Quantstamp Smart Contract Audit

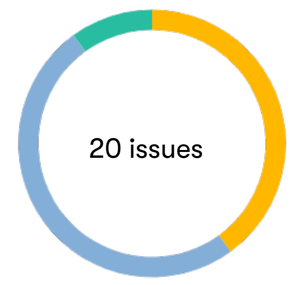
Thunder Bridge

This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

Quantstamp helps to secure blockchain applications such as smart contracts. We are developing a new protocol for smart contract verification, performing professional audits and consultations, and developing security tools. Quantstamp also has expertise in application security and secure software development.

Executive Summary

Type	ERC20-to-ERC20 Token Bridge Contract
Auditors	Yohei Oka, Forward Deployed Engineer Martin Derka, Senior Research Engineer Nadir Akhtar, Research Engineer Chang Yu, Research Engineer
Timeline	2019-07-09 through 2019-07-19
EVM	Constantinople
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	README.md
Source Code	Repository: thunder_bridge — Commit: afeed3f
Total Issues	**20** (12 Resolved)
High Risk Issues	0
Medium Risk Issues	0
Low Risk Issues	**8** (4 Resolved)
Informational Risk Issues	**10** (7 Resolved)
Undetermined Risk Issues	**2** (1 Resolved)



Overall Assessment

Upon the delivery of the first report, ThunderCore provided clarification on the issues that were pointed out. Quantstamp has since updated the report and are waiting on the updated code to verify the fixes.

While a high level overview is provided as part of the `README`s, Quantstamp also requests that more detailed documentation is provided, namely those regarding expected flows and contract calls for token swaps, contract deployment steps, and contract upgrade steps.

Severity Categories	
High	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.

Goals

- Check that user funds are safe
- Check for potential upgradability issues
- Check that token swaps are handled correctly
- Check for other security vulnerabilities

Changelog

- 2019-07-15 - Initial Report
- 2019-07-19 - Updated Report

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The below notes outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Truffle v4.0.11
- Mythril v0.20.4
- MAIAN commit sha: ab387e1
- Securify
- NodeJS v10.15.1

Steps taken to run the tools:

1. Installed Truffle: `npm install -g truffle`
2. Flattened the source code using `truffle-flattener` to accommodate the auditing tools.
3. Installed the Mythril tool from Pypi: `pip3 install mythril`
4. Ran the Mythril tool on each contract: `myth -x path/to/contract`
5. Ran the Securify tool: `java -Xmx6048m -jar securify-0.1.jar -fs contract.sol`
6. Cloned the MAIAN tool: `git clone --depth 1 https://github.com/MAIAN-tool/MAIAN.git maian`
7. Ran the MAIAN tool on each contract: `cd maian/tool/ && python3 maian.py -s path/to/contract contract.sol`
8. Installed NodeJS from here

Assessment

Findings

Inconsistent Layout of State Variables

Severity: *Low*

Contract(s) affected: `EternalStorageProxy.sol`, `HomeBridgeErcToErc.sol`, `ForeignBridgeErcToErc.sol`

Description:
The layout of the state variables between the `EternalStorageProxy.sol` and either `HomeBridgeErcToErc.sol` or `ForeignBridgeErcToErc.sol` are inconsistent. The layout of `EternalStorageProxy.sol` is

```
_upgradeabilityOwner: address
_version: uint256
_implementation: address
uintStorage: mapping(bytes32 => uint256)
stringStorage: mapping(bytes32 => string)
addressStorage: mapping(bytes32 => address)
bytesStorage: mapping(bytes32 => bytes)
boolStorage: mapping(bytes32 => bool)
intStorage: mapping(bytes32 => int256)
```

when it should be

```
uintStorage: mapping(bytes32 => uint256)
stringStorage: mapping(bytes32 => string)
addressStorage: mapping(bytes32 => address)
bytesStorage: mapping(bytes32 => bytes)
boolStorage: mapping(bytes32 => bool)
intStorage: mapping(bytes32 => int256)
_upgradeabilityOwner: address
_version: uint256
_implementation: address
```

It is unlikely that this will lead to data corruption with the current implementation. However discrepancy could lead to potential unexpected behavior in the future depending on how the Proxy and upgraded contracts are implemented.

Exploit Scenario:
The tests read and write the storage from the context of the implementation contracts, which would show as consistent. However, writing from the context of the implementation and reading from the context of the proxy contract will lead to inconsistency. For example, appending the following function to `EternalStorageProxy`:

```
contract EternalStorageProxy is OwnedUpgradeabilityProxy, EternalStorage {

    function gasPriceAlternate() public view returns(uint256) {

        return uintStorage[keccak256(abi.encodePacked("gasPrice"))];

    }

}
```

would lead to differing results when calling `gasPrice()` and `gasPriceAlternate()`.

Recommendation:
Flip the order of the inherited contracts in `EternalStorageProxy` to be

```
contract EternalStorageProxy is EternalStorage, OwnedUpgradeabilityProxy
```

Please also refer to the following link for more information on best practices concerning upgradable contracts.

https://blog.trailofbits.com/2018/09/05/contract-upgrade-anti-patterns/

Unprotected Initialization Function

Severity: *Low*

Status: Fixed

Contract(s) affected: `BridgeValidators.sol`, `ForeignBridgeErcToErc.sol`, `HomeBridgeErcToErc.sol`

Description:
[Update] The ThunderCore team is aware of this issue and will use `upgradeToAndCall()` *when performing upgrades.*

The contracts `BridgeValidators.sol`, `ForeignBridgeErcToErc.sol`, and `HomeBridgeErcToErc.sol` have a function called `initialize()` that can be called by anyone. Furthermore, the other functions in the contracts don't check if the contract has been initialized yet. While most of the functions are bound to fail due to lack of initialization, they shouldn't be able to be called before contract initialization.

Exploit Scenario:
While the contracts don't provide much value without being initialized, a malicious actor to could call `initialize()` beforehand and could potentially steal funds if a deposit is made before the real owner calls `initialize()`.

Step:

1. `ForeignBridgeErcToErc.sol` is deployed
2. `ERC20` tokens are deposited to the contract for future use
3. Malicious actor calls `initialize()` before the real owner and uses a bogus address for the `_erc20token` parameter
4. Malicious actor withdraws tokens using `claimTokens()`

Recommendation:
Move initialization code to the constructor so it is called on deployment.

Old Solidity Version

Severity: *Low*

Status: Fixed

Description:
[Update] Feedback from the ThunderCore team: "We cannot use higher version of solidity as ThunderCore EVM does not support St. Petersburg or Constantinople changes to the EVM. Latest Solidity versions presume the latest EVM version."

As security standards develop, so does the Solidity language. In order to stay up to date with current practices, it's important to use a recent version of Solidity and recent conventions.

Recommendation:
Upgrade to the latest Solidity version.

Inconsistent enforcement of conditions

Severity: *Low*

Contract(s) affected: `ForeignBridgeErcToErc.sol`, `HomeBridgeErcToErc.sol`, `BasicBridge.sol`

Description:
[Update] Currently being addressed

Certain conditions are maintained and checked in one direction but not the other.

Eg. When setting `executionMaxPerTx`, `executionMaxPerTx` is required to be less than `executionDailyLimit`. However, when `executionDailyLimit` is updated, it is not checked whether it is greater than `executionMaxPerTx`, breaking the invariant that `executionMaxPerTx` is always less than `executionDailyLimit`. The same issue can be applied to `minPerTx`, `maxPerTx`, and `dailyLimit`.

Recommendation:
Check that conditions are satisfied in both directions when updating numbers.

Gas Usage / `for` Loop Concerns

Severity: *Low*

Status: Fixed

Contract(s) affected: `BridgeValidators.sol`, `ForeignBridgeErcToErc.sol`, `HomeBridgeErcToErc.sol`, `Message.sol`

Description:
[Update] ThunderCore has confirmed that they expect the maximum number of validators to be around 5. For the practical situation, 5 validators will not cause significant gas and transaction cost issues, but moving to a much higher number of validators will significantly increase the transaction costs of signing and verifying validators. In the case of having 155 validators, the cost of calling initialize in BridgeValidators.sol is almost the block gas limit on the Ropsten testnet, with having 156 validators exceeding the limit.

Gas usage is a main concern for smart contract developers and users, since high gas costs may prevent users from wanting to use the smart contract. Even worse, some gas usage issues may prevent the contract from providing services entirely. For example, if a `for` loop requires too much gas to exit, then it may prevent the contract from functioning correctly entirely. It is best to break such loops into individual functions as possible.

One example is `removeValidator()` in `BridgeValidators.sol`. If the number of validators becomes too big, `removeValidator()` may perpetually fail due to gas issues. The same thing can be said about `onExecuteMessage()` in `ForeignBridgeErcToErc.sol` and `onExecuteAffirmation()` in `HomeBridgeErcToErc.sol`

Recommendation:
Break up `for` loops if possible. Implement more efficient functions that don't require looping through an entire list. If a `for` loop is absolutely necessary, conduct stress testing to figure out the maximum number of iterations that can be made without running out of gas.

Inability to remove malicious validators

Severity: *Low*

Status: Fixed

Contract(s) affected: `BridgeValidators.sol`

Description:
[Update] In the scenario below, ThunderCore plans to add an honest validator before removing the malicious one

Validators cannot be removed if the number of validators is less than or equal to the required number of signatures due to the check on L51 `require(validatorsCount > requiredSignatures());`. This becomes problematic when there is a malicious validator that needs to be urgently removed. One way to get around this restriction is to temporarily reduce the number of required signatures, remove the validator, and reset the number of required signatures. However, the malicious validator may take advantage of this temporary reduction in the number of required signatures.

Recommendation:
Re-evaluate whether validators can be removed regardless of the number of required signatures.

`BridgeContract` can receive `BridgeTokens` without callback

Severity: *Low*

Contract(s) affected: `ERC677BridgeToken.sol`

Description:
[Update] Currently being addressed

`ERC677BridgeToken` is implemented so that if the tokens are sent to the registered `BridgeContract`, the `onTokenTransfer()` fallback function has to be called (L43, 62). However, tokens can be sent to the `BridgeContract` using the `transferFrom()` method.

Recommendation:
Implement fallback/callback functionality for the `transferFrom()` method.

Unchecked `delegatecall` Input

Severity: *Low*

Contract(s) affected: `Proxy.sol`

Description:
[Update] Currently being addressed

The default behavior of `delegatecall` returns `true` if the contract does not exist. This behavior could lead to unintended consequences with initializing the bridge contracts. See https://blog.trailofbits.com/2018/09/05/contract-upgrade-anti-patterns/ for additional information.

Recommendation:
Check `extcodesize > 0` before executing `delegatecall`.

