SOLIDIFIED

Audit Report for Polymath. February 14, 2018.

Summary

Audit Report prepared by Solidified for Polymath covering the revised version of the Polymath-core contracts.

Process and Delivery

This is an amendment of the original report that was delivered on February 2, 2018 that is based on an updated version of the code and client's feedback to the original list of issues.

Three (3) independent Solidified experts performed an unbiased and isolated audit of the below contracts. The debrief took place on February 12, 2018 and the final results are presented here.

Audited Files

The following files from commit `ad40f07504471ed02d39102daa1f4d8dda8596f7` were audited:

- Compliance.sol
- Customers.sol
- SecurityToken.sol
- SecurityTokenRegistrar.sol
- Template.sol

Intended Behavior

The purpose of these contracts is to facilitate the creation and initial offering of regulatorily compliant security tokens on the Polymath platform.

SOLIDIFIED

Audit Report for Polymath. February 14, 2018.

Critical Issues

1. Multiple trustlessness issues in Customers contract

Right now KYC providers can:

- Retrieve payment from their customers without providing them with the accreditation
- Give customers different accreditation than agreed
- Unilaterally change or revoke accreditation of their customers at any time

Possible fund stealing attack

A malicious KYC provider watches the chain for incoming transactions from clients calling the `approve` function in the POLY contract. As soon as the approve transaction is mined, the KYC rushes to verify the customer in Customers.sol with junk data to get its KYC fee.

Other issues

Because direct payment towards KYC providers has to be pre-approved before verifyCustomer is called, KYC providers can transfer this fee to their address without providing any service, there is also no mechanism ensuring that customer agreed to the exact details of the accreditation that the KYC provider is registering for the customer. And finally, if KYC providers set their fee to 0, they can arbitrarily change the accreditations of all of their customers at any time, even cancelling them.

Suggested changes

- Let customers sign details of their accreditation (countryJurisdiction, divisionJurisdiction, role, accredited, expires) with their private key and transfer this signature to the verifiers off chain, add this signature as an argument to verifyCustomer function and verify it before finalizing the process. To prevent replay attacks, a nonce should be added to the Customer structure and included in the signed data, this nonce should be incremented after each accreditation, if no previous accreditation exists, it should be assumed to be 0.

- Let customers pre-approve POLY token transfer not to the KYC provider, but to the Customers contract and only transfer POLY tokens to the address of the KYC provider after the desired accreditation has been given to the customer inside the verifyCustomer function.

AMENDED [2018-02-14]:
This issue has been fixed by the Polymath team and is not present in commit `ad40f07504471ed02d39102daa1f4d8dda8596f7`

Major Issues

2. Race condition due to setRegistrarAddress

There is no authorization check in setRegistrarAddress function, someone could set up a script that calls this function as soon as the contract is deployed to the network, irreversibly breaking the system.

```
    function setRegsitrarAddress(address _STRegistrar) public returns
(bool) {
        require(STRegistrar == address(0));
        STRegistrar = SecurityTokenRegistrar(_STRegistrar);
        return true;
    }
```

Recommendation
SecurityTokenRegistrar contract should be deployed by the Compliance contract.

Client's response
Because of block gas limits in deploying our contracts we have left this functionality largely unchanged (the Compliance contract is first deployed, then the SecurityTokenRegistrar contract, then setRegistrarAddress is called on the Compliance contract). This allows a possible attack during the deployment process, but is easy to check (since the deployment would fail in this case) and only possible for a small moment in time. Recovery would be straight-forward (redeploying the contracts).

Our response
Dedicated malicious party can continuously set incorrect SecurityTokenRegistrar addresses as soon as a new instance of Compliance contract gets deployed to the blockchain, and since this attack is significantly less gas intensive than deployment of the Compliance contract, attacker can in the long run outlast the author. This attack could be improbable, but it is theoretically possible.
Since block gas limit reportedly prevents all contracts being deployed inside one transaction, a possible alternative is to deploy Compliance contract with a hash of the bytecode of the SecurityTokenRegistrar

contract and then check that this hash matches the contract that calls setRegistrarAddress and that address of POLY token contract and Customers contract set in the SecurityTokenRegistrar also matches these set in the Compliance contract.
This system should prevent the above mentioned attack.

