SOLIDIFIED

Audit Report for CardStack - July 27, 2021

## Summary

Audit Report prepared by Solidified covering the CardStack smart contracts.

## Process and Delivery

Three (3) independent Solidified experts performed an unbiased and isolated audit of the code. The debrief on 31 May 2021.

A second audit was performed by the same team of auditors on an expanded codebase and concluded on 27 July 2021.

## Audited Files

The source code has been supplied in the form of two GitHub repositories:

https://github.com/cardstack/card-protocol-xdai/tree/audit-2/

Commit number: `1f4bc845367b207e4bea5ff224fe26b7cfa6b495`

https://github.com/cardstack/tokenbridge-contracts/tree/audit-2

Commit number: `e44429d86e05bb39552823dcd5976b69e2e7dca1`

The scope of the audit was limited to the following files:

```
contracts
├── ERC677BridgeToken.sol
├── ERC677BridgeTokenRewardable.sol
├── ERC677MultiBridgeToken.sol
├── Migrations.sol
├── PermittableToken.sol
├── interfaces
│   ├── ERC677.sol
│   ├── ERC677Receiver.sol
│   ├── ERC721.sol
│   ├── IAMB.sol
│   ├── IBlockReward.sol
│   ├── IBridgeMediator.sol
│   ├── IBridgeUtils.sol
│   ├── IBridgeValidators.sol
│   ├── IBurnableMintableERC677Token.sol
│   ├── IChai.sol
│   ├── IGasToken.sol
```

```
│   ├── IMediatorFeeManager.sol
│   ├── IMintHandler.sol
│   ├── IPot.sol
│   ├── IRewardableValidators.sol
│   └── IUpgradeabilityOwnerStorage.sol
├── libraries
│   ├── Address.sol
│   ├── ArbitraryMessage.sol
│   ├── Bytes.sol
│   ├── Message.sol
│   ├── SafeERC20.sol
│   └── TokenReader.sol
├── upgradeability
│   ├── EternalStorage.sol
│   ├── EternalStorageProxy.sol
│   ├── OwnedUpgradeabilityProxy.sol
│   ├── Proxy.sol
│   ├── README.md
│   ├── UpgradeabilityOwnerStorage.sol
│   ├── UpgradeabilityProxy.sol
│   └── UpgradeabilityStorage.sol
└── upgradeable_contracts
    ├── BaseBridgeValidators.sol
    ├── BaseERC677Bridge.sol
    ├── BaseFeeManager.sol
    ├── BaseMediatorFeeManager.sol
    ├── BaseOverdrawManagement.sol
    ├── BaseRewardAddressList.sol
    ├── BasicAMBMediator.sol
    ├── BasicBridge.sol
    ├── BasicForeignBridge.sol
    ├── BasicHomeBridge.sol
    ├── BasicTokenBridge.sol
    ├── BlockRewardBridge.sol
    ├── BlockRewardFeeManager.sol
    ├── BridgeValidators.sol
    ├── ChaiConnector.sol
    ├── ChooseReceiverHelper.sol
    ├── Claimable.sol
    ├── DecimalShiftBridge.sol
    ├── ERC20Bridge.sol
    ├── ERC677Bridge.sol
    ├── ERC677BridgeForBurnableMintableToken.sol
    ├── ERC677Storage.sol
    ├── FeeTypes.sol
    ├── GasTokenConnector.sol
    ├── HomeOverdrawManagement.sol
    ├── Initializable.sol
    ├── InitializableBridge.sol
    ├── InterestReceiver.sol
    ├── MediatorBalanceStorage.sol
```

