ASSESSMENT REPORT

RIPPLE LABS INC.

SIDE CHAINS SECURITY ASSESSMENT 2023 — EVM

JULY 24, 2023

Bishop Fox Contact Information:
+1 (480) 621-8967
contact@bishopfox.com
8240 S. Kyrene Road
Suite A-113
Tempe, AZ 85284

TABLE OF CONTENTS

EXECUTIVE REPORT

Project Overview

Ripple Labs Inc. engaged Bishop Fox to assess the security of the Ripple EVM side chain feature. The following report details the findings identified during the course of the engagement, which started on June 13, 2023.

Goals

- Identify vulnerabilities on systems and services exposed on the internet-facing services related to the EVM side chain bridging infrastructure
- Assess the overall security of the EVM bridging protocol, witness server, and RPC interfaces
- Enumerate any weaknesses or potential hardening opportunities within the EVM bridging feature whose remediation could improve its security posture

FINDING COUNTS
3 Medium
8 Low

11 Total findings

SCOPE
EVM bridge packages

EVM bridge infrastructure

DATES
06/13/2023
Kickoff

06/19/2023 – 07/13/2023
Active testing

07/24/2023
Report delivery

Summary of Findings

The assessment team performed a security assessment of the EVM bridging feature, including the `xrp-evm` repository that implemented the EVM bridging protocol and the corresponding witness server. During this assessment, the team determined that the RPC networking interfaces were well-protected against injection-based attacks and identified no core issues with the bridging functionality.

In contrast, the assessment team did discover multiple issues related to the applications' build processes, such as the use of outdated dependencies. While such issues were not directly exploited during the assessment, they reduce the security posture of the affected applications and their build environments. The team also identified unsafe command and code execution patterns through which certain functions unsafely passed arguments to a command line interface or code execution engine. While the team did not identify any exploitation paths for these unsafe patterns, any additional functionality that uses these features or changes to the application through future development may permit an attacker to compromise the integrity of witness servers.

Additionally, the team identified multiple issues with lesser impact that did not present an immediate risk to the EVM bridging functionality, including minor cryptographic issues and minor issues related to deployed infrastructure that did not follow best practices.

Overall, despite attempting multiple potential attack paths against the EVM bridge, the team did not identify mechanisms for a remote attacker to violate the operational integrity of the EVM bridge applications or forge bridging transactions.

RECOMMENDATIONS

Update Software Dependencies — Remove outdated dependencies from affected packages and integrate dependency update checks into the CI/CD pipeline.

Remove Unsafe Execution Patterns — Avoid injecting arbitrary data into command and code execution functionality.

ASSESSMENT REPORT

Hybrid Application Assessment

The assessment team performed a hybrid application assessment with the following target in scope:

- EVM packages (`https://github.com/Peersyst/xrp-evm`)

Identified Issues

1 ARBITRARY CODE EXECUTION **MEDIUM**

Definition

Arbitrary code execution occurs when attackers execute code on a target machine within the same context as a compromised process. As a result, the machine cannot differentiate between trusted and untrusted code and will execute commands under the privileges of the original process.

Details

The assessment team identified one instance of arbitrary code execution in the `IsValidXAddress` function within the `bridge-witness` package. While the assessment team did not identify a mechanism to exploit this vulnerability, if a vector for passing arbitrary strings to this function were identified, an attacker could potentially execute arbitrary JavaScript on the bridge witness server within the `v8go` sandbox.

To identify the issue, the team reviewed the `IsValidXAddress` code and determined that it leveraged the `v8go` library's `Context.RunScript()` function to execute code within the V8 sandbox:

```
func (xrplJs *XrplJs) IsValidXAddress(account string) bool {
id := "a" + strconv.FormatUint(consumeId(), 10)
_, err := xrplJs.ctx.RunScript("let "+id+" = xrpl.isValidXAddress('"+account+"')",
 "call-isValidXAddress.js")
…omitted for brevity…
}
```

FIGURE 1 - `IsValidXAddress` function calling `RunScript` on string input

In the function above, the `account` input was insecurely concatenated onto the JavaScript code passed into the `v8go` engine. The team developed an exploit that, if injectable, could overwrite the `isValidXAddress` function within an initialized `v8go` instance to always return `true`:

```
func main() {
engine := NewXrplJs()
overwrite :=
engine.IsValidXAddress("X7YDPC4TJvjVxLc4QNDgCfaAocYVbWBE8jzpaKPZBy8mKDf');
xrpl.isValidXAddress = function(x) { return true; } //")
println("Response for `X7YDPC4TJvjVxLc4QNDgCfaAocYVbWBE8jzpaKPZBy8mKDf` address:")
println(overwrite)
println("Response for `notanaddress` address:")
exploited := engine.IsValidXAddress("notanaddress")
println(exploited)
}
```

FIGURE 2 - Injecting additional JavaScript into V8 to override `isValidAddress` function

In the above proof-of-concept (PoC), the first call to the Golang `IsValidXAddress` function passed a valid `account` to `isValidXAddress` that terminated the JavaScript statement by including [`');`] at the end of the address. The payload then overwrote the `xrpl.isValidXAddress` JavaScript function to always return `true`. Once run, the PoC returned the following response:

```
$ go run XrplJs.go
Response for `X7YDPC4TJvjVxLc4QNDgCfaAocYVbWBE8jzpaKPZBy8mKDf` address:
true
Response for `notanaddress` address:
true
```

FIGURE 3 - Running PoC to overwrite `v8go` function

As this PoC illustrated, attackers could leverage different attack strategies to disrupt the operation of the witness server. For example, an attacker could include an infinite loop in their payload and potentially cause a denial-of-service (DoS) scenario.

As previously mentioned, the assessment team did not identify a direct vector that could be used to inject arbitrary payloads into the affected function. However, if additional features were added that used the function against data retrieved from an attacker-controlled source, malicious users could significantly modify the intended functionality of the witness server.

Affected Locations

Source Code

`xrp-evm/packages/bridge-witness/external/xrpl.js/XrplJs.go:147`

Total Instances **1**

Recommendations

To remediate this instance of arbitrary code execution, the team recommends the following steps:

- Build an allowlist of the namespaces and commands allowed to be executed by the daemon, and strictly check all commands against the allowlist before execution.
- Perform strong input validation of all user-supplied data sent to the daemon.

Additional Resources

CWE-94: Improper Control of Generation of Code (‘Code Injection’)
http://cwe.mitre.org/data/definitions/94.html

2 ARBITRARY COMMAND INJECTION

MEDIUM

Definition

Arbitrary command injection occurs when a user passes maliciously crafted input into an application, which then uses the unchecked data in a function that executes at the operating system level. The system cannot differentiate between these malicious commands and regular application commands and executes calls within the authority context of the original application.

Details

The assessment team identified an instance of arbitrary command injection within the `cli` package. The command injection vulnerability existed within a command line application and could only be exploited by tricking end users into running the `cli` application with a malicious configuration file.

To identify the issue, the assessment team reviewed the `exprd` function and determined that it made an `exec` call with arbitrary input:

```
export function exrpd(arg: string, homeDir = "", dockerImage = "peersyst/xrp-evm-
client:latest", pre = ""): string {
   return exec(`${pre} docker run --rm -i -v ${homeDir}:/root/.exrpd ${dockerImage}
exrpd ${arg}`, {
       encoding: "utf8",
       stdio: "pipe",
   });
}
```

FIGURE 4 - `exrpd` function with insecure `exec` call

The assessment team generated the following PoC script to demonstrate how additional commands could be injected into the function if an attacker controlled the input:

```
import { execSync as exec } from "child_process";
export function exrpd(arg: string, homeDir = "", dockerImage = "peersyst/xrp-evm-
client:latest", pre = ""): string {
   return exec(`${pre} docker run --rm -i -v ${homeDir}:/root/.exrpd ${dockerImage}
exrpd ${arg}`, {
       encoding: "utf8",
       stdio: "pipe",
   });
}
// modified to remove calls to class properties
function parseHexAddressToBech32Poc(address: string): string {
   const { formats } = JSON.parse(exrpd(`keys parse ${address.replace("0x", "")} --
output json`));
   return formats[0];
}
function proofOfConcept1() {
 console.log("Running poc1...")
 let payload = "somecommand ; echo 'poc1.txt created via exrpd injection' >
poc1.txt; #"
 console.log("Payload:")
 console.log(payload)
   exrpd(payload, "", "hello-world")
}
function proofOfConcept2() {
 console.log("Running poc2...")
 let payload = "89B4dE433558cbEeA95cD57bfCA4357A4FEA4Ace --output json 2>/dev/null;
echo 'poc2.txt created via parseHexAddressToBech32Poc' > poc2.txt; echo
'{\"formats\":[0]}'; #";
 console.log("Payload:")
 console.log(payload)
  \parseHexAddressToBech32Poc(payload);
}
proofOfConcept1();
console.log("")
proofOfConcept2();
```

FIGURE 5 - PoC script to inject additional commands into function

The assessment team ran the PoC script and observed the following output:

```
$ ts-node poc.ts && cat poc1.txt poc2.txt
Running poc1...
Payload:
somecommand ; echo 'poc1.txt created via exrpd injection' > poc1.txt; #
Running poc2...
Payload:
89B4dE433558cbEeA95cD57bfCA4357A4FEA4Ace --output json 2>/dev/null; echo 'poc2.txt
created via parseHexAddressToBech32Poc' > poc2.txt; echo '{"formats":[0]}'; #

poc1.txt created via exrpd injection
poc2.txt created via parseHexAddressToBech32Poc
```

FIGURE 6 - PoC script demonstrating ability to inject arbitrary commands

The assessment team did not identify any external code paths into this functionality that an attacker could directly exploit to inject code into a victim's system. However, an attacker could exploit this vulnerability to compromise their target's system by coercing them into running the application against a maliciously crafted configuration file.

Affected Locations

Affected File

packages/cli/src/util/exrpd.ts:3

Total Instances 1

Recommendations

To mitigate the risk of arbitrary command injection, the assessment team recommends the following actions:

- Avoid arbitrarily building commands from strings and passing them to a shell interface.
- Perform strict input sanitization and validation against any values passed to the command line.
- Alternatively, do not leverage the command line interface (CLI) to execute Docker commands and instead utilize an HTTP interface that cleanly passes the command to a Docker container.

Additional Resources

CWE-94: Improper Control of Generation of Code ('Code Injection')
http://cwe.mitre.org/data/definitions/94.html

OWASP Command Injection
https://owasp.org/www-community/attacks/Command_Injection

OWASP Cheat Sheet Series - Input Validation
https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html

3 VULNERABLE SOFTWARE

MEDIUM

Definition

Vulnerable software exists when an application has not been updated with the latest security patches. These insecure versions of software can contain issues (e.g., arbitrary remote code execution or SQL injection) that could allow a malicious user to gain elevated access to the application itself or its supporting infrastructure.

Details

When analyzing the `xrp-evm` repository's use and integration of third-party dependencies, the team identified several dependencies and libraries that included potential security issues. When an application uses unpatched third-party libraries with known vulnerabilities or missing security updates, vulnerabilities or security weaknesses may be introduced into the application.

For example, the assessment team reviewed the `package.json` files leveraged by the `xrp-evm` build processes and found multiple dependencies with known security concerns:

```
$ npm audit
# npm audit report
parse-path  <5.0.0
Severity: high
Authorization Bypass in parse-path - https://github.com/advisories/GHSA-3j8f-xvm3-
ffx4
fix available via `npm audit fix --force`
Will install lerna@5.6.2, which is outside the stated dependency range
…omitted for brevity…
```

FIGURE 7 - Running `npm-audit` against `xrp-evm`'s `package.json`

Additionally, the team also identified potential issues in Golang dependencies present in the `xrp-evm` repository. For example, the `bridge-witness` package leveraged a vulnerable version of `v8go` that suffered from multiple type-confusion issues.

Finally, the team also identified outdated and potentially vulnerable software within multiple referenced Docker container builds in the `xrp-evm` repository, as shown below:

```
$ cat packages/cli/Dockerfile
FROM node:18.15.0 as install
ARG NPM_TOKEN
ENV NPM_TOKEN ${NPM_TOKEN}
```

FIGURE 8 - Example reference to node `18.15.0` within `cli` package

The team identified multiple uses of outdated versions of `node` and `alpine` Docker images that lacked up to date security fixes within the `xrp-evm` repository.

Affected Locations

Dependencies
For a full list of affected dependencies, please see the attached spreadsheet.

Total Instances **378**

Recommendations

To mitigate the risk of vulnerable software, the assessment team recommends the following action:

- Update the affected packages to their latest versions.

Strategic Considerations

- Integrate dependency update scans into the CI/CD pipeline and block merges when issues are identified.
- Regularly review security vulnerability lists and vendor advisory pages related to applications used within the environment. Apply all security patches released for these systems in a timely manner.
- Establish an expedited process for applying updates to critical vulnerabilities.

Additional Resources

OWASP Top Ten 2021 - Vulnerable and Outdated Components
https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/

CWE-937: Using Components with Known Vulnerabilities
https://cwe.mitre.org/data/definitions/937.html

List of Known v8 Vulnerabilities
https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=V8+Chrome

4 INSECURE NETWORK TRANSMISSION LOW

Definition

Insecure network transmission occurs when sensitive information is sent over a network without adequate protection. When data is sent across insecure communication channels, it may be susceptible to interception and modification by third parties, resulting in unauthorized information disclosure.

Details

The assessment team discovered that two EVM sidechain endpoints did not strictly enforce the use of HTTPS. Neither application contained sensitive customer information. However, the availability of the cleartext HTTP service could allow attackers to perform Man-in-the-Middle (MitM) attacks against users, through which users could be coerced into sending tokens to the attacker's wallet.

The team demonstrated this issue by directly browsing the HTTP version of the application and determining that the server did not return an automatic redirect to the secure HTTPS version, as shown below in the HTTP 200 response to a request sent without TLS encryption:

Request
```
GET / HTTP/1.1
Host: witness-evm-sidechain.peersyst.tech
Cache-Control: max-age=0
…omitted for brevity…
```

Response
```
HTTP/1.1 200 OK
Date: Fri, 14 Jul 2023 01:24:49 GMT
Content-Type: text/html; charset=utf-8
Connection: close
…omitted for brevity…
```

The lack of HTTPS could allow an attacker to inject a phishing site into a user's application, then trick the user into sending funds to the attacker's wallet.

Affected Locations

URLs

- `http://witness-evm-sidechain.peersyst.tech`
- `http://evm-poa-sidechain.peersyst.tech/`

Total Instances **2**

Recommendations

To mitigate the threat of data interception and modification, the assessment team recommends the following step:

- Enforce the use of TLS for all communications that transmit sensitive information (including data such as cookies and authentication credentials).

Strategic Consideration

- When switching entirely to HTTPS, consider enabling HTTP Strict Transport Security (HSTS).

Additional Resources

OWASP Cheat Sheet Series - Transport Layer Protection
https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html

CWE-319: Cleartext Transmission of Sensitive Information
https://cwe.mitre.org/data/definitions/319.html

Wikipedia - HTTP Strict Transport Security
http://en.wikipedia.org/wiki/HTTP_Strict_Transport_Security

5 INSECURE SSL/TLS CONFIGURATION LOW

Definition

The Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols allow secure communication between a client and a server. Known vulnerabilities introduced by insecure SSL/TLS configurations can potentially result in a successful Man-in-the-Middle (MitM) attack.

Details

The assessment team observed that multiple side chain endpoints supported the use of insecure protocols TLS 1.0 and TLS 1.1, which are subject to known vulnerabilities. Additionally, the team discovered insecure cipher suites on the affected endpoints. These insecure configurations could allow an attacker to access cleartext communications between the client and the server.

To confirm this issue, the assessment team leveraged the utility `testssl.sh` against the affected endpoints, with one result as an example shown below:

```
$ ./testssl.sh witness-evm-sidechain.peersyst.tech
…omitted for brevity…
 Testing protocols via sockets except NPN+ALPN
SSLv2      not offered (OK)
SSLv3      not offered (OK)
TLS 1      offered (deprecated)
TLS 1.1    offered (deprecated)
TLS 1.2    offered (OK)
TLS 1.3    offered (OK): final
NPN/SPDY   h2, http/1.1 (advertised)
ALPN/HTTP2 h2, http/1.1 (offered)

Testing cipher categories
NULL ciphers (no encryption)                      not offered (OK)
Anonymous NULL Ciphers (no authentication)        not offered (OK)
Export ciphers (w/o ADH+NULL)                     not offered (OK)
LOW: 64 Bit + DES, RC[2,4], MD5 (w/o export)      not offered (OK)
Triple DES Ciphers / IDEA                         offered
Obsoleted CBC ciphers (AES, ARIA etc.)            offered
Strong encryption (AEAD ciphers) with no FS       offered (OK)
Forward Secrecy strong encryption (AEAD ciphers)  offered (OK)

…omitted for brevity…
```

FIGURE 9 - Sample output from `testssl.sh` enumerating TLS configuration issues

These vulnerable configurations could allow attackers to threaten the confidentiality of network activity, access its cleartext, and steal sensitive information.

Affected Locations

URLs

- `https://witness-evm-sidechain.peersyst.tech`
- `https://evm-sidechain.xrpl.org`
- `https://evm-sidechain.peersyst.tech`
- `https://evm-poa-sidechain.peersyst.tech`

Total Instances **4**

Recommendations

The assessment team recommends the following methods to properly deploy SSL/TLS services:

- Disable TLS 1.0 and TLS 1.1 support in vulnerable services.
- Avoid the following insecure ciphers and algorithms: 3DES, SHA1, CBC.

Additional Resources

SSL/TLS Vulnerability Cheat Sheet: Certificate issues
https://github.com/IBM/tls-vuln-cheatsheet#certificate-issues

Testing TLS/SSL encryption
https://testssl.sh

SSL and TLS Deployment Best Practices
https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices

6 INSECURE SOFTWARE CONFIGURATION

LOW

Definition

Insecure software configuration occurs when applications and infrastructure are configured in a manner that is inconsistent with industry best practices. These misconfigurations could allow unauthorized access to affected systems, the disclosure of sensitive information, and the exposure of critical application logs

Details

The assessment team identified insecure software configurations in the `xrp-evm` code repository. These configurations provide opportunities for hardening the `xrp-evm` applications within the repository.

MISSING NODEJS PRODUCTION SETTING

The team identified two packages, `bridge-client-frontend` and `bridge-client-backend`, that contained Docker builds that did not set the `NODE_ENV` environment variable to `production`. To identify the issue, the team reviewed the Dockerfile configurations within these packages and discovered that the `NODE_ENV` setting had been commented out, as shown below:

```
FROM node:16.13.0 AS build
#ENV NODE_ENV=production
WORKDIR /app
COPY ["package.json", "yarn.lock", ".npmrc", "./"]
RUN yarn
```

FIGURE 10 – `bridge-client-backend` Dockerfile with `NODE_ENV` not set to `production`

Setting `NODE_ENV` to `production` provides multiple security benefits due to automatic hardening. For example, when `NODE_ENV` is set to `production`, node will minimize logging and avoid generating verbose error messages. Additionally, setting `NODE_ENV` to `production` also improves performance due to increased caching.

SERVER VERSION LEAKAGE

The assessment team determined that the `witness-evm-sidechain.peersyst.tech` endpoint returned default error messages that contained the server's version number. To identify the issue, the team sent the following request and reviewed the information returned in the HTTP 403 response:

Request

```
GET /static/ HTTP/2
Host: witness-evm-sidechain.peersyst.tech
```

```
…omitted for brevity…
```

Response

```
HTTP/2 403 Forbidden
Date: Fri, 14 Jul 2023 16:38:13 GMT
Content-Type: text/html
…omitted for brevity…

<html>
<head><title>403 Forbidden</title></head>
<body>
<center><h1>403 Forbidden</h1></center>
<hr><center>nginx/1.25.1</center>
</body>
</html>
…omitted for brevity…
```

While the issue was not directly exploitable, the leakage of the server's version could provide an attacker with useful information when constructing an exploit against the service.

Affected Locations

Locations

- `packages/bridge-client-frontend/Dockerfile:3`
- `packages/bridge-client-backend/Dockerfile:2`
- `https://witness-evm-sidechain.peersyst.tech`

Total Instances **3**

Recommendations

To prevent vulnerabilities introduced by insecure software configuration, the assessment team recommends the following steps:

- Ensure `NODE_ENV` is set to `production` when deploying an application.
- Disable all default error messages when deploying a reverse proxy and ensure server versions are not being disclosed to the public.

Additional Resources

Node.js, the difference between development and production
https://nodejs.dev/en/learn/nodejs-the-difference-between-development-and-production/

How To Configure Nginx to Use Custom Error Pages on Ubuntu 22.04
https://www.digitalocean.com/community/tutorials/how-to-configure-nginx-to-use-custom-error-pages-on-ubuntu-22-04

7 MISSING SECURITY HEADERS

LOW

Definition

HTTP security headers activate features in modern web browsers that help protect users against cross-site scripting (XSS), UI redress, and Man-in-the-Middle (MitM) attacks.

Details

The assessment team discovered that multiple EVM side chain endpoints omitted multiple modern security header directives that support additional security features in browsers. The lack of HTTP security headers is a frequently overlooked opportunity for additional protection against client-side and MitM attacks.

Descriptions of the missing security headers are provided below:

Header Name	Description
`Strict-Transport-Security`	This header enforces HTTP Strict Transport Security (HSTS), which instructs browsers to communicate with the application over HTTPS and prevents any communications from being sent over HTTP.
`X-Frame-Options`	This header restricts which domains can render the resource within a frame, iframe, or object tag, which provides protection against UI redress attacks. The application can deny this ability entirely, restrict it to the same origin as the embedding page, or specify a safelist of allowed origins. While this header is considered deprecated, it is still widely supported by browsers.
`X-Content-Type-Options`	Setting the value of this header to `nosniff` prevents Internet Explorer and Google Chrome from attempting to determine the content type by inspecting the response. This helps protect users from untrusted content being rendered as HTML or other content types.

FIGURE 11 - Missing security headers

Enabling these headers provides browsers with additional information regarding the security constraints that should be applied to the site. For backward compatibility reasons, browsers do not always implement these features automatically; they must be explicitly enabled by setting and including the HTTP security headers.

Affected Locations

URLs

- `https://witness-evm-sidechain.peersyst.tech`
- `https://custom.xrpl.org`
- `https://evm-poa-sidechain.peersyst.tech`
- `https://evm-sidechain.xrpl.org`
- `https://evm-sidechain.peersyst.tech`

Total Instances **5**

Recommendations

The assessment team recommends the following changes to leverage the security capabilities of modern browsers:

- Enable the `Strict-Transport-Security` header and set the `max-age` directive to a value greater than zero. If possible, HSTS should be enabled for subdomains using the `includeSubdomains` directive. It is best to start with a small value for the `max-age` directive, such as 86400 seconds, or one day, to ensure that any issues can be resolved before enabling HSTS for a longer period. If testing and user acceptance is successful, the value for the `max-age` directive should be gradually increased to one year, or 31536000 seconds.
- Enable the `X-Content-Type-Options` header and set it to `no-sniff`.
- Enable the `X-Frame-Options` header and set it to either `DENY` or `SAMEORIGIN`.

Additional Resources

OWASP Secure Headers Project
https://owasp.org/www-project-secure-headers

Strict-Transport-Security
https://developer.mozilla.org/en-US/docs/Web/Security/HTTP_strict_transport_security

X-Frame-Options
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Frame-Options

X-Content-Type-Options
https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/X-Content-Type-Options

CWE-693: Protection Mechanism Failure
https://cwe.mitre.org/data/definitions/693.html

8 OUTDATED SOFTWARE

LOW

Definition

Outdated dependencies exist when an application has not been updated with the latest patches or is using an outdated or deprecated version of a third-party library. Software that has not been kept up to date could prove more difficult to upgrade at pace when a vulnerability is made public.

Details

The assessment team reviewed the dependencies of the `xrp-evm` repository, including the Golang, Docker, and JavaScript dependencies leveraged by the multiple packages within the repository, and found instances of outdated dependencies in the repository. For example, the following excerpt from the `go.mod` configuration within the `bridge-witness` package shows the inclusion of outdated versions of AWS libraries:

```
module peersyst/bridge-witness-go
go 1.19
require (
	github.com/aws/aws-sdk-go-v2/config v1.1.1
	github.com/aws/aws-sdk-go-v2/service/kms v1.20.1
	github.com/ethereum/go-ethereum v1.10.26
	github.com/gorilla/websocket v1.4.2
	…omitted for brevity…
```

FIGURE 12 - Golang dependency configuration with outdated libraries

As shown above, the package relied on the v1.1.1 version of the `aws-sdk-go-v2/config` and v1.20.1 version of `aws-sdk-go-v2/service/kms`. However, both packages have more recent versions available that were not integrated into the `bridge-witness` package. While the assessment team did not identify any vulnerabilities or unpatched security updates related to the outdated packages, regularly updating packages ensures that unidentified or unreported vulnerabilities are not introduced into an application.

Affected Locations

Dependencies

For a full list of affected dependencies, please see the attached spreadsheet.

Total Instances 32

Recommendations

To mitigate the risk of outdated software, the assessment team recommends the following actions:

- Ensure that dependencies utilized by an application are still actively supported by upstream maintainers.
- Integrate automatic dependency updates in the CI/CD pipeline to ensure that dependencies are continuously being patched.

Additional Resources

OWASP Top Ten 2021 - Vulnerable and Outdated Components
https://owasp.org/Top10/A06_2021-Vulnerable_and_Outdated_Components/

CWE-937: Using Components with Known Vulnerabilities
https://cwe.mitre.org/data/definitions/937.html

9 SENSITIVE INFORMATION DISCLOSURE LOW

Definition

Sensitive information disclosure occurs when private data is exposed to unauthorized parties. This may include financial data, personal privacy information, health records, proprietary information, or other important data.

Details

The assessment team discovered that the `xrp-evm` repository contained multiple instances of potentially sensitive credentials within the source code. When sensitive credentials exist in a source code repository, any user with access to that repository can leverage those credentials to gain unauthorized access to services. Additionally, if source code were ever made public, the credentials would continue to exist in Git history and be available to the public.

The team determined that the repository contained `terraform.tfstate` and `terraform.tfstate.backup` files. Terraform state files often contain sensitive files generated when deploying infrastructure. In this case, the state files appeared to contain `tendermint` private keys, as shown below:

```
…omitted for brevity…
{
 "module": "module.blockchain.module.prepare_network",
 "mode": "data",
 "type": "local_file",
 "name": "keys",
 "provider": "provider[\"registry.terraform.io/hashicorp/local\"]",
 "instances": [
   {
     "index_key": 0,
     "schema_version": 0,
     "attributes": {
       "content": "-----BEGIN TENDERMINT PRIVATE KEY-----\nkdf: bcrypt\nsalt:
[REDACTED]\ntype: eth_secp256k1\n\n[REDACTED]\n=1zyW\n-----END TENDERMINT PRIVATE
KEY-----\n",
       "content_base64": "[REDACTED]",
       "filename": "/Users/[REDACTED]/GIT/xrp-evm/infra/devnet/.data/validator-
0.key",
       "id": "c763c08ba63cfe51b6614ba0b72119fa02e57335"
     },
     "sensitive_attributes": []
   },
…omitted for brevity…
```

FIGURE 13 - Terraform state file containing multiple `tendermint` private keys

Additionally, the team identified several NPM authentication tokens embedded in the repository. For example, the `bridge-client-backend` package contained a `.npmrc` file with an unencrypted authentication token:

```
$ cat packages/bridge-client-backend/.npmrc
//registry.npmjs.org/:_authToken=npm_[REDACTED]
legacy-peer-deps=true
```

FIGURE 14 - Identifying unencrypted NPM authentication tokens in package

The assessment team escalated these issues and confirmed with the application team that the NPM authentication tokens previously had read-only access to private NPM repositories but have since been revoked.

Finally, the assessment discovered that two endpoints, `evm-poa-sidechain.peersyst.tech` and `evm-sidechain.peersyst.tech`, appeared to publicly leak `prometheus` logs, as shown below:

Request

```
GET /metrics HTTP/1.1
Host: evm-sidechain.peersyst.tech
…omitted for brevity…
```

Response

```
HTTP/1.1 200 OK
Date: Thu, 13 Jul 2023 22:00:45 GMT
…omitted for brevity…

# TYPE phoenix_channel_receive_duration_microseconds histogram
# HELP phoenix_channel_receive_duration_microseconds Phoenix channel receive handler
time in microseconds
# TYPE phoenix_controller_call_duration_microseconds histogram
# HELP phoenix_controller_call_duration_microseconds Whole controller pipeline
execution time in microseconds.
# TYPE ecto_queue_duration_microseconds histogram
# HELP ecto_queue_duration_microseconds The time spent to check the connection out
in microseconds.
ecto_queue_duration_microseconds_bucket{result="ok",le="10"} 3723604574
ecto_queue_duration_microseconds_bucket{result="ok",le="100"} 3723724517
ecto_queue_duration_microseconds_bucket{result="ok",le="1000"} 3723724525
```

While such logs did not appear to contain sensitive data, attackers may still find them useful for understanding the state of the server or application.

Affected Locations

Affected Locations

- `xrp-evm/infra/devnet/terraform.tfstate`
- `xrp-evm/infra/devnet/terraform.tfstate.backup`

- `xrp-evm/.npmrc`
- `xrp-evm/packages/bridge-client-backend/.npmrc`
- `xrp-evm/packages/bridge-client-frontend/.npmrc`
- `xrp-evm/packages/bridge-node/.npmrc`
- `xrp-evm/packages/bridge-witness/.npmrc`
- `xrp-evm/packages/stress-tester/.npmrc`
- `http://evm-sidechain.peersyst.tech/metrics`
- `http://evm-poa-sidechain.peersyst.tech/metrics`

Total Instances **10**

Recommendations

To address the issue of sensitive information disclosure, the assessment team recommends the following remediation action:

- Remove all credential files from the affected Git repository and rotate the credentials found in these files.

Additional Resources

CWE-200: Exposure of Sensitive Information to an Unauthorized Actor
http://cwe.mitre.org/data/definitions/200.html

10 WEAK CONTENT SECURITY POLICY (CSP) LOW

Definition

Content Security Policy (CSP) is an HTML5 standard primarily designed to mitigate the issue of cross-site scripting (XSS) and other content injection vulnerabilities within web applications. Weak CSPs stem from overly permissive or ineffective content source directives

Details

The assessment team identified many endpoints deployed to support the EVM side chain infrastructure that either did not have a Content Security Policy (CSP) or had policies that did not follow security best practices. An insecure or improperly configured CSP could allow an attacker to exploit most XSS vulnerabilities in the affected applications.

The team determined that the `witness-evm-sidechain.peersyst.tech`, `custom.xrpl.org`, and `evm-poa-sidechain.peersyst.tech` endpoints did not contain a CSP in their responses. For example, when navigating to the `witness-evm-sidechain.peersyst.tech` application in a browser, the following response without a CSP was returned:

Request
```
GET / HTTP/2
Host: witness-evm-sidechain.peersyst.tech
…omitted for brevity…
```

Response
```
HTTP/2 200 OK
Date: Thu, 13 Jul 2023 18:39:29 GMT
Content-Type: text/html
Last-Modified: Tue, 27 Jun 2023 13:27:19 GMT
Cf-Cache-Status: DYNAMIC
Report-To:
{"endpoints":[{"url":"https:\/\/a.nel.cloudflare.com\/report\/v3?s=oPD4AyhKdXIvaa%2B
PY%2BR8MLgGdgf4VT1gbj%2FiWIMuCUw8MIBoOduI0V3AGlDVkylHlU5lYYIgl0pEoqe4EMlyPvIV052YddB
HbEyaYk8AB52uHMYV3eWx%2Fo29hoDclM2WEZ%2BpeZyuqF9RlHEG%2BmH4VEREblHyaQ%3D%3D"}],"grou
p":"cf-nel","max_age":604800}
Nel: {"success_fraction":0,"report_to":"cf-nel","max_age":604800}
Server: cloudflare
Cf-Ray: 7e63a61fef692e73-DFW
Alt-Svc: h3=":443"; ma=86400

<!doctype html>
<html lang="en">
…omitted for brevity…
```

Additionally, although the `evm-sidechain.xrpl.org` and `evm-sidechain.peersyst.tech` endpoints implemented a CSP, the assessment team observed that the policies that did not conform to best practices by including a `script-src` with a host allowlist and omitting an `object-src` directive, as shown below:

Request

```
GET / HTTP/2
Host: evm-poa-sidechain.peersyst.tech
…omitted for brevity…
```

Response

```
HTTP/2 200 OK
…omitted for brevity…
Content-Security-Policy: connect-src 'self' ws://evm-poa-sidechain.peersyst.tech
wss://evm-poa-sidechain.peersyst.tech wss://*.bridge.walletconnect.org/
https://request-global.czilladx.com/
https://raw.githubusercontent.com/trustwallet/assets/
https://registry.walletconnect.org/data/wallets.json https://*.poa.network;
default-src 'self';        script-src 'self' 'unsafe-inline' 'unsafe-eval'
https://coinzillatag.com https://www.google.com https://www.gstatic.com;
style-src 'self' 'unsafe-inline' 'unsafe-eval' https://fonts.googleapis.com;
img-src 'self' * data:;        media-src 'self' * data:;        font-src 'self'
'unsafe-inline' 'unsafe-eval' https://fonts.gstatic.com data:;        frame-src
'self' 'unsafe-inline' 'unsafe-eval' https://request-global.czilladx.com/
https://www.google.com;
…omitted for brevity…

<!DOCTYPE html>
<html lang="en-US">
…omitted for brevity…
```

When possible, it is best practice to set the `script-src` parameter to a restricted location such as `strict-dynamic` with nonces and script hashes as well as without `unsafe-inline` or `unsafe-eval`. In this case, the `coinzillatag.com` domain pointed to an advertising platform integrated into the upstream version of the block explorer. The `coinzillatag.com` and `request-global.czilladx.com` domains should be omitted from the `xrp-evm` version to reduce the likelihood of malware being injected via an advertisement campaign. Additionally, setting `object-src` to explicit values such as `self` or none increases the security posture of the application.

Affected Locations

URLs

- `https://witness-evm-sidechain.peersyst.tech`
- `https://custom.xrpl.org`
- `https://evm-poa-sidechain.peersyst.tech`
- `https://evm-sidechain.xrpl.org`
- `https://evm-sidechain.peersyst.tech`

Total Instances **5**

Recommendations

To mitigate a weak CSP, the assessment team recommends the following actions:

- Never allow `unsafe-inline` or `unsafe-eval` for active content such as `default-src`, `script-src`, `style-src`, or `object-src`.
- Avoid the use of wildcards [`*`] within any content sources.
- Avoid hosting user-supplied files on origins allowed by active content directives such as `script-src`.
- If an application does not use a content source, explicitly set corresponding CSP directives to `none`.

Additional Resources

HTML5 Rocks - An Introduction to Content Security Policy
http://www.html5rocks.com/en/tutorials/security/content-security-policy/

W3C - Content Security Policy
https://w3c.github.io/webappsec-csp/

OWASP Cheat Sheet Series - Content Security Policy
https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html

CWE-1021: Improper Restriction of Rendered UI Layers or Frames
https://cwe.mitre.org/data/definitions/1021.html

11 WEAK CRYPTOGRAPHY

LOW

Definition

Weak cryptography occurs when a weakness in an implemented cryptographic routine or function undermines the security of the encrypted data. The severity of the issue can range from theoretical weaknesses to weaknesses that allow an attacker to fully recover cleartext data without cryptographic keys.

Details

The team discovered that the `cli` package contained a contract deployment that did not provide the `saltNonce` with sufficient entropy. Due to the risk of a nonce collision, if a user attempted to generate many contracts with the same configuration, the package would not generate the intended number of contracts.

To identify the issue, the team analyzed the use of `deployeSafe` within the `EvmBridgeChainProvider.ts` file, as shown below:

```
const safe: Safe = await safeFactory.deploySafe({
   safeAccountConfig: {
       threshold: 1,
       owners: isSignerWitness ? witnesses : [...witnesses, signer.address],
   },
   saltNonce: BigNumber.from(crypto.randomInt(1_000_000)).toString(),
   options: {
       gasLimit: 3_000_000,
   },
});
```

FIGURE 15 - Call to `deploySafe` with insufficient entropy for `saltNonce`

According to Ripple's `safe-global` documentation, the `saltNonce` should have 256 bits of entropy. However, due to the restrictions imposed when generating the nonce via `randomInt`, the generated `saltNonce` only had 20 bits of entropy. The assessment team escalated this issue and determined that although a collision would result in unintended behavior and the generation of fewer contracts than specified, all generated contracts would still be controlled by the end user.

The assessment team also identified the use of weak algorithms to verify the integrity of NPM modules. To identify the issue, the teams reviewed the `package-lock.json` and `yarn.lock` files within the `xrp-evm` repository and determined that some modules leveraged SHA1 for integrity checks:

```
# THIS IS AN AUTOGENERATED FILE. DO NOT EDIT THIS FILE DIRECTLY.
# yarn lockfile v1

…omitted for brevity…
set-blocking@^2.0.0:
  version "2.0.0"
  resolved "https://registry.npmjs.org/set-blocking/-/set-blocking-2.0.0.tgz"
  integrity sha1-BF+XgtARrppoA93TgrJDkrPYkPc=
…omitted for brevity…
```

FIGURE 16 - Use of SHA1 within dependency `bridge-contracts` lock file

When weak algorithms are used to verify the integrity of files or packages, an attacker may bypass integrity checks and replace such files with malicious payloads.

Affected Locations

File Paths

- `packages/cli/src/bridge/create/EvmBridgeChainProvider.ts:95-104`
- `packages/explorer/apps/block_scout_web/assets/package-lock.json`
- `packages/explorer/apps/explorer/package-lock.json`
- `packages/blockchain/vue/package-lock.json`
- `packages/bridge-contracts/yarn.lock`
- `packages/bridge-client-backend/yarn.lock`

Total Instances **6**

Recommendations

To properly leverage cryptography in an application, the assessment team recommends the following actions:

- Ensure that nonce values are properly typed and have sufficient capacity.
- Review algorithms utilized by NPM to verify the integrity of packages.

Additional Resources

CWE-326: Inadequate Encryption Strength
https://cwe.mitre.org/data/definitions/326.html

CWE-327: Use of a Broken or Risky Cryptographic Algorithm
https://cwe.mitre.org/data/definitions/327.html

APPENDIX A — MEASUREMENT SCALES

Finding Severity

Bishop Fox determines severity ratings using in-house expertise and industry-standard rating methodologies such as the Open Web Application Security Project (OWASP) and the Common Vulnerability Scoring System (CVSS).

The severity of each finding in this report was determined independently of the severity of other findings. Vulnerabilities assigned a higher severity have more significant technical and business impact and achieve that impact through fewer dependencies on other flaws.

Critical	Vulnerability is an otherwise high-severity issue with additