Boson Protocol

Smart Contract Audit Report

October 31, 2021

Introduction

1. About Boson Protocol

Boson Protocol's vision is to provide for a decentralized commerce ecosystem by funding and enabling the development of a stack of specialized applications to disrupt, demonopolize and democratize commerce. Boson Protocol enables this via an open tokenized economy of Things which:

1. Automates the redemption of digital rights for physical assets using NFTs and carefully-designed deposit transfers.
2. Disrupts e-commerce platforms by tokenizing Things and their data within a liquid digital market, built on DeFi.

This is enabled by a design with five modular and substitutable components. The first of these components is a commitment to perform a future commercial exchange represented as a tokenized voucher. Second, a core mechanism for autonomous coordination of commercial exchange. Third, a token model for incentivizing actors, and for capturing and distributing value. Fourth, a Web3 data marketplace for monetizing data. And finally, an evolving governance system for directing and controlling the protocol throughout its lifecycle.

Visit https://www.bosonprotocol.io/ to know more about.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 105+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit http://immunebytes.com/ to know more about the services.

Documentation Details

The TrueFi team has provided the following doc for the purpose of audit:

1. https://docs.bosonprotocol.io/introduction/

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: Boson Protocol
- Contracts Names: BosonRouter, Cashier, VoucherKernel, DaiTokenWrapper, ERC1155ERC721, ERC1155NonTransferable, Gate, TokenRegistry, VoucherKernel
- Languages: Solidity(Smart contract), Typescript (Unit Testing)
- Github commit hash for audit: 5d175848db1beea65f5e12706684c02c4529ec2d
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck, SFuzz

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
    a. Correctness
    b. Readability
    c. Sections of code with high complexity
    d. Quantity and quality of test coverage

Security Level References

Every issue in this report was assigned a severity level from the following:

Admin/Owner Privileges can be misused either intentionally or unintentionally.

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	High	Medium	Low
Open	1	-	4
Closed	-	-	-

High Severity Issues

1. **VoucherKernel.sol : cancelOrFault() function includes invalid access control in the function Line no - 598**

   Explanation:
   The **cancelOrFault()** function in the contract allows the seller to admit to a fault and stop the deal. Quite similar to most of the functions in the current architecture of the protocol, this function is also supposed to be accessed via the BosonRouter contract.

   However, no such access control modifier was found to be associated with the **cancelOrFault()** function. This allows anyone to trigger this function and leads to an unwanted scenario where the function can be invoked on behalf of any seller as there is no adequate access control assigned to the function.

   Recommendation:
   The function must include an **onlyFromRouter()** modifier to ensure that it can only be called from the router contract.

Medium severity issues

No issues were found.

Low severity issues

1. **VoucherKernel.sol: Use of Require statements should be preferred over IF-ELSE Statement. Line: 874-879**

   Explanation:
   The **triggerFinalizeVoucher()** function in the contract aims to mark a **final** status to the given voucher token.
   Before assigning the **final** status to the given voucher Id, the function involves a series of checks to ensure whether or not the status can be assigned to the given voucher ID. This is done by assigning **TRUE** to a local boolean variable **(mark)**, which indicates that the status can be marked as **FINAL** to the given voucher ID.

   However, as a final step, an IF statement is included to check if the **mark** variable is TRUE so that the status of the voucher token can be changed to **FINAL**.

```
873
874         if (mark) {
875             vouchersStatus[_tokenIdVoucher].status = determineStatus(
876                 tStatus,
877                 IDX_FINAL
878             );
879             emit LogFinalizeVoucher(_tokenIdVoucher, msg.sender);
880         }
881     }
```

In such scenarios of strict validations, where the further execution of a function strictly depends on a value, it's comparatively effective to use **require statements** instead of IF statements.

Recommendation:
The validation in the **triggerFinalizeVoucher()** function, before updating the status of the voucher token, can be modified with a require statement as follows:

```
require(mark, "Status cannot be set to FINAL");

vouchersStatus[_tokenIdVoucher].status = determineStatus(
  tStatus,
  IDX_FINAL
);
emit LogFinalizeVoucher(_tokenIdVoucher, msg.sender);
```

2. **BosonRouter.sol: Invalid Error message found in require statement**
   Line no - 598

Explanation:
The requestVoucherTKNTKNSameWithPermit() function in the BosonRouter.
The smart contract includes a require statement to check whether or not the token deposit address and the token price address are similar.

However, the error message in this statement mentions IC, i.e., Invalid Caller.
This is not an adequate error message as the require statement's condition doesn't really involve any check on the caller of the function.

Recommendation:
It's recommended to include adequate error messages in the **require** statements.

3. **Absence of Error messages in Require Statements**
   Line no - 788, 824

   Explanation:
   The **Cashier** contract includes a few require statements, at the above-mentioned lines, that don't contain any error message.

   While this makes it troublesome to detect the reason behind a particular function revert, it also reduces the readability of the code.

   Recommendation:
   Error Messages must be included in every **require** statement in the contract

4. **TokenRegistry.sol: Adequate Input or Range validations not found**
   Line no - 34-37, 44-52

   Explanation:
   The **setETHLimit()** and **setTokenLimit()** in the TokenRegistry contract, do not implement proper input validations for the **uint256** type argument, i.e., **_newLimit**.

   Moreover, the functions do not involve any lower or upper threshold for this value which might result in an unexpected scenario if an inappropriate argument is mistakenly passed to the function.

   Recommendation:
   The above-mentioned functions should include effective validations to ensure no invalid uint argument is passed to the function.

Recommendations

1. **Cashier.sol: Redundant initialization of Boolean Variable**
   Line no - 85

   The Cashier smart contract involves the redundant update of the boolean state variable, **disasterState** to **False** in the constructor of the contract.

   A boolean variable is by default initialized to **FALSE** whereas a uint256 is initialized to **ZERO**. Hence, such state variables do not need to be initialized explicitly.

   Recommendation:
   Redundant initialization of state variables should be avoided.

2. **Cashier.sol: “emit” keyword not used during event emissions**
   Line no - 329, 377, 469, 475, 481, 567, 574, 621, 666, 711,117, 132, 152

   Explanation:
   The events emitted at the line numbers mentioned above do not use the **emit** keyword.

   Recommendation:
   It is recommended to include the **emit** keyword every time an event is emitted in the contract.