Audit Report for CardStack - July 27, 2021

```
├── MessageRelay.sol
├── OtherSideBridgeStorage.sol
├── Ownable.sol
├── README.md
├── ReentrancyGuard.sol
├── RewardableBridge.sol
├── RewardableMediator.sol
├── RewardableValidators.sol
├── Sacrifice.sol
├── TokenBridgeMediator.sol
├── TokenSwapper.sol
├── TransferInfoStorage.sol
├── Upgradeable.sol
├── Validatable.sol
├── ValidatorStorage.sol
├── ValidatorsFeeManager.sol
├── VersionableBridge.sol
├── amb_erc20_to_native
│   ├── BasicAMBErc20ToNative.sol
│   ├── BlockReward.sol
│   ├── ForeignAMBErc20ToNative.sol
│   ├── HomeAMBErc20ToNative.sol
│   └── HomeFeeManagerAMBErc20ToNative.sol
├── amb_erc677_to_erc677
│   ├── BasicAMBErc677ToErc677.sol
│   ├── BasicStakeTokenMediator.sol
│   ├── ForeignAMBErc677ToErc677.sol
│   ├── ForeignStakeTokenMediator.sol
│   ├── HomeAMBErc677ToErc677.sol
│   ├── HomeStakeTokenFeeManager.sol
│   ├── HomeStakeTokenMediator.sol
│   └── callflows.md
├── arbitrary_message
│   ├── BasicAMB.sol
│   ├── BasicForeignAMB.sol
│   ├── BasicHomeAMB.sol
│   ├── ForeignAMB.sol
│   ├── ForeignAMBWithGasToken.sol
│   ├── HomeAMB.sol
│   ├── MessageDelivery.sol
│   ├── MessageProcessor.sol
│   └── VersionableAMB.sol
├── erc20_to_erc20
│   ├── BasicForeignBridgeErcToErc.sol
│   ├── FeeManagerErcToErcPOSDAO.sol
│   ├── ForeignBridgeErc677ToErc677.sol
│   ├── ForeignBridgeErcToErc.sol
│   ├── HomeBridgeErcToErc.sol
│   ├── HomeBridgeErcToErcPOSDAO.sol
│   └── RewardableHomeBridgeErcToErc.sol
├── erc20_to_native
```

```
    │   ├── FeeManagerErcToNative.sol
    │   ├── FeeManagerErcToNativePOSDAO.sol
    │   ├── ForeignBridgeErcToNative.sol
    │   ├── HomeBridgeErcToNative.sol
    │   └── RewardableHomeBridgeErcToNative.sol
    ├── erc721_to_erc721
    │   ├── ERC721BurnableMintable.sol
    │   ├── ERC721Proxy.sol
    │   ├── interfaces
    │   │   ├── IForeignNftMediator.sol
    │   │   └── IHomeNftMediator.sol
    │   └── mediator
    │       ├── BasicNftMediator.sol
    │       ├── ERC721Bridge.sol
    │       ├── ForeignNftMediator.sol
    │       └── HomeNftMediator.sol
    ├── multi_amb_erc20_to_erc677
    │   ├── BasicMultiAMBErc20ToErc677.sol
    │   ├── BasicMultiTokenBridge.sol
    │   ├── ForeignMultiAMBErc20ToErc677.sol
    │   ├── HomeFeeManagerMultiAMBErc20ToErc677.sol
    │   ├── HomeMultiAMBErc20ToErc677.sol
    │   ├── MultiTokenBridgeMediator.sol
    │   ├── MultiTokenForwardingRules.sol
    │   ├── TokenProxy.sol
    │   └── callflows.md
    └── native_to_erc20
        ├── FeeManagerNativeToErc.sol
        ├── FeeManagerNativeToErcBothDirections.sol
        ├── ForeignBridgeNativeToErc.sol
        ├── HomeBridgeNativeToErc.sol
        ├── RewardableForeignBridgeNativeToErc.sol
        └── RewardableHomeBridgeNativeToErc.sol
```

```
contracts
├── ActionDispatcher.sol
├── BridgeUtils.sol
├── Exchange.sol
├── MerchantManager.sol
├── PrepaidCardManager.sol
├── RevenuePool.sol
├── RewardPool.sol
├── SupplierManager.sol
├── TokenManager.sol
├── action-handlers
│   ├── PayMerchantHandler.sol
│   ├── RegisterMerchantHandler.sol
│   ├── SplitPrepaidCardHandler.sol
│   └── TransferPrepaidCardHandler.sol
├── core
│   ├── Safe.sol
│   └── Versionable.sol
├── dev
```

```
│   ├── DevDependencies.sol
│   └── MockDIAOracle.sol
├── oracles
│   ├── ChainlinkFeedAdapter.sol
│   ├── DIAOracleAdapter.sol
│   ├── IDIAOracle.sol
│   ├── IPriceOracle.sol
│   └── ManualFeed.sol
├── roles
│   └── SPENDMinterRole.sol
└── token
    ├── ERC677Token.sol
    ├── ERC677TransferReceiver.sol
    ├── IERC677.sol
    ├── ISPEND.sol
    └── SPEND.sol
```

