Incognito Ethereum Bridge Audit

coinspect

incognito

Smart Contract Audit: Incognito Ethereum Bridge

Prepared for Incognito • April 2021

v210429

1. Executive Summary

In February 2021, Incognito engaged Coinspect to perform a source code review of the smart contracts that comprise the Incognito-Ethereum bridge. The goal of the project was to evaluate the security of the smart contracts.

The assessment was conducted on the `master` branch of the public git repository at https://github.com/incognitochain/bridge-eth/tree/master/bridge/contracts as of commit `4879219669a38d601265582f815596b67775855b6` of January 6, and fixes were verified as of commit `232fef72ac3e6bf4cd8e795df11818960099172b` of March 22.

The off-chain components were out of scope for this assessment, and it is recommended that they be audited in the future.

The following issues were identified during the assessment:

High Risk	Medium Risk	Low Risk
-	3	8

The three medium-risk issues include partially arbitrary calls from the `Vault` contract controllable by users, insufficient checks when swapping committees in the `IncognitoProxy` contract, and a race condition that can be triggered when swapping committees. The other eight findings are low risk, but nevertheless we recommend fixing them because some of them could potentially have a high impact if exploited (see the impact field of each issue in section 5). As of April 8, all issues have been fixed and verified except IBE-006 that will be considered for future improvements.

It is important to mention that the contracts are upgradeable; this means that they are not truly autonomous and the Incognito organization has full power over the contracts and the deposited funds. Upgradability allows the organization to fix bugs and potentially mitigate ongoing attacks by pausing a contract, but at the same time it also poses a risk. Special care should be taken with the admin keys that allow access to this functionality. It is also recommended to consider resigning admin access in the future to make the contracts fully autonomous or contemplate options for decentralized governance.

2. Introduction

The audit started on February 8 and was conducted on the `master` branch of the public git repository at https://github.com/incognitochain/bridge-eth/tree/master/bridge/contracts as of commit `4879219669a38d601265582f815596b6775855b6` of January 6. Fixes were verified as of commit `232fef72ac3e6bf4cd8e795df11818960099172b` of March 22.

The scope of the audit was limited to the following Solidity source files, shown here with their latest sha256sum hash:

```
62cd5be9ebc888300bde5ab44129a5a65995a290edf6aca034aad283bac1d265  IERC20.sol
0f471c4a0d1d1b08aaa69de76938d9bbc218ce40323b10246d062fdf1769d9b1  incognito_proxy.sol
aadb09572636fca5346de029396d79a1c50eae49cf0e5187a7c2753d1a568bcc  pause.sol
9893b4abe402131979cc7174ce64f8285d085a869d2df992489a362ae0b1e533  proxy.sol
6641afbf26a4b8839cad3269fe3d232d202c1b75e69852a26a512eae5f630574  transparentUpgraded.sol
2a25156260c05cb03bb3c89ebc168bd2b8cf0f6f61934b8369daf9cd71f54cd6  upgradableProxy.sol
98755abc8d49c808d2d20b33ba3c43fc634ec35240f02366ba676a1d46122675  vault.sol
```

3. Assessment

The main contracts are:

- `IncognitoProxy`: stores beacon and bridge committee members of the Incognito Chain, and other contracts can query this contract to check if an instruction is confirmed on the Incognito Chain.
- `Vault`: responsible for deposits and withdrawals; it holds assets (Ether or ERC20 tokens) and emits events that the Incognito Chain interprets as minting instructions; when presented with a burn proof created over at the Incognito Chain, it releases the assets back to the user.

All contracts are specified to be compiled with the Solidity version 0.6.6, except `IncognitoProxy` and `AdminPausable` that are compiled with Solidity 0.5.12. It is recommended to update to a newer Solidity version. The latest version of the 0.6.x series is 6.6.12, and 0.5.17 for the 0.5.x series. Both contain numerous bug fixes and improvements. (See IBE-001.)

The contracts compile without warnings. However, linting with `solhint` produces many warnings and errors, most of them about aesthetic issues. Nevertheless, it is advisable to review the output of `solhint` and improve the source code. Also, linting should be incorporated into the development process.

The repository includes tests. It is recommended to make coverage reports (for example, with the `solidity-coverage` tool) and make sure to add the necessary tests to achieve full coverage and verify that during development, new tests are added as needed to always ensure full coverage.

The reviewed Solidity code is well written and clear. Some comments, however, have become outdated and should be corrected. For example, some comments say that some functions require the `Vault` contract to be unpaused, but actually the new version of the `Vault` contract is no longer pausable (pausing functionality was moved to the transparent proxy instead). Also, the event `UpdateIncognitoProxy` is never emitted by the `Vault` contract, and if it is not needed, it is recommended that it be removed from the source code.

The contracts contain multiple uses of `require` without an error string. This is problematic for several reasons. For example, it makes it impossible to write unit tests that check certain failure conditions. (It is possible for a test to verify that a determined function call produces a revert, but it is not possible to know for sure that the revert happened due to the condition being tested or for some other reason). It is advisable to always include an error string in every `require` and always write tests that verify that the call is reverted in each case that it should be reverted due to failing requirements. (See IBE-002.)

Vault

The function `depositERC20` in the `Vault` contract is *`payable`*. This is unnecessary and creates the risk that someone will call this function with ether and the funds will be lost (see IBE-003). A similar problem happens with the `receive` function that, according to the comment, is intended to be called only from the previous vault. If so, it would be safer to restrict it to accept calls only from `prevVault`, but this is not currently possible because it also receives calls from other contracts called through the `execute` function.

The function `signToAddress` returns `address(0)` when the signature is invalid. Because of this, the internal function `verifySigData` can return `address(0)` and then, if there were any funds wrongly assigned to `address(0)`, an attacker could call function `requestWithdraw` with an invalid signature and withdraw the funds. Function `verifySigData` should check the value returned by `ecrecover` and revert if it is `address(0)`, and `BurnInstData` should never be accepted if the field *to* is `address(0)`. (See IBE-004.)

The functions `withdraw` and `submitBurnProof` contain integer overflows when multiplying the token amount by a power of 10 to convert the number of decimals. These are not easily triggerable by an attacker since the data is signed by members of the committees and has to pass checks done off-chain. However, it is advisable to fix it to avoid any potential problems. (See IBE-005.)

The function `execute` allows anyone to make calls with the `Vault` as `msg.sender`. There are some restrictions regarding the return values, so calls are not totally arbitrary; this is a dangerous practice. It is recommended to restrict the calls as much as possible — for example, by adding a whitelist of explicitly allowed destination addresses and function signatures. Although the destination address and calldata are signed by the user and this mitigates the risk of a third party attacking the user, there is still risk of a user attacking the vault. For example, this mechanism allows a user to make the vault call an ERC20 contract. With the current restrictions on the return values, it does not seem possible that a user could perform this kind of attack to steal funds, at least with standard ERC20 contracts, but since the system accepts arbitrary token contracts, this possibility cannot be ruled out. (See IBE-006.)

Also, in function `execute`, the caller is free to choose parameters `token` and `recipientToken` since they are not covered by the signature. This could potentially allow an attacker front-running the user transaction to steal funds, depending on the implementation of the particular contracts that are called. It is recommended to include both the `token` and `recipientToken` parameters in the data that is required to be signed. (See IBE-007.)

Furthermore, there is ambiguity between signed data for functions `execute` and `requestWithdraw`, and a signature can be valid for both functions, potentially allowing attackers to use a signature intended for one of the functions in a call to the other one. It is recommended to avoid clashing by adding a header or a value indicating “type” (`requestWithdraw` or `execute`) to the data that is hashed and signed. (See IBE-008.)

Note also that the function `execute` can increase asset balances and does not check limits as do the `deposit` and `depositERC20` functions (`10**27` weis in the case of ether, and `10**18` including up to 9 decimals in the case of ERC20 tokens). The relevance of this issue depends on the implementation details of off-chain and Incognito Chain components.

IncognitoProxy

In the `IncognitoProxy` contract, when swapping bridge or beacon committees, the number of members (variable `numVals`) should be required to be > 0. Otherwise, the new committee would be empty, meaning it will not be possible to change it any more and the funds in the vault will be locked. However, it would be possible to recover from this problem as long as there is an admin set for the vault proxy that would be able to set a new `IncognitoProxy`. (See IBE-009.)

In the function `extractCommitteeFromInstruction`, it should be required for each member address to be different from 0. Otherwise, any invalid signature would look like a valid signature from a legitimate member of the committee. It should also be checked that there are no duplicate addresses or the repeated one would have more voting power than the others. All this can also be enforced off-chain. (See also IBE-009.)

Also, if the new committee `startBlock` is older than the current block of the Incognito Chain, some instructions already processed could become invalid after swapping a committee, which could produce inconsistencies in the protocol. For example, if the latest committee has `startBlock` 10, and the `IncognitoProxy` checks an instruction with height 20 signed by the committee it will accept it, but just before checking the instruction the committee is swapped with a new committee with `startBlock` 15, the instruction could be rejected. This is a race condition between committee swapping in the `IncognitoProxy` and users calling `withdraw` and `submitBurnProof` in the `Vault`. (See IBE-010.)

Upgrades

The vault contract currently deployed in `mainnet` is upgradeable. It implements an upgrading mechanism that involves first pausing the contract (which effectively prevents any user operations including deposits, withdrawals, and calls to `execute`), then calling the function `migrate` with the address of the newly deployed vault (a transparent proxy), and then calling (one or more times) the function `moveAssets` with the list of assets to be transferred to the new vault. The function `moveAssets` transfers ERC20 tokens or ether to the new vault and then calls the function `updateAssets` in the new vault to update the mapping `totalDepositedToSCAmount`. Only the *admin* address can call these functions to perform the upgrade.

It is worth noting that the function `moveAssets` in the `mainnet` vault assumes that the new vault does not have any ether or any token and transfers all assets without checking the final balance has not increased beyond maximum values (`10**27` weis in the case of ether, and `10**18` including up to 9 decimals in the case of ERC20 tokens).

It is very important that the old vault remains paused forever after upgrading it to a new version. No new deposits or any other user operations should be accepted by the old vault once it has been upgraded.

The new version of the `Vault` contract implements upgradeability using transparent proxies, following OpenZeppeling's scheme for upgradeable contracts.

The main differences between the `mainnet` vault and the new version of the `Vault` contract is that the new version is not pausable and does not have an admin and a setter function for the `IncognitoProxy` (this functionality has been moved from the logic contract to the transparent proxy itself).

Another subtle difference is that the new version of the `Vault` contract checks that the length of the burn instruction is `>= 130` before calling the function `parseBurnInst`. However, it is recommended to move this check to the `parseBurnInst` function itself.
The proxy contract is implemented in `TransparentUpgradeableProxy` that inherits from `UpgradeableProxy` that in turn inherits from `Proxy`. The functionality of the `AdminPausable` contract has been implemented in the `TransparentUpgradeableProxy`. It is worth mentioning that the new pausing functionality behaves differently from the original `AdminPausable` because pausing the proxy now causes *all* function calls to revert (instead of specific functions as before). Also, the function `claim` has been implemented in such a way that it is not transparent. (See IBE-011.)

Finally, it is recommended to test the upgrade on a fork of mainnet before performing the actual upgrade. This is easy to do using `’ganache-cli --fork’` or with hardhat’s forking feature.

4. Summary of Findings

ID	Description	Risk	Fixed
IBE-001	Outdated Solidity version	Low	✔
IBE-002	Uses of require without error string	Low	✔
IBE-003	Unnecessary payable function	Low	✔
IBE-004	Invalid signatures accepted as signed by address(0)	Low	✔
IBE-005	Integer overflows in functions withdraw and submitBurnProof	Low	✔
IBE-006	Partially arbitrary external calls controllable by users	Medium	✘
IBE-007	Free parameters in function execute controllable by front-runners	Low	✔
IBE-008	Ambiguous signatures in requestWithdraw and execute	Low	✔
IBE-009	Insufficient checks when swapping committees	Medium	✔
IBE-010	Inconsistencies with committee swapping lead to race condition	Medium	✔
IBE-011	Function claim in TransparentUpgradableProxy is not transparent	Low	✔

5. Findings

IBE-001	Outdated Solidity version	
Total Risk **Low**	Impact Low	Location *.sol
Fixed ✓	Likelihood -	

Description

Currently, the contract code specifies with the `pragma` statement that it is meant to be built with a version of the Solidity compiler older than the latest production release. Newer versions have added additional warnings that can help to detect problems, solve bugs, and enforce new rules to enhance security.

All contracts are specified to be compiled with Solidity version 0.6.6 except IncognitoProxy that is compiled with Solidity 0.5.12.

Recommendation

It is recommended to update to a newer Solidity version. The latest version of the 0.6.x series is 6.6.12 and 0.5.17 for the 0.5.x series, and both contain numerous bug fixes and improvements.

Status

Fixed in commit `2596f1e19c14ccd6fae12ceb77c65c0e331d3199`, now using version 0.6.12.

IBE-002 Uses of require without error string

Total Risk	Impact	Location
Low	Low	incognitoProxy.sol, vault.sol, upgradableProxy.sol, transparentUpgraded.sol
Fixed	Likelihood	
✓	-	

Description

The contracts contain multiple uses of `require` without an error string. This is problematic for several reasons. For example, it makes it impossible to write unit tests that check certain failure conditions. In other words, it is possible for a test to verify that a determined function call produces a revert, but it is not possible to know whether the revert happened due to the condition being tested or for some other reason.

Recommendation

It is advisable to always include an error string in every `require` and always write tests that verify that the call is reverted in each case that it should be reverted due to the corresponding failing requirements.

Status

Fixed in commit `2596f1e19c14ccd6fae12ceb77c65c0e331d3199`.

IBE-003 Unnecessary payable function

Total Risk Low	Impact High	Location vault.sol
Fixed ✓	Likelihood Low	

Description

The function `depositERC20` in the `Vault` contract is *`payable`*:

```
function depositERC20(address token, uint amount, string calldata incognitoAddress)
external payable nonReentrant {
[...]
```

This is unnecessary and creates the risk that someone will call this function with ether and the funds will be lost.

Recommendation

Remove the `payable` modifier from the `depositERC20` function.

Status

The `payable` modifier was removed from the `depositERC20` function in commit `2596f1e19c14ccd6fae12ceb77c65c0e331d3199`.

IBE-004 Invalid signatures accepted as signed by address(0)

Total Risk	Impact	Location
Low	Low	vault.sol
Fixed ✓	Likelihood Low	

Description

The function `signToAddress` returns `address(0)` when the signature is invalid. Because of this, the internal function `verifySigData` can return `address(0)` and then, if there were any funds wrongly assigned to `address(0)`, an attacker could call function `requestWithdraw` with an invalid signature and withdraw the funds.

Recommendation

Function `verifySigData` should check the value returned by `ecrecover` and revert if it is `address(0)`.

Also, consider rejecting `BurnInstData` if the field `to` is `address(0)`. For example, change the function `parseBurnInst` to revert if `to` is `0`. Transfers to the zero address are not defined in the ERC20 standard, and ERC20 implementations are inconsistent in regard to this (some implementations revert, but not all of them).

Status

Commit `2596f1e19c14ccd6fae12ceb77c65c0e331d3199` fixed function `verifySigData`.

IBE-005 Integer overflows in functions withdraw and submitBurnProof

Total Risk	Impact	Location
Low	High	vault.sol
Fixed ✓	Likelihood Low	

Description

Functions `withdraw` and `submitBurnProof` contain integer overflows:

```
uint8 decimals = getDecimals(data.token);
if (decimals > 9) {
    data.amount = data.amount * (10 ** (uint(decimals) - 9));
}
```

If `data.amount` is very big, this multiplication would overflow, producing a result that is less than expected, potentially leading to inconsistencies and loss of funds.

However, these integer overflows are not easily triggerable by an attacker since the data is signed by members of the committees and has to pass checks done off-chain.

Recommendation

It is advisable to fix these integer overflows to avoid any potential problems and not only rely on off-chain checks. Use `safeMul` for the multiplication instead of *.

Status

Fixed in commit `2596f1e19c14ccd6fae12ceb77c65c0e331d3199`.

IBE-006 Partially arbitrary external calls controllable by users

Total Risk **Medium**	Impact High	Location vault.sol
Fixed ✘	Likelihood Low	

Description

The function `execute()` allows anyone to make calls with the `Vault` as `msg.sender`. There are some restrictions regarding the return values, so calls are not totally arbitrary; this is a dangerous practice. It is recommended to restrict the calls as much as possible. For example, add a whitelist of explicitly allowed destination addresses and function signatures. Although the destination address and calldata are signed by the user and this mitigates the risk of a third party attacking the user, there is still risk of a user attacking the vault. For example, this mechanism allows a user to make the vault call an ERC20 contract. With the current restrictions on the return values, it does not seem possible that a user could perform this kind of attack to steal funds, at least with standard ERC20 contracts, but since the system accepts arbitrary token contracts, this possibility cannot be ruled out.

Recommendation

It is advisable to restrict the contracts and functions that can be called through the `execute` function. One possibility is to add whitelists of allowed destination addresses and function signatures. It could also be possible to redesign the `execute` mechanism to make the external call from another contract, a “call forwarder” different from the vault that does not hold all the user’s funds.

Status

It is the intention of the Incognito team to allow arbitrary external calls so that every smart contract developer can integrate his/her dApps with the Incognito platform.

Whitelisting as suggested by Coinspect will be taken into account for future improvements.

IBE-007 Free parameters in function execute controllable by front-runners

Total Risk	Impact	Location
Low	High	vault.sol
Fixed ✓	Likelihood Low	

Description

In function `execute`, the caller is free to choose parameters `token` and `recipientToken` since they are not covered by the signature:

```
function execute(
    address token,
    uint amount,
    address recipientToken,
    address exchangeAddress,
    bytes calldata callData,
    bytes calldata timestamp,
    bytes calldata signData
) external payable nonReentrant {
    //verify ower signs data from input
        address  verifier  =  verifySignData(abi.encode(exchangeAddress,  callData,
timestamp, amount), signData);
[...]
```

This could potentially allow an attacker front-running the user transaction to steal funds, depending on the implementation of the particular contracts that are called.

Recommendation

It is recommended to include both the `token` and `recipientToken` parameters in the data signed by the user.

Status

Fixed in commit `2596f1e19c14ccd6fae12ceb77c65c0e331d3199`.

IBE-008 Ambiguous signatures in requestWithdraw and execute

Total Risk	Impact	Location
Low	High	vault.sol
Fixed ✓	Likelihood Low	

Description

There is ambiguity between signed data for functions `execute` and `requestWithdraw`. A signature can be valid for both functions, potentially allowing attackers to use a signature intended for one of the functions in a call to the other one.

Recommendation

An actual attack seems unlikely because it would be hard to avoid a revert due to failing requirements. However, the fix is worthwhile because it is very simple and would rule out any possibility of exploitation.

It is recommended to avoid clashing by adding a header or a value indicating "type" (`requestWithdraw` or `execute`) to the data that is hashed and signed.

Status

Fixed in commit `2596f1e19c14ccd6fae12ceb77c65c0e331d3199`.

IBE-009 Insufficient checks when swapping committees

Total Risk	Impact	Location
Medium	High	incognito_proxy.sol
Fixed ✓	Likelihood Low	

Description

In the `IncognitoProxy` contract, when swapping bridge or beacon committees with functions `swapBridgeCommittee` or `swapBeaconCommittee`, the number of members (variable `numVals`) is not required to be > 0. If an empty committee is set, it becomes impossible to change the committee any more because it would require at least one member signature. Also, for the same reason, any burn instruction would be rejected, and the funds in the vault would be locked.

However, it is possible to recover from this situation as long as there is an admin set for the vault proxy because the admin would be able to set a new `IncognitoProxy`.

Also, the function `extractCommitteeFromInstruction` does not check that none of the members of the new committee is `address(0)`. If a member is `address(0)`, a signature from that member is trivially forgeable because an invalid signature would look as if signed by `address(0)`. This is also possible because the function `verifySig` does not check the return value of `ecrecover` is not `address(0).`

Furthermore, there are no checks for duplicated members. If a member appeared in a committee more than once, it would have more "voting power" than the other members and would invalidate the requirement of a valid signature of at least two-thirds of the members.

Recommendation

Although all these checks could be done off-chain, it is also recommended to include the checks in the IncognitoProxy contract and not rely solely on off-chain checks.

Status

Commit `2596f1e19c14ccd6fae12ceb77c65c0e331d3199` and commit `a4f86cea21eb7a52e42ba0cbc1260284eca2d87c` add a requirement for new committees to be nonempty and fixes function `verifySig` to reject `address(0)` as a valid signer.

The contract still accepts `address(0)` as a committee member, although now that `verifySig` rejects `address(0)`, this cannot be used to forge instructions. Committees with duplicated members are also still accepted. The responsibility to avoid pathological committee swaps lies on the current committee.

IBE-010 Inconsistencies with committee swapping lead to race condition

Total Risk	Impact	Location
Medium	High	incognitoProxy.sol
Fixed ✓	Likelihood: Low	

Description

When swapping bridge or beacon committees in the `IncognitoProxy` contract, if the new committee `startBlock` is older than the current block of the Incognito Chain, some instructions already processed could become invalid, which could produce inconsistencies in the protocol.

For example, if the latest committee has `startBlock` 10 and the `IncognitoProxy` checks an instruction with height 20 signed by the committee, it will accept it. But if just before checking the instruction the committee is swapped with a new committee with `startBlock` 15, the instruction could be rejected. This produces a race condition between committee swapping in the `IncognitoProxy` and users calling `withdraw` and `submitBurnProof` in the `Vault`.

Recommendation

It is recommended to make changes in committee swapping to avoid race conditions if possible or provide mitigations in case this race condition is triggered. When swapping committees, the new `startHeight` should be higher than the block height of any instruction already signed by the current committee. Otherwise, an approved instruction could become invalid.

Status

Commit `f921e4c099f090363735c13bc7cdae3a4546bf1f` introduces changes that force committee swap instructions to be fed into the contract in block height order. It is still important for new committees to have a `startHeight` higher than the block height of any instruction already signed by the current committee so far, and this can be enforced off-chain.

IBE-011 Function claim in TransparentUpgradableProxy is not transparent

Total Risk Low	Impact Low	Location transparentUpgaded.sol
Fixed ✓	Likelihood Low	

Description

The `TransparentUpgradableProxy` contract implements the *transparent proxy* pattern. All external functions are supposed to be visible only to the admin, and a call from any other address is delegated to the logic contract (`Vault` in this case).

The function claim breaks this pattern because it is visible to any address, and if the logic contract implemented this function, it would not be visible in the proxy contract:

```
function claim() external {
    require(msg.sender == _successor(), "TransparentUpgradeableProxy: unauthorized");
    emit Claim(_successor());
    _setAdmin(_successor());
}
```

Recommendation

It is recommended to make this function transparent by making it visible only to the *successor*:

```
function claim() external {
    if(msg.sender == _successor()) {
        emit Claim(_successor());
        _setAdmin(_successor());
    } else {
        _fallback();
    }
}
```

Status

Fixed in commit `2596f1e19c14ccd6fae12ceb77c65c0e331d3199`.

6. Disclaimer

The information presented in this document is provided “as is” and without warranty. Source code reviews are a “point in time” analysis, and as such, it is possible that something in the code could have changed since the tasks reflected in this report were executed. This report should not be considered a perfect representation of the risks threatening the analyzed system.