Centralization of Power

Severity: *Undetermined*

Status: Fixed

Description:
[Update] ThunderCore is aware of this and will bear the responsibility of educating their users

Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. However, this centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

The contracts feature centralization through the use of multiple `onlyOwner` modifiers. Such features include being able to add and remove validators, being able to upgrade the contract at any time, unlimited minting of `ERC677BridgeToken`s, being able to set different parameters including fees and daily limits, and so on.

Recommendation:
Some of these features may be necessary to operate the platform but should be clearly communicated to users upfront. User should be informed of such centralization characteristics before deciding to use the platform.

Unable to verify security of potential upgrades

Severity: *Undetermined*

Description:
Proxy contracts are used as a mechanism to allow updates. The owner of the proxy contracts can update the contracts that implement the functionality at any time. At this moment, we cannot verify the security of potential upgrades as it depends on future implementations.

Recommendation:
Users should be informed well in advance when future upgrades are scheduled. Users should also be informed that the contracts can be upgraded at any moment.

`ForeignBridge` has no daily limit

Severity: *Informational*

Status: Fixed

Contract(s) affected: `ForeignBridgeErcToErc.sol`

Description:
[Update] We have confirmed with ThunderCore that this is an explicit design decision

`ForeignBridgeErcToErc` does not enforce a `dailyLimit`, `maxPerTx`, and `minPerTx` while `HomeBridgeErcToErc` does.

Recommendation:
Verify that such constraints are not necessary for the `ForeignBridge` contract and document why these decisions were made.

Unchecked Return Value

Severity: *Informational*

Contract(s) affected: `HomeBridgeErcToErc.sol`

Description:
[Update] `ERC677BridgeToken` *will revert in the case of any failure when minting so an explicit check of the return value may be unnecessary. However, we still recommend checking the return value so other ERC677 tokens can be supported and to be consistent with the* `ForeignBridgeErcToErc` *implementation.*

Most functions will return a `true` or `false` value upon success. Some functions, like `send()`, are more crucial to check than others. It's important to ensure that every necessary function is checked.

The return value of the `erc677token.mint()` function called within `onExecuteAffirmation()` is used only upon returning. However, it's possible that some of these intermediary mints may fail as well, and there is no guarantee that the token implementation is defined to revert upon failure. This means that possibility of error must be handled by the calling contract as well.

Recommendation:
To ensure that there is no unhandled failure, each of these return values should be considered, possibly reverting if appropriate, or returning the conjunction of all the return values.

Verify `parseMessage` logic

Severity: *Informational*

Status: Fixed

Contract(s) affected: `Message.sol`

Description:
[Update] The ThunderCore team has tested this behavior and confirmed that it works as expected. Quantstamp still recommends writing unit tests for this library.

The `contractAddress` is saved from offset 116 but because `mload()` always reads 32 bytes, the offset when reading the bytes is set to 104 (L42), similar to how the recipient address at offset 32 is read (offset is set to 20). The difference is that in the case of the `recipient` (L39), the prepended bytes from offset 20-31 are zerobytes, where as the prepended bytes from offset 104-115 are part of the `txHash` bytes and may not be zero.

Recommendation:
Write unit tests to verify that prepending 12 random bytes still results in reading the address correctly.

Reentrancy

Severity: *Informational*

Status: Fixed

Contract(s) affected: `ERC677BridgeToken.sol`

Description:
[Update] The current implementation doesn't pose a reentrancy threat where user funds can be drained. However, future updates and token contracts should be mindful of such vulnerability.

A reentrancy vulnerability is a scenario where an attacker can repeatedly call a function from itself, unexpectedly leading to potentially disastrous results. Here's a basic example representing the very attack which impacted The DAO in 2016:

```
function withdraw_with_reentrancy(uint256 _amount) public {
    msg.sender.call.value(_amount)();   // ATTACKER CAN CALL AGAIN BEFORE
                                        // FUNCTION TERMINATES because `call`
                // allows msg.sender to execute any code using fallback function
    balances[msg.sender] -= _amount;    // Balance only updated AFTER funds withdrawn
}
```

Exploit Scenario:
The ability for any contract to implement a `onTokenTransfer` function that acts like Ether's default fallback for ERC677 presents a potential reentrancy vulnerability. Consider a following lottery contract that uses ERC677 and implements the following method:

```
function withdrawWinnings() {
    // Allows multiple withdrawals because gas is not checked
    require(transferAndCall(winner, 100, bytes(0)));
}
```

A malicious winner could implement an😉 `onTokenTransfer()` fallback that can call this method, draining the entire lottery contract's ERC677 supply through reentrancy.

Recommendation:
The vulnerability can be prevented by adhering to the Checks-Effects-Interactions practice. Consider adding reentrancy protection in the token functions

eg. https://github.com/OpenZeppelin/openzeppelin-solidity/blob/master/contracts/utils/ReentrancyGuard.sol

Inconsistent implementation of abstract contracts

Severity: *Informational*

Status: Fixed

Contract(s) affected: `BasicHomeBridge.sol`, `ERC677Bridge.sol`, `ERC677Receiver.sol`

Description:
[Update] ThunderCore has confirmed the contract works as expected in their internal testing and has decided not to take any action at this moment. Quantstamp recommends fixing it for code consistency

Some contracts are kept abstract and require implementation in subclasses while some have default implementations and marked as abstract using comments.

- `BasicForeignBridge.sol` is an abstract class with functions left unimplemented (eg. L43)
- `BasicHomeBridge.sol` should be an abstract class but includes functions that always return `true` (eg. L152)
- `ERC677Bridge.sol` should be an abstract class but includes the comments `"has to be defined"` (L27)

Furthermore, contracts that don't have any functions implemented can be written as interfaces. Please refer to the latest version of OpenZeppelin contracts for guidance on naming conventions as well. For example, the code may be easier to read if abstract contracts are prepended with the word `Abstract`.

Recommendation:

- Keep functions unimplemented instead of using a default or empty implementation for abstract classes.
- Use `interface` instead of `contract` for contracts that don't have any implementation

Multiple Inheritance

Severity: *Informational*

Status: Fixed

Contract(s) affected: `BasicBridge.sol`, `HomeBridgeErcToErc.sol`, `BasicHomeBridge.sol`, `FeeManager.sol`, `BridgeValidators.sol`

Description:
[Update] ThunderCore has confirmed the contract works as expected in their internal testing and has decided not to take any action at this moment. Quantstamp recommends fixing it to remove code redundancy

Some contract have duplicated inheritance of the same parent contract. For example, `HomeBridgeErcToErc` inherits `ERC677Bridge` and `BasicBridge` even though `ERC677Bridge` already inherits `BasicBridge`.

Recommendation:
Remove duplicated and unnecessary inheritance.

Hardcoded keys

Severity: *Informational*

Status: Fixed

Description:
[Update] ThunderCore has confirmed the contract works as expected in their internal testing and has decided not to take any action at this moment. Quantstamp recommends fixing it as a best practice

The contracts rely heavily on the `EternalStorage` that is accessed using keys implemented as `keccak256(abi.encodePacked("key"))`. Hardcoding the keys may lead to unexpected typos and bugs.

Recommendation:
Use constants to prevent bugs caused by typos or erroneous copy-pastes.

Eg.

```
string public constant DEPLOYED_AT_BLOCK = "deployedAtBlock";

function deployedAtBlock() public view returns(uint256) {
    return uintStorage[keccak256(abi.encodePacked(DEPLOYED_AT_BLOCK))];
}
```

Inconsistent use of `external` vs `public`

Severity: *Informational*

Status: Fixed

Description:
[Update] ThunderCore has confirmed the contract works as expected in their internal testing and has decided not to take any action at this moment. Quantstamp recommends fixing it for code consistency

Throughout the contracts, both `external` and `public` visibility are used. For example, in `BasicBridge.sol`, L33 `setRequiredBlockConfirmations()` is `public` but `setExecutionMaxPerTx()` is `external`.

Recommendation:
Review all functions and maintain consistency around what visibility is used.

Improper Contract Bytecode Size Check

Severity: *Informational*

Contract(s) affected: `ERC677BridgeToken.sol`, `BasicBridge.sol`, `ERC677Bridge.sol`, `ForeignBridgeErcToErc.sol`, `HomeBridgeErcToErc.sol`

Description:
[Update] Currently being addressed

Checking `extcodesize > 0` does not always check whether an address is a contract because contracts that have not been fully constructed will return an `extcodesize` of 0.

Exploit Scenario:
If the negation of the `isContract` modifier is checked, a malicious user could call the function via a contract constructor and bypass the modifier.

Recommendation:
The current contracts don't check for the negation of `isContract`. If this check is required in the future, use `require(tx.origin == msg.sender)` instead.

Issues and Recommendations per contract

Severity: *Informational*

Description:

ERC677BridgeToken.sol

- L7 `ERC677Receiver` is imported but not used
- L16 For best practices, save contracts as the interface type rather than as the address type
- L18 Add indexing for this event
- L18, L36, L70, 82 `uint` -> `uint256`
- L26 This function should emit some kind of event
- L31 Best practice: modifiers should be defined before functions
- L37 `validRecipient()` is only checked for `transferAndCall()` but not for `transfer()` or `transferFrom()`
- L40 Event overload: `Transfer()` is fired twice with different arguments
- L52-55 `superTransfer()` is redundant code because `super.transfer()` exists
- L129 `transferFrom()` does not handle contract callbacks

ERC677Receiver.sol

- Should be an `interface`

IBurnableMintableERC677Token.sol

- L6 For consistency, `mint()` should also have named parameters

libraries/Message.sol

- L20, 42 Add unit tests to make sure that prepending 12 bytes from `txHash` works
- L70, 71 Comment and code mismatch
- L75 This function should return a boolean and the result should be handled in the caller
- L83 Should also validate the size of `_rs` and `_ss`
- L90 Wrap in a `require` statement instead of using `if() { revert()}`

libraries/SafeMath.sol

- Import using OpenZeppelin's npm package

upgradeability/OwnedUpgradeabilityProxy.sol

- L17 Add indexing

upgradeability/UpgradeabilityProxy.sol

- L25 Should check that `implementation` is not `0x0`

upgradeability/UpgradeabilityStorage.sol

- L17 Comment mismatch, comment specifies `string`, actual return value is `uint256`

upgradeable_contracts/BasicBridge.sol

- Clean up duplicated inheritance
- L14-17 Common practice to log both the old and new values
- L138 `this` should be casted to an `address`
- L145 `uint` -> `uint256`

