SOLIDIFIED

Audit Report for Polymath. July 7, 2018.

Summary

Audit Report prepared by Solidified for Polymath covering the Polymath Core Tore release.

Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the below token swap. The debrief took place on July 07, 2018 and the final results are presented here.

Audited Files

All the contracts present in Polymath Core Toro release.

Intended Behavior

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

The audit was based on commit `364ea2a6669206aea854d5bfa0202c009e08ea83`.

Issues Found

Critical

1. IModuleFactory contracts are treated as trusted in SecurityToken contract which allows several rules to be broken

---

Owner of the `SecurityToken` contract can pass an arbitrary contract to the `addModule` function and have it treated as a trusted module factory. This is allowed by two facts: function `useModule` of the `ModuleRegistry` contract accepts any contract as long as the address it returns through the `.owner()` function call returns an identical address to the `SecurityToken`'s owner and even if this wouldn't be true, the owner of the `SecurityToken` contract can change the `SecurityTokenRegistry`, and consequently the associated `ModuleRegistry` at any time through the `changeSecurityTokenRegistryAddress` function.
By using a malicious `ModuleFactory` contract, `SecurityToken` owner can bypass several rules. They can unlock locked module type by making the factory contract return a different value when it's called in lines `SecurityToken.sol:191` and `SecurityToken.sol:200`. Using a similar technique, they can push module into an already locked category. They can also bypass the `MAX_MODULES` limit.

Recommendation
Heighten the requirements for adding modules to the security token, perhaps only allowing Polymath approved modules.

Client's response:
We want issuers to be able to attach arbitrary modules which have not been verified by Polymath, provided they are the owner of those modules.

To address this issue we have:
- made `addModule` non-reentrant.
- only called `getType` once in `_addModule` so that the returned value cannot be changed by a malicious module factory between calls
- removed the ability for issuers to unilaterally modify the address of the `SecurityTokenRegistry` or `ModuleRegistry` contracts (see #11 for more details).

This should mean that whilst an issuer can add any module to their contract (provided they own the corresponding module factory, or specifically that the module factory returns their address in a call to `owner()`) their added modules must respect the type rules and not re-enter `addModule` as part of their deployment.

AMENDED [2018-7-24]:
This issue is no longer present in commit `8d1f586cccea2721cf93cebb2d0e84e159991241`.

Major

2. Mint function in SecurityToken allows creating a positive balance for address(0) with fatal consequences

---

The mint function in the `SecurityToken` contract doesn't contain any check preventing the zero address from being the beneficiary. This opens a way to deflate the `investorCount` counter through subsequent minting, because `adjustInvestorCount` will consider each minting transaction with value equivalent to the balance of the zero address as a loss of an investor. Leading to a complete freeze of `SecurityToken` transactions when `investorCount` underflows.

Recommendation
Validade the address input within the mint function.

AMENDED [2018-7-24]:
This issue is no longer present in commit `8d1f586cccea2721cf93cebb2d0e84e159991241`.

3. changeModuleBudget function of SecurityToken is vulnerable to approve frontrunning attack

Original ERC20 token standard contains a vulnerability allowing addresses that are approved for transfer by other addresses to withdraw their tokens just before the new updated approval is included in the blockchain and then withdrawing again based on the new approval. The `increaseApproval` function was added to the ERC20 standard to work around this problem. This attack is conceivable from the module's side when token owner is changing the budget.

Recommendation
Use the `increaseApproval` and `decreaseApproval` functions when modifying a module's budget

AMENDED [2018-7-24]:
This issue is no longer present in commit `8d1f586cccea2721cf93cebb2d0e84e159991241`.

4. Possible future re-entrancy vulnerability in CappedSTO based contracts

By adding a `TransferManager` module when `_deliverTokens` is called, `SecurityToken` owner can execute arbitrary code, potentially re-entering the `CappedSTO` contract. This isn't immediately exploitable, but one could easily be introduced after `_updatePurchasingState` and `_postValidatePurchase` are implemented in a derived contract.

Recommendation
Add a reentrancy lock mechanism in the capped STO contract to prevent future issues.

AMENDED [2018-7-24]:
This issue is no longer present in commit `8d1f586cccea2721cf93cebb2d0e84e159991241`.

5. Registries can be front run and “ransomed”

Ticker reservation currently allows a potential attacker to monitor mempool for reservation transactions and frontrun them to squat the desired ticker, then possibly offering it to the original buyer for a “ransom” via smart contract.

Recommendation
Consider using a more sophisticated reservation method, e.g. commit-reveal or follow ENS’ approach.

Client’s response:
We don't want to over complicate the token registration process. We understand there's the potential for someone to build a service to front-run token registration. Still, they would need to pay the POLY fee, and if such a thing happened we can always deploy a new ticker registry which implements other "security" measures down the road.

6. Dividend mechanism may return incorrect checkpoint Balances

The function `getValueAt` may return incorrect results if checkpoints are in far enough in the past, mostly because it doesn't consider that a specific `checkpointId` might not be present in the checkpoints array.

Example:
If the function is called with this parameters:

```
getValueAt([{0,0},{8,10},{15,20}], 5, 20)
```

The return will be 10 for the value at the id 5. But in reality, this user still had 0 balance in the fifth checkpoint. This same scenario happens with a lot of other inputs.

Recommendation
Adapt this function to consider bigger jumps in the continuity of `checkpointsId` inside the checkpoints array. When the desired id is between the current index and next, return the lowest one.

Client’s response:
The checkpoint functionality stores the current (i.e. before the `transfer` is processed) balance if there has been a new checkpoint added since the previous `transfer` operation.

So in the given example, the correct balance at id 5 would be 10, and this would be returned.

i.e. Suppose Alice has 0 tokens, and receives 10 tokens at block 8, the code will store {8, 0} in the checkpoint (i.e. at block 8, the balance before the transfer is 0).

Please recheck your understanding of the above logic and the correctness of the above.

AMENDED [2018-7-24]:
Given the client's response, this has been verified as not an issue.

Minor

7. Ticker string is not converted to uppercase in addCustomSecurityToken of SecurityTokenRegistry contract, leading to possible creation of unreachable tokens

---

In contrast to `generateSecurityToken`, `addCustomSecurityToken` does not convert the provided ticker string to uppercase before creating an entry in the symbols mapping. This can lead to registering tokens that are unreachable by the `getSecurityTokenAddress` function.

Recommendation
Call the `upper()` function before creating the symbol.

AMENDED [2018-7-24]:
This issue is no longer present in commit `8d1f586cccea2721cf93cebb2d0e84e159991241`.

8. Wrong update of contribution counters in allocateTokensMulti function of PresaleSTO contract

---

ETH and POLY contribution arguments in the `allocateTokensMulti` function are implemented as totals for all contributions in the batch, but the numbers are added for each contributor.

Recommendation
Adjust this function to consider the total amount of tokens only once.

AMENDED [2018-7-24]:
This issue is no longer present in commit `8d1f586cccea2721cf93cebb2d0e84e159991241`.

9. Inconsistency in expiry limit

The comment in the code contradicts the actual variable value.

TickerRegistry.sol / Line 19-20

```
    // For now it's value is 90 days;
     uint256 public expiryLimit = 15 * 1 days;
```

Recommendation
Adjust either the comment or the variable to make it consistent.

AMENDED [2018-7-24]:
This issue is no longer present in commit `8d1f586cccea2721cf93cebb2d0e84e159991241`.

10. No validation that owner != address(0) when registering ticker

A ticker registration with `address(0)` as owner will appear to succeed and user will be charged. However, such a registration will always return false when `expiryCheck` is called on it. Impact is very limited, but a user who mistakenly did not provide an owner address will end up paying for nothing with no clear feedback of what went wrong.

Recommendation
Add a require statement forbidding address(0) as an owner.

AMENDED [2018-7-24]:
This issue is no longer present in commit `8d1f586cccea2721cf93cebb2d0e84e159991241`.

11. No way to deprecate address keys in Registry

Once a key is registered there's no way to deprecated/deactivate it: you can only change the address it's mapped to. It may prove useful to add this functionality.

Recommendation
Consider implementing a deprecating mechanism.

Client's response:
We added a `PolymathRegistry` contract which holds the addresses of:
- `polymathRegistry`;
- `moduleRegistry`;
- `securityTokenRegistry`;
- `tickerRegistry`;
- `polyToken`;

This contract is passed into all of our relevant contracts, and is referenced via `RegistryUpdater.sol` whenever our contracts need these addresses.

`PolymathRegistry` is centrally maintained by Polymath and can be updated to reflect new deployments. If addresses are modified, dependent contracts would call `updateFromRegistry` to pick up new addresses (i.e. new addresses are pulled in, not pushed in).

`PolymathRegistry` will only ever be deployed once, and it's address cannot be updated in any dependent contracts.

12. Percentage transfer manager only works with 18 decimal tokens

---

While is currently only possible to deploy tokens with 18 decimals; it is possible to manually add STOs with different decimal places and the percentage manager only expects tokens with 18 decimals.

Recommendation
Read the STO's `decimals()` function and use it when calculating percentages.

AMENDED [2018-7-24]:
This issue is no longer present in commit `8d1f586cccea2721cf93cebb2d0e84e159991241`.

13. Polymath can seize security token registrations

Polymath can change which contract a registry entry points to without restriction. This gives Polymath significant privileges, most notably: by swapping out which address `getAddress("TickerRegistry")` points to, Polymath can overwrite any ST registration (the new contract would not respect the fact that the ticker was already registered). This could be done and swapped back to the original address in one transaction, causing no disruption to users otherwise.

