2.5 Recommendation Summary

Severity

Issues	🔴 High	🟡 Medium	🟢 Low	🔵 Informational
Open	1	0	2	1
Resolved	1		2	1
Acknowledged				
Partially Resolved				

- **Open:** Unresolved security vulnerabilities requiring resolution.
- **Resolved:** Previously identified vulnerabilities that have been fixed.
- **Acknowledged:** Identified vulnerabilities noted but not yet resolved.
- **Partially Resolved:** Risks mitigated but not fully resolved.

3.0 Checked Vulnerabilities

We examined the smart contract for widely recognized and specific vulnerabilities. Below are some of the common vulnerabilities considered.

Category	Check Items
Source Code Review	➔ Reentrancy Vulnerabilities ➔ Ownership Control ➔ Time-Based Dependencies ➔ Gas Usage in Loops ➔ Transaction Sequence Dependencies ➔ Style Guide Compliance ➔ EIP Standard Compliance ➔ External Call Verification ➔ Mathematical Checks ➔ Type Safety ➔ Visibility Settings ➔ Deployment Accuracy ➔ Repository Consistency
Functional Testing	➔ Business Logic Validation ➔ Feature Verification ➔ Access Control and Authorization ➔ Escrow Security ➔ Token Supply Management ➔ Asset Protection ➔ User Balance Integrity ➔ Data Reliability ➔ Emergency Shutdown Mechanism

Issue#1 [Resolved]

[H-1] Order Execution Vulnerability Due to Lack of Nonce Validation

Severity

High

Location	Functions
~/AdminController.ts	`processTransaction`

Issue Description

The `processTransaction` endpoint in the `AdminController` does not validate or track a nonce for each transaction. In blockchain and bridge applications, nonce validation is critical to maintaining transaction order and preventing replay attacks. Without it, previous requests can be replayed, leading to duplicate transaction execution, double spending, and inconsistencies in the bridge state.

```
@Post(ExternalRoutes.EXECUTE_ORDER)
async processTransaction(@Body() { chainType }: ExecuteOrderDto):
Promise<any> {
    try {
        return await this.ordersService.eprocessTransaction(chainType);
    } catch (error) {
        throw new HttpException(error, HttpStatus.BAD_REQUEST);
    }
}
```

Recommendation

Implement Nonce Validation:

Add a `nonce` parameter to the request and verify it against a stored record of previously used nonces.

Prevent Replay Attacks:

Ensure that each nonce is unique and follows an incrementally sequential order for each user or chain type to prevent duplicate transactions.

Update DTOs:

Modify `ProcessTransactionDto` to include a `nonce` field, ensuring that each transaction carries a unique identifier for validation and replay protection.

```
@Post(ExternalRoutes.EXECUTE_ORDER)
async processTransaction(
    @Body() { chainType, nonce }: ProcessTransactionDto
): Promise<any> {
    try {
        // Validate nonce
        const isValidNonce = await
this.ordersService.validateAndUpdateNonce(chainType, nonce);
        if (!isValidNonce) {
            throw new HttpException('Invalid nonce or replay attempt',
HttpStatus.BAD_REQUEST);
        }

        // Execute order
        return await this.ordersService.executeOrder(chainType);
    } catch (error) {
        throw new HttpException(error, HttpStatus.BAD_REQUEST);
    }
}

// DTO
class ProcessTransactionDto extends ExecuteOrderDto {
    @IsNumber()
    @IsNotEmpty()
    nonce: number;
}
```

Status

Issued Fixed

Issue#2 [Resolved]

[L-1] Insufficient Input Validation in `handleTrigger`

Severity

Low

Location	Functions
~/orderController.ts	`handleTrigger`

Issue Description

The `handleTrigger` function processes inputs without proper validation:

```
trigger = await bridgeContract.populateTransaction.trigger(
    orderDetails.tokenAddress,
    orderDetails.user,
    BigInt(orderDetails.amount),
    orderDetails.orderId,
    Number(CHAIN_TYPE_NUMBER[orderDetails.chainType]),
    tokenDetails.decimals,
    orderDetails.isWrappedToken,
    orderDetails.isNativeCurrency,
    tokenDetails.symbol,
    tokenDetails.name
);
```

Potential Risks:

Malicious inputs can lead to unexpected behavior, including:

- Denial of Service (DoS): Malformed transactions could cause system instability.
- Unauthorized Token Transfers: Incorrect tokenAddress values might lead to unintended token transfers.

Resolution:

The file was removed from the project.

Recommendation

Implement strict validation checks for all input parameters before processing:

```
if (!ethers.utils.isAddress(orderDetails.tokenAddress)) {
    throw new Error("Invalid token address");
}

if (!ethers.utils.isAddress(orderDetails.user)) {
    throw new Error("Invalid user address");
}

if (!Number.isSafeInteger(orderDetails.amount)) {
    throw new Error("Invalid amount");
}
```

Status

Issued Fixed

Issue#3 [Resolved]
[L-2] Inadequate CORS Configuration

Severity
Low

Location	Functions
~/main.ts	`app.enableCors`

Issue Description
In a blockchain bridge application, improper CORS (Cross-Origin Resource Sharing) settings may allow unauthorized external domains to interact with backend services. If the `whitelistedDomain` variable is not enforced, sensitive API endpoints could be exposed to malicious actors.

Recommendation
Implement a strict whitelist for CORS configuration to prevent unauthorized access:

```
const whitelistedDomain = configService.get("WHITELIST_DOMAINS");

app.enableCors({
    origin: whitelistedDomain.split(','),
    credentials: true,
});
```

Status
Issued Fixed

Issue#4 [Resolved]

[I-1] Logging Disabled in Non-Stage Environments

Severity

Info

Location	Functions
~/main.ts	NA

Issue Description

Logging plays a crucial role in monitoring, debugging, and auditing, especially in production environments. Completely disabling logs in non-stage environments can hinder incident response and troubleshooting efforts.

```
if (process.env.NODE_ENV !== "stage") {
    console.log = () => {};
    console.error = () => {};
    console.warn = () => {};
}
```

Recommendation

Maintain essential logging for production monitoring and debugging while ensuring that sensitive information is not exposed.

```
if (process.env.NODE_ENV === "production") {
    app.useLogger(WinstonModule.createLogger(LoggerConfig));
} else {
    console.log = console.log;
    console.error = console.error;
    console.warn = console.warn;
}
```

Status

Issued Fixed