3. Function selectTemplate in SecurityToken can be called repeatedly leading to multiple problems

The fact selectTemplate can be called multiple times leads to two problems:

- It allows artificially inflating the usedBy statistic inside the Compliance contract
- It can lead to creation of multiple token allocations to different delegates, exceeding the provided POLY balance for compensation and rewarding delegates that proposed templates that haven't been ultimately used

Recommendation
To solve this problem, there should be a check that assures that selectTemplate function can be successfully called only once

AMENDED [2018-02-14]:
This issue has been fixed by the Polymath team and is not present in commit `ad40f07504471ed02d39102daa1f4d8dda8596f7`

4. Security token contract can lock owner's POLY

The token owner needs to send poly to the Security Token contract in order to select a template and an offering. When extra POLY is sent, the owner can only withdraw it before a template is selected, otherwise it gets locked.

Recommendation
Change withrawPOLY function to allow withdraw before the offering starts, restricted to extra amount.

AMENDED [2018-02-14]:
This issue has been fixed by the Polymath team and is not present in commit `ad40f07504471ed02d39102daa1f4d8dda8596f7`

5. Owner of SecurityToken contract can prevent buyers from freezing raised funds by blacklisting them

In the event of `SecurityToken``'s owner believing their investors will freeze their funds, they can blacklist enough contributors to prevent the vote to freeze fund from succeeding.

Recommendation
Allow blacklisted contributors to vote for freeze.

Client's response
No action taken - we recognise the potential problem and are tracking this as an issue, but any fix may form part of a larger rethink of this voting process.

Our response
We strongly recommend listing this as a known issue in the future bug bounty, to prevent misunderstandings.

6. `SecurityToken` is not ERC20 compliant

1. SecurityToken's `approve` function prevents setting approval to 0: this breaks ERC20 standard compatibility.

2. SecurityToken's `transfer` and `transferFrom` functions do not allow transfers of 0 values this breaks ERC20 standard compatibility

Recommendation
Remove checks against 0 values in the identified functions.

AMENDED [2018-02-14]:
This issue has been fixed by the Polymath team and is not present in commit `ad40f07504471ed02d39102daa1f4d8dda8596f7`

7. Overflow vulnerability on vestingPeriod variable

```
require(now > endSTO + allocations[msg.sender].vestingPeriod);
```

Because safeMath library is not used in withdrawPoly function, it's possible to break the vesting obligation by causing an overflow, furthermore, the value that causes this overflow also makes the tokens unfreezable.

Recommendation
Make use of a safe addition operation

8. newProvider function being callable on behalf of any address is potentially problematic

---

The fact that anybody can create a provider record in the Customers contract on behalf of any address in combination with name and details variables being unchangeable once they are set, is potentially problematic. A malicious party can create record, for example, on behalf of a well know multisig/DAO contract that plans to begin providing KYC services, with the wrong name and details attributes and the owners of the contract have no way to correct this information.

9. Expiration date missing from signed data in verifyCustomer function

---

The parameter _expires is not part of the data signed by the KYC customer.

Minor Issues

10. Security token creation fee can be gamed

Making both _host and _fee parameters of the SecurityTokenRegistrar createSecurityToken function makes it possible for callers to bypass the fee, either setting it to 0 or sending to themselves.

Recommendation
Store this info as a state variable which can only be updated by PolyMath (so it can later be adjusted).

AMENDED [2018-02-14]:
This issue has been fixed by the Polymath team and is not present in commit `ad40f07504471ed02d39102daa1f4d8dda8596f7`

Suggestions

11. Variables TimesUsed and AmountRaised in template reputation are not used

Both variables remain constant since no function is provided to alter them. It's advisable to either remove or consider them in "updateTemplateReputation"

AMENDED [2018-02-14]:
This issue has been fixed by the Polymath team and is not present in commit `ad40f07504471ed02d39102daa1f4d8dda8596f7`

12. Redundancies in addToWhitelist and addToBlacklist functions of SecurityToken contract

Logging caller address of addToWhitelist and addToBlacklist function and storing it in Shareholder struct is unnecessary because only contract owner can call these function and the address of the owner can't change after deployment.

AMENDED [2018-02-14]:
This issue has been fixed by the Polymath team and is not present in commit `ad40f07504471ed02d39102daa1f4d8dda8596f7`

13. Redundant verified flag in Customer struct of Customers contract

The verified flag in the Customer struct is always checked together with the expires variable: it can be omitted without any loss of functionality.

```
    struct Customer {                                                    //
Structure use to store the details of the customers
        bytes32 countryJurisdiction;                                     //
Customers country jurisdiction as ex - ISO3166
        bytes32 divisionJurisdiction;                                    //
Customers sub-division jurisdiction as ex - ISO3166
        uint256 joined;                                                  //
Timestamp when customer register
        uint8 role;                                                      //
role of the customer
        bool verified;                                                   //
Boolean variable to check the status of the customer whether it is verified
or not
        bool accredited;                                                 //
Accrediation status of the customer
        bytes32 proof;                                                   //
Proof for customer
        uint256 expires;                                                 //
Timestamp when customer verification expires
    }
```

AMENDED [2018-02-14]:
This issue has been fixed by the Polymath team and is not present in commit `ad40f07504471ed02d39102daa1f4d8dda8596f7`

14. ChangeFee function should use onlyProvider modifier

Instead of a require in the body of function ensuring the caller is registered provider, the existing onlyProvider modifier should be used to remove code duplication and increase readability.

```
    function changeFee(uint256 _newFee) public returns (bool success) {
        require(providers[msg.sender].details != 0x0);
```

AMENDED [2018-02-14]:
This issue has been fixed by the Polymath team and is not present in commit `ad40f07504471ed02d39102daa1f4d8dda8596f7`

15. Redundancies in addToWhitelist and addToBlacklist functions of SecurityToken contract

Logging caller address of addToWhitelist and addToBlacklist function and storing it in Shareholder struct is unnecessary because only contract owner can call these function and the address of the owner can't change after deployment.

AMENDED [2018-02-14]:
This issue has been fixed by the Polymath team and is not present in commit `ad40f07504471ed02d39102daa1f4d8dda8596f7`

16. Poor separation of concerns between Security Token and security token Offering

The process of buying a new security token is divided between both the offering contract and the token contract. It's recommended each contract deals only with it's own concerns and all information regarding a token offering is delegated to the STO contract, that only calls transfer functions in the token contract when all sanity checks have been made.

AMENDED [2018-02-14]:
This issue has been fixed by the Polymath team and is not present in commit `ad40f07504471ed02d39102daa1f4d8dda8596f7`

17. Avoid constant keyword for functions

The constant keyword is deprecated in preference of the pair view/pure. It is advisable to use the newer syntax, since "constant" is misleading regarding its meaning.

AMENDED [2018-02-14]:
This issue has been fixed by the Polymath team and is not present in commit `ad40f07504471ed02d39102daa1f4d8dda8596f7`

18. Prefer using 'bytes32' over 'string'

Strings are dynamically sized and therefore need to store both the data and the memory information. To save gas, since storage is especially expensive, it's advised to use bytes32.

Client's response

No action taken - this is a possible improvement which we are tracking but not fixing in this version.

19. New KYC provider fee is not present

The whitepaper states that new KYC providers must pay a fee in order to avoid spam. Such a mechanism is not present in the KYC registration process. It's advisable to either add it or remove it from the paper

Client's response

No action taken - we are tracking this as an issue, but currently not changing the logic (in the decentralised system it is not clear who would receive this fee as we don't want Polymath to be directly involved beyond the protocol layer).

20. Isolate code intended solely for test net deployment

Code such as `STOContract.sol` which is not intended for production, should be clearly separated from audited platform code as to not provide a false sense of security. Clearly demarcate or remove code not intended for production.

AMENDED [2018-02-14]:
This issue has been fixed by the Polymath team and is not present in commit `ad40f07504471ed02d39102daa1f4d8dda8596f7`

21. Inefficiency in reputation tracking in the Compliance contract

It seems inefficient to record full SecurityToken address with each sale (updateTemplateReputation, updateOfferingFactoryReputation), if there's a need to track number of sales on top of their volume, it is cheaper to have an integer counter for each template or offering factory, or combination of template/security token, or offering factory/security token.

22. Inconsistently enforced rule in SecurityTokenRegistrar

In the createSecurityToken function, a security token's name is not permitted to be an empty string, this rule is not enforced in the changeName function of the SecurityToken contract.

23. Unnecessary logging of old variable values in SecurityToken contract

In the functions changeTotalSupply, changeDecimals and changeName, the old value is logged along with the new value, but since the record of the old value already must exist in the log, this might not be necessary.

24. Unecessary flag hasOfferingStarted in the SecurityToken contract

Instead of the flag hasOfferingStarted something like offering != address(0) should be sufficient to check that offering contract hasn't been set yet.

25. Unnecessary logging of KYC address in addToWhitelist

Since KYC for a SecurityToken contract can't change once it has been set, all records resulting from addToWhitelist will contain the same KYC address, it's therefore unnecessary to include the address in the records.

26. Unnecessary argument in the createOffering function in the IOfferingFactory interface

Since in the current system, _securityToken will always be identical with msg.sender it might can be omitted without any loss in functionality.

27. Unnecessary requires on return values

It doesn't make sense to call functions that can't return false (because they throw on failure) inside a require. The functions in the requires should either return false on failure, or the require should be removed. This redundancy is present in several instances in the code. An example is:

```
require(addToWhitelist(_whitelistAddresses[i]));
```

in addToWhitelistMulti function of the SecurityToken contract.

Potential Issues

28. Confusing relationship between Security Tokens and Security Token Offerings

Some parts of the contracts suggest that a single offering contract can be reused for multiple tokens, awarding a reputation, for example, while other bits suggests that there's a 1:1 correlation.

AMENDED [2018-02-14]:
This issue has been fixed by the Polymath team and is not present in commit `ad40f07504471ed02d39102daa1f4d8dda8596f7`

29. The function addToBlacklist can't be reliably reversed

If there's some change of the buyers accreditation on the KYC provider side, addToBlacklist can't be reliably reversed by calling addToWhitelist.

AMENDED [2018-02-14]:
This issue has been fixed by the Polymath team and is not present in commit `ad40f07504471ed02d39102daa1f4d8dda8596f7`

30. Allowing contracts to hold vested POLY balances and security tokens is potentially problematic

If the purpose of the system is to ensure that the security tokens can't be transferred to holders that don't satisfy regulatory requirements, allowing smart contracts to own security tokens can be problematic. By changing the ownership of the contract that owns the security tokens, the tokens can be effectively transferred even if the receiving counterparty doesn't have the required certification. A similar problem arises with vested balances of POLY tokens.

Client's response

No action taken - it will be up to the KYC providers to decide whether or not to allow contract addresses to be whitelisted, and there are tools in the polymath.js library to allow them to review this. Allowing some contract addresses to hold tokens may be needed for exchanges / multi-sig wallets and so on.

31. Whitelist mechanism is potentially limiting

Because each investor has to be added to the whitelist manually, it's currently impossible to open the sale to all users that satisfy the regulatory requirements, this could be potentially limiting, especially if the pool of eligible buyers is very large.

Client's response

No action taken - we have an open issue on GitHub for this, but see this as a future enhancement rather than security flaw.

Audit Report for Polymath. February 14, 2018.

Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of the Polymath platform. This audit does not provide a security or correctness guarantee of the audited smart contracts. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

The individual audit reports are anonymized and combined during a debrief process, in order to provide an unbiased delivery and protect the auditors of Solidified platform from legal and financial liability.

Solidified Technologies Inc.