## Intended Behavior

The smart contracts implement a payment protocol implemented on the XDai L2 chain and the related bridge smart contracts that allow migrating the assets to and from the Ethereum network.

## Code Complexity and Test Coverage

Smart contract audits are an important step to improve the security of smart contracts and can find many issues. However, auditing complex codebases has its limits and a remaining risk is present (see disclaimer).

Users of a smart contract system should exercise caution. In order to help with the evaluation of the remaining risk, we provide a measure of the following key indicators: **code complexity**, **code readability**, **level of documentation**, and **test coverage**.

**Note, that high complexity or lower test coverage does equate to a higher risk. Certain bugs are more easily detected in unit testing than a security audit and vice versa. It is, therefore, more likely that undetected issues remain if the test coverage is low or non-existent.**

| Criteria | Status | Comment |
|---|---|---|
| Code complexity | Medium | - |
| Code readability and clarity | High | - |
| Level of Documentation | Medium-High | - |
| Test Coverage | High | - |

SOLIDIFIED

Audit Report for CardStack - July 27, 2021

## Issues Found

Solidified found that the CardStack contracts contain no critical issues, 1 major issue, 4 minor issues, in addition to 9 informational notes.

We recommend all issues are amended, while the notes are up to the team's discretion, as they refer to best practices.

| Issue # | Description | Severity | Status |
|---|---|---|---|
| 1 | Car Protocol: Action Handlers: All action-handlers do not validate the msg.sender within onTokenTransfer() functions | Critical | Pending |
| 2 | Card Protocol: Erroneous / Unconventional usage OpenZeppelin upgradability system | Major | Non-issue |
| 3 | Card Protocol: All action-handlers trust actionData part which is parsed from data parameter | Minor | Pending |
| 4 | Card Protocol: core/Exchange.sol: Listing tokens by Symbol may lead to collisions | Minor | Acknowledged |
| 5 | Token bridge: Re-entrancy fix not propagated from original POA bridge codebase | Minor | Resolved |
| 6 | Token bridge: Insecure random number generation in fee payout distribution | Minor | Pending |
| 7 | BlockReward.sol has hardcoded value that cannot make it to deploy | Minor | Resolved |
| 8 | Card Protocol: Enable Merchants to add signers to safe | Note | - |
| 9 | Card Protocol: Spelling mistake in PrepaidCardManager.sol | Note | Resolved |
| 10 | Card Protocol: ForeignAMBErc20ToNative.sol does not consistently use safe math | Note | - |
| 11 | Token bridge: BridgesAllowed.sol: function | Note | Resolved |

| | | | |
|---|---|---|---|
| | reward() takes unnecessary parameters | | |
| 12 | Card Protocol: Operations.md: Incorrectly documents the behaviour of RevenuePool.sol contract | Note | Resolved |
| 13 | Card Protocol: BridgeUtils.sol: updating a non-existent token adds a new one | Note | - |
| 14 | Token bridge: ForeignMultiAMBErc20ToErc677.sol: Bridge is not compatible with tokens that charge a fee on transfers | Note | - |
| 15 | Token bridge:  PermittableToken.sol uses now instead of block.timestamp | Note | Resolved |