upgradeable_contracts/BasicForeignBridge.sol

- L10 `SafeMath` is not used in this contract
- L12 Add indexing
- L12 `uint` -> `uint256`
- L13 Verify that anyone can call this function
- L14 `Message.hasEnoughValidSignatures()` should return a boolean and the result should be verified here

upgradeable_contracts/BasicHomeBridge.sol

- Clean up duplicated inheritance
- L12, 13, 16 Add indexing
- L29, 57, 66 Use `SafeMath` to be safe
- L37 Typo "couse"
- L56 Typo "overflew"
- L80, 108 should be consistent. use either `_hash` or `_message`
- L84, 152, 156 These should be left unimplemented to keep contract abstract

upgradeable_contracts/BridgeValidators.sol

- L26 Can be simplified as negation using `!`
- L26 Use `require()` instead of `assert()`
- L27 This can be tracked as a variable and set once after the loop iteration is finished to save gas
- L33 Use `abi.encodePacked()`
- L50 `uint` -> `uint256`
- L67 Don't update if number hasn't changed
- L91 `== true` is redundant
- L86-88 and L90-92 have the same functionality

upgradeable_contracts/ERC677Bridge.sol

- Should inherit `ERC677Receiver` as it implements `ERC677Receiver`
- L26 Should be left unimplemented to keep contract abstract

upgradeable_contracts/FeeManager.sol

- L5 `OwnedUpgradeability` is imported but unused

- L12 Common practice to log both old and new values
- L12, 34 `uint` -> `uint256`
- L20, 35 Define `10000` as a constant instead of hardcoding

upgradeable_contracts/OverdrawManagement.sol

- L11 Add indexing

upgradeable_contracts/Ownable.sol

- L16 Add indexing
- Add a constructor so that the owner is automatically set and an owner always exists

upgradeable_contracts/erc20_to_erc20/ForeignBridgeErcToErc.sol

- L13 Already defined on `BasicForeignBridge`
- L13 `uint` -> `uint256`
- L35-41 Use setter functions that already implement this with appropriate condition checks
- L38 `maxPerTx` isn't validated
- L61 Call `transfer()` directly instead of using `call()`. If the intention is to allow exceptions, use `send()`

upgradeable_contracts/erc20_to_erc20/HomeBridgeErcToErc.sol

- Clean up duplicated inheritance
- L16 Add indexing
- L41-50 Use setter functions that already implement this with appropriate condition checks
- L62 This is unnecessary as this is the default behavior since Solidity 0.4.0
- L76, 82, 83 Return the aggregated boolean result instead of just the last call. See the implementation in `ForeignBridgeErcToErc`
- L82 Use `SafeMath`