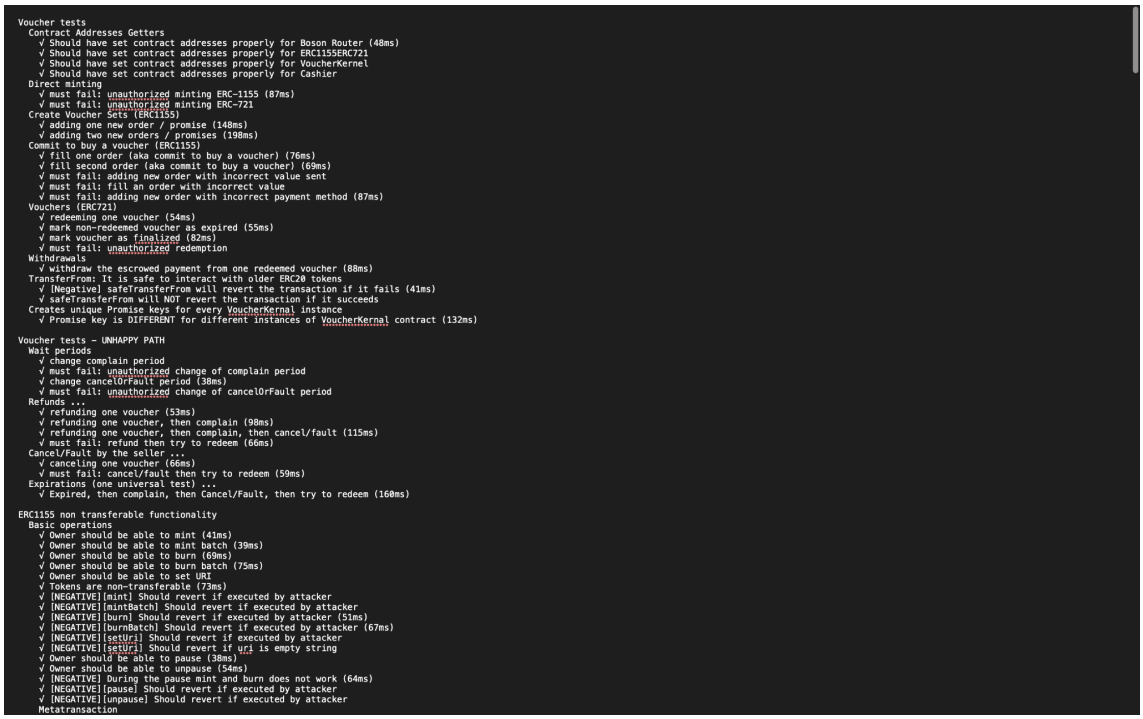
3. **ERC1155ERC721.sol: Inadequate address validation before approval.**
   Line no - 231-244

   Explanation:
   The **approve()** function at the above-mentioned line number does not include zero address validations for the address being passed as arguments.

   Recommendation:
   A require statement should be included in such functions to ensure no zero address is passed in the arguments.

