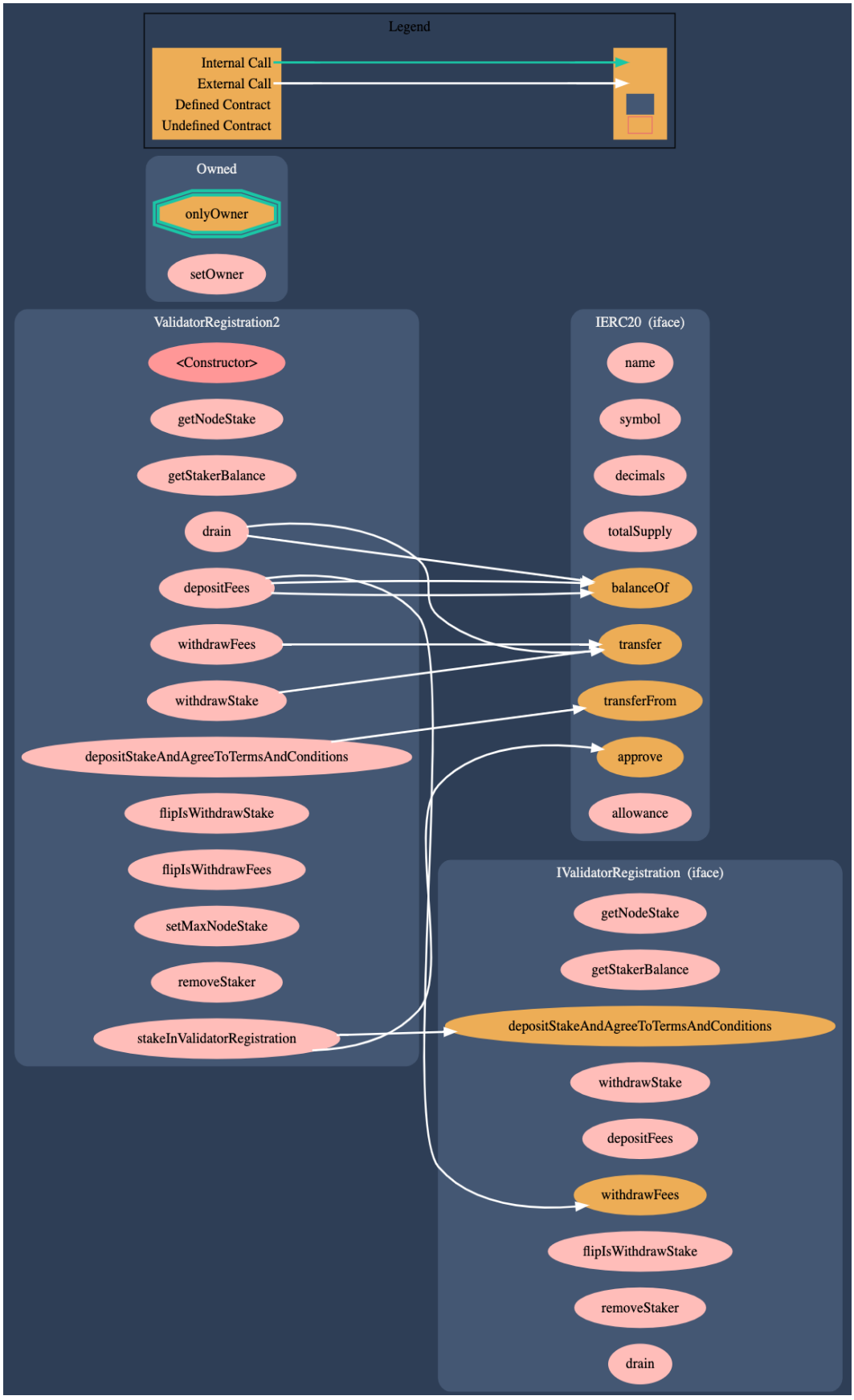
- Renamed depositStake() to depositStakeAndAgreeToTermsAndConditions()
- Changed visibility from public to external in the following functions (following our recommendations):
    - withdrawStake()
    - depositStake()
    - depositFees()
    - withdrawFees()
    - flipIsWithdrawStake()
    - drain()
- Added **require(amount > 0, "No amount to be withdrawn");** in withdraw fees.

