PUBLIC

Code Assessment

of the Xgov Smart Contracts

September 17, 2025

Produced for

Curve

by

CHAINSECURITY

Contents

1 Executive Summary

Dear Curve team,

Thank you for trusting us to help Curve with this security audit. Our executive summary provides an overview of subjects covered in our audit of the latest reviewed contracts of Xgov according to Scope to support you in forming an opinion on their security risks.

Curve implements xGov, a system that extends the capabilities of the Curve DAO, allowing it to interact with contracts on different networks.

The most critical subjects covered in our audit are the Merkle Patricia Proof verifier correctness, access control, and cross-chain message decoding. Security regarding all the aforementioned subjects is high.

Other general subjects covered are denial of service, gas optimization, and RLP decoding. Security regarding all the aforementioned subjects is high.

In summary, we find that the codebase provides a high level of security.

It is important to note that security audits are time-boxed and cannot uncover all vulnerabilities. They complement but don't replace other vital measures to secure a project.

The following sections will give an overview of the system, our methodology, the issues uncovered, and how they have been addressed. We are happy to receive questions and feedback to improve our service.

Sincerely yours,

ChainSecurity

1.1 Overview of the Findings

Below we provide a brief numerical overview of the findings and how they have been addressed.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

2 Assessment Overview

In this section, we briefly describe the overall structure and scope of the engagement, including the code commit which is referenced throughout this report.

2.1 Scope

The assessment was performed on the source code files inside the Xgov repository based on the documentation files. The table below indicates the code versions relevant to this report and when they were received.

curve-xgov

V	Date	Commit Hash	Note
1	28 April 2025	f2abf673c949d4852a382925856ebd7e07114384	Initial Version
2	18 August 2025	f2547d36711ca1159db89465f4cd636ae565fab3	Final Version

storage-proofs

V	Date	Commit Hash	Note
1	29 July 2025	aaa88a56bf0b778b3957f253daf35bf3e7324b24	Initial Version
2	15 September 2025	3831873fa248d8269e4bad51d860f2ab78e296c0	Final Version

For the solidity smart contracts, the compiler version `0.8.18` was chosen. For the vyper smart contracts, the compiler version `0.3.10` was chosen.

The following contracts were included in the scope of the assessment:

xgov:

```
contracts/xyz/XYZBroadcaster.vy
contracts/xyz/XYZRelayer.vy
```

storage-proofs:

```
contracts/xgov/verifiers/MessageDigestVerifier.sol
contracts/xdao/contracts/libs/MerklePatriciaProofVerifier.sol
contracts/xdao/contracts/libs/StateProofVerifier.sol
```

2.1.1 Excluded from scope

Anything not listed in scope of the assessment is considered out of scope. This includes tests, scripts and external libraries such as `Solidity-RLP@2.0.7`.

2.2 System Overview

This system overview describes the initially received version (Version 1) of the contracts as defined in the Assessment Overview.

Furthermore, in the findings section, we have added a version icon to each of the findings to increase the readability of the report.

Curve implements xGov, a system that extends the capabilities of the Curve DAO allowing it to interact with contracts on different networks.

The system is designed to forward messages from Ethereum Mainnet to other networks.

2.2.1 Ethereum Contracts

On Ethereum, the `XYZBroadcaster` contract is responsible for broadcasting messages to other networks. It has three privileged roles (agents) that must be given to one address each, where each address can only hold one role at a time:

- **OWNERSHIP**
- **PARAMETERS**
- **EMERGENCY**

When calling `broadcast`, the privileged role holder must provide:

- The destination chain
- A list of messages to be sent
- A time to live (TTL) for the messages

The contract writes the digest (hash) of the messages and their deadline (based on the TTL) to state, indexed by the agent that sent it, the chain ID, and the current nonce:

```
nonce: public(HashMap[Agent, HashMap[uint256, uint256]])
# agent -> chainId -> nonce

digest: public(HashMap[Agent, HashMap[uint256, HashMap[uint256, bytes32]]])
# agent -> chainId -> nonce -> messageDigest

deadline: public(HashMap[Agent, HashMap[uint256, HashMap[uint256, uint256]]])
# agent -> chainId -> nonce -> deadline
```

Restricted Functions:

- `broadcast`: Can be called by the privileged roles to send messages to other networks.
- `commit_admins`: Allow the **OWNERSHIP** role to commit a new set of privileged roles.
- `apply_admins`: Allow the **OWNERSHIP** role to apply the committed set of privileged roles.

2.2.2 Other Chain Contracts

For the security of the message forwarding system, a trusted block-hash oracle is used to obtain block hashes from Ethereum Mainnet. Given such a block-hash and Merkle Patricia proofs, it is possible to permissionlessly provide messages to the `MessageDigestVerifier` contract on the destination chain and verify that the messages were indeed sent by the `XYZBroadcaster` contract on Ethereum Mainnet.

2.2.2.1 MessageDigestVerifier

The `MessageDigestVerifier` contract is responsible for verifying provided messages against the state of Ethereum Mainnet's `XYZBroadcaster` contract using the blockhash oracle. The contract is permissionless and implements two entry points:

- `verifyMessagesByBlockHash`
- `verifyMessagesByStateRoot`

Both functions are very similar, while the first one allows for providing an RLP encoded block header to be validated against the block-hash oracle, the second one allows for providing a block number to directly query the block-hash oracle for the corresponding state root.

In both cases, the caller must provide:

- The Agent that sent the message (1, 2, or 4 for OWNERSHIP, PARAMETERS, and EMERGENCY respectively)
- The list of messages to be verified
- Three RLP-encoded Merkle Patricia proofs:
    1. The proof for the account of the `XYZBroadcaster` contract in the state root
    2. The proof for the message digest slot in the account's storage root
    3. The proof for the deadline slot in the account's storage root

Once all proofs are verified, and if the deadline has not passed, the messages are forwarded to the `XYZRelayer` contract.

2.2.2.2 XYZRelayer

The `XYZRelayer` contract is responsible for relaying the messages to the right contracts for processing.

Using `relay()`, the messenger (here the `MessageDigestVerifier` contract) can forward the verified messages to the appropriate destination contracts. The function will match the message to the destination contract based on the provided agent.

Restricted Functions:

- `relay`: Can be called by the set `messenger` to forward messages to the destination contracts.
- `set_messenger`: Can be called by the **OWNERSHIP** role to set the messenger address.

2.2.3 Changelog

In Version 2 of the system, only fixes to previously reported issues were applied. No new features were added.

2.3 Trust Model

The xGov system operates with a multi-tier trust model involving privileged roles, external dependencies, and permissionless components. Below is a comprehensive analysis of the trust assumptions:

2.3.1 General Trust Assumptions

- The `XYZBroadcaster` contract is deployed on Ethereum Mainnet at address `0x7BA33456EC00812C6B6BB6C1C3dfF579c34CC2cc`.
- All other contracts are deployed on other networks.

2.3.2 Privileged Roles

OWNERSHIP Agent

- **Trust Level**: Fully trusted
- **Capabilities**:
    - Broadcast messages to all destination chains

    - Commit and apply new admin sets via `commit_admins()` and `apply_admins()` in `XYZBroadcaster`
    - Modify the messenger address in relayer contracts via `set_messenger()`
- **Risk Profile**: Complete control over the system. Can:
    - Broadcast potentially malicious messages
    - Replace all privileged roles (including themselves)
    - Disable message forwarding by setting invalid messenger addresses

PARAMETER Agent

- **Trust Level**: Fully trusted
- **Capabilities**:
    - Broadcast parameter update messages
- **Risk Profile**: Can execute arbitrary parameter changes on destination contracts but cannot modify system roles or infrastructure

EMERGENCY Agent

- **Trust Level**: Fully trusted
- **Capabilities**:
    - Broadcast emergency messages
- **Risk Profile**: Can execute arbitrary emergency changes on destination contracts but cannot modify system roles or infrastructure

2.3.3 External Dependencies

Block Hash Oracle (`IBlockHashOracle`)

- **Trust Level**: Fully trusted
- **Critical Functions**:
    - `get_block_hash(uint256 _number)` - Returns Ethereum mainnet block hashes
    - `get_state_root(uint256 _number)` - Returns state root for given block number
- **Trust Assumptions**:
    - Oracle provides accurate, uncensorable access to Ethereum mainnet state
    - Oracle remains operational and accessible
    - Oracle data is not manipulated or delayed beyond acceptable bounds
- **Risk Profile**: Complete compromise of message verification system if oracle is compromised. Malicious oracle could:
    - Allow verification of fraudulent messages
    - Prevent legitimate message verification

2.3.4 Contract Upgradeability

All contracts in the system are immutable once deployed.

3 Limitations and use of report

Security assessments cannot uncover all existing vulnerabilities; even an assessment in which no vulnerabilities are found is not a guarantee of a secure system. However, code assessments enable the discovery of vulnerabilities that were overlooked during development and areas where additional security measures are necessary. In most cases, applications are either fully protected against a certain type of attack, or they are completely unprotected against it. Some of the issues may affect the entire application, while some lack protection only in certain areas. This is why we carry out a source code assessment aimed at determining all locations that need to be fixed. Within the customer-determined time frame, ChainSecurity has performed an assessment in order to discover as many vulnerabilities as possible.

The focus of our assessment was limited to the code parts defined in the engagement letter. We assessed whether the project follows the provided specifications. These assessments are based on the provided threat model and trust assumptions. We draw attention to the fact that due to inherent limitations in any software development process and software product, an inherent risk exists that even major failures or malfunctions can remain undetected. Further uncertainties exist in any software product or application used during the development, which itself cannot be free from any error or failures. These preconditions can have an impact on the system's code and/or functions and/or operation. We did not assess the underlying third-party infrastructure which adds further inherent risks as we rely on the correct execution of the included third-party technology stack itself. Report readers should also take into account that over the life cycle of any software, changes to the product itself or to the environment in which it is operated can have an impact leading to operational behaviors other than those initially determined in the business specification.

4 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- *Likelihood* represents the likelihood of a finding to be triggered or exploited in practice
- *Impact* specifies the technical and business-related consequences of a finding
- *Severity* is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.

Likelihood	Impact		
	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

5 Open Findings

In this section, we describe any open findings. Findings that have been resolved have been moved to the Resolved Findings section. The findings are split into these different categories:

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0

6 Resolved Findings

Here, we list findings that have been resolved during the course of the engagement. Their categories are explained in the Open Findings section.

Below we provide a numerical overview of the identified findings, split up by their severity.

Critical-Severity Findings	0
High-Severity Findings	0
Medium-Severity Findings	0
Low-Severity Findings	0
Informational Findings	7

- Error Messages Code Corrected
- Magic Value Code Corrected
- Misleading Comments Code Corrected
- Missing Events Code Corrected
- Missing Future Admin Check Code Corrected
- Missing NatSpec Comments Code Corrected
- Unused Variable Code Corrected

6.1 Error Messages

Informational Version 1 Code Corrected

CS-CURVE-XGOV-002

The `MessageDigestVerifier`, `StateProofVerifier` and `MerklePatriciaProofVerifier` often do not provide specific error when verification fails. This can make debugging and identifying issues more difficult for developers. Implementing more descriptive error could improve the developer experience and facilitate easier troubleshooting.

Similarly, the `StateProofVerifier` provide an error string in `verifyBlockHeader`. Instead of using error strings, custom errors could be used to reduce deployment and runtime cost.

Code corrected:

Descriptive error messages were added through the codebase. In `MerklePatriciaProofVerifier` and `StateProofVerifier`, custom errors were introduced to replace error strings.

6.2 Magic Value

Informational Version 1 Code Corrected

CS-CURVE-XGOV-004

In `MerklePatriciaProofVerifier`, the magic value `0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421` could be replaced by a constant defined to explicitly represent that this value is the Keccak256 hash of `0x80`.

Code corrected:

The magic value was not replaced by a constant, but a comment was added to explain its meaning: `// keccak256(0x80)`.

6.3 Misleading Comments

Informational **Version 1** **Code Corrected**

CS-CURVE-XGOV-006

The following comments may be misleading or inaccurate:

- In `MessageDigestVerifier`, several NatSpec comment mentions gauges and gauge types, even though the system is expected to be used in a broader context.
- In `MerklePatriciaProofVerifier`, the following comments are misleading, as in the first case, `node[1]` is expected to contain `rlp(Keccak256(rlp(child)))`:

```
if (!node[1].isList()) {
    // rlp(child) was at least 32 bytes. node[1] contains
    // Keccak256(rlp(child)).
    nodeHashHash = node[1].payloadKeccak256();
} else {
    // rlp(child) was less than 32 bytes. node[1] contains
    // rlp(child).
    nodeHashHash = node[1].rlpBytesKeccak256();
}
```

Code corrected:

Both comments were updated to be more accurate.

6.4 Missing Events

Informational **Version 1** **Code Corrected**

CS-CURVE-XGOV-007

- In `XYZRelayer`, no events are emitted by the function `relay()`.
- In the `MessageDigestVerifier`, no events are emitted.

Code corrected:

Event emissions were added to `relay()` in `XYZRelayer` and to `_verifyMessages()` in `MessageDigestVerifier`.

6.5 Missing Future Admin Check

Informational | Version 1 | Code Corrected

CS-CURVE-XGOV-008

In the `XYZBroadcaster`, the current admin can give ownership to a new admin by calling first `commit_admins()`, which sets the storage variable `future_admins` and then `apply_admins()`, which reads it and apply the changes. Since there are no check in `apply_admins()` to verify if `future_admins` was written to, calling `apply_admins()` without a prior call to `commit_admins()` will result in wiping the current `admins` structure and make the contract unusable.

Code corrected:

A check was added in `apply_admins()` to ensure that `future_admins` is not empty before applying the changes.

6.6 Missing NatSpec Comments

Informational | Version 1 | Code Corrected

CS-CURVE-XGOV-009

Across the codebase, several functions are missing NatSpec comments:

XYZBroadcaster:

- `__init__()`
- `commit_admins()`

XYZRelayer:

- `__init__()`
- `set_messenger()` (param `_messenger`)

Code corrected:

All missing NatSpec comments were added to the functions listed above.

6.7 Unused Variable

Informational | Version 1 | Code Corrected

CS-CURVE-XGOV-012

In `MerklePatriciaProofVerifier`, the variable `value` is defined but never used. As this led in the past to confusion in this library, removing it and explicitly reverting if the loop terminates would be beneficial even if this should be an unreachable state provided a valid `rootHash`.

Code corrected:

The unused variable was removed and an explicit revert was added at the end of the function to indicate that this point should be unreachable.

7 Informational

We utilize this section to point out informational findings that are less severe than issues. These informational issues allow us to point out more theoretical findings. Their explanation hopefully improves the overall understanding of the project's security. Furthermore, we point out findings which are unrelated to security.

7.1 Canonicality & Type Checks

Informational **Version 1** **Acknowledged**

CS-CURVE-XGOV-001

In `MerklePatriciaProofVerifier`, across the verification flow we always assume a trusted `state` or `storage` root. Under that threat model, having the system accepting some non-canonical RLP/MPT encodings or loose types in the proof does not enable forgery, since every node is always hash-linked to the trusted root. However, adding strict canonicality & type checks would make verification fail faster on malformed inputs and improve overall robustness.

The following non-exhaustive list of checks show potential areas for improvement:

In `MerklePatriciaProofVerifier.extractProofValue`:

- Enforce child pointer canonicality (branch/extension child)
    - Inline child must be a non-empty list and `len(RLP(child)) < 32 bytes`.
    - Hashed child must be the RLP encoding of a 32-byte short string.
- Enforce hex-prefix compact path canonicality:
    - For even paths in leaf and extension nodes, the low half-nibble of first byte must be 0.
    - The extension path must be non-empty.
    - The parity of the decoded nibbles must match flag;
- Reject node leafs that are not length 2 or 17.
- Enforce proof structure:
    - Revert at function end if inconclusive (no silent empty return).
- Enforce `path.length == 32` if the library should be specific for Ethereum proofs and not generic
- List header sanity in `RLPItem.toList()`:
    - For short list: `item.len == 1 + declaredLen`.
    - For long list: `item.len == 1 + lenOfLen + declaredLen`.

In `StateProofVerifier`:

- State root / storage root / code hash type: enforce exact 32-byte RLP string.

Acknowledged:

Curve has acknowledged this informational finding, and decided to not implement the suggested improvements as they do not affect correctness, and without them, the code remains more generic.

7.2 Gas Savings

Informational | Version 1 | Code Partially Corrected

CS-CURVE-XGOV-003

1. In `MessageDigestVerifier`, the function `_verifyMessages()` reads `nonce` twice from storage, one `SLOAD` could be saved.
2. In the function `extractProofValue` of `MerklePatriciaProofVerifier`, `rlpValue` is declared at the beginning of the function, only to be used before one of the return statements. It could be declared just before that return statement, saving some gas.

Code partially corrected:

The second gas saving was implemented, for the first one, Curve answered:

```
Left `++nonce` update as is, so no intermediary code affects `cur_nonce`.
```

7.3 Merkle Library Limitations

Informational | Version 1 | Acknowledged

CS-CURVE-XGOV-005

While the system always expect to pass a `path` of length `32` `bytes` to `MerklePatriciaProofVerifier.extractProofValue()`, in theory, the library is designed to handle paths of varying lengths. However, given the type of the argument (`bytes`), it is not possible to pass a path having an odd length in nibbles.

Acknowledged:

Curve acknowledged the limitation.

7.4 Missing Sanity Checks

Informational | Version 1 | Code Partially Corrected

CS-CURVE-XGOV-010

The following checks are missing:

1. None of the addresses in `_admins` is not validated to ensure they are not the zero address in `XYZBroadcaster.__init__()`
2. Message are not ensured to have non-empty payloads and the chain-id is not validated (non-zero, not current chain) in `XYZBroadcaster.broadcast()`
3. None of the addresses in `_future_admins` is validated to ensure they are not the zero address in `XYZBroadcaster.commit_admins()`
4. The new `_messenger` is not validated to ensure it is not the zero address in `XYZRelayer.set_messenger()`
5. The address parameters `_block_hash_oracle` and `_relayer` in the `MessageDigestVerifier`'s constructor are not validated to ensure they are not the zero address.

6. In `MessageDigestVerifier`, `verifyMessagesByStateRoot` does not validate the `state_root` to be non-zero.

Code partially corrected:

- Check 3 has been partially implemented as the `ownership` role is now validated to be non-zero.
- Check 4, 5 and 6 have been implemented.

Other missing sanity checks were not addressed. Curve answered:

```
Broadcaster is already in prod,
changes are not that important to update
```

7.5 Outdated Vyper Version

Informational **Version 1** **Acknowledged**

CS-CURVE-XGOV-011

Across the system, the Vyper version `0.3.10` is used, which is outdated, and should be updated to a more recent version in the `0.4.x` series.

Acknowledged:

Curve acknowledged the use of an outdated Vyper version.

7.6 Vulnerability in Solidity-RLP Library

Informational **Version 1** **Acknowledged**

CS-CURVE-XGOV-013

All released versions of the `RLPReader` library suffer from a vulnerability that allows an attacker to read out-of-bound memory when parsing a byte array of RLP-encoded data.

Vulnerability Description

The `toList()` function in `RLPReader.sol` is vulnerable to out-of-bounds (OOB) memory reads. When decoding an RLP-encoded list, `toList()` iterates over the payload and constructs `RLPItem` objects for each sub-item. However, it does not verify that each sub-item's memory range is fully contained within the bounds of the original buffer. As a result, a maliciously crafted RLP input can cause `toList()` to create `RLPItem` objects that reference memory outside the intended buffer, leading to OOB reads and potential leakage of adjacent memory contents.

Proof of Concept

The following Foundry test shows the vulnerability in action, as `0x70` is printed on the console, showing that the out-of-bounds read can access unintended memory—in this case, the memory location of the variable *a*.

```
pragma solidity 0.8.18;

import "forge-std/Test.sol";
import "forge-std/console.sol";
```

```
import {RLPReader} from "hamdiallam/Solidity-RLP@2.0.7/contracts/RLPReader.sol";

contract RlpReaderTest is Test {

    using RLPReader for RLPReader.RLPItem;
    using RLPReader for bytes;

    function test_RlpDecoding() public {
        // This RLP-encoded data is crafted to trigger an out-of-bounds read and for the
        // out-of-bound to happen on the most significant byte of the next word in memory.
        // It is a list with 16 items, each being a list containing an empty list.
        // Except for the last item, which contains a list whose content is out of bound.
        // Note that the first `C1` is malformed, but the library don't check this.
        bytes memory rlpEncodedData = hex"C1C1C0C1C0C1C0C1C0C1C0C1C0C1C0C1C0C1C0C1C0C1C0C1C0C1C0C1C0C1C0C1C0C1C0C1";
        uint256[] memory a = new uint256[](1);
        assembly {
            mstore(a, 0x7000000000000000000000000000000000000000000000000000000000000000)
        }
        RLPReader.RLPItem[] memory rlpItem = rlpEncodedData.toRlpItem().toList();
        assertEq(rlpItem.length, 16);
        assertEq(rlpItem[15].toList().length, 1);
        console.logBytes(rlpItem[15].toList()[0].toBytes());
    }
}
```

Impact

No impact was found on the system, because:

1. An out-of-bounds read alone is not problematic, since the caller could have provided a byte array containing the same content that was read out-of-bounds.
2. An issue would arise if multiple reads were to access different data in the out-of-bounds memory, for example, if the first read passed a validation check and the second read returned unexpected data.
3. What is described in point 2 was not observed in the system, as user-provided byte arrays of RLP-encoded data are followed by other memory allocations that are not mutated once the validation begins.

Remediation

As of commit `da526be21b744d427d796a1cba6cfbc94934eaf3`, the Solidity-RLP library has been updated to include bounds checking in the `toList()` function. This prevents the creation of `RLPItem` objects that reference memory outside the original buffer.

Acknowledged:

Curve acknowledged the vulnerability.

7.7 `future_admins` Not Cleared

Informational **Version 1** **Acknowledged**

CS-CURVE-XGOV-014

In `XYZBroadcaster`, after applying new admins, the `future_admins` storage variable is not cleared and will keep the last value set by `commit_admins()`. This can lead to confusion when reading the public getter for `future_admins`.

Acknowledged:

Curve acknowledged the informational finding, but will not fix it as the code is already deployed.

8 Notes

We leverage this section to highlight further findings that are not necessarily issues. The mentioned topics serve to clarify or support the report, but do not require an immediate modification inside the project. Instead, they should raise awareness in order to improve the overall understanding.

8.1 Commit and Apply Admins

Note Version 1

In the `XYZBroadcaster`, the current admin can give ownership to a new admin by calling first `commit_admins()` and then `apply_admins()`. Note that in this scheme, the old admin must call `apply_admins()`, and not the future new admin.

8.2 Reverting Messages

Note Version 1

If a received message at nonce n fail to execute and consistently revert, the system will not be able to process any further messages at nonce n+1 or higher until the deadline for the message at nonce n expires, this is behavior is by design.

```
if (block.timestamp <= deadline.value) {
    IRelayer(RELAYER).relay(_agent, _messages);
}
```