Unit Test

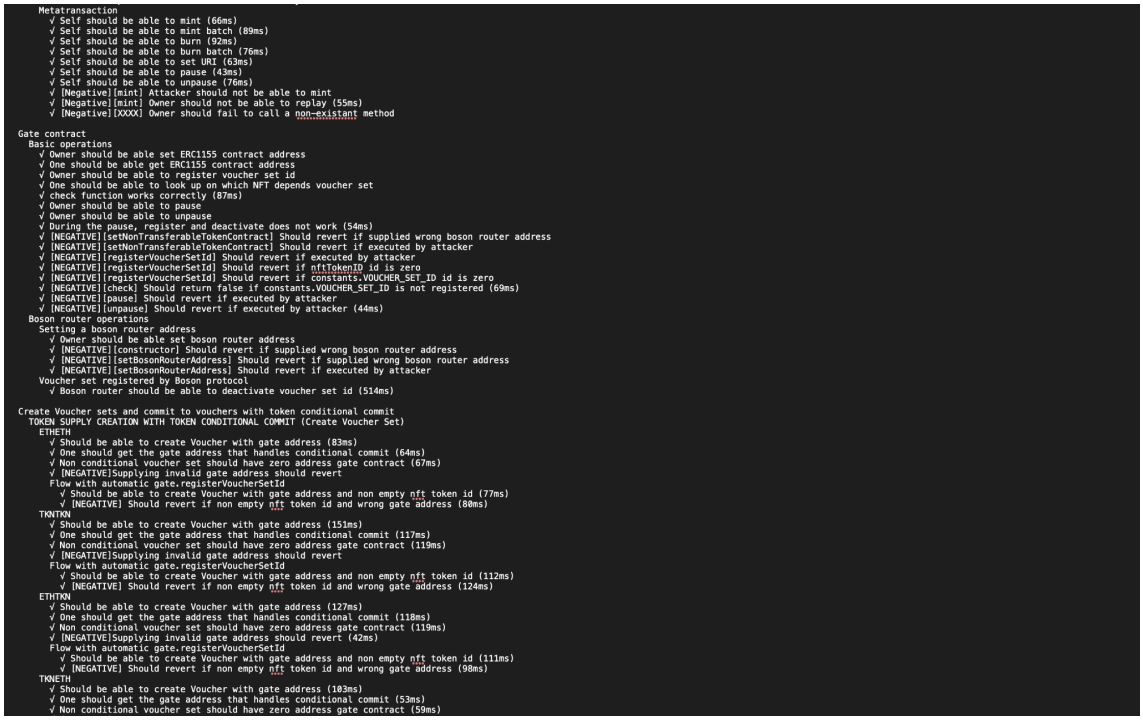
All unit tests provided by the team are passing without issues.



```
Voucher tests
  Contract Addresses Getters
    √ Should have set contract addresses properly for Boson Router (48ms)
    √ Should have set contract addresses properly for ERC1155ERC721
    √ Should have set contract addresses properly for VoucherKernel
    √ Should have set contract addresses properly for Cashier
  Direct minting
    √ must fail: unauthorized minting ERC-1155 (87ms)
    √ must fail: unauthorized minting ERC-721
  Create Voucher Sets (ERC1155)
    √ adding one new order / promise (148ms)
    √ adding two new orders / promises (198ms)
  Commit to buy a voucher (ERC1155)
    √ fill one order (aka commit to buy a voucher) (76ms)
    √ fill second order (aka commit to buy a voucher) (69ms)
    √ must fail: adding new order with incorrect value sent
    √ must fail: fill an order with incorrect value
    √ must fail: adding new order with incorrect payment method (87ms)
  Vouchers (ERC721)
    √ redeeming one voucher (54ms)
    √ mark non-redeemed voucher as expired (55ms)
    √ mark voucher as finalized (82ms)
    √ must fail: unauthorized redemption
  Withdrawals
    √ withdraw the escrowed payment from one redeemed voucher (88ms)
  TransferFrom: It is safe to interact with older ERC20 tokens
    √ [Negative] safeTransferFrom will revert the transaction if it fails (41ms)
    √ safeTransferFrom will NOT revert the transaction if it succeeds
  Creates unique Promise keys for every VoucherKernal instance
    √ Promise key is DIFFERENT for different instances of VoucherKernal contract (132ms)

Voucher tests - UNHAPPY PATH
  Wait periods
    √ change complain period
    √ must fail: unauthorized change of complain period
    √ change cancelOrFault period (38ms)
    √ must fail: unauthorized change of cancelOrFault period
  Refunds ...
    √ refunding one voucher (53ms)
    √ refunding one voucher, then complain (98ms)
    √ refunding one voucher, then complain, then cancel/fault (115ms)
    √ must fail: refund then try to redeem (66ms)
  Cancel/Fault by the seller ...
    √ canceling one voucher (66ms)
    √ must fail: cancel/fault then try to redeem (59ms)
  Expirations (one universal test) ...
    √ Expired, then complain, then Cancel/Fault, then try to redeem (160ms)

ERC1155 non transferable functionality
  Basic operations
    √ Owner should be able to mint (41ms)
    √ Owner should be able to mint batch (39ms)
    √ Owner should be able to burn (69ms)
    √ Owner should be able to burn batch (75ms)
    √ Owner should be able to set URI
    √ Tokens are non-transferable (73ms)
    √ [NEGATIVE][mint] Should revert if executed by attacker
    √ [NEGATIVE][mintBatch] Should revert if executed by attacker
    √ [NEGATIVE][burn] Should revert if executed by attacker (51ms)
    √ [NEGATIVE][burnBatch] Should revert if executed by attacker (67ms)
    √ [NEGATIVE][setUri] Should revert if executed by attacker
    √ [NEGATIVE][setUri] Should revert if uri is empty string
    √ Owner should be able to pause (38ms)
    √ Owner should be able to unpause (54ms)
    √ [NEGATIVE] During the pause mint and burn does not work (64ms)
    √ [NEGATIVE][pause] Should revert if executed by attacker
    √ [NEGATIVE][unpause] Should revert if executed by attacker
    Metatransaction
```



```
    Metatransaction
      √ Self should be able to mint (66ms)
      √ Self should be able to mint batch (89ms)
      √ Self should be able to burn (92ms)
      √ Self should be able to burn batch (76ms)
      √ Self should be able to set URI (63ms)
      √ Self should be able to pause (43ms)
      √ Self should be able to unpause (76ms)
      √ [Negative][mint] Attacker should not be able to mint
      √ [Negative][mint] Owner should not be able to replay (55ms)
      √ [Negative][XXXX] Owner should fail to call a non-existant method

Gate contract
  Basic operations
    √ Owner should be able set ERC1155 contract address
    √ One should be able get ERC1155 contract address
    √ Owner should be able to register voucher set id
    √ One should be able to look up on which NFT depends voucher set
    √ check function works correctly (87ms)
    √ Owner should be able to pause
    √ Owner should be able to unpause
    √ During the pause, register and deactivate does not work (54ms)
    √ [NEGATIVE][setNonTransferableTokenContract] Should revert if supplied wrong boson router address
    √ [NEGATIVE][setNonTransferableTokenContract] Should revert if executed by attacker
    √ [NEGATIVE][registerVoucherSetId] Should revert if executed by attacker
    √ [NEGATIVE][registerVoucherSetId] Should revert if nftTokenID id is zero
    √ [NEGATIVE][registerVoucherSetId] Should revert if constants.VOUCHER_SET_ID id is zero
    √ [NEGATIVE][check] Should return false if constants.VOUCHER_SET_ID is not registered (69ms)
    √ [NEGATIVE][pause] Should revert if executed by attacker
    √ [NEGATIVE][unpause] Should revert if executed by attacker (44ms)
  Boson router operations
    Setting a boson router address
      √ Owner should be able set boson router address
      √ [NEGATIVE][constructor] Should revert if supplied wrong boson router address
      √ [NEGATIVE][setBosonRouterAddress] Should revert if supplied wrong boson router address
      √ [NEGATIVE][setBosonRouterAddress] Should revert if executed by attacker
    Voucher set registered by Boson protocol
      √ Boson router should be able to deactivate voucher set id (514ms)

Create Voucher sets and commit to vouchers with token conditional commit
  TOKEN SUPPLY CREATION WITH TOKEN CONDITIONAL COMMIT (Create Voucher Set)
    ETHETH
      √ Should be able to create Voucher with gate address (83ms)
      √ One should get the gate address that handles conditional commit (64ms)
      √ Non conditional voucher set should have zero address gate contract (67ms)
      √ [NEGATIVE]Supplying invalid gate address should revert
      Flow with automatic gate.registerVoucherSetId
        √ Should be able to create Voucher with gate address and non empty nft token id (77ms)
        √ [NEGATIVE] Should revert if non empty nft token id and wrong gate address (80ms)
    TKNTKN
      √ Should be able to create Voucher with gate address (151ms)
      √ One should get the gate address that handles conditional commit (117ms)
      √ Non conditional voucher set should have zero address gate contract (119ms)
      √ [NEGATIVE]Supplying invalid gate address should revert
      Flow with automatic gate.registerVoucherSetId
        √ Should be able to create Voucher with gate address and non empty nft token id (112ms)
        √ [NEGATIVE] Should revert if non empty nft token id and wrong gate address (124ms)
    ETHTKN
      √ Should be able to create Voucher with gate address (127ms)
      √ One should get the gate address that handles conditional commit (118ms)
      √ Non conditional voucher set should have zero address gate contract (119ms)
      √ [NEGATIVE]Supplying invalid gate address should revert (42ms)
      Flow with automatic gate.registerVoucherSetId
        √ Should be able to create Voucher with gate address and non empty nft token id (111ms)
        √ [NEGATIVE] Should revert if non empty nft token id and wrong gate address (98ms)
    TKNETH
      √ Should be able to create Voucher with gate address (103ms)
      √ One should get the gate address that handles conditional commit (53ms)
      √ Non conditional voucher set should have zero address gate contract (59ms)
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

