Multichain Stellar

Security Assessment

coinspect

Stellar CrossChain-Router

Security Assessment

V230330 Prepared for Multichain • March 2023

1. Executive Summary

2. Assessment and Scope

3. Summary of Findings

4. Detailed Findings

MTC-01 Bridge minimum and maximum swap amount limits not enforced

MTC-02 Project lacks unit and integration tests

MTC-03 Errors are returned to callers when functions succeed

MTC-04 Bridge transactions will fail during fee price surge

MTC-05 Bridge panic after GetAccount swallows errors

MTC-06 GetOperations and GetBlockTxids incorrect retry logic

MTC-07 Loss of funds caused by overflow on big.Int cast to int64

MTC-08 SendTransaction callers ignore failed transactions

MTC-09 BuildRawTransaction returns incorrect error message

MTC-10 Can’t sign transactions with more than one operation

MTC-11 Hard-coded constants and code readability improvements

MTC-12 Wrong address regular expression

5. Disclaimer

6. About Coinspect

1. Executive Summary

In **March 2023,** Multichain engaged Coinspect to perform a source code review of the **Multichain cross-chain router's new adapter code for the Stellar network**. The objective of the project was to evaluate the security of the off-chain adapter that integrates the Stellar network to the already existing set of supported networks of the Multichain cross-chain router.

During the security audit, Coinspect identified high risk issues related to missing enforcement of bridge swapping limits and several issues related to security and code quality, including lack of unit and integration tests. The Multichain team swiftly addressed the identified issues, which led to an enhanced security level for the final product.

It is strongly recommended that unit and integration tests with good coverage are developed and implemented before deploying the project to production.

The other reported issues involve potential business impacts but based on our analysis of their likelihood we decided to reduce their overall risk level to Medium and Low.

The following issues were identified during the initial assessment:

High Risk	Medium Risk	Low Risk
Open **0**	Open **0**	Open **0**
Fixed 1	Fixed 5	Fixed 1
Reported 2	Reported 5	Reported 1

2. Assessment and Scope

The audit started on **March 13, 2023** and was conducted on the `feature/stellar` branch of the git repository at https://github.com/anyswap/CrossChain-Router/ as of commit `3d93a94f5b43e2e11f6125f31e46437e8e554fcd` of **February 27, 2023**.

The scope of the audit was limited to the following source files located in the `tokens/stellar` directory, shown here with their `sha256sum` hash:

```
c7176512fdf76816ffe69c6adea7ba61217d673d4e1d8b8442da8cddb294df61  verifytx.go
07f3fb1ecb5ca85fb91f97ff88240b4b7060c25edfd04ff9b0a3337a5f2c3eb4  register.go
992fac231159d634f4b350578e1d89482e8d279a01a69904782679a927df66c3  tools/sendPaymentTx/main.go
1f32302fe1d8368948a1161680820ac79a272f7d18c45bea093c4d4b1f728a39  tools/publicKeyToAddress/main.go
c2bcd16497b07550fc1583a04e4eb1fac6a299a2e0706d3f456dffdb1eabb843  tools/addressToPublickey/main.go
8f52c86588c1036884d8a814c7d8489fe751b3232798e3675b6c1c1f099eea83  tools/sendTrustLineTx/main.go
3400cec87c7054ddad2d11503574802764b9733e9965e12de6ff38a0eb117d28  tools/buildSwapMemo/main.go
5421bafa68857b9bc2cff5d09c2fbb6cd64aa1b635c9df23f79af6bb390ed62e  tools/getStubChainID/main.go
5b93b5073ee5f8fa80a43c8d51ba58c44a101bbeeabb8333e00d141104b1c38f  address.go
e8866f069024e37c6de07a520243161232372b5080288600208acf3fd4c42e30  sendtx.go
39e51f784a049a68f78b64d814577ac7deafbeaaedcd9721a95377fee4409dc5  bridge.go
9b1bccea800599e7eed739698b3fb586628f7c43cbb0d9dba2f9026a9aba7106  signtx.go
f42d9ee65778866cb05099ce2852728caddcaa4c3c7d427d5aa1827b351c4f01  buildtx.go
a7eb730f28275bdcfaff16f25f8c3ccea69c1b54f7992298d1ccdf76251358e1  init.go
```

