Smart Contract
Security Audit
V1

Zeus’ Bounty Smart Contract

https://zeusesbounty.io/

12/1/2022

https://saferico.com/

business@saferico.com
https://t.me/SFI_ANN

–

Table of Contents

Background

The purpose of the audit was to achieve the following:

- Ensure that the smart contract functions as intended.
- Identify potential security issues with the smart contract.

The information in this report should be used to understand the risk exposure of the smart contract, and as a guide to improve the security posture of the smart contract by remediating the issues that were identified.

Project Information

- **Website**: https://zeusesbounty.io/ / https://zeusesbounty.com/
- **Telegram group:** https://t.me/zeusbounty
- **Twitter**: https://twitter.com/ZeusBounty
- **Instagram**: https://www.instagram.com/zeusbounty/
- **YouTube**: https://www.youtube.com/c/ZeusBounty/featured
- **Facebook:** https://www.facebook.com/ZeusBounty
- **Platform**: Binance Smart Chain
- **Contract Address**: 0xFa7F765c0880e905Eb3937bf61B1aABd62c5604C

Contracts address deployed to BSC

Zeuses Bounty Smart Contract on BSCSCAN.COM
https://bscscan.com/address/0xFa7F765c0880e905Eb3937bf61B1aABd62c5604C

Executive Summary

According to our assessment, the customer`s solidity smart contract is **Secured**.because the issues fix in version 2.

Well Secured	
Secured	✔
Poor Secured	
Insecure	

Automated checks are with remix IDE. All issues were performed by the team, which included the analysis of code functionality, manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the Project Information section and all issues found are located in the audit overview section.

Team found 0 critical, 0 high, 0 medium, 3 low, 0 very low-level issues and 2 notes in all solidity files of the contract

The files:

zeus.sol

File and Function Level Report

File in Scope:

Contract Name	SHA 256 hash	Contract Address
zeus.sol	**87b4088fb4a9d04ff133228f66198f99366f7c147e043a7e3f999c303a3906e9**	0xFa7F765c0880e905Eb3937bf61B1aABd62c5604C

- Contract: ZeusesBounty
- Inherit: ReentrancyGuard, ChainlinkClient, Ownable
- Observation: All passed including security check
- Test Report: passed
- Score: passed
- Conclusion: passed

Function	Test Result	Type / Return Type	Score
stringToBytes32	✔	Read / public	**Passed**
owner	✔	Read / public	**Passed**
withdrawable	✔	Write / public	**Passed**
transferOwnership	✔	Write / public	**Passed**
register	✔	Write / payable	**Passed**
fulfillWithdrawable	✔	Write / public	**Passed**
deposit	✔	Write / payable	**Passed**

Issues Checking Status

No.	Issue Description	Checking Status
1	Compiler warnings.	**Passed**
2	Race conditions and Reentrancy. Cross-function race conditions.	**Passed**
3	Possible delays in data delivery.	**Passed**
4	Oracle calls.	**Passed**
5	Front running.	**Passed**
6	Timestamp dependence.	**Passed**
7	Integer Overflow and Underflow.	**Passed**
8	DoS with Revert.	**Passed**
9	DoS with block gas limit.	**Passed**
10	Methods execution permissions.	**Passed**
11	Economy model. If application logic is based on an incorrect economic model, the application would not function correctly and participants would incur financial losses. This type of issue is most often found in bonus rewards systems, Staking and Farming contracts, Vault and Vesting contracts, etc.	**Passed**
12	The impact of the exchange rate on the logic.	**Passed**
13	Private user data leaks.	**Passed**
14	Malicious Event log.	**Passed**
15	Scoping and Declarations.	**Passed**
16	Uninitialized storage pointers.	**Passed**
17	Arithmetic accuracy.	**Passed**
18	Design Logic.	**Passed**

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial functions
Medium	Medium-level vulnerabilities are important to fix; however, they can’t lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can’t have significant impact on execution
Note	Lowest-level vulnerabilities, code style violations and info statements can’t affect smart contract execution and can be ignored.

Audit Findings

Critical:

No critical severity vulnerabilities were found.

High:

No High severity vulnerabilities were found

Medium:

No Medium severity vulnerabilities were found.

Low:

#Contract has many pragmas and the main Pragma version not fixed

Description

It is a good practice to lock the solidity version for a live deployment (use 0.8.0 instead of ^0.8.0). contracts should be deployed with the same compiler version and flags that they have been tested the most with. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs. Contracts may also be deployed by others and the pragma indicates the compiler version intended by the original authors. and the contract has many libraries with many pragmas.

Remediation

Remove the ^ sign to lock the pragma version and make all libraries with same pragma.

Status Closed. Fixed in version 2.

#Missing receive ETH function

Description

The receive function is executed on a call to the contract with empty calldata. This is the function that is executed on plain Ether transfers (e.g. via .send() or .transfer()). If no such function exists, but a payable fallback function exists, the fallback function will be called on a plain Ether transfer. If neither a receive Ether nor a payable fallback function is present, the contract cannot receive Ether through regular transactions and throws an exception.

And the contract has only fallback function it will be better if we add the receive function.

Remediation

Add receive payable function in the contract.

Status: Closed. Fix in version 2.

#This declaration has the same name as another declaration

Description

In the smart contract there is 3 functions with same names with another declaration

```
function setSubnodeOwner(
    bytes32 node,
    bytes32 label,
    address owner
  ) external;

 function setResolver(bytes32 node, address resolver) external;

 function setOwner(bytes32 node, address owner) external;

 function setTTL(bytes32 node, uint64 ttl) external;
```

Remediation

Comment or delete unuse functions from the contracts.

Status: Closed. Fix in version 2.

Very Low:

No Very Low severity vulnerabilities were found.

Notes:

#Missing SPDX-License-Identifier:

Warning: SPDX license identifier not provided in source file. Before publishing, consider adding a comment containing "SPDX-License-Identifier: <SPDX-License>" to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. Please see https://spdx.org for more information.

Remediation

Add License Identifier

// SPDX-License-Identifier: UNLICENSE

Status: Closed. Fix in version 2.

#Unnecessary use of SafeMath

Description

Solidity version 0.8 was released with SafeMath checks inbuilt, we can avoid using an explicit safe math library.

Remediation

Remove SafeMath Library to save gas fees.

Status: closed. Fix in version 2.

Automatic Testing

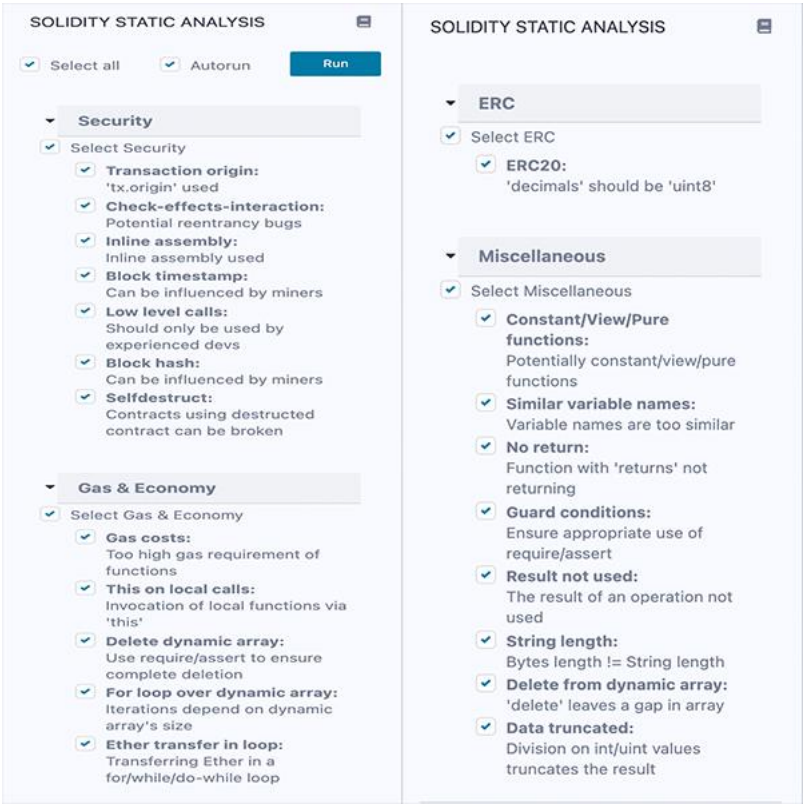
1- Check for security

87b4088fb4a9d04ff133228f66198f99366f7c147e043a7e3f999c303a3906e9

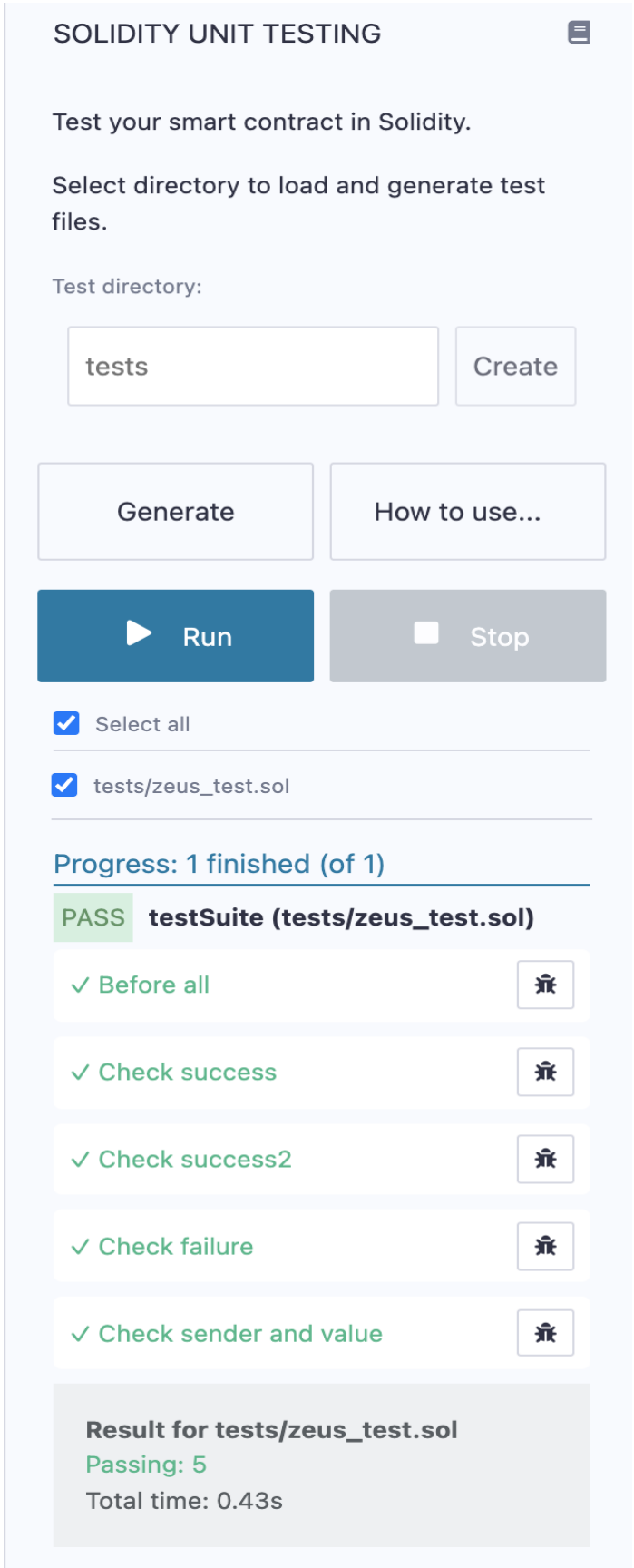
File: zeus.sol | Language: solidity | Size: 43857 bytes | Date: 2022-01-12T10:15:51.070Z

Critical	High	Medium	Low	Note
0	0	0	0	0

2- SOLIDITY STATIC ANALYSIS



3- SOLIDITY UNIT TESTING



Functions signature

```
40429946  =>
oracleRequest(address,uint256,bytes32,address,bytes4,uint256,uint256,bytes)
44955163  =>  write(buffer,uint256,bytes32,uint256)
50188301  =>  withdrawable()
66188463  =>  decreaseApproval(address,uint256)
3b3b57de  =>  addr(bytes32)
38cc4831  =>  getAddress()
4ab0d190  =>  fulfillOracleRequest(bytes32,uint256,address,bytes4,uint256,bytes32)
fa00763a  =>  isAuthorizedSender(address)
f3fef3a3  =>  withdraw(address,uint256)
6ee4d553  =>  cancelOracleRequest(bytes32,uint256,bytes4,uint256)
3c6d41b9  =>  operatorRequest(address,uint256,bytes32,bytes4,uint256,uint256,bytes)
6ae0bc76  =>  fulfillOracleRequest2(bytes32,uint256,address,bytes4,uint256,bytes)
902fc370  =>  ownerTransferAndCall(address,uint256,bytes)
8fde83a9  =>  distributeFunds(address,uint256[])
2408afaa  =>  getAuthorizedSenders()
ee56997b  =>  setAuthorizedSenders(address[])
a0042526  =>  getForwarder()
dd62ed3e  =>  allowance(address,address)
095ea7b3  =>  approve(address,uint256)
70a08231  =>  balanceOf(address)
313ce567  =>  decimals()
d73dd623  =>  increaseApproval(address,uint256)
06fdde03  =>  name()
95d89b41  =>  symbol()
18160ddd  =>  totalSupply()
a9059cbb  =>  transfer(address,uint256)
4000aea0  =>  transferAndCall(address,uint256,bytes)
23b872dd  =>  transferFrom(address,address,uint256)
06ab5923  =>  setSubnodeOwner(bytes32,bytes32,address)
1896f70a  =>  setResolver(bytes32,address)
5b0fc9c3  =>  setOwner(bytes32,address)
14ab9038  =>  setTTL(bytes32,uint64)
02571be3  =>  owner(bytes32)
0178b8bf  =>  resolver(bytes32)
16a25cbd  =>  ttl(bytes32)
b92f5e20  =>  init(buffer,uint256)
215b3e32  =>  fromBytes(bytes)
84993de3  =>  resize(buffer,uint256)
6d5433e6  =>  max(uint256,uint256)
0329aaf4  =>  truncate(buffer)
e1c2615c  =>  write(buffer,uint256,bytes,uint256)
e5882e91  =>  append(buffer,bytes,uint256)
d69190b6  =>  append(buffer,bytes)
9ebc63c9  =>  writeUint8(buffer,uint256,uint8)
aa813663  =>  appendUint8(buffer,uint8)
b81b0225  =>  writeBytes20(buffer,uint256,bytes20)
2249b392  =>  appendBytes20(buffer,bytes20)
44f40653  =>  appendBytes32(buffer,bytes32)
d6e21663  =>  writeInt(buffer,uint256,uint256,uint256)
854d9b30  =>  appendInt(buffer,uint256,uint256)
1e15446d  =>  encodeFixedNumeric(BufferChainlink.buffer,uint8,uint64)
af53ef09  =>  encodeIndefiniteLengthType(BufferChainlink.buffer,uint8)
6ef13d84  =>  encodeUInt(BufferChainlink.buffer,uint256)
3803cb34  =>  encodeInt(BufferChainlink.buffer,int256)
2df9cccd  =>  encodeBytes(BufferChainlink.buffer,bytes)
```

```
34a05063  =>  encodeBigNum(BufferChainlink.buffer,uint256)
9aeeba15  =>  encodeSignedBigNum(BufferChainlink.buffer,int256)
d994655b  =>  encodeString(BufferChainlink.buffer,string)
e715127c  =>  startArray(BufferChainlink.buffer)
161a73cf  =>  startMap(BufferChainlink.buffer)
dc0a81c5  =>  endSequence(BufferChainlink.buffer)
914b2bce  =>  initialize(Request,bytes32,address,bytes4)
973ab6e4  =>  setBuffer(Request,bytes)
03afcbe9  =>  add(Request,string,string)
6a2e9482  =>  addBytes(Request,string,bytes)
5bc4f3a3  =>  addInt(Request,string,int256)
ab6a2cb7  =>  addUint(Request,string,uint256)
c53370af  =>  addStringArray(Request,string,string[])
56db6353  =>  buildChainlinkRequest(bytes32,address,bytes4)
76b919d2  =>  buildOperatorRequest(bytes32,bytes4)
a06a20f5  =>  sendChainlinkRequest(Chainlink.Request,uint256)
7918fe63  =>  sendChainlinkRequestTo(address,Chainlink.Request,uint256)
cff3487b  =>  sendOperatorRequest(Chainlink.Request,uint256)
2f08c649  =>  sendOperatorRequestTo(address,Chainlink.Request,uint256)
af64137f  =>  _rawRequest(address,uint256,uint256,bytes)
0e088c98  =>  cancelChainlinkRequest(bytes32,uint256,bytes4,uint256)
9b70126d  =>  getNextRequestCount()
7a9b0412  =>  setChainlinkOracle(address)
31d0e3f5  =>  setChainlinkToken(address)
987c4311  =>  setPublicChainlinkToken()
7020b511  =>  chainlinkTokenAddress()
fe21f306  =>  chainlinkOracleAddress()
8387aa39  =>  addChainlinkExternalRequest(address,bytes32)
a8b65043  =>  useChainlinkWithENS(address,bytes32)
4d6f49b3  =>  updateChainlinkOracleWithENS()
e5f982a4  =>  validateChainlinkCallback(bytes32)
b67d77c5  =>  sub(uint256,uint256)
771602f7  =>  add(uint256,uint256)
119df25f  =>  _msgSender()
8b49d47e  =>  _msgData()
8da5cb5b  =>  owner()
f2fde38b  =>  transferOwnership(address)
89c19ddb  =>  concat(string,string)
cfb51928  =>  stringToBytes32(string)
2c6e7598  =>  toAsciiString(address)
69f9ad2f  =>  char(bytes1)
89b2521c  =>  withdrawable(address,bool)
8cf2dd80  =>  fulfillWithdrawable(bytes32,uint256,uint256)
e8c1b308  =>  register(address,address,uint256,string,string)
47e7ef24  =>  deposit(address,uint256)
```

Conclusion

The contracts are written systematically. Team found no critical issues. So, it is good to go for production.

Since possible test cases can be unlimited and developer level documentation (code flow diagram with function level description) not provided, for such an extensive smart contract protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan Everything.

Security state of the reviewed contract is “secured”.

- ✔ No mint function.
- ✔ No volatile code.
- ✔ Not many high severity issues were found.

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as of the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against the team on the basis of what it says or doesn't say, or how team produced it, and it is important for you to conduct your own independent investigations before making any decisions. team go into more detail on this in the below disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Saferico and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (Saferico s) owe no duty of care towards you or any other person, nor does Saferico make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Saferico hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Saferico hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Saferico, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.