```
Cashier && VK
  Pausing Scenarios
    BOSON ROUTER
      COMMON PAUSING
        √ Should not be paused on deployment
        √ Owner should pause the contract
        √ Owner should unpause the contract (69ms)
        √ [NEGATIVE] Attacker should not be able to pause the contract
        √ [NEGATIVE] Attacker should not be able to unpause the contract (40ms)
      ETHETH
        √ [NEGATIVE] Should not create voucher supply when contract is paused (54ms)
        √ Should create voucher supply when contract is unpaused (83ms)
        √ [NEGATIVE] Should not create voucherID from Buyer when paused (131ms)
      [WITH PERMIT]
        ETHTKN
          √ [NEGATIVE] Should not create voucher supply when contract is paused (64ms)
          √ Should create voucher supply when contract is unpaused (101ms)
          √ [NEGATIVE] Should not create voucherID for Buyer when paused (180ms)
        TKNETH
          √ [NEGATIVE] Should not create voucher supply when contract is paused (48ms)
          √ Should create voucher supply when contract is unpaused (100ms)
          √ [NEGATIVE] Should not create voucherID for Buyer when paused (159ms)
        TKNTKN
          √ [NEGATIVE] Should not create voucher supply when contract is paused (66ms)
          √ Should create voucher supply when contract is unpaused (95ms)
          √ [NEGATIVE] Should not create voucherID for Buyer when paused (205ms)
    VOUCHER KERNEL
      COMMON PAUSING
        √ Should not be paused on deployment
        √ Should be paused from BR (44ms)
        √ Should be unpaused from BR (84ms)
        √ [NEGATIVE] Pause should not be called directly
        √ [NEGATIVE] Unpause should not be called directly
      ETHETH
        √ [NEGATIVE] Should not process refund when paused (124ms)
        √ [NEGATIVE] Should not process complain when paused (173ms)
        √ [NEGATIVE] Should not process redeem when paused (130ms)
        √ [NEGATIVE] Should not process cancel when paused (140ms)
      [WITH PERMIT]
        ETHTKN
          √ [NEGATIVE] Should not process refund when paused (173ms)
          √ [NEGATIVE] Should not process complain when paused (202ms)
          √ [NEGATIVE] Should not process redeem when paused (180ms)
          √ [NEGATIVE] Should not process cancel when paused (208ms)
        TKNETH
          √ [NEGATIVE] Should not process refund when paused (162ms)
          √ [NEGATIVE] Should not process complain when paused (187ms)
          √ [NEGATIVE] Should not process redeem when paused (116ms)
          √ [NEGATIVE] Should not process cancel when paused (164ms)
        TKNTKN
          √ [NEGATIVE] Should not process refund when paused (185ms)
          √ [NEGATIVE] Should not process complain when paused (256ms)
          √ [NEGATIVE] Should not process redeem when paused (191ms)
          √ [NEGATIVE] Should not process cancel when paused (225ms)
        TKNTKN Same
          √ [NEGATIVE] Should not process refund when paused (173ms)
          √ [NEGATIVE] Should not process complain when paused (190ms)
          √ [NEGATIVE] Should not process redeem when paused (161ms)
          √ [NEGATIVE] Should not process cancel when paused (178ms)
    CASHIER
      COMMON PAUSING
        √ Should not be paused on deployment
        √ Should be paused from BR (44ms)
        √ Should be unpaused from BR (85ms)
        √ [NEGATIVE] Pause should not be called directly
        √ [NEGATIVE] Unpause should not be called directly
        √ Owner should set the Cashier to disaster state (55ms)
        √ Should not be unpaused after disaster
      ETHETH
```

```
    ERC1155721
      √ Owner should be the deployer
      √ Owner should be able to set VK address
      √ [NEGATIVE][setVoucherKernelAddress] Should revert if executed by attacker
      √ [NEGATIVE][setVoucherKernelAddress] Should revert if ZERO address is provided
      √ Owner should be able to set Cashier address
      √ [NEGATIVE][setCashierAddress] Attacker should not be able to set Cashier address
      √ [NEGATIVE][setCashierAddress] Owner should not be able to set ZERO Cashier address
    VoucherKernel
      √ Owner should be the deployer
      √ Owner should be able to set Cashier address
      √ [NEGATIVE][setCashierAddress] Attacker should not be able to set Cashier address
      √ [NEGATIVE][setCashierAddress] Owner should not be able to set ZERO Cashier address
      √ Owner should be able to set BR address
      √ [NEGATIVE][setBosonRouterAddress] Should revert if executed by attacker
      √ [NEGATIVE][setBosonRouterAddress] Should revert if ZERO address is provided
      √ [NEGATIVE][setBosonRouterAddress] Should revert if ZERO address is provided at deployment

  Token Wrappers
    DAI Token Wrapper
      √ Should allow owner to set the token address
      √ Should call permit on the DAI token (262ms)
      √ Should call permit on the DAI token if deadline is zero (217ms)
      √ Should revert if token address is zero when contract is deployed (68ms)
      √ Should revert if owner sets token address to zero address
      √ Should revert if attacker tries to set token address (55ms)
      √ Should revert when token owner address is zero address (191ms)
      √ Should revert when token spender address is zero address (188ms)
      √ Should revert if deadline has expired (192ms)
      √ Should revert if signatue portion r is invalid (203ms)
      √ Should revert if signatue portion s is invalid (208ms)
      √ Should revert if the DAI token reverts (173ms)

  Create Voucher sets and commit to vouchers with token wrapper
    TOKEN SUPPLY CREATION WITH TOKEN WRAPPER (Create Voucher Set)
      [WITH PERMIT]
        ETHTKN
          √ Should emit the correct events and set correct state (57ms)
          √ Should update escrow correctly
          √ Should create payment method ETHTKN
          √ [NEGATIVE] Should revert if token doesn not have a registered token wrapper (226ms)
          √ [NEGATIVE] Should revert if token wrapper reverts because of invalid deadline (266ms)
          √ [NEGATIVE] Should revert if token wrapper reverts because of invalid signature (258ms)
        TKNTKN
          √ Should emit the correct events and set correct state (54ms)
          √ Should update escrow correctly
          √ Should create payment method TKNTKN
    VOUCHER CREATION (Commit to buy)
      [WITH PERMIT]
        ETHTKN
          √ Should emit the correct events and set correct state
          √ Should update escrow correctly
        TKNTKN
          √ Should emit the correct events and set correct state
          √ Should update Cashier contract's token balance correctly
          √ Should update escrow correctly
        TKNTKN Same
          √ Should emit the correct events and set correct state
          √ Should update escrow correctly
          √ [NEGATIVE] Should revert if Price Token and Deposit Token are diff contracts (348ms)
        TKNETH
          √ Should emit the correct events and set correct state
          √ Should update escrow correctly


  537 passing (6m)
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Coverage Report