On **28 March, 2023** the Multichain team shared an update with bug fixes for the issues and new tests on commit `8d795c7a13477a5681b969f7d75dc36bff548469`. The updated `sha256sum` values are:

```
b7685709d00f9988a5461754375fb0210177db1b03133de41718995e8a2c565f  bridge.go
e0b135c11bcdecd3ae57a1fb42dc6b0dba112724e1eef36220d3650c348de482  buildtx.go
4c95c26948998dc29d23531afae63297ff61410db38afb09eece8a7792652f8b  register.go
29de8c30a704df9755b5623d5641842950d61c9952b51a10504ee07d50368daa  sendtx.go
e460197aeaddc45ba01b3429d9c8bb11cebf345ebaa7951aba6194f2dbaecf0c  signtx.go
3a9a24b59a250d51d15047288b5406f7fa3eb00be19dd162955c753c82074e7e  verifytx.go
f087538db4fa464d4b23ef5b4569a454aec211596cfd42eab95e3544f2a561da  address.go
```

Coinspect auditors focused on the Multichain-Stellar integration; however, to correctly evaluate the security of this component, it was necessary to review, at least partially, other modules of the CrossChain Router base framework in order to understand how both these components interact.

The Multichain cross-chain router provides a key mechanism for sending tokens between different chains. For this, the project supports multiple tokens in different networks. It does so by providing common interfaces that must be satisfied for

adding a network into the platform. The main interface is the `IBridge` interface defined in the `interface.go` file.

This audit focuses on the files introduced in `tokens/stellar` which implement the `Bridge` struct, a structure that satisfies the `IBridge` interface, with support for the Stellar network. The Stellar adapter relies on the "memo" mechanism, and does not support the `anyCall` interface, only `ERC20SwapType` swaps are handled.

During the audit, Coinspect assumed that the Multichain cross-chain router framework was correctly working for every other network. This included process management, funding of the bridge agents, database design, RPC requests throttling, and especially the MPC implementation.

Coinspect auditors reviewed the code looking for flaws that would enable attackers to duplicate swap transactions, crash the router node workers, fake swap-ins and swap-outs, and network availability and DoS attacks.

Unit tests are minimum and report a coverage of `17.0%` by the go cover tool. Tests should exercise both happy paths and errors, covering corner cases and reaching a high coverage value.

The most important issues detailed in this report are related to:

1. Missing security checks required by the bridge framework
2. Incorrect error handling
3. Handling of big integers
4. Lack of testing

Here is a quick description of the issues:

- High-priority issues include unenforced minimum and maximum swap amount limits (MTC-01) and a lack of unit and integration tests (MTC-02).
- Medium-priority issues involve erroneous error handling (MTC-03), bridge transaction failures during Stellar network fee price surges (MTC-04), bridge panics (MTC-05), incorrect transaction retry logic (MTC-06), and loss of funds due to an big integer overflow (MTC-07).
- Low-priority and informational issues include ignored failed transactions (MTC-08), incorrect error message propagation (MTC-09), the inability to

sign transactions with multiple operations (MTC-10), and hard-coded constants and readability concerns (MTC-11).

All the issues identified by Coinspect have been fixed during the engagement.

3. Summary of Findings

Id	Title	Total Risk	Fixed
MTC-01	Bridge minimum and maximum swap amount limits not enforced	High	✔
MTC-02	Project lacks unit and integration tests	High	!
MTC-03	Errors are returned to callers when functions succeed	Medium	✔
MTC-04	Bridge transactions will fail during fee price surge	Medium	✔
MTC-05	Bridge panic after GetAccount swallows errors	Medium	✔
MTC-06	GetOperations and GetBlockTxids incorrect retry logic	Medium	✔
MTC-07	Loss of funds caused by overflow on big.Int cast to int64	Medium	✔
MTC-08	SendTransaction callers ignore failed transactions	Low	✔
MTC-09	BuildRawTransaction returns incorrect error message	Info	✔
MTC-10	Can't sign transactions with more than one operation	Info	✔
MTC-11	Hard-coded constants and code readability improvements	Info	✔
MTC-12	Wrong address regular expression	Info	✔

4. Detailed Findings

MTC-01 Bridge minimum and maximum swap amount limits not enforced

Total Risk	Impact	Location
High	High	`tokens/stellar/verifytx.go` `tokens/base.go`
Fixed ✔	Likelihood: High	

Description

It is possible to bypass the bridge enforced limits for maximum and minimum swap values.

The `VerifyTransaction` and `verifySwapoutTx` functions are missing a call to the `CheckTokenSwapValue` function in `tokens/base.go`. This function, implemented in the base framework code is responsible for enforcing minimum and maximum swap amounts, besides whitelisting and important sanity checks. Network adapters are expected to call this function. It is worth noting that all other network adapters available in the repository behave as expected and call this function.

Recommendation

Call the `CheckTokenSwapValue` function from the `verifySwapoutTx` in order to guarantee the limits are correctly enforced by the bridge. Coinspect suggests the bridge base code is modified to always call this function, independently of the network adapter implementation, in order to avoid this scenario where an adapter implementation error can affect the bridge's security.

Status

Validations were added in the new `checkSwapoutInfo` function.

MTC-02 Project lacks unit and integration tests

Total Risk	Impact	Location
High	High	*
Fixed	Likelihood	
!	High	

Description

The project lacks unit and integration tests.

Recommendation

Test the new network adaptor. The Multichain test framework located in `tests/` can be leveraged to create tests.

Status

Acknowledged.

Stellar support to the test framework and unit tests were added. However, the coverage is still low. The MultichainThe team has recognized the necessity for enhancement in this area and declared that additional testing will be conducted prior to the project's launch into production.

MTC-03 Errors are returned to callers when functions succeed

Total Risk	Impact	Location
Medium	Medium	stellar/bridge.go
Fixed ✔	Likelihood Medium	

Description

Errors are wrongly returned to callers on multiple functions when consulting RPC nodes for chain states.

For example, the `GetTransaction` function returns an error if a single request fails:

```
func (b *Bridge) GetTransaction(txHash string) (tx interface{}, err error) {
   for i := 0; i < rpcRetryTimes; i++ {
       for _, r := range b.Remotes {
           resp, err1 := r.TransactionDetail(txHash)
           if err1 != nil {
               log.Warn("Try get transaction failed", "error", err1)
               err = err1
               continue
           }
           tx = &resp
           return
       }
       time.Sleep(rpcRetryInterval)
   }
   return
}
```

The `err` return variable is set on the first time that the `TransactionDetail` fails. This value will be returned even if the `TransactionDetail` call succeeds on a retry attempt.

Similar errors can be found on multiple functions: `GetLatestBlockNumber`, `GetBlockHash`, `GetAsset`, and `GetBlockTxids`.

Recommendation

Clean errors on success.

Status

Fixed except for `GetBlockTxids` which is not currently used. We recommend deleting the unused function.

MTC-04 Bridge transactions will fail during fee price surge

Total Risk	Impact	Location
Medium	Medium	`stellar/bridge.go`
Fixed ✔	Likelihood: Medium	

Description

The fee used to send Stellar transactions is hardcoded to the minimum admissible fee. As a consequence, swapins will fail during Stellar network fee surge conditions.

```
// GetFee get fee
func (b *Bridge) GetFee() int {
    return txnbuild.MinBaseFee
}
...

// MinBaseFee is the minimum transaction fee for the Stellar network of 100 stroops
(0.00001 XLM).
const MinBaseFee = 100
```

During high demand scenarios this may cause transactions to not be included in blocks.

Recommendation

The following Stellar documentation suggests best practices to handle network fees: Fees, Surge Pricing, and Fee Strategies | Stellar Documentation

Status

Fixed. It now checks for the suggested price provided by the connected node.

MTC-05 Bridge panic after GetAccount swallows errors

Total Risk	Impact	Location
Medium	High	stellar/bridge.go
Fixed ✔	Likelihood Medium	

Description

The `GetAccount` function does not return errors produced by remotes. Instead it returns `(nil, nil)` on error, which causes the Bridge to panic while dereferencing `nil`.

```
func (b *Bridge) GetAccount(address string) (acct *hProtocol.Account, err error) {
    destAccountRequest := horizonclient.AccountRequest{
        AccountID: address,
    }
    for i := 0; i < rpcRetryTimes; i++ {
        for _, r := range b.Remotes {
            resp, err1 := r.AccountDetail(destAccountRequest)
            if err1 != nil {
                continue
            }
            err = err1
            acct = &resp
            return
        }
        time.Sleep(rpcRetryInterval)
    }
    return
}
```

Any error returned by the `AccountDetail` call will cause the inner for loop iteration to end prematurely without assigning the value to the `err` return variable. If the error persists in the following retries, the function will eventually return `(nil, nil)` a value not handled correctly on all callers.

Example:

```
fromAccount, err := b.GetAccount(args.From)
if err != nil {
    return nil, err
}
```

```
// check XLM
if !b.checkXlmBalanceEnough(fromAccount) {
    return nil, tokens.ErrMissTokenConfig
}
```

Recommendation

Return an error when all attempts fail.

Status

Fixed: an error is returned if all attempts fail.

MTC-06 GetOperations and GetBlockTxids incorrect retry logic

Total Risk	Impact	Location
Medium	Medium	`bridge.go`
Fixed ✔	Likelihood Medium	

Description

The `GetOperation` and `GetBlockTxids` functions fail to obtain the required transaction information if only one RPC Stellar node fails to answer.

The intended for loop returns before trying to next available nodes in case of an error when calling the `NextOperationsPage` function.

```
if len(resp.Embedded.Records) >= int(rpcQueryLimit) {
    resp, err1 = r.NextOperationsPage(resp)
    if err1 != nil {
        err = err1
        log.Warn("Try get block tx ids failed", "error", err1)
        return nil, err
    }
} else {
    nextPage = false
}
```

Recommendation

Instead of returning, continue the loop in order to keep trying with the other configured nodes.

Status

Fixed. Methods now retry on failure.

MTC-07 Loss of funds caused by overflow on big.Int cast to int64

Total Risk	Impact	Location
Medium	High	`stellar/buildtx.go`
Fixed ✔	Likelihood: Low	

Description

Amount value cast to `Int64` may overflow and result in loss of funds or undefined behavior.

While processing swaps the `BuildRawTransaction` function in the Stellar adapter code is called by the bridge framework. This function then calls `getPaymentAmount` which incorrectly handles the amount passed as a parameter:

```
func getPaymentAmount(amount *big.Int, token *tokens.TokenConfig) string {
    value := float64(amount.Int64()) / math.Pow10(int(token.Decimals))
    return fmt.Sprintf("%.7f", value)
}
```

Then, the returned string value is used to call `NewUnsignedPaymentTransaction` to create the payment transaction corresponding to the swap operation, which is then signed and sent. As a consequence, the payment amount will not match the amount retrieved from the transaction that initiated the swap, leading to a loss of funds.

Recommendation

Use big integers for operations and formatting.

Status

Fixed: big values are handled correctly.

MTC-08 SendTransaction callers ignore failed transactions

Total Risk	Impact	Location

Low	Low	`stellar/sendtx.go`
		`stellar/tools/SendPaymentTx/main.go`
		`stellar/tools/SendTrustLineTx/main.go`

Fixed ✔	Likelihood: Low	

Description

The `SendTransaction` function returns the hash of the transaction even when it was not successful. However, its callers consider the transaction as successful without checking its status.

This is the code handling the submitted transaction in `SendTransaction`:

```
resp, err = r.SubmitTransaction(tx)
if err != nil {
    log.Warn("Try sending transaction failed", "error", err)
    continue
}
if !resp.Successful {
    log.Warn("send tx with error result", "result", resp.Successful, "message")
}
txHash = resp.Hash
success = true
```

Then, for example, in the `main` function in `SendPaymentTx`, the status of the transaction is ignored:

```
txHash, err = b.SendTransaction(signedTx)
if err != nil {
    log.Fatal("send tx failed", "err", err)
}
log.Info("send tx success", "txHash", txHash)
```

The same situation occurs in `SendTrustLineTx`.

Recommendation

Check the transaction status.

Status

The issue is fixed.

MTC-09 BuildRawTransaction returns incorrect error message

Total Risk	Impact	Location
Info	-	`buildtx.go`
Fixed ✔	Likelihood -	

Description

The `checkXmlBalanceEnough` function returns an incorrect error message when the account balance is less than required.

```
// check XLM
if !b.checkXlmBalanceEnough(fromAccount) {
    return nil, tokens.ErrMissTokenConfig
}
```

Recommendation

Return an error signaling the lack of balance.

Status

It now returns `ErrTokenBalanceNotEnough`.

MTC-10 Can’t sign transactions with more than one operation

Total Risk	Impact	Location
Info	-	`signtx.go`
Fixed ✔	**Likelihood** -	

Description

Because of how `verifyTransactionWithArgs` is implemented, it is not possible to sign transactions with more than one op:

```
for i := 0; i < len(opts); i++ {
    op, ok := opts[i].(*txnbuild.Payment)
    if !ok {
        continue
    }

    to := op.Destination

    if !strings.EqualFold(to, checkReceiver) {
        return fmt.Errorf("[sign] verify tx receiver failed")
    }
}
```

This issue is only information and does not currently represent a risk because the function is currently only being called to verify transactions created with the `NewUnsignedPaymentTransaction` function in `stellar/buildtx.go`.

Recommendation

Consider parsing all ops instead of returning error if the first one has not the expected receiver.

Status

The code was changed so only one operation per transaction is explicitly supported.

MTC-11 Hard-coded constants and code readability improvements

Total Risk	Impact	Location
Info	-	*
Fixed	**Likelihood**	
✔	-	

Description

There are multiple instances of coding practices that affect code readability and make maintaining the source code harder.

For example:

a. Single case switch

```
switch args.SwapType {
    case tokens.ERC20SwapType:
    default:
        return nil, tokens.ErrSwapTypeNotSupported
    }
```

b. Single iteration loop

```
startIndex, endIndex := 0, len(opts)
if logIndex != 0 {
    if logIndex >= endIndex || logIndex < 0 {
        return nil, []error{tokens.ErrLogIndexOutOfRange}
    }
    startIndex = logIndex
    endIndex = logIndex + 1
}
errs := make([]error, 0)
swapInfos := make([]*tokens.SwapTxInfo, 0)
for i := startIndex; i < endIndex; i++ {
    op := getPaymentOperation(opts[i])
    if op == nil {
        continue
    }
    si, err := b.buildSwapInfoFromOperation(txres, op, i)
    errs = append(errs, err)
    swapInfos = append(swapInfos, si)
}
```

c. Hard-coded constants: many constants are hard-coded in code without clear indication.

   i. retry attempts in the `CheckNativeBalance` function

```
func CheckNativeBalance(b IBridge, account string, needValue *big.Int)
(err error) {
    var balance *big.Int
    for i := 0; i < 3; i++ {
        balance, err = b.GetBalance(account)
        if err == nil {
            break
        }
```

ii. decimals used for token amount conversions

```
fixedFee := ConvertTokenValue(feeCfg.MinimumSwapFee, 18, fromDecimals)
```

Recommendation

Coinspect recommends improving the source code's readability and maintainability.

Status

Acknowledged.

MTC-12 Wrong address regular expression

Total Risk	Impact	Location
Info	-	`tokens/stellar/address.go`
Fixed ✔	**Likelihood** -	

Description

The regular expression used to match the address is incorrect. The value used is

```
var rAddressReg = "^[1-9A-Z]{56}$"
```

This regular expression matches values that should be rejected.

This doesn't impose any risk because the use of the regular expression is followed by the `strkey.Decode` function which correctly validates the values used.

Relevant documentation:

1. https://github.com/stellar/stellar-protocol/blob/master/ecosystem/sep-0023.md
2. https://www.rfc-editor.org/rfc/rfc4648#section-6

Recommendation

Adjust the regular expression.

Status

The issue was fixed by adjusting the regular expression.

5. Disclaimer

The information presented in this document is provided “as is” and without warranty. Security Audits are a “point in time” analysis, and as such, it’s possible that something in scope may have changed since the tasks reflected in this report were executed. This report shouldn’t be considered a perfect representation of the risks threatening the analyzed systems and/or applications in scope.

6. About Coinspect

Coinspect is a boutique-style security consulting firm focused on Blockchain, Cryptocurrencies and Web3 technologies. Founded in 2014 by a team of information security professionals with currently +20 years of experience providing valuable and outstanding results to cutting-edge technology companies worldwide.

- Team of expert auditors (with +6y of experience in crypto / web3)
- Tailored service delivery to each client and their specific targets(s).
- Own manual security audit methodology (not a product/platform centric).
- Contributing to the community by publishing papers, techniques, and tools.

We have contributed to secure key technologies and services of the cryptocurrency ecosystem by auditing, reporting vulnerabilities, and proposing improvements for new blockchain (L1/L2) implementations, smart contracts, dApps, mobile apps, hardware wallets, compilers, developer tools, DeFi protocols, and exchanges.

Delivering True Value to our Customers

- Lower Cost of Vulnerability Remediation; by delivering a detailed vulnerability report and included mitigation recheck services, our team will interact closely with Customers' team and educate them on findings and recommendations.
- Increase Confidence and Expand Adoption; by providing a reliable technology from a security standpoint the users will increase the circulating value in the platforms.
- Lessen Business Impact of Incidents; our state-of-the-art manual security assessment ensures that any potential threats are identified and addressed quickly and effectively, minimizing the business impact of incidents.

For more information, please visit our website https://www.coinspect.com/