Test Results

Test Suite Results

The tests cover a decent amount of the contract functionality. A plus is seeing how much gas was consumed running each test. It would be helpful if more tests checked that incorrect values or circumstances caused the contract to fail as intended.

```
Contract: BridgeValidators
    #initialize
       ✓ sets values (307370 gas)
    #addValidator
       ✓ adds validator (97937 gas)
       ✓ cannot add already existing validator (47484 gas)
    #removeValidator
       ✓ removes validator (50026 gas)
       ✓ cannot remove if it will break requiredSignatures (51383 gas)
       ✓ cannot remove non-existent validator (50908 gas)
    #setRequiredSignatures
       ✓ sets req signatures (52361 gas)
       ✓ cannot set more than  validators count (23259 gas)
    #upgradable
       ✓ can be upgraded via upgradeToAndCall (657386 gas)

  Contract: ERC677BridgeToken
     ✓ default values
    #bridgeContract
       ✓ can set bridge contract (4899798 gas)
       ✓ only owner can set bridge contract (4923048 gas)
       ✓ fail to set invalid bridge contract address (46208 gas)
    #mint
       ✓ can mint by owner (68113 gas)
       ✓ no one can call finishMinting (21710 gas)
       ✓ cannot mint by non-owner (23564 gas)
    #transfer
       ✓ sends tokens to recipient (129728 gas)
       ✓ sends tokens to bridge contract (198084 gas)
       ✓ sends tokens to contract that does not contains onTokenTransfer method (170144 gas)
       ✓ fail to send tokens to bridge contract out of limits (258467 gas)
    #burn
       ✓ can burn (108546 gas)
    #transferAndCall
       ✓ calls contractFallback (610648 gas)
       ✓ sends tokens to bridge contract (201125 gas)
       ✓ fail to sends tokens to contract that does not contains onTokenTransfer method (171001 gas)
       ✓ fail to send tokens to bridge contract out of limits (263663 gas)
    #claimtokens
       ✓ can take send ERC20 tokens (1618949 gas)
    #transfer
       ✓ if transfer called on contract, onTokenTransfer is also invoked (483621 gas)
       ✓ if transfer called on contract, still works even if onTokenTransfer doesnot exist (1626561 gas)

Contract: ForeignBridge_ERC20_to_ERC20
    #initialize
       ✓ should initialize (5208505 gas)
    #executeSignatures
       ✓ should allow to executeSignatures (112947 gas)
       ✓ should executeSignatures with broken ERC20 (USDT) (5428819 gas)
       ✓ should allow second withdrawal with different transactionHash but same recipient and value (249327 gas)
       ✓ should not allow second withdraw (replay attack) with same transactionHash but different recipient (217563 gas)
       ✓ should not allow withdraw over home max tx limit (88656 gas)
       ✓ should not allow withdraw over daily home limit (313871 gas)
    #withdraw with 2 minimum signatures
       ✓ withdraw should fail if not enough signatures are provided (235583 gas)
       ✓ withdraw should fail if duplicate signature is provided (60115 gas)
       ✓ works with 5 validators and 3 required signatures (7221638 gas)
    #upgradeable
       ✓ can be upgraded (10943335 gas)
       ✓ can be deployed via upgradeToAndCall (3953467 gas)
    #claimTokens
```

```
      ✓ can send erc20 (7062931 gas)
    #Fee
      ✓ works with 5 validators and 3 required signatures (7306830 gas)

  Contract: HomeBridge_ERC20_to_ERC20
    #initialize
      ✓ sets variables (301762 gas)
      ✓ cant set maxPerTx > dailyLimit (61661 gas)
      ✓ can be deployed via upgradeToAndCall (729909 gas)
      ✓ cant initialize with invalid arguments (1011688 gas)
    #fallback
      ✓ reverts (21046 gas)
    #setting limits
      ✓ #setMaxPerTx allows to set only to owner and cannot be more than daily limit (76829 gas)
      ✓ #setMinPerTx allows to set only to owner and cannot be more than daily limit and should be less than maxPerTx (77830 gas)
    #executeAffirmation
      ✓ should allow validator to withdraw (194165 gas)
      ✓ test with 2 signatures required (8647327 gas)
      ✓ should not allow to double submit (193525 gas)
      ✓ should not allow non-authorities to execute deposit (30357 gas)
      ✓ doesnt allow to deposit if requiredSignatures has changed (8785008 gas)
      ✓ works with 5 validators and 3 required signatures (8975042 gas)
      ✓ should not allow execute affirmation over foreign max tx limit (98979 gas)
      ✓ should fail if txHash already set as above of limits (168095 gas)
      ✓ should not allow execute affirmation over daily foreign limit (455544 gas)
    #isAlreadyProcessed
      ✓ returns  (4855366 gas)
    #submitSignature
      ✓ allows a validator to submit a signature (275247 gas)
      ✓ when enough requiredSignatures are collected, CollectedSignatures event is emitted (531125 gas)
      ✓ works with 5 validators and 3 required signatures (9075848 gas)
      ✓ attack when increasing requiredSignatures (606057 gas)
      ✓ attack when decreasing requiredSignatures (471693 gas)
    #requiredMessageLength
      ✓ should return the required message length
    #fixAssetsAboveLimits
      ✓ Should reduce outOfLimitAmount and not emit any event (143083 gas)
      ✓ Should reduce outOfLimitAmount and emit UserRequestForSignature (144556 gas)
      ✓ Should not be allow to be called by an already fixed txHash (343748 gas)
      ✓ Should fail if txHash didnt increase out of limit amount (131061 gas)
      ✓ Should fail if not called by proxyOwner (172583 gas)
    #OwnedUpgradeability
      ✓ upgradeabilityAdmin should return the proxy owner (5299747 gas)
    #Fee
      ✓ test with 2 signatures and fee (8735746 gas)
      ✓ only owner can set fee percent (6673524 gas)

·-----------------------------------------------------------------------------------------|-----------------------------------·
|                                          Gas                                            ·  Block limit: 17592186044415 gas  |
·-------------------------------------------------|---------------------------------------|···································
|  Methods                                        ·              1 gwei/gas               ·          268.66 usd/eth           |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  Contract               ·  Method               ·  Min      ·  Max      ·  Avg      ·  # calls         ·  usd (avg)     |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  BridgeValidators       ·  addValidator         ·        -  ·        -  ·    74116  ·               2  ·          0.02  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  BridgeValidators       ·  initialize           ·   229714  ·   437814  ·   333763  ·               8  ·          0.09  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  BridgeValidators       ·  removeValidator      ·        -  ·        -  ·    26218  ·               2  ·          0.01  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  BridgeValidators       ·  setRequiredSignatures·        -  ·        -  ·    29790  ·               5  ·          0.01  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  ERC677BridgeToken      ·  burn                 ·        -  ·        -  ·    18249  ·               1  ·          0.00  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  ERC677BridgeToken      ·  claimTokens          ·        -  ·        -  ·    44119  ·               1  ·          0.01  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  ERC677BridgeToken      ·  mint                 ·    38433  ·    68433  ·    63837  ·              20  ·          0.02  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  ERC677BridgeToken      ·  setBridgeContract    ·    29432  ·    44432  ·    41426  ·              10  ·          0.01  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  ERC677BridgeToken      ·  transfer             ·    37691  ·    86589  ·    56079  ·               8  ·          0.02  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  ERC677BridgeToken      ·  transferAndCall      ·    88260  ·   166206  ·   127233  ·               2  ·          0.03  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  ERC677BridgeToken      ·  transferOwnership    ·        -  ·        -  ·    30924  ·               4  ·          0.01  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  EternalStorageProxy    ·  upgradeTo            ·    35871  ·    65871  ·    59871  ·               5  ·          0.02  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  EternalStorageProxy    ·  upgradeToAndCall     ·   278876  ·   351399  ·   313262  ·               3  ·          0.08  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  ForeignBridgeErcToErc  ·  executeSignatures    ·    82947  ·   344441  ·   161504  ·              10  ·          0.04  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  ForeignBridgeErcToErc  ·  initialize           ·   244909  ·   260037  ·   256255  ·               4  ·          0.07  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  HomeBridgeErcToErc     ·  executeAffirmation   ·    81947  ·   320753  ·   146146  ·              16  ·          0.04  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  HomeBridgeErcToErc     ·  initialize           ·   287530  ·   302658  ·   298540  ·               8  ·          0.08  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  HomeBridgeErcToErc     ·  setFeePercent        ·        -  ·        -  ·    29534  ·               1  ·          0.01  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  HomeBridgeErcToErc     ·  setMaxPerTx          ·        -  ·        -  ·    29469  ·               1  ·          0.01  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  HomeBridgeErcToErc     ·  setMinPerTx          ·        -  ·        -  ·    30036  ·               1  ·          0.01  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  HomeBridgeErcToErc     ·  submitSignature      ·   159410  ·   275247  ·   220246  ·              10  ·          0.06  |
·-------------------------|-----------------------|-----------|-----------|-----------|-----------------|-----------------·
|  Deployments                                    ·                                   ·  % of limit      ·                 |
·-------------------------------------------------|-----------|-----------|-----------|-----------------|-----------------·
|  BridgeValidators                               ·        -  ·        -  ·  1466802  ·            0 %  ·          0.39  |
·-------------------------------------------------|-----------|-----------|-----------|-----------------|-----------------·
|  ERC677BridgeToken                              ·  1463536  ·  1464560  ·  1464071  ·            0 %  ·          0.39  |
·-------------------------------------------------|-----------|-----------|-----------|-----------------|-----------------·
|  EternalStorageProxy                            ·        -  ·        -  ·   378510  ·            0 %  ·          0.10  |
·-------------------------------------------------|-----------|-----------|-----------|-----------------|-----------------·
|  ForeignBridgeErcToErc                          ·        -  ·        -  ·  3265447  ·            0 %  ·          0.88  |
·-------------------------------------------------|-----------|-----------|-----------|-----------------|-----------------·
|  HomeBridgeErcToErc                             ·        -  ·        -  ·  4855366  ·            0 %  ·          1.30  |
·-------------------------------------------------|-----------|-----------|-----------|-----------------|-----------------·

  73 passing (1m)
```

Code Coverage

Due to the tests being run via `npm test` and not working via `truffle test`, we were not able to run the `solidity-coverage` tool.

Automated Analyses

Mythril

Mythril detected primarily "External calls to fixed addresses" warnings due to the many fixed calls to specific contracts, primarily for data. However, this is a risk already accepted in tandem with the "Centralization of Power" issue denoted earlier. Mythril also detected "Integer overflow/underflow" and other such warnings, but these were deemed false positives.

MAIAN

MAIAN detected no issues.

Securify

Securify detected several issues, primarily regarding locked up ether. However, these were deemed false positives.

Adherence to Specification

Contracts adhere to the high level specs provided as part of the `README`s. However, the documentation lacks detailed implementation specifications that are necessary to verify functionality. Quantstamp requests that more detailed documentation is provided, namely those regarding expected flows and contract calls for token swaps, contract deployment steps, and contract upgrade steps.

Code Documentation

A majority of the main contracts `ForeignBridgeErcToErc.sol`, `HomeBridgeErcToErc.sol`, `BasicBridge.sol`, `BasicForeignBridge.sol`, `BasicHomeBridge.sol`, `BridgeValidators.sol` lack sufficient comments and documentation. Please include comments that provide more context and requirements.

Adherence to Best Practices

The contracts rely on the delegatecall-based proxies for future upgradability. While upgradability may be useful when an unexpected bug is found in the future, it significantly increases the complexity of the contracts and the risk of unexpected behavior. Upgradability should only be added when the developers of the project are fully aware of the consequences and trade offs.

Appendix

File Signatures

The following are the SHA-256 hashes of the audited contracts and/or test files. A smart contract or file with a different SHA-256 hash has been modified, intentionally or otherwise, after the audit. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the audit.

Contracts

```
fa85baeb1ce252d01f8ce1c57cf649cd7857b2c35b9bb05024d51c8b83e023fb
./contracts/ERC677Receiver.sol

2f6475829695129ec64f0855e82bc422d893d479d7727895d957a4c50ba0b064
./contracts/IOwnedUpgradeabilityProxy.sol

a967de62735b62169e3f6c0a49557fa869cf683b9ff6ef603133e5cce506d309
./contracts/Migrations.sol

56f826f7b8b210760432a70880dd172bde90d14a87e00eda07e5ceb8740d2b73
./contracts/IBurnableMintableERC677Token.sol

b1700e1afae3323d25d16015ac7b03ccfd0771b521af081df0e6fc5243cd4655
./contracts/ERC677.sol

247c29f20256d96bef1c107b8d3afdfc5c829a2796ab46e24985c382e2e5c2c2
./contracts/ERC677BridgeToken.sol

fc9d5956884f7593edb5e2b720e2e51baefd53a3be69ed97009905cdb0bf4e25
./contracts/IBridgeValidators.sol

8c7346e16766c735a8f6c5d0210faf5f561435e2869365df6db8f43373672e74
./contracts/upgradeability/OwnedUpgradeabilityProxy.sol

7598ccd71071596608a7f39f830c5bf67b337af3641dfdfef05b470b56627f9d
./contracts/upgradeability/EternalStorageProxy.sol

908284b2a7dfec5a424e251aa80060088ac7f46c958407c9539b41d737e1e297
./contracts/upgradeability/Proxy.sol

d998717e0f9712798a7da32cbe01e1cf99bae312ab25444f2e6a8ceef2f01f75
./contracts/upgradeability/UpgradeabilityOwnerStorage.sol

d364054ea9cb5cd08e69cf406f5ae59dcc928e82b576d6a0ef477dd5a65e8613
./contracts/upgradeability/EternalStorage.sol

2b559ad9399f3cb69316c6ecbc524f65ba8e9bf422ef967dd0ecfcef42673397
./contracts/upgradeability/UpgradeabilityProxy.sol

e80b4fff5ad2cb958efc81ee79df297315a6fa45136fd8ac5e0594c5519ed0ea
./contracts/upgradeability/UpgradeabilityStorage.sol

ae1c7539b4788dc8b6cc77843cd7f2777109de66db55d69ce3f542a5a17f3267
./contracts/libraries/SafeMath.sol

3b63d13bb1264c9a7e8dd08dccc5117302ed32f094eaa0c87f42b33365ace058
./contracts/libraries/Message.sol

c6f50e73d03961337f149310de5f8cee6de7a70871e101d0dc8582b8a882c33c
./contracts/upgradeable_contracts/BasicBridge.sol

750c58243f3b3f6102605b045510382e2d0ebe52d936687e62b85e3c9903bd30
./contracts/upgradeable_contracts/ERC677Bridge.sol

b9c22978ab757ee81073e6d3db99276432ad3cb1f2db713b04014cc73d5557d1
./contracts/upgradeable_contracts/Ownable.sol

76fb6779fdeb16befc9fb7efd68ece43443de136f971d492e0ac824276b29797
./contracts/upgradeable_contracts/FeeManager.sol

b2cdbded0dafa215a7976970666a1453b32542e37c6b963fc059c39bfc328814
./contracts/upgradeable_contracts/BridgeValidators.sol

aeb9f1d85f286a7c112d9ef49e541eff8affbb4a69674e44103c1d641f42826d
./contracts/upgradeable_contracts/Validatable.sol

0b2073534fb5f1170ab499331ba3b972f3c2fd97a1c185b21dcf421f1b27bad2
./contracts/upgradeable_contracts/BasicForeignBridge.sol

5bb7a6f0f5bd63df2921a32b726684b8c409f818f19f686a8c24ea49241e4f65
./contracts/upgradeable_contracts/OverdrawManagement.sol

77a7b11ece503729db82eba4135a1a7e4ea299ac4531137cb082daf6e133eeee
./contracts/upgradeable_contracts/BasicHomeBridge.sol

72c4a5548aaddc68b56336378f35ae09b643d8c3f59ded9459b4fb554f76bdb7
./contracts/upgradeable_contracts/OwnedUpgradeability.sol

dc1169018c79aa9dfcd9cf0d5e36d6a8a43198773d0f5f203a34b90ca6c384db
./contracts/upgradeable_contracts/erc20_to_erc20/HomeBridgeErcToErc.sol

a74ea41286d941355c21d687eec0a9ab832872a5e43d61ccadd437c6b9e9a13f
./contracts/upgradeable_contracts/erc20_to_erc20/ForeignBridgeErcToErc.sol
```

Tests

```
6f268f7373df8e44c741441c4f3086481c3e91722a5586e2caba0520fa14f8fe
./test/validators_test.js

c060af35d885072f52588e8501163489e49cc217ed1dc8f85d7def818b4afa57
./test/poa20_test.js

bfbdc03e8dfb15e833c2cb94a1ef04b497ff0b702400761c4b9b5f06ab6db6ad
./test/setup.js

787eb2ed67a669ae327e305cf63c311b5e154e3362bbbbab9a5ef777646bfa72
./test/helpers/helpers.js

65a02c5e29329dfcd577c578c5c5f3bfb0865d5ee582e16881e3d6e5e76482e4
./test/testContracts/RevertFallback.sol

f404265edf8684425bccffcfbc52f5bb009b1829cb9023f25f54aa701417cf90
./test/testContracts/TetherToken.sol

a31db1ca88f4d6c0b8de2f108daad7796e5fd92888379cb0e5940413eae4b8a3
./test/testContracts/ForeignBridgeV2.sol

d7557beb4297e1fcd86dfa2e23f3bffdefc95d831d0775a3bccb7d158f026483
./test/testContracts/ERC677ReceiverTest.sol

074e44e5615df84b760e3b486a1343ae9fcb034feb03ce1016cf86e358a2bcc3
./test/erc_to_erc/home_bridge.test.js

e600a217e72e87595040a5e7bfe0652e19e534265797e9ccf46754dd723272a3
./test/erc_to_erc/foreign_bridge.test.js
```

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure smart contracts at scale using computer-aided reasoning tools, with a mission to help boost adoption of this exponentially growing technology.

Quantstamp's team boasts decades of combined experience in formal verification, static analysis, and software verification. Collectively, our individuals have over 500 Google scholar citations and numerous published papers. In its mission to proliferate development and adoption of blockchain applications, Quantstamp is also developing a new protocol for smart contract verification to help smart contract developers and projects worldwide to perform cost-effective smart contract security audits.

To date, Quantstamp has helped to secure hundreds of millions of dollars of transaction value in smart contracts and has assisted dozens of blockchain projects globally with its white glove security auditing services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Finally, Quantstamp's dedication to research and development in the form of collaborations with leading academic institutions such as National University of Singapore and MIT (Massachusetts Institute of Technology) reflects Quantstamp's commitment to enable world-class smart contract innovation.