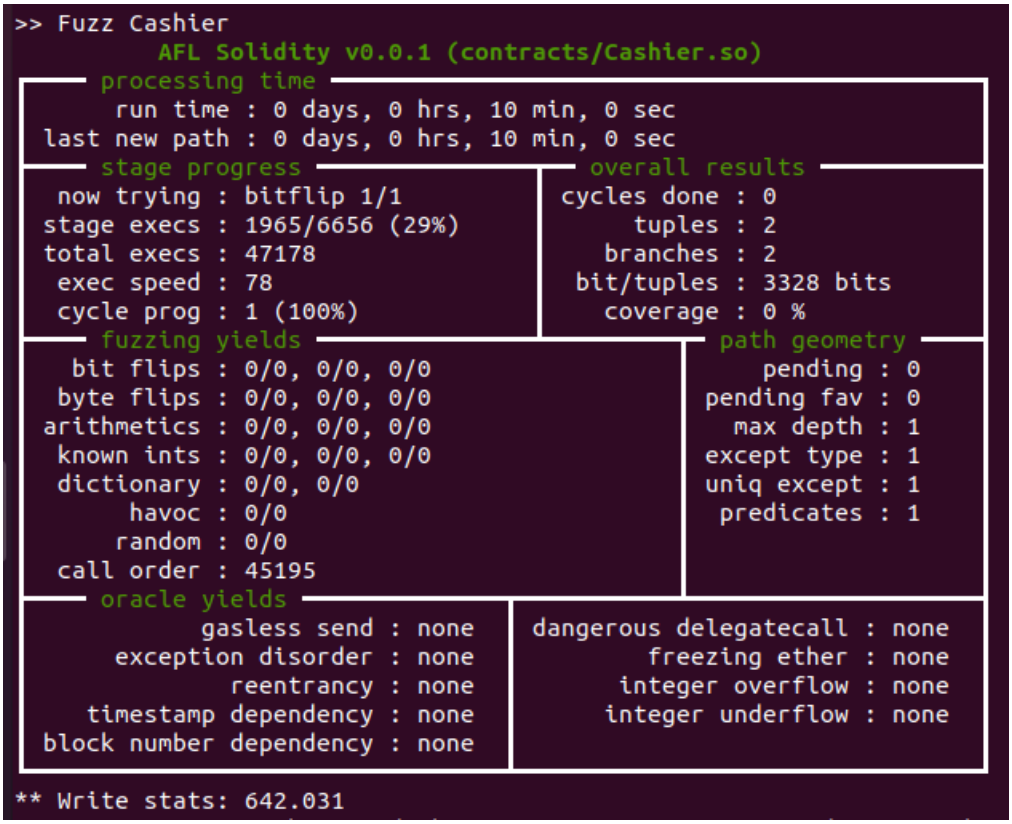
Test coverage of smart contracts:

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts\	0.56	0.25	2.35	0.68	
BosonRouter.sol	0	0	0	0	... 7,1170,1171
Cashier.sol	0	0	0	0	... 0,1114,1129
DAITokenWrapper.sol	20	16.67	40	27.27	... 64,77,78,91
ERC1155ERC721.sol	0	0	0	0	... 850,863,876
ERC1155NonTransferable.sol	8.33	0	20	8.33	... 136,137,138
Gate.sol	0	0	0	0	... 145,152,159
MetaTransactionReceiver.sol	0	0	0	0	... 60,67,74,75
TokenRegistry.sol	9.09	0	12.5	8.33	... 5,89,90,104
UsingHelpers.sol	0	100	0	0	... 0,88,97,110
VoucherKernel.sol	0	0	0	0	... 0,1314,1327
contracts\interfaces\	100	100	100	100	
IBosonRouter.sol	100	100	100	100	
ICashier.sol	100	100	100	100	
IDAI.sol	100	100	100	100	
IERC1155ERC721.sol	100	100	100	100	
IERC1155NonTransferable.sol	100	100	100	100	
IERC20WithPermit.sol	100	100	100	100	
IGate.sol	100	100	100	100	
ITokenRegistry.sol	100	100	100	100	
ITokenWrapper.sol	100	100	100	100	
IVoucherKernel.sol	100	100	100	100	
contracts\libs\	0	0	0	0	
SafeERC20WithPermit.sol	0	0	0	0	26,50,55,58
contracts\mocks\	0	0	0	0	
ERC20WithPermit.sol	0	0	0	0	... 147,148,153
MockBosonRouter.sol	0	0	0	0	... 9,1201,1202
MockERC20Permit.sol	0	0	0	0	... 28,32,36,40
All files	0.45	0.21	1.87	0.55	

Fuzz Testing

1. Cashier.sol: -

    a. Terminal Output
    [With the use of: “ -g -r 0 -d 600 ”]

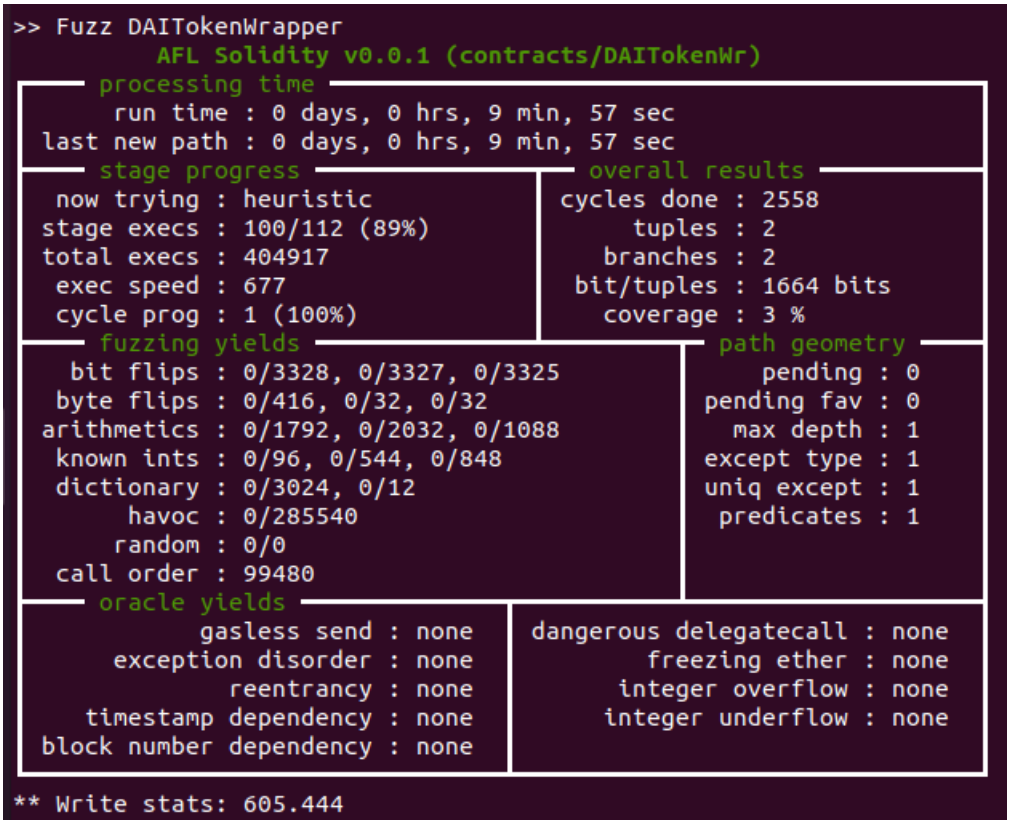


- Excel Sheet of States for the Output of Fuzz Testing
  [With the use of: “ -g -r 1 -d 600 ”]

  https://drive.google.com/file/d/1OxOk9rtnsdZ4FcqMWbBJqJBp9vEVStux/view?usp=sharing

2. DAITokenWrapper.sol: -

    a. Terminal Output
    [With the use of: " -g -r 0 -d 600 "]