# Critical Issues

## 1. Car Protocol: Action Handlers: All action-handlers do not validate the msg.sender within onTokenTransfer() functions

The action handlers (`PayMerchantHandler.sol`, `RegisterMerchantHandler.sol`, `SplitPrepaidCardHandler.sol` and `TransferPrepaidCardHandler.sol`) in their `onTokenTransfer()` function implementation do not check if `msg.sender` is the expected token's contract.
As a result the caller of those functions can specify any `from`, `amount` and `data` parameters. For example a malicious actor could call the `PayMerchantHandler.onTokenTransfer()` from a contract which implements ERC-20 interface and returns a well known `symbol()` to mint SPEND tokens for free to a `merchantSafe`. This particular attack would utilize the issue #2 to succeed - the `Exchange.sol` relies on the token's symbol as a unique identifier (issue #2).

**Recommendation**
Consider validating `msg.sender` in `onTokenTransfer()` function implementations.

# Major Issues

## 2. Card Protocol: Erroneous / Unconventional usage OpenZeppelin upgradability system

The CardStack protocol contracts follow a proxy / delegatecall upgradability pattern, declaring storage gaps and using `setup()` functions instead of constructors. The upgradable versions of OpenZeppelin libraries are used and the OpenZeppelin upgradability plugin for Truffle is used. However, these OZ libraries are not initialized using the OpenZeppelin-provided `initialize()` function. This could lead to unexpected consequences.
In addition, initialization code is provided in the `setup()` function, meaning the OZ deployment system will not automatically invoke it. If the intention is to invoke `setup()` manually, there is no code to check that the function has been called before contracts are used. This also means that setup can be called several times, which may or may not be intentional.

**Recommendation**

Consider using the OpenZeppelin provided support for upgradability as described at: https://docs.openzeppelin.com/contracts/3.x/upgradeable

**Status:** Non issue

During further verification with the team it has been established that the OpenZeppelin-provided `initialize()` function is used in relevant places. The use of a `setup()` function that can be called several times is intentional and these parameters have been carefully selected to be modifiable.

## Minor Issues

### 3. Card Protocol: All action-handlers trust actionData part which is parsed from data parameter

The function `onTokenTransfer()` in `ActionHandler.sol` does not validate `actionData` which is parsed from `data` parameter. The `actionData` can have arbitrary values set by a malicious user transferring tokens to `ActionHandler.sol`.

**Recommendation**
Perform basic data validation on these inputs

### 4. Card Protocol: core/Exchange.sol: Listing tokens by Symbol may lead to collisions

The contract stores the `ExchangeInfo` data structure per token indexed by the hash of the token symbol. This may lead to collisions with tokens with the same symbol. The issue is somewhat mitigated by the fact that tokens can only be added by the owner of the contract. However, there are no checks for a token already being registered, so accidental overwrite could occur.
In addition, ERC-20 tokens have occasionally changed their symbol. One well-known example is the original DAI token becoming SAI.

**Recommendation**
Consider using the address of the token instead for indexing exchange data.

**Status:** Acknowledged

**Team Reply:**

*“There should not be a collision here. The TokenBridge has an "allow list" of the specific token addresses that are allowed to be bridged into the L2 network, and specifically the TokenBridgecalls the BridgeUtils contract to set the specific token addresses that the cardpay protocol can accepts as valid token payments. The reason we are using token symbols here is that the token contracts that we will use in layer 2 are actually created on-the-fly by the token bridge contract, so there is no way that we will know in advance what the actual CPXD token contract addresses will be. However, we have complete control over which token contracts we accept as valid tokens from the token bridge's allow list, and hence which token contracts we will be deriving exchange rates for. As such, it is not possible for a  token to actually participate in our protocol, and hence the only exchange rates we will be generating for tokens are the specific token addresses that we allow into our network.”*

## 5. Token bridge: Re-entrancy fix not propagated from original POA bridge codebase

The original codebase from which the token bridge was forked has received a security fix in the form of a reentrancy patch:

https://github.com/poanetwork/tokenbridge-contracts/commit/44c84e4be06945c543ec05c59f0e36ff823631cb

This patch has not been incorporated into the current codebase.

**Recommendation**
Consider implementing the fix in the forked codebase.

## 6. Token bridge: Insecure random number generation in fee payout distribution

The token bridge uses the previous block’s blockhash in a number of places to generate a random number:

```
function random(uint256 _count) internal view returns (uint256) {
        return uint256(blockhash(block.number.sub(1))) % _count;
}
```

This seems to be used to randomize account lists for fee distribution. Using the blockhash for random number generation is insecure, since it is a value that can be known by a calling smart contract. However, it would be hard and probably not cost effective for a user to exploit this.

**Recommendation**
Consider using a more secure random number generator.

## 7. BlockReward.sol has hardcoded value that cannot make it to deploy

The function bridgesAllowed() hardcodes a return value of the 0x address, with a note that this value must be updated before deployment.

**Recommendation**
Use a state variable that gets initialized in the constructor to prevent this incorrect value from making it into production.

## Informational Notes

## 8. Card Protocol: Enable Merchants to add signers to safe

The system uses a Gnosis Safe for each merchant. This is created with 1 owner and a signature threshold of 1. Such a setup has no additional security compared to a standard smart contract wallet or external wallet.

**Recommendation**
Consider providing a UI option for merchants to interact directly with the Gnosis Safe and encourage them to add signatories and increase the threshold.

## 9. Card Protocol: Spelling mistake in PrepaidCardManager.sol

The constant `TRANSER_AND_CALL` is spelled incorrectly.

**Recommendation**
Fix change spelling to `TRANSFER_AND_CALL`.

## 10. Card Protocol: ForeignAMBErc20ToNative.sol does not consistently use safe math

The function `fixMediatorBalance()` uses safe math inconsistently. On line 126 safe math is not used, and on 133 it is.

**Recommendation**
Add `SafeMath` to all mathematical operations.

## 11. Token bridge: BridgesAllowed.sol: function reward() takes unnecessary parameters

The function `reward` takes in two arrays that are then required to be empty.

**Recommendation**
Remove arrays.

## 12. Card Protocol: Operations.md: Incorrectly documents the behavior of RevenuePool.sol contract

The `Operations.md` sections `Add Tally` and `Remove Tally` state that `SPENT` tokens can be redeemed on the merchant's behalf. However, only the Payable Tokens can be redeemed.

**Recommendation**
Update `Operations.md` to match the implementation.

## 13. Card Protocol: BridgeUtils.sol: updating a non-existent token adds a new one

The function `updateToken()` adds a payable token if it has not been added before. This may or may not be intentional but is counterintuitive.

**Recommendation**
Consider only allowing a token to be updated if it has been added before.

## 14. Token bridge: `ForeignMultiAMBErc20ToErc677.sol`: Bridge is not compatible with tokens that charge a fee on transfers

The token bridge is not compatible with tokens that charge a fee on transfers. This is documented in the original implementation: https://github.com/poanetwork/tokenbridge-contracts/commit/1166e00c5b976295914d625504f3b53b7bfc945c. In general, care must be taken with ERC-20 tokens that implement additions to the usual behavior, such as changing fees, or limiting transfers, etc.

**Recommendation**
Ensure that only compatible tokens are used.

## 15. Token bridge: `PermittableToken.sol` uses `now` instead of `block.timestamp`

The function `_now()` returns `now`. `now` has been deprecated in later versions of solidity.

**Recommendation**
Replace `now` with `block.timestamp`.

Audit Report for CardStack - July 27, 2021

## Disclaimer

Solidified audit is not a security warranty, investment advice, or an endorsement of CardStack or its products. 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.

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.*