



SMART CONTRACT SECURITY AUDIT

Xenea

Bridge Contracts

CLIENT

Xenea

LANGUAGE

Solidity

REVIEW

October 2025

DOCUMENT

Document information

Project	Xenea, Bridge Contracts
Client	Xenea
Platform	Not recorded in the original report
Language	Solidity
Repository	Not recorded in the original report
Commit	Not recorded in the original report
Methods	Not recorded in the original report
Assessed	October 2025

Document history

VERSION	DATE	CHANGE
1.0	October 2025	Original report issued View original report
1.1	October 2026	Reformatted into the Safe Edges standard template. Findings unchanged

This report was reformatted into the Safe Edges standard template in October 2026. Findings, severities, recommendations and code references are unchanged from the original report, which is preserved unchanged in the [Safe Edges Audit Portfolio on GitHub](#).

CONTENTS

Contents

01	Executive summary	3
02	Scope	4
03	Methodology	5
04	Severity and status	6
05	Findings summary	7
06	Detailed findings	8
	H-01 Insufficient Signature Malleability Check in SignatureUtils:recoverSigner	8
	H-02 Insufficient Multi-Chain Replay Attack Protection in SignatureUtils Contract	9
	H-03 Missing Constructor and _disableInitializers() to Prevent Unauthorized Initialization	10
	M-01 Missing Call to __UUPSUpgradeable_init() in Xeneacraft:initialize	10
	M-02 Potential Returndata Gas Bomb Attack in Xeneacraft:unlock	11
	L-01 Missing Zero-Address Validation Leading to Potential State Corruption	11
	L-02 Missing Explicit Function to Set and Validate Fees in Xeneacraft	12
	I-01 Use Ownable2StepUpgradeable Instead of OwnableUpgradeable	12
	I-02 Missing Event Emissions for Significant State Changes	12
	G-01 Use external Instead of public for Gas Optimization	13
	G-02 Use Custom Errors Instead of require Statements to Save Gas	13
	Disclaimer	14

Executive summary

About Xenea

Xenea Bridge moves assets between chains using signed lock, unlock and mint operations, implemented as upgradeable Solidity contracts.

Summary of the review

Safe Edges reviewed the Xenea Bridge contracts in October 2025. Eleven issues were identified: three High, two Medium, two Low and four Informational, two of which are gas optimisations. The High issues covered signature malleability, cross-chain signature replay and an unprotected initializer on the upgradeable implementation, and all three were fixed. Eight issues were fixed in total and three were acknowledged.

Findings by severity



Findings by status

STATUS	ISSUES
Resolved	8
Acknowledged	3

Scope

Repository	Not recorded in the original report
Commit	Not recorded in the original report
Platform	Not recorded in the original report
Language	Solidity

Only the code at the commit above was reviewed. Changes made after this commit are not covered by this report.

Methodology

Methods recorded for this engagement: Not recorded in the original report.

Manual review

Auditors read the in-scope code line by line, tracing control flow, state changes and trust boundaries against the intended behaviour described by the client.

Threat modelling

We map privileged roles, external dependencies and value flows, then test the assumptions the code makes about each of them.

Functional and automated testing

Where tooling exists for the language and chain, we run static analysis and targeted tests to confirm suspected issues and check edge cases.

Fix review

After the client addresses the findings, we review the updated code and record the final status of each issue.

Severity and status

Severity levels

SEVERITY	DEFINITION
Critical	Direct loss or permanent lock of funds, or full compromise of the protocol, with few or no preconditions. Must be fixed before deployment.
High	Significant loss of funds, broken core functionality or manipulation of critical state under realistic conditions. Should be fixed before deployment.
Medium	Limited loss or degraded functionality, or issues that need specific conditions or privileged access to exploit. Should be fixed.
Low	Minor impact, low likelihood, or deviations from best practice. Fix when practical.
Informational	Code quality, documentation, gas and style observations with no direct security impact.

Status definitions

STATUS	DEFINITION
Resolved	The issue was fixed and the fix was verified.
Partially resolved	The risk was reduced but not fully removed.
Acknowledged	The client accepted the issue without changing the code.
Open	The issue was not addressed at the time of the report.
Not recorded	The original report did not record a resolution status for this issue.

Findings summary

ID	TITLE	SEVERITY	STATUS
H-01	Insufficient Signature Malleability Check in SignatureUtils:recoverSigner	High	● Resolved
H-02	Insufficient Multi-Chain Replay Attack Protection in SignatureUtils Contract	High	● Resolved
H-03	Missing Constructor and _disableInitializers() to Prevent Unauthorized Initialization	High	● Resolved
M-01	Missing Call to __UUPSUpgradeable_init() in Xeneacraft:initialize	Medium	● Resolved
M-02	Potential Returndata Gas Bomb Attack in Xeneacraft:unlock	Medium	● Acknowledged
L-01	Missing Zero-Address Validation Leading to Potential State Corruption	Low	● Resolved
L-02	Missing Explicit Function to Set and Validate Fees in Xeneacraft	Low	● Acknowledged
I-01	Use Ownable2StepUpgradeable Instead of OwnableUpgradeable	Informational	● Resolved
I-02	Missing Event Emissions for Significant State Changes	Informational	● Resolved
G-01	Use external Instead of public for Gas Optimization	Informational	● Acknowledged
G-02	Use Custom Errors Instead of require Statements to Save Gas	Informational	● Resolved

Detailed findings

H-01 Insufficient Signature Malleability Check in SignatureUtils:recoverSigner

Severity **High** Status ● Resolved

DESCRIPTION

The `recoverSigner` function attempts to recover the signers address from a given message hash and signature. However, after extracting the v, r, and s values from the signature, it fails to enforce a crucial check on the s value. The s value in an ECDSA signature should always be in the lower half order to ensure signature uniqueness. Without this validation, the function allows for malleable (non-unique) signatures, leading to potential replay attacks. The EVMs `ecrecover` precompile permits multiple valid signatures for the same hash, meaning an attacker could generate two different signatures for the same message, potentially exploiting the system.

IMPACT

Allows signature malleability, making the system vulnerable to replay attacks.

RECOMMENDED MITIGATION

Incorporate an additional check similar to OpenZeppelins implementation to ensure s is within the lower half order:

In `SignatureUtils:recoverSigner`

```
assembly {
    r := mload(add(signature, 0x20))
    s := mload(add(signature, 0x40))
    v := byte(0, mload(add(signature, 0x60)))
}
+ if (uint256(s) > 0x7FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF5D576E7357A4501DDFE92F46681B20A0) {
+     return (address(0));
+ }
if (v < 27) {
    v += 27;
}
```

H-02 Insufficient Multi-Chain Replay Attack Protection in SignatureUtils Contract

Severity **High** Status ● Resolved

DESCRIPTION

The contract generates a signature hash using the `getEthSignedMessageHash` function. However, this function currently lacks additional contextual elements such as `chainId()` and `address(this)`. Without these elements, an attacker could reuse the same signature across multiple chains, leading to cross-chain replay attacks.

IMPACT

An attacker could use a valid signature from one blockchain on another chain, executing unintended transactions.

RECOMMENDED MITIGATION

Enhance the signature hash creation by incorporating `block.chainId` and `address(this)`, ensuring uniqueness across different chains:

```
function getEthSignedMessageHash(
    bytes32 _messageHash
) internal pure returns (bytes32) {
    return
        keccak256(
            abi.encodePacked(
                "\x19Ethereum Signed Message:\n32",
                block.chainId,
                address(this),
                _messageHash
            )
        );
}
```

H-03 Missing Constructor and `_disableInitializers()` to Prevent Unauthorized Initialization

Severity **High** Status ● Resolved

DESCRIPTION

Upgradeable contracts should explicitly disable their initializer function to prevent unauthorized initialization of the implementation contract. Failing to do so allows anyone to call the initializer function and take control of the contract.

IMPACT

A malicious actor could initialize the contract, set themselves as an owner/admin, and gain unauthorized control.

RECOMMENDED MITIGATION

Include `_disableInitializers()` in the constructor:

```
+ constructor() {  
+   _disableInitializers();  
+ }
```

M-01 Missing Call to `__UUPSUpgradeable_init()` in `Xenecraft:initialize`

Severity **Medium** Status ● Resolved

DESCRIPTION

The Xenecraft contract inherits UUPSUpgradeable, but fails to call its initializer function. This could cause unexpected behavior in upgradeable deployments.

RECOMMENDED MITIGATION

Add the missing initializer call:

```
function initialize() public initializer {  
    __Ownable_init(msg.sender);  
    __Pausable_init();  
    __ReentrancyGuard_init();  
+   __UUPSUpgradeable_init();  
    adminLists[msg.sender] = true;  
    signer = msg.sender;  
}
```

M-02 Potential Returndata Gas Bomb Attack in Xeneart:unlock

Severity **Medium** Status ● Acknowledged

DESCRIPTION

The unlock function executes a low-level call to `curUser_`:

```
(bool success, ) = address(curUser_).call{value: amount_}("");
require(success, "UnLock: Fail transfer native");
```

A malicious `curUser_` can exploit this by returning excessive data, causing a gas exhaustion attack, known as a "returndata bombing" attack. Although the user is responsible for covering the gas, they could exploit this vulnerability to force a revert with a dust amount.

RECOMMENDED MITIGATION

Use `ExcessivelySafeCall` with `maxCopy` set to 0 or 32 bytes to prevent excessive memory allocation.

L-01 Missing Zero-Address Validation Leading to Potential State Corruption

Severity **Low** Status ● Resolved

DESCRIPTION

Certain functions fail to validate addresses, potentially allowing incorrect state changes.

EXAMPLES

```
function setSigner(address user_) external onlyAdmins {}
function setMinter(address minter_) external onlyOwner{}
```

RECOMMENDED MITIGATION

Add zero-address validation in relevant functions:

```
require(user_ != address(0), "Invalid address");
```

L-02 Missing Explicit Function to Set and Validate Fees in Xenearaft

Severity **Low** Status ● Acknowledged

DESCRIPTION

The contract allows zero-fee transactions if `fee == 0`, potentially undermining the lock and burn mechanism.

IMPACT

Users may exploit fee-free transactions to bypass intended economic restrictions.

RECOMMENDED MITIGATION

Introduce a setter function with `minFee` and `maxFee` constraints.

I-01 Use Ownable2StepUpgradeable Instead of OwnableUpgradeable

Severity **Informational** Status ● Resolved

DESCRIPTION

`OwnableUpgradeable` lacks a two-step ownership transfer process, making accidental or malicious transfers possible.

RECOMMENDED MITIGATION

Use `Ownable2StepUpgradeable` instead:

```
- import "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";  
+ import "@openzeppelin/contracts-upgradeable/access/Ownable2StepUpgradeable.sol";
```

I-02 Missing Event Emissions for Significant State Changes

Severity **Informational** Status ● Resolved

DESCRIPTION

State changes should emit events for better off-chain tracking.

EXAMPLES

```
setAcceptedToken(..)  
setMinter(address minter_)
```

RECOMMENDED MITIGATION

Add events to relevant functions.

G-01 Use external Instead of public for Gas Optimization

Severity Informational

Status ● Acknowledged

Category Gas

DESCRIPTION

Functions that are never called internally should be marked external for gas efficiency.

EXAMPLES

```
- function initialize() public initializer {}  
+ function initialize() external initializer {}
```

G-02 Use Custom Errors Instead of require Statements to Save Gas

Severity Informational

Status ● Resolved

Category Gas

DESCRIPTION

Using Solidity v0.8.4+ custom errors reduces gas consumption compared to string-based require statements.

RECOMMENDED MITIGATION

Replace require with custom errors:

```
error Unauthorized();  
require(msg.sender == owner, Unauthorized());
```

Disclaimer

This report is based on the code at the commit listed in the Scope section and on information the client provided during the review. It reflects Safe Edges' assessment at the time of the review and is not a guarantee that the code is free of vulnerabilities.

Changes made after the reviewed commit, deployment configuration, off-chain components and third-party dependencies are outside the scope of this report unless stated otherwise.

Security is ongoing. We recommend continued testing, monitoring and bug bounty coverage alongside any audit.

This report is not investment advice and is not an endorsement of the project, its team or any token.



Safe Edges is a Web3 security firm and official security partner for the Fuel, ENI and XI EcoChain ecosystems. We work across smart contract audits, Rust and WASM reviews, ZK circuit review, AI and MCP security, and penetration testing.

200+

PROJECTS AUDITED

\$4B+

TVL SECURED

0

POST-AUDIT HACKS