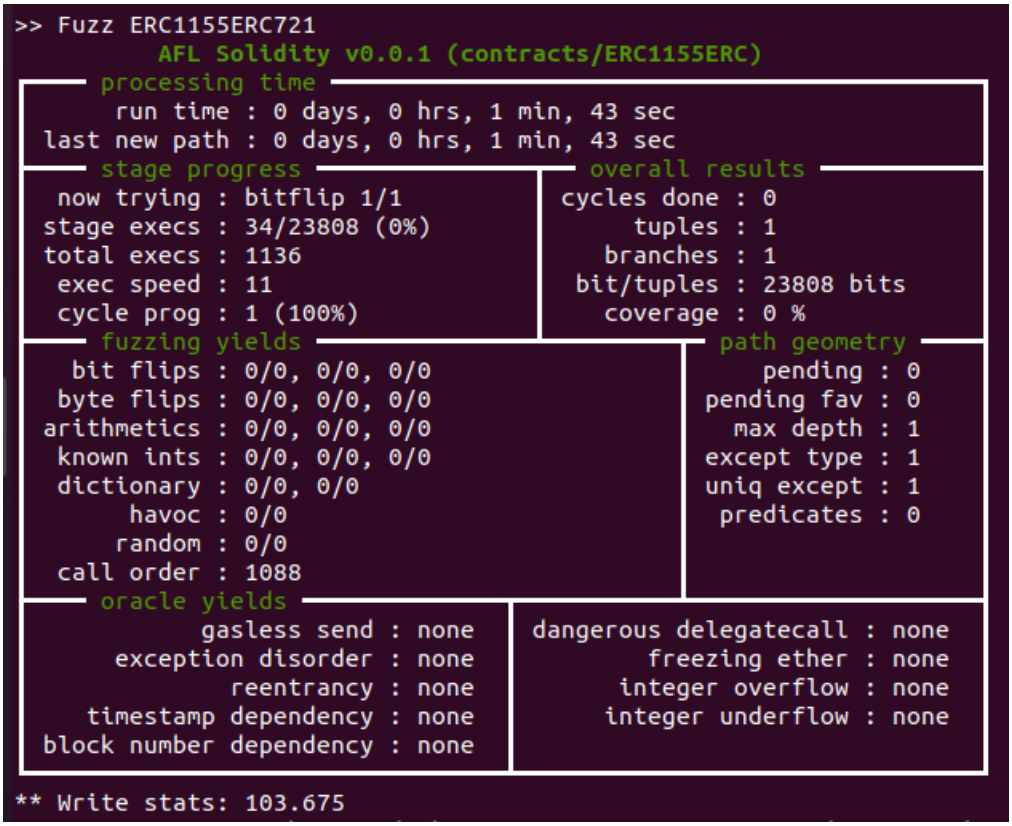


- Excel Sheet of States for the Output of Fuzz Testing
  [With the use of: " -g -r 1 -d 600 "]

  https://drive.google.com/file/d/1u_tztIlUcCJXPk_5BOJ4tZ4UPiaDoSaM/view?usp=sharing

3. ERC1155ERC721.sol: -

    a. Terminal Output
    [With the use of: " -g -r 0 -d 600 "]

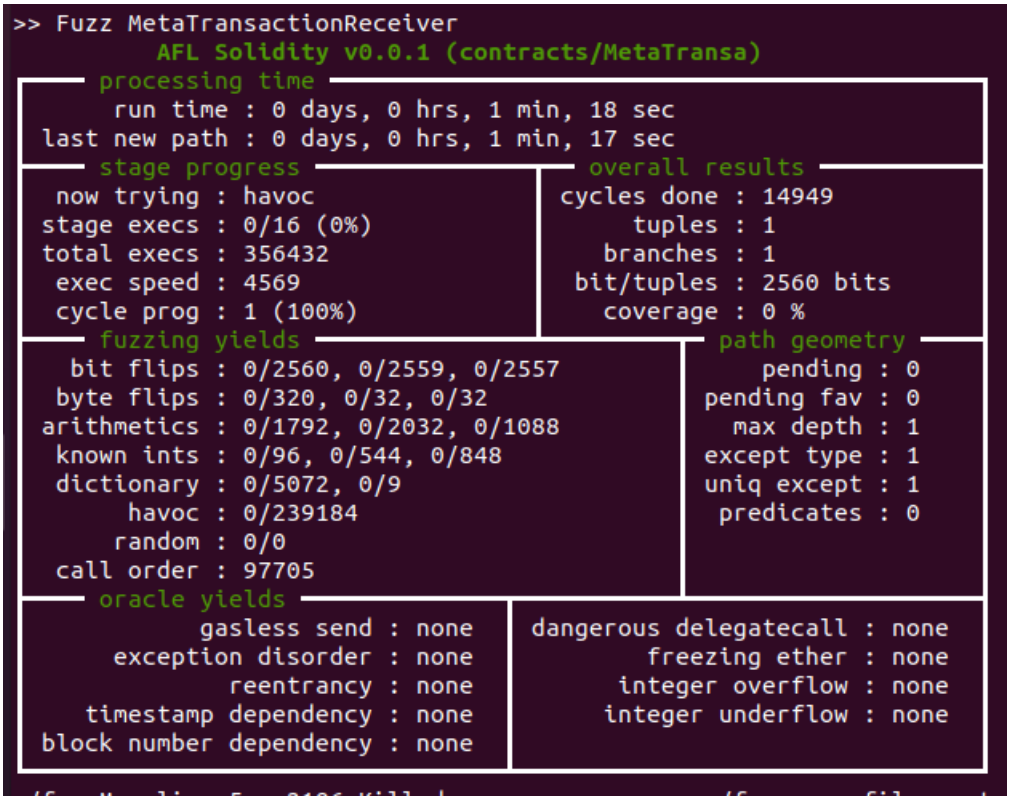


- Excel Sheet of States for the Output of Fuzz Testing
  [With the use of: " -g -r 1 -d 600 "]

  https://drive.google.com/file/d/1ra1WRKwk2RB-gK-QPT9YpC5wMOLd2RoP/view?usp=sharing

4. MetaTransactionReceiver.sol: -

    a. Terminal Output
    [With the use of: " -g -r 0 -d 600 "]