Similarly, `setProtocolVersion` allows Polymath to overwrite any given version, which seems contrary to the intent of versioning.

Recommendation
Consider adding a speed bump to Registry updates/protocol upgrades.

Client's response:
We need to launch with some privileges, even though we will not use them.

Notes

14. Consider implementing proxied delegate call for gas savings

The factory pattern is widely used, but it's significantly more expensive than the delegate proxy pattern with respect to deploying new contracts. The delegate proxy approach allows the bulk of the required code to be deployed only once; switching to it would enable massive gas savings in the long term. The recommended approach is described here.

Client's response:
Adding to backlog for investigation in a future release.

15. Consider restricting the usage of STO modules

Token owners have the ability to use STO modules very freely. It's possible to use multiple STO contracts simultaneously, to deploy a `PRESALESTO` at any given time, etc. While there's no security issue by itself, token offerings should be as transparent as possible; and this flexibility opens the possibility for exploitative behaviour by token owners (e.g. distributing more tokens, diluting the shares of the previous holders).

Client's response:
We added the ability for the issuer to transparently freeze all new minting from STOs at the SecurityToken level. This will give investors a guarantee that the issuer can no longer mint additional tokens.

We also removed the "locked module" functionality in favour of this more transparent approach for STO modules.

We will also be building a ST20 token explorer showing investors all attached modules (including STOs) for investor transparency.

16. Interface naming convention is misleading

The entire project makes heavy use of contracts that are nominally interfaces, but in actuality contain implementations. Partial implementations are more appropriately termed “abstract contracts”, especially since `interface` is a keyword in Solidity that enforces specific restrictions. Calling these contracts “interfaces” obfuscates the fact that logic has been implemented in them that will be inherited.

Client’s response:
Doesn't cause any immediate concerns. Will gradually fix in future releases.

17. CappedSTO related issues

1. Final transaction in the `CappedSTO` must be exact
A more user friendly approach would be to accept any amount and send the rest back as change. This way, the transaction that tops the cap does not need to be exact.

2. Consider adding whale protection mechanism in the `CappedSTO`
While transfer can be restricted in the `PercetageTransferManager`, this two modules might not work well if the token sale is starting (since the total supply is near zero), and therefore there's no effective way to prevent a whale from buying the majority of the tokens in a offering.

3. `CappedSTO` can last forever
Token owners can avoid ever closing the offering by pausing and unpausing with a different end date. While this mechanism is useful it also opens up the possibility of never ending sales.

Client’s response:
CappedSTO can be seen as a model STO that will probably not find many real life use and will need to be customized further.

18. Misc

- `IModuleFactory` (`changeFactorySetupFee`, `changeFactoryUsageFee`, `changeFactorySubscriptionFee`): variable juggling unnecessary if event emitted first
- `validAddressKeys` in `Registry` contract could be removed and `storedAddresses` value could be checked instead
- hashing before comparison in the `isSecurityToken` function of `SecurityTokenRegistry` probably unnecessary
- The POLY paid as a fee in the `SecurityTokenRegistry` could be transferred directly to the owner removing the need to periodically withdraw
- `securityTokenVersion` in `SecurityToken` contract should probably be declared as a constant
- `transferFunctions` could be precalculated constant hashes
- module type locking can happen in `addModule`, but doesn't emit event, inconsistent
- malicious module factory tokens can bypass `.owner()` and `.getType()` checks in `ModuleRegistry` contract
- `DummySTO` doesn't use the declared `ADMIN` permission
- Consider breaking loops that finalised their state, specifically those in onlyModule and pruneInvestors functions of `SecurityToken` contract
- Consider using modulo for check in `checkGranularity` function of `SecurityToken` contract
- `GeneralPermissionManagerFactory.sol:72` wrong array size
- `IModule`: `FEE_ADMIN` should be a constant

Client's response:
All fixed as per recommendations except for:
"transferFunctions could be precalculated constant hashes" where we have decided to not use pre-calculated values for easier transparency / readability.

AMENDED [2018-7-24]:
These issues is no longer present in commit `8d1f586cccea2721cf93cebb2d0e84e159991241`.

SOLIDIFIED

Audit Report for Polymath. July 7, 2018.

Closing Summary

Critical and major issues were found during the audit that could break the intended behaviour. The project generally takes an approach of allowing operators (both Polymath and Security Token owners) to inject arbitrary code with very little validation and does not account for the possibility of front-running. This significantly increases the project's attack surface. We recommend fixing these issues before a production release.

AMENDED [2018-7-24]:
Polymath has addressed all major and critical issues reported.

Disclaimer

Solidified Technologies Inc.