CONSENSYS STANDARD BOUNTIES AUDIT

November 10, 2017

ARGUS AUDITS

Observe. Report. Defend.

IN COLLABORATION WITH

BLOCKCHAIN AT BERKELEY

CONSENSYS

STANDARD BOUNTIES AUDIT

The Argus team conducted a security audit of StandardBounties at initial commit hash 7da4a5365ab677baf1b329cd97b6733470182320 and updated commit hash f31e11cf425209675e8cfb2d9ed0c05dbc22832b. The updated StandardBounties contract is deployed on Ethereum mainnet at 0x066128b9f7557b5398db3d4ed141f2e64245ffa1.

The team was contracted to audit StandardBounties, in partnership with ConsenSys, on October 4, 2017. An initial audit was submitted to StandardBounties team on October 19, 2017.

Following the initial audit report, updates were made to the StandardBounties library in response to our feedback.

The team was contracted again to audit StandardBounties, certifying the changes made in response to the initial audit report.

This is the final audit report, completed on November 10, 2017.
No major smart contract code vulnerabilities were found to put funds at risk in the updated audit. The final audit report includes analysis from the initial audit and updated audit of StandardBounties.

SCOPE

The team reviewed the code with the following metrics in mind:

Security

Identification of potential security flaws/attacks including but not limited to:

- Data compliance
- Integer overflow and underflow
- Race conditions
- DoS attacks
- Timestamp dependence
- Code quality
- Reentrancy attacks
- Sybil attacks
- Front running
- Transaction-ordering dependence
- Call depth attacks

CODE QUALITY

Review of smart contract code to follow Argus best practices and ConsenSys best practices in:

- Syntax
- Readability
- Consistency
- Complexity
- Scalability

Reivew of smart contract architecture with emphasis on:

- Game theory
- Incentives
- Pain points
- Usability

INITIAL ANALYSIS

The following contains findings from the initial audit by the Argus team.

General Findings

- Consider timestamp dependence issues when using now as opposed to block number.
- Concerns regarding sybil resistance.
- Miners can see fulfillments, withhold those transactions, and submit the data for themselves, frontrunning the bounty.
- There are no checks for invalid addresses (at the very least != 0).
- IPFS persistence may be an issue. Files with less demand will go down if hosts go offline.
- The hash of any given data on chain is not able to be verified.
- Remove ^ from pragma statement to concretely set version.

Modifiers

amountIsNotZero

This modifier should verify if the value is greater than 0.

transferredAmountEqualValue

This modifier should assert the difference in balance equals the transferred amount.
It should require(msg.value == 0) before the transferFrom if statement to avoid unnecessary computation.

isBeforeDeadline

This is often used before checking the validity of the _bountyId, which could cause an array out of bounds error.

validateFulfillmentArrayIndex

For clean code convention, this modifier should be moved up next to other index check modifier.

enoughFundsToPay

The comparison operator should be changed to be consistent with previous comparison in activateBounty.
(E.g. consistently follow x > y rather than x <= y)

Functions

issueBounty

The function should verify that the token address is valid.

issueandActivateBounty

The function needs to check on msg.value == 0 if _paysTokens == true.
The function should verify that the token address is valid.

activateBounty

The function needs to cal the modifier validateBountyArrayIndex first, as other modifiers rely on the validity of this array index to contain a non-null value.

INITIAL ANALYSIS

Functions

killBounty

Issuer can drain all the money without refunding any contributors. This disincentivizes contributors from participating in the bounty.

Best practice to have difference > 0 in the case that the EVM/solc converts uint types into int for comparison. Also better for good practice and clarity.

transferIssuer

This function does not check if _newIssuer is a valid address. An issuer could grief attack the bounty by transferring to the 0x0 address. The contract is then stuck in the same state and no one can remove their token.

Plausible deniability for a malicious issuer. Issuers can steal all data, swap the issuer, and claim that the new issuer is the malicious one (even when owned by same person).

Blame for malicious activity can be directed to the new issuer, B as long as the original issuer, A claims to have been attacked/decieved by B.

changeBountyDeadline

The @dev comment is incorrect for the function.

changeBountyData

The @dev comment is incorrect for the function.

changeBountyFulfillmentAmount

The @dev comment is incorrect for the function.

changeBountyArbiter

The @dev comment is incorrect for the function.

changeBountyPaysTokens

By changing the bounty payout token type, the issuer can drain the contract of all funds, including from contributors. The issuer can steal funds from contributors with or without intending to do so.

Should check that oldPaysTokens and _newPaysTokens if tokenContract[_boundyId] already points to a token, to see if the bounty transitions from a HumanStandardToken to ether. In this case, the _newTokenContract parameter should not be considered and the previous tokenContract[_bountyId] address should be deleted.

When a bounty transitions from ether to a token, the new token address should be checked to be legitimate.

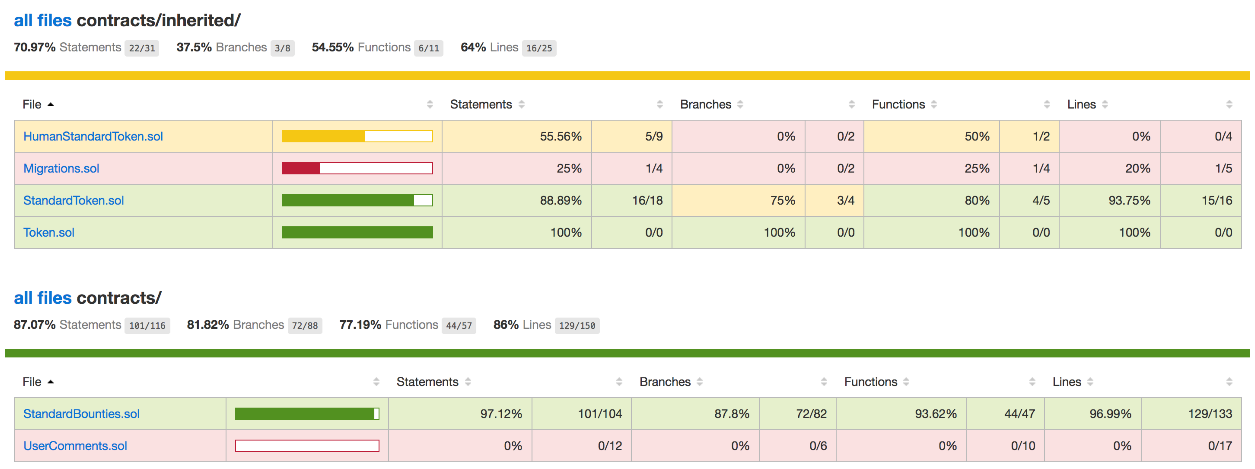
contribute

This function needs to first call the modifier, validateBountyArrayIndex, since other modifiers rely on the validity of this array index to contain a non-null value.

INITIAL CODE COVERAGE AND OYENTE

Code coverage using the Solidity-Coverage tool was used to measure the what portions of the codebase was run with the given test suite.

The test coverage results are located in the code coverage directory and can be viewed by opening the index.html file or by observing the image below:



all files contracts/inherited/

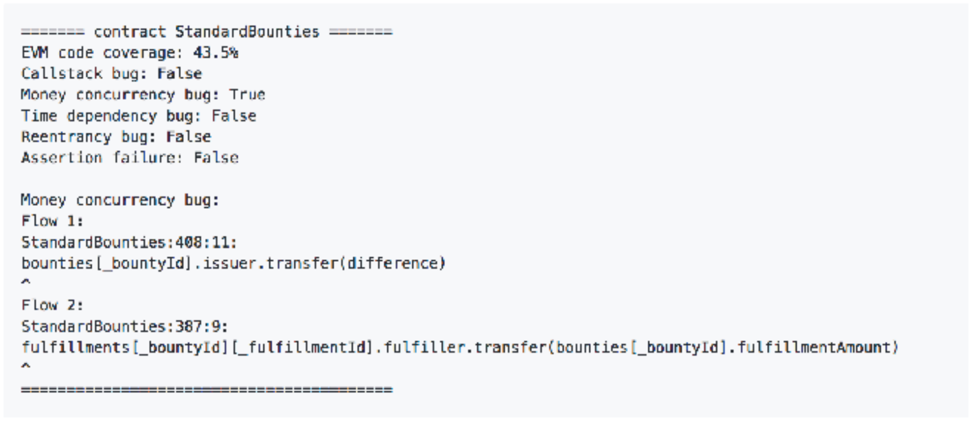
70.97% Statements 22/31 **37.5%** Branches 3/8 **54.55%** Functions 6/11 **64%** Lines 16/25

File		Statements		Branches		Functions		Lines	
HumanStandardToken.sol		55.56%	5/9	0%	0/2	50%	1/2	0%	0/4
Migrations.sol		25%	1/4	0%	0/2	25%	1/4	20%	1/5
StandardToken.sol		88.89%	16/18	75%	3/4	80%	4/5	93.75%	15/16
Token.sol		100%	0/0	100%	0/0	100%	0/0	100%	0/0

all files contracts/

87.07% Statements 101/116 **81.82%** Branches 72/88 **77.19%** Functions 44/57 **86%** Lines 129/150

File		Statements		Branches		Functions		Lines	
StandardBounties.sol		97.12%	101/104	87.8%	72/82	93.62%	44/47	96.99%	129/133
UserComments.sol		0%	0/12	0%	0/6	0%	0/10	0%	0/17



```
======= contract StandardBounties =======
EVM code coverage: 43.5%
Callstack bug: False
Money concurrency bug: True
Time dependency bug: False
Reentrancy bug: False
Assertion failure: False

Money concurrency bug:
Flow 1:
StandardBounties:408:11:
bounties[_bountyId].issuer.transfer(difference)
^
Flow 2:
StandardBounties:387:9:
fulfillments[_bountyId][_fulfillmentId].fulfiller.transfer(bounties[_bountyId].fulfillmentAmount)
^
=========================================
```

UPDATED ANALYSIS

November 9th, Commit Hash : f31e11cf425209675e8cfb2d9ed0c05dbc22832b

General Findings

- The variable numAccepted is iterated on but never used.
- Use block number. Instead of now and timestamp.
- Consider using a linter for formatting, syntax, and spacing.

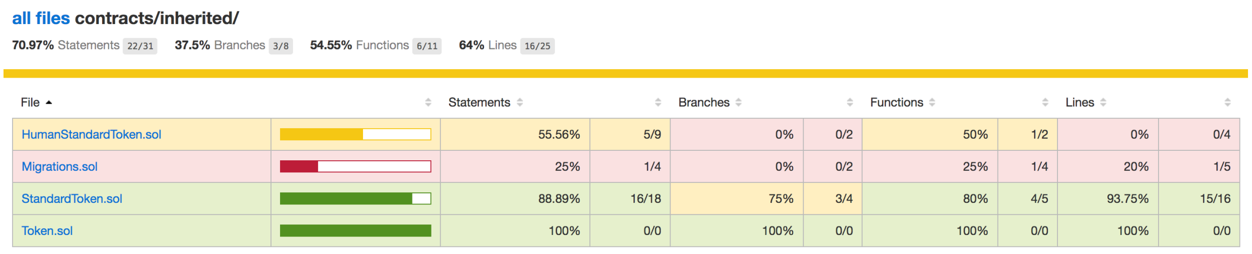
UserComments.sol

Consider emitting bountyID in the event when a comment is added otherwise, only the sender and reciever's addresses are emitted.

Comments will scale quickly. Users will pay gas to store and add comments proportional to the length of the text. Consider using a centralized database solution for faster and cheaper storage, or storing hashes of comments on-chain.

UserComments.sol is not documented like StandardBounties.sol. It is assumed that comments are parsed and fetched via events rather than scanning through the comments array. In this case, there is no need to store the comments in the smart contract when comment contents are contained in events.

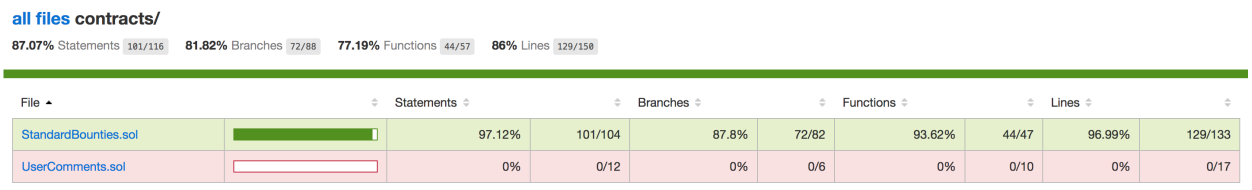
UPDATED CODE COVERAGE AND OYENTE



all files contracts/inherited/

70.97% Statements 22/31 **37.5%** Branches 3/8 **54.55%** Functions 6/11 **64%** Lines 16/25

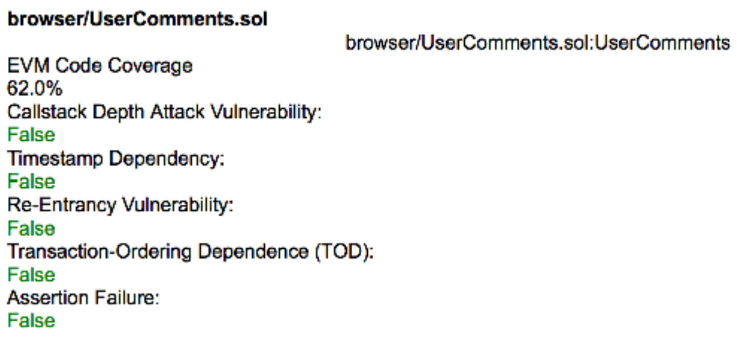
File ▲		Statements		Branches		Functions		Lines	
HumanStandardToken.sol		55.56%	5/9	0%	0/2	50%	1/2	0%	0/4
Migrations.sol		25%	1/4	0%	0/2	25%	1/4	20%	1/5
StandardToken.sol		88.89%	16/18	75%	3/4	80%	4/5	93.75%	15/16
Token.sol		100%	0/0	100%	0/0	100%	0/0	100%	0/0



all files contracts/

87.07% Statements 101/116 **81.82%** Branches 72/88 **77.19%** Functions 44/57 **86%** Lines 129/150

File ▲		Statements		Branches		Functions		Lines	
StandardBounties.sol		97.12%	101/104	87.8%	72/82	93.62%	44/47	96.99%	129/133
UserComments.sol		0%	0/12	0%	0/6	0%	0/10	0%	0/17



browser/UserComments.sol

browser/UserComments.sol:UserComments

EVM Code Coverage
62.0%
Callstack Depth Attack Vulnerability:
False
Timestamp Dependency:
False
Re-Entrancy Vulnerability:
False
Transaction-Ordering Dependence (TOD):
False
Assertion Failure:
False

ARGUS AUDITS

Authors

Ali Mousa
ali@argus.observer

Howard Wu
howard@argus.observer

Collin Chin
collin@argus.observer

Pranav Gaddamadugu
pranav@argus.observer

Raymond Chu
ray@argus.observer

Advisor

Dawn Song
Professor, University of California, Berkeley
dawn@argus.observer

Contact

Argus Audit Team
audit@argus.observer