SALUS SECURITY

JAN 2025

CODE SECURITY ASSESSMENT

YALA

Overview

Project Summary

- Name: Yala - Notary_Smartcontracts
- Platform: EVM-compatible chains
- Language: Solidity
- Repository:
  - https://github.com/yalaorg/yala-notary-smartcontracts
- Audit Range: See Appendix - 1

Project Dashboard

Application Summary

Name	Yala - Notary_Smartcontracts
Version	v2
Type	Solidity
Dates	Jan 02 2025
Logs	Nov 26 2024; Jan 02 2025

Vulnerability Summary

Total High-Severity issues	3
Total Medium-Severity issues	3
Total Low-Severity issues	2
Total informational issues	2
Total	10

Contact

E-mail: support@salusec.io

Risk Level Description

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for clients' reputations or serious financial implications for clients and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to a moderate financial impact.
Low Risk	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 pose an immediate risk, but is relevant to security best practices or defense in depth.

Content

Introduction

1.1 About SALUS

At Salus Security, we are in the business of trust.

We are dedicated to tackling the toughest security challenges facing the industry today. By building foundational trust in technology and infrastructure through security, we help clients to lead their respective industries and unlock their full Web3 potential.

Our team of security experts employ industry-leading proof-of-concept (PoC) methodology for demonstrating smart contract vulnerabilities, coupled with advanced red teaming capabilities and a stereoscopic vulnerability detection service, to deliver comprehensive security assessments that allow clients to stay ahead of the curve.

In addition to smart contract audits and red teaming, our Rapid Detection Service for smart contracts aims to make security accessible to all. This high calibre, yet cost-efficient, security tool has been designed to support a wide range of business needs including investment due diligence, security and code quality assessments, and code optimisation.

We are reachable on Telegram (https://t.me/salusec), Twitter (https://twitter.com/salus_sec), or Email (support@salusec.io).

1.2 Audit Breakdown

The objective was to evaluate the repository for security-related issues, code quality, and adherence to specifications and best practices. Possible issues we looked for included (but are not limited to):

- Risky external calls
- Integer overflow/underflow
- Transaction-ordering dependence
- Timestamp dependence
- Access control
- Call stack limits and mishandled exceptions
- Number rounding errors
- Centralization of power
- Logical oversights and denial of service
- Business logic specification
- Code clones, functionality duplication

1.3 Disclaimer

Note that this security audit is not designed to replace functional tests required before any software release and does not give any warranties on finding all possible security issues with the given smart contract(s) or blockchain software, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues.

Findings

2.1 Summary of Findings

ID	Title	Severity	Category	Status
1	Adding accounts with leaked private keys as notary	High	Business Logic	Resolved
2	Defects of signature verification mechanism	High	Business Logic	Resolved
3	Missing signatures length verification allows unauthorized ether extraction	High	Data Validation	Resolved
4	Transferring ether to the Bitcoin chain is not allowed	Medium	Business Logic	Resolved
5	Lack of ether rebalancing mechanism	Medium	Business Logic	Resolved
6	Centralization risk	Medium	Centralization	Acknowledged
7	BTC’s decimals is inconsistent	Low	Inconsistency	Resolved
8	Weth’s permit silent revert	Low	Business Logic	Resolved
9	Missing zero address checks	Informational	Data Validation	Resolved
10	Gas optimization suggestions	Informational	Gas Optimization	Resolved

2.2 Notable Findings

Significant flaws that impact system confidentiality, integrity, or availability are listed below.

1. Adding accounts with leaked private keys as notary

Severity: High	Category: Business Logic

Target:

- src/YALABridge.sol

Description

The `YALABridge` contains test code, which should be removed before deployment.

The `notaries` control the `receiveMessage` and the `receiveEther` function. The `setupTestnet` function will add the `0xf39F...2266` to `notaries`, but the private key corresponding to `0xf39F...2266` has been leaked. So the action is very dangerous. The attacker can steal assets belonging to the cross-chain bridge by signing with a leaked private key.

src/YALABridge.sol:L82-93

```
function setupTestnet() public {
    localTokens[uint8(TokenType.BTC)] = Token({
        id: uint8(TokenType.BTC),
        symbol: "YBTC",
        contractAddress: 0x5FbDB2315678afecb367f032d93F642f64180aa3
    });

    notaries[0xf39Fd6e51aad88F6F4ce6aB8827279cffFb92266] = true;

    remoteTokens[258][1] = true; // Enable BTC on Bitcoin
    remoteTokens[1][1] = true; // Enable BTC on Ethereum
}
```

Recommendation

Remove the `setupTestnet` function.

Status

The team has resolved this issue in commit 94aa8ce.

2. Defects of signature verification mechanism

Severity: High	Category: Business Logic

Target:

- src/YALABridge.sol

Description

The `receiveMessage` function processes cross-domain token transfer messages, validating their authenticity using signatures from enough approved notaries, ensuring message uniqueness, and verifying input integrity. But the loop is missing to check if `signatures[i]` are different. The attacker can use the same `signatures[i]` to bypass the check using signatures from enough approved notaries.

src/YALABridge.sol:L219-248

```
function receiveMessage(
) external whenNotPaused {
    require(signatures.length >= minimumSignatures, "Invalid signatures");
    ...
    bytes32 messageId = MessageHashUtils.toEthSignedMessageHash(message);
    require(!processedMessages[messageId], "Message already processed");
    processedMessages[messageId] = true;

    for (uint i = 0; i < signatures.length; i++) {
        address recoveredAddress = recoverSigner(messageId, signatures[i]);
        require(notaries[recoveredAddress], "Invalid notary");
    }
```

The `recoverSigner` function extracts the signer's address from a hashed message and its corresponding signature using the `ecrecover` method. But using the `ecrecover` method is vulnerable to malleability attacks, because it is possible to produce another valid signature for the same message (which also means the same digest).

src/YALABridge.sol:L246-L260

```
function recoverSigner(
    bytes32 hash,
    bytes memory signature
) internal pure returns (address) {
    require(signature.length == 65, "Invalid signature length");

    bytes32 r;
    bytes32 s;
    uint8 v;

    assembly {
        r := mload(add(signature, 32))
        s := mload(add(signature, 64))
        v := byte(0, mload(add(signature, 96)))
    }

    return ecrecover(hash, v, r, s);
}
```

Recommendation

1. Make sure the signature will not be reused.
2. Using OpenZeppelin’s ECDSA library instead of EVM's ecrecover.

Status

The team has resolved this issue in commit 94aa8ce.

3. Missing signatures length verification allows unauthorized ether extraction

Severity: High	Category: Data Validation
Target: - src/YALABridge.sol	

Description

In the `receiveMessage` function, there is a check to ensure that the number of provided signatures meets the `minimumSignatures` threshold:

src/YALABridge.sol:L178-L212

```
function receiveMessage(
    ...
    bytes[] calldata signatures
) external whenNotPaused {
    require(signatures.length >= minimumSignatures, "Invalid signatures");
    ...
}
```

However, in the `receiveEther` function, this check is absent:

src/YALABridge.sol:L213-L244

```
function receiveEther(
   …
   bytes[] calldata signatures
) public whenNotPaused {
   require(nativeTokenType > 0, "native token is not allowed");
   require(amount > 0, "Invalid amount");
   require(receiver != address(0) && sender.length >= 20, "Invalid receiver or sender");
   require(isValidHash(fromHash), "Invalid fromHash");

   bytes memory message = abi.encodePacked(
       fromDomainIdentifier, domainIdentifier, nativeTokenType, amount, fromHash,
fromIndex, receiver, sender
   );

   bytes32 messageId = MessageHashUtils.toEthSignedMessageHash(message);
   require(!processedMessages[messageId], "Message already processed");
   processedMessages[messageId] = true;

   for (uint256 i = 0; i < signatures.length; i++) {
       address recoveredAddress = recoverSigner(messageId, signatures[i]);
       require(notaries[recoveredAddress], "Invalid notary");
   }

   (bool success,) = receiver.call{value: amount}("");
   require(success, "Ether transfer failed");

   emit MessageReceived(receiver, fromDomainIdentifier, nativeTokenType, amount,
fromHash, fromIndex, messageId);
}
```

An attacker can exploit this missing check by calling the `receiveEther` function with an empty signatures array, thus skipping the for loop, and the function will not perform any signature verification, enabling unauthorized extraction of all ether held by the contract.

Recommendation

Add the Minimum Signatures Check:

```
require(signatures.length >= minimumSignatures, "Invalid signatures");
```

Status

The team has resolved this issue in commit d37e2e4.

4. Transferring ether to the Bitcoin chain is not allowed

Severity: Medium	Category: Business Logic
Target: - src/YALABridge.sol	

Description

In the YALABridge contract, the `sendEther` function is designed to facilitate the cross-chain transfer of ether. However, the function contains a logical issue where it allows ether to be sent to the Bitcoin chain (`ChainType.BITCOIN`), which is not allowed.

src/YALABridge.sol:L161-L176

```
function sendEther(uint16 toDomainIdentifier, bytes calldata receiver) public payable
whenNotPaused {
   require(nativeTokenType > 0, "native token is not allowed");
   require(msg.value > 0, "Invalid amount");
   require(remoteTokens[toDomainIdentifier][nativeTokenType], "Unsupported destination
domain token");

   ChainType destinationChain = ChainType(toDomainIdentifier >> 8);
   if (destinationChain == ChainType.EVM) {
       require(isValidETHAddress(receiver), "Invalid receiver");
   } else if (destinationChain == ChainType.BITCOIN) {
       require(isValidBitcoinAddress(receiver), "Invalid receiver");
   } else {
       revert("Unknown chain type");
   }

   emit MessageSent(_msgSender(), domainIdentifier, nativeTokenType, toDomainIdentifier,
receiver, msg.value);
}
```

Users may mistakenly believe they can transfer ether to the Bitcoin network, but their Bitcoin address won't receive the money.

Recommendation

Remove the logic in the `sendEther` function.

Status

The team has resolved this issue in commit 5ca1868.

5. Lack of ether rebalancing mechanism	
Severity: Medium	Category: Business Logic
Target: - src/YALABridge.sol	

Description

The YALABridge contract serves as a cross-chain bridge deployed on multiple EVM-compatible blockchains, facilitating the transfer of ether and tokens across different networks. For ether cross-chain transfers, the `sendEther` and `receiveEther` functions rely on the contract holding sufficient ether on each chain to process user transactions. Specifically, when users initiate a cross-chain ether transfer, the `receiveEther` function on the destination chain must have enough ether in the contract to send to the user.

There is no mechanism in place within the YALABridge contract to rebalance ether holdings across the different chains it operates on. This absence can lead to situations where:

- Imbalance of ether holdings: Some chains may have a surplus of ether locked in the contract, while others may experience shortages, preventing successful cross-chain transfers.
- Service disruption: Users attempting to receive ether on a chain where the contract lacks sufficient balance will face transaction failures, degrading the user experience.

Recommendation

Introduce a reBalancer role: using role-based access control to manage ether rebalancing tasks.

Status

The team has resolved this issue in commit 5ca1868.

6. Centralization risk

Severity: Medium	Category: Centralization
Target: - src/YALABridge.sol	

Description

The YALABridge contract has privileged accounts assigned with the `DEFAULT_ADMIN_ROLE`. The privileged accounts have extensive control over the contract’s critical functions. Specifically, they can:

Pause the contract: By utilizing the `whenNotPause` modifier on the `receiveMessage` and `receiveEther` functions, privileged accounts can pause the contract on the destination chain. When paused, users are unable to receive their tokens or ether, leading to uncertainty and potential loss of trust, as users may not understand why their assets are not being minted or when the service will resume.

Remove notaries from the mapping: Privileged accounts can remove notaries from the contract’s notary mapping. Notaries are essential for validating cross-chain transactions. If a notary is removed, users’ calls to `receiveMessage` may revert due to invalid or insufficient signatures, forcing users to obtain new signatures from other notaries. This can cause significant delays and operational challenges.

If the privileged accounts are plain EOA accounts, this can be worrisome and pose a risk to the other users.

Recommendation

We recommend transferring privileged accounts to multi-sig accounts with timelock governors for enhanced security. This ensures that no single person has full control over the accounts and that any changes must be authorized by multiple parties.

Status

This issue has been acknowledged by the team.

7. BTC’s decimals is inconsistent

Severity: Low	Category: Inconsistency
Target: - src/YALABridge.sol	

Description

Function `` `receiveMessage` `` mint amount for users exact number, as received. The BTC’s decimals are 8 in the Bitcoin chain. The yBTC’s decimals are 18 in the EVM-compatible chains. So it needs recalculation on the destination chain.

src/YALABridge.sol:L219-L270

```
function receiveMessage(
    uint8 tokenType,
    uint256 amount,
    …
) external whenNotPaused {
    ...
    IERC20MintBurnable(token).mint(receiver, amount);
    …
}
```

Recommendation

Add precision conversion logic.

Status

The team has resolved this issue in commit 43d7d88.

8. Weth’s permit silent revert

Severity: Low	Category: Business Logic

Target:

- src/YALABridge.sol

Description

Most ERC20 have the permit function to approve a spender if a valid signature is provided.

However WETH does not. Surprisingly, when permit is called on WETH, the function call will execute without any errors.

src/YALABridge.sol:L128-L151

```
function sendMessageWithPermit(
    uint8 tokenType,
    uint256 amount,
    uint16 toDomainIdentifier,
    bytes calldata receiver,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) external whenNotPaused {
    address token = localTokens[tokenType].contractAddress;
    require(token != address(0), "Unknown token");

    IERC20Permit(token).permit(
        _msgSender(),
        address(this),
        amount,
        deadline,
        v,
        r,
        s
    );
    sendMessage(tokenType, amount, toDomainIdentifier, receiver);
}
```

Recommendation

Avoid adding cross-chain support for such tokens.

Status

The team has resolved this issue in commit a0339f8.

2.3 Informational Findings

9. Missing zero address checks

Severity: Informational	Category: Data Validation
Target: - src/YBTC.sol	

Description

It is considered a security best practice to verify addresses against the zero address during initialization or setting. However, this precautionary step is absent for address variables `` `DEFAULT_ADMIN_ROLE` ``, `` `PAUSER_ROLE` `` and `` `MINTER_ROLE` ``.

src/YBTC.sol:L15-22

```
constructor(address defaultAdmin, address pauser, address minter)
    ERC20("YALA BTC", "yBTC")
    ERC20Permit("YALA BTC")
{
    _grantRole(DEFAULT_ADMIN_ROLE, defaultAdmin);
    _grantRole(PAUSER_ROLE, pauser);
    _grantRole(MINTER_ROLE, minter);
}
```

Recommendation

Consider adding zero address checks for address variables `` `DEFAULT_ADMIN_ROLE` ``, `` `PAUSER_ROLE` `` and `` `MINTER_ROLE` ``.

Status

The team has resolved this issue in commit 94aa8ce.

10. Gas optimization suggestions

Severity: Informational	Category: Gas Optimization

Target:
- src/YALABridge.sol
- src/BatchMessageReceiver.sol

Description

Memory reading saves more gas than storage reading multiple times when the state is not changed. So caching the storage variables in memory and using the memory instead of storage reading is effective. Cache array length outside of the loop can save gas.

src/YALABridge.sol:L245,L304

```
for (uint i = 0; i < signatures.length; i++) {
```

src/YALABridge.sol:L355

```
for (uint256 i = 0; i < tokens.length; i++) {
```

src/YALABridge.sol:L387,LL401

```
for (uint i = 0; i < addr.length; i++) {
```

src/YALABridge.sol:L410

```
for (uint i = 0; i < hash.length; i++) {
```

src/BatchMessageReceiver.sol:L46

```
for (uint256 i = 0; i < tokenTypes.length; i++) {
```

Recommendation

Consider using the above suggestions to save gas.

Status

The team has resolved this issue in commit 94aa8ce.

Appendix

Appendix 1 - Files in Scope

This audit covered the following files in commit c84c914:

File	SHA-1 hash
src/YBTC.sol	3944652fd650d38e83add6d2a45dba72b3ad0914
src/YALABridge.sol	8faea542e70ebdcc7989a28c55efad0a070613da
src/BatchMessageReceiver.sol	9d9b9b3bf3f226a9cd44b1195a4c9f2d5acdb521