1.3 ValidatorRegistration compared to ValidatorRegistration2

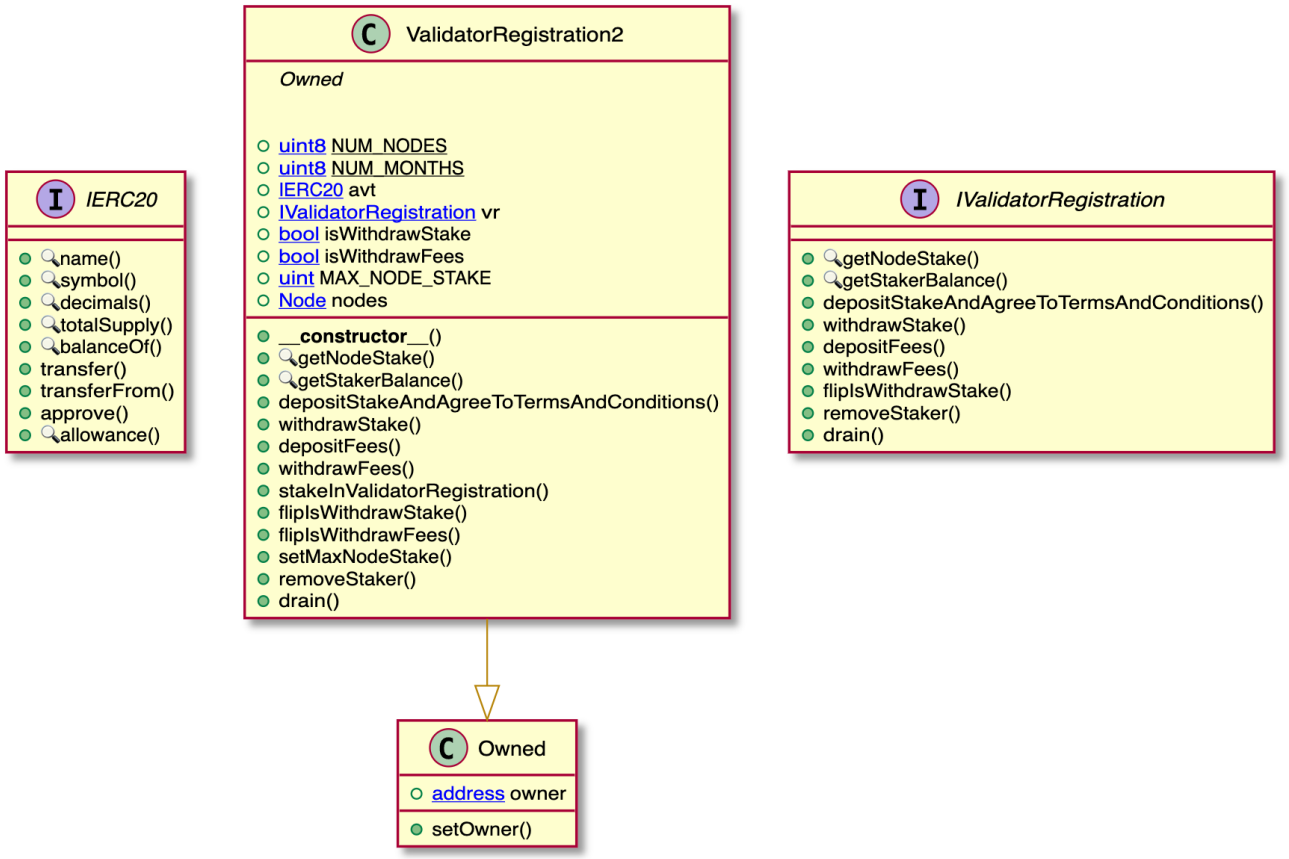
- added **import "./IValidatorRegistration.sol";**
- Rebranded the contract (appended a 2 to its name).
- Deleted the following constant and replaced it with an array of uints:
    - **uint public constant MAX_NODE_STAKE = 250000 ether;**

- Added the following interface:
    - **IValidatorRegistration public vr;**
- Constructor now needs a new argument (the address of the old contract), also sets isWithdrawFees to false:
- Updated instances of MAX_NODE_STAKE to use the new array of uints instead:
- Updated depositFees()

1.4 Contract Interaction



1.5 Contract Structure



1.6 Disclaimer

The audit makes no statements or warranty about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

2 Technical Findings – Smart Contracts Audit

The remainder of this document is technical in nature and provides additional detail about the items already discussed, for the purposes of remediation and risk assessment.

2.1 Function inputs should be checked for validity

Function inputs should be checked for validity	CVSSv2 Score: 1 - 3.9
Risk Rating	Low
Status	Closed

Description:

In the following functions some inputs are not checked for valid values.

```
function getNodeStake(uint8 node)
function getStakerBalance(uint8 node, address staker)
function depositStakeAndAgreeToTermsAndConditions(uint256 amount,uint8 node)
function withdrawStake(uint256 amount, uint8 node)
function depositFees(uint8 node, uint8 month)
function withdrawFees(uint8 _node, uint8 month)
function stakeInValidatorRegistration(uint8 node, uint256 amount)
function setMaxNodeStake(uint8 node, uint256 amount)
function removeStaker(address staker, uint8 _node)
function drain(address dst)
```

Recommendation:

Use a require statement to check, for example that the node parameter is within the correct range, or an address parameter is not the zero address.

RETESTED on 05 Feb 2021

Require statements have been added for example

require(node <= 9, "Invalid node index specified");

We would recommend using NUM_NODES -1 rather than 9, but this does not present a security problem.

Affects:

Smart Contract

ValidatorRegistration2

2.2 Use events to monitor contract activity

Use events to monitor contract activity	CVSSv2 Score: 0 – 0.9
Risk Rating	Informational
Status	Closed

Description:

For functions that change state an event should be emitted to describe the transition.

Recommendation:

Events should be considered for the following functions.

```
function withdrawStake(uint256 amount, uint8 node)
function depositFees(uint8 node, uint8 month)
function withdrawFees(uint8 _node, uint8 month)
function stakeInValidatorRegistration(uint8 node, uint256 amount)
function flipIsWithdrawStake()
function flipIsWithdrawFees()
function setMaxNodeStake(uint8 node, uint256 amount)
function removeStaker(address staker, uint8 _node)
function drain(address dst)
```

RETESTED on 05 Feb 2021

The client decided to use external tools to monitor the state rather than events.

Affects:

Smart Contract
ValidatorRegistration2

2.3 Consistent coding style

Consistent coding style	CVSSv2 Score: 0 – 0.9
Risk Rating	Informational
Status	Closed

Description:

A consistent coding style makes code easier to understand.

In functions withdrawFees, and removeStaker an intermediate variable is used for a specific node

Node storage node = nodes[_node];

in other functions simply

nodes[_node]

is used.

Recommendation:

Be consistent in the use of an intermediate variable.

RETESTED on 05 Feb 2021

The client has followed this recommendation

Affects:

Smart Contract
ValidatorRegistration2

2.4 Meaningful Error Messages

Meaningful Error Messages	CVSSv2 Score: 0 – 0.9
Risk Rating	Informational
Status	Closed

Description:

Require statements allow for an error message to be returned, a message should always be given, and the messages should be meaningful and specific.
The following require statements are missing an error message

```
function stakeInValidatorRegistration
        require(avt.approve(address(vr), amount));

function drain(address dst)
        require(avt.transfer(dst, avt.balanceOf(address(this))));
```

The following error message could be stated more clearly, for example
“Insufficient funds have been approved for the deposit”

```
function depositStakeAndAgreeToTermsAndConditions
 require(
            avt.transferFrom(msg.sender, address(this), amount),
            "Approved insufficient funds for deposit"
```

Recommendation:

Always provide an error message and give as much relevant information as possible in the error message.

RETESTED on 05 Feb 2021

The client has followed this recommendation

Affects:

Smart Contract
ValidatorRegistration2

3 Tailored Methodologies

3.1 Audit Goals

We will audit the code in accordance with the following criteria:

Sound Architecture

This audit includes assessments of the overall architecture and design choices. Given the subjective nature of these assessments, it will be up to the development team to determine whether any changes should be made.

Smart Contract Best Practices

This audit will evaluate whether the codebase follows the current established best practices for smart contract development.

Code Correctness

This audit will evaluate whether the code does what it is intended to do.

Code Quality

This audit will evaluate whether the code has been written in a way that ensures readability and maintainability.

Security

This audit will look for any exploitable security vulnerabilities, or other potential threats to the users.

Testing and testability

This audit will examine how easily tested the code is and review how thoroughly tested the code is. Although we have commented on the application design, issues of crypto-economics, game theory and suitability for business purposes as they relate to this project are beyond the scope of this audit.

3.2 Test Methodology

The security audit is performed in three phases:

Independent Code Review

The code was inspected separately by four team members checking for software errors and known vulnerabilities.

Static Analysis

The code was subjected to static analysis using Mythx and SmartCheck

Tests

The unit tests provided were run against the contract.

3.3 Tool List

The following tools were used during the assessment:

Tools Used	Description	Resources
MythX	Dynamic analysis	https://mythx.io/
SmartCheck	Static analysis	https://tool.smartdec.net/
Surya	Code visualizer	https://github.com/ConsenSys/surya
SWC Registry	Vulnerability database	https://swcregistry.io/