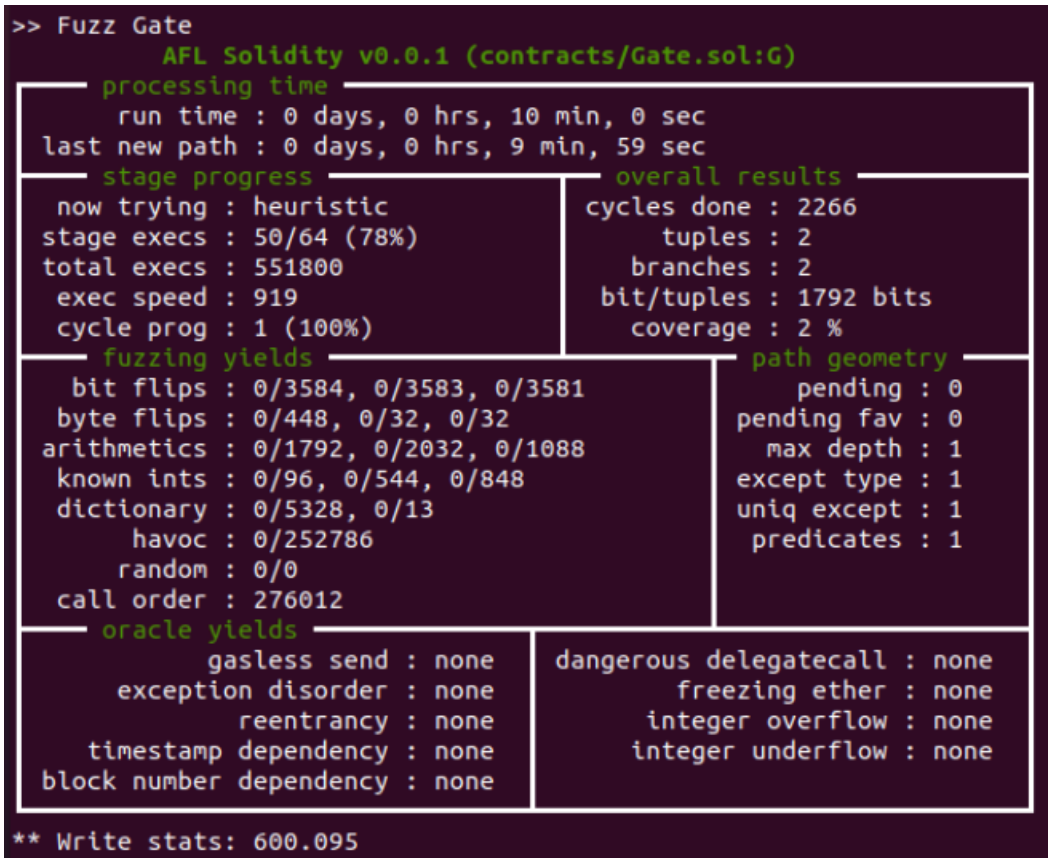


- Excel Sheet of States for the Output of Fuzz Testing
  [With the use of: " -g -r 1 -d 600 "]

  https://drive.google.com/file/d/1u6mkf2dMbvbJICLYHneB3RvRu8wR12sL/view?usp=sharing

5. Gate.sol: -

    a. Terminal Output
    [With the use of: “ -g -r 0 -d 600 ”]

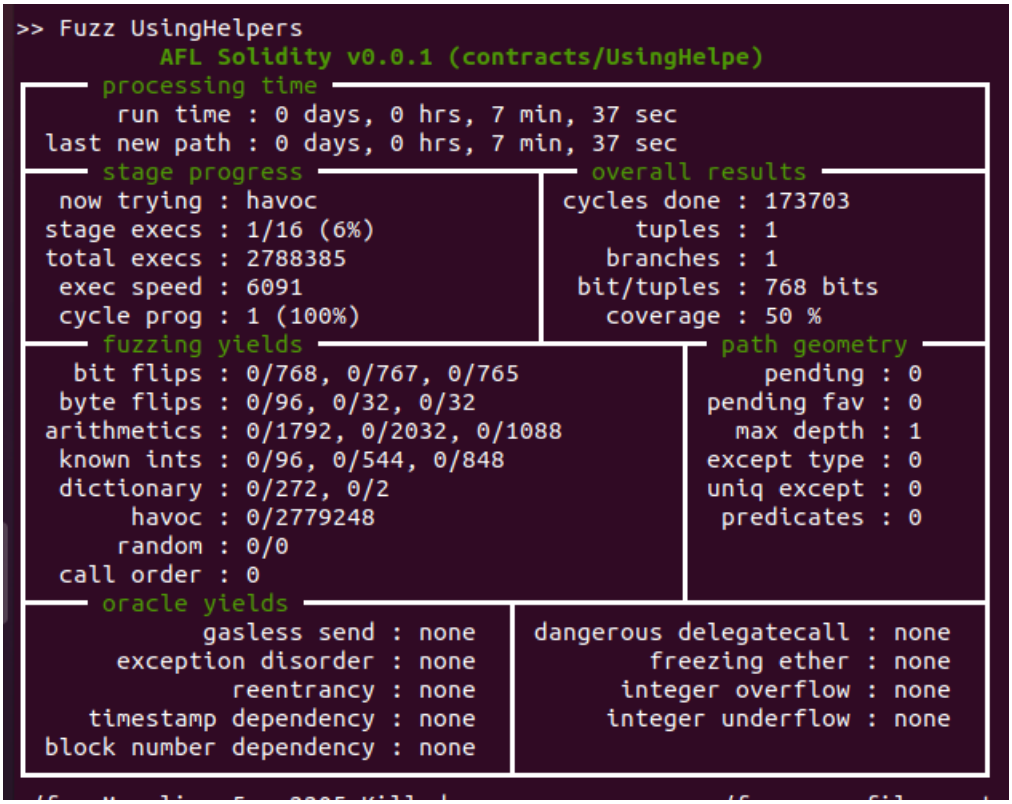


- Excel Sheet of States for the Output of Fuzz Testing
  [With the use of: “ -g -r 1 -d 600 ”]

  https://drive.google.com/file/d/1sWhVwrw2LLmICJujlpnfX1KStW1d3u25/view?usp=sharing

6. UsingHelpers.sol: -

    a. Terminal Output
    [With the use of: " -g -r 0 -d 600 "]

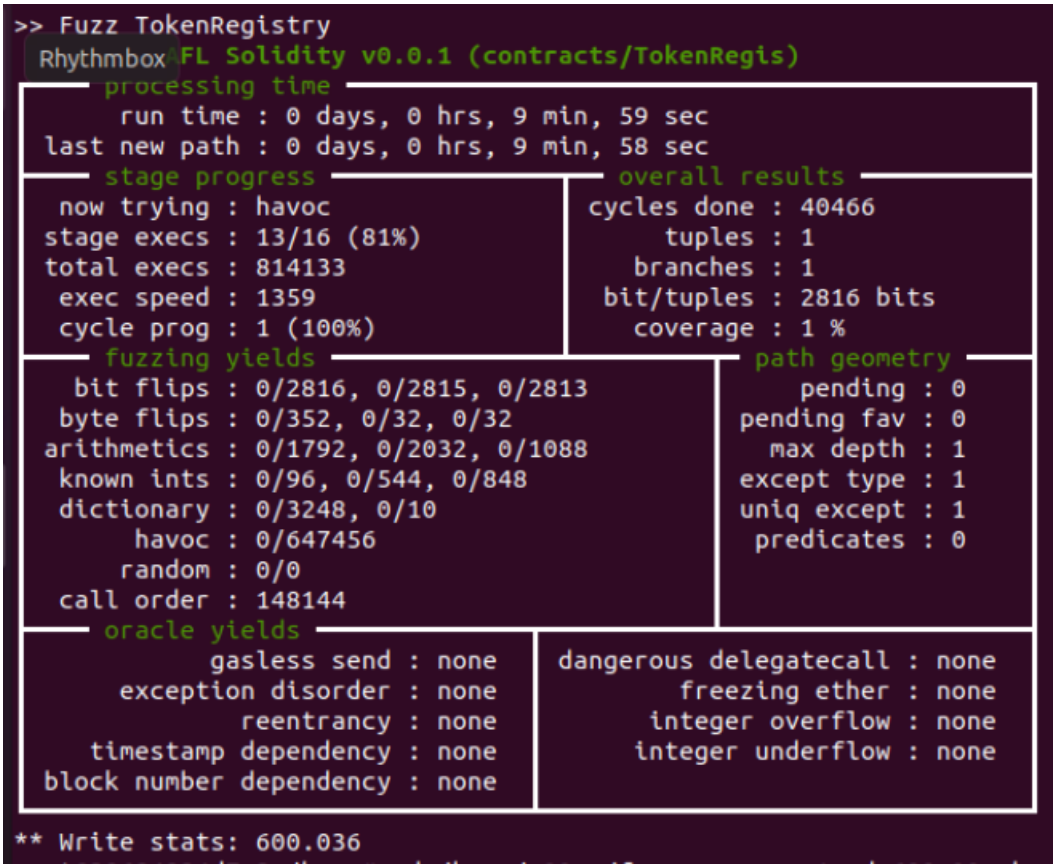


- Excel Sheet of States for the Output of Fuzz Testing
  [With the use of: " -g -r 1 -d 600 "]

  https://drive.google.com/file/d/16fvn6LUR3UAte4_8miicH9IfBI5DwwTq/view?usp=sharing

7. TokenRegistry.sol: -

    a. Terminal Output
    [With the use of: " -g -r 0 -d 600 "]

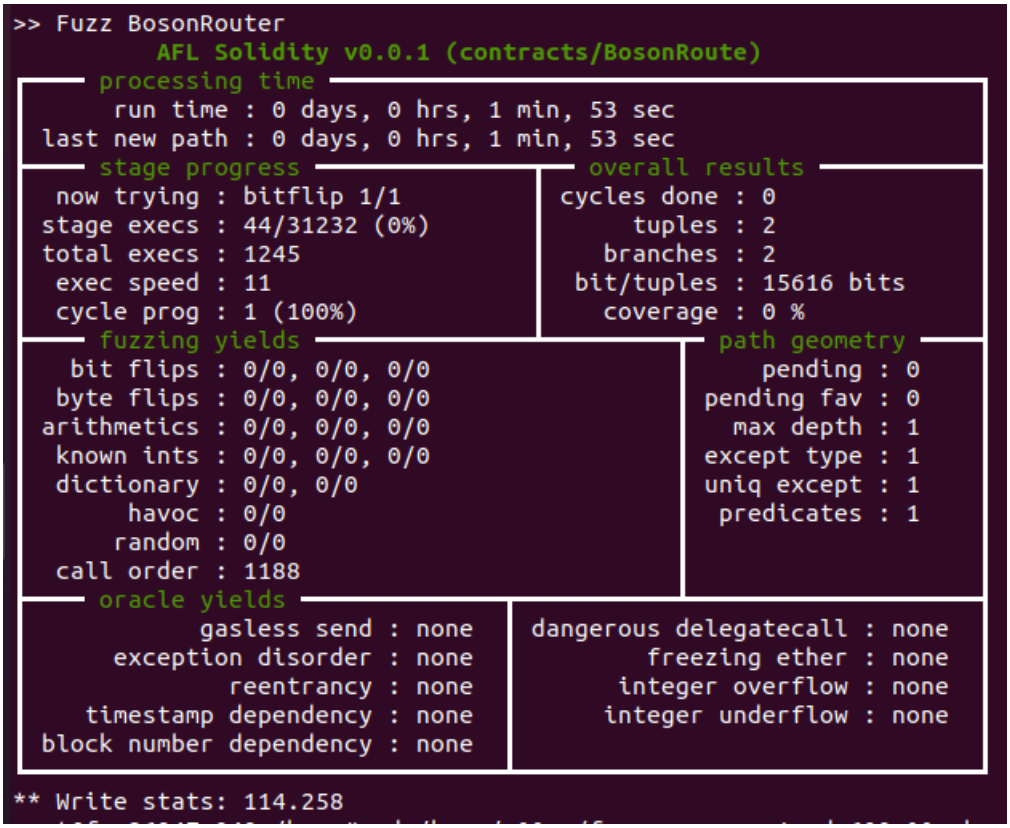


- Excel Sheet of States for the Output of Fuzz Testing
  [With the use of: " -g -r 1 -d 600 "]

  https://drive.google.com/file/d/1DMKwLL7CJiCSbCKxafID3Db-qFw1i66b/view?usp=sharing

8. BosonRouter.sol: -

    a. Terminal Output
    [With the use of: " -g -r 0 -d 600 "]



- Excel Sheet of States for the Output of Fuzz Testing
  [With the use of: " -g -r 1 -d 600 "]

  https://drive.google.com/file/d/1SPdOZ5ldV39b4CruEm5LVUzPbzcH371Z/view?usp=sharing

Vulnerability Checks

TYPES	ORACLES	WHEN A VULNERABILITY IS DETECTED	WHY IT IS VULNERABLE	Results
Error	Gasless Send	Function sends or transfer is called and receiver has a costly fallback function	RunOufOfGasexception	PASSED
Error	Exception Disorder	There is an exception in the call chain but the. These functions hide exceptions	Root of the call chain does not throw exception	PASSED
Error	Timestamp Dependency	The test case evaluates a condition based on timestamp and then sends ether	Miners control the values of timestamp	PASSED
Error	Block Number Dependency	The test case evaluates a condition based on block number and then sends ether	Miners control the values of block number.	PASSED
Error	Danger Delegate Call	delegatecall is executed via msg.data.	The attacker can call any function.	PASSED
Error	Reentrancy	A contract function is called via fallback function from another contract and sends ether.	Refer to the DAO vulnerability	PASSED
Error	Integer Overflow/Underflow	If b >0 and a + b < a or b > 0 and a − b > b or ⋯	Arithmetic error	PASSED
Error	Integer Overflow/Underflow	If b >0 and a + b < a or b > 0 and a − b > b or ⋯	Arithmetic error	PASSED
Warning	Freezing Ether	After all test case, nosend()or transfer() function is executed	The contract is a blackhole for ether	PASSED

Concluding Remarks

While conducting the audits of the Boson Protocol smart contracts, it was observed that the contracts contained High and Low severity issues.

Our auditors suggest that High and Low severity issues should be resolved by the developers. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the Boson Protocol platform or its product nor this audit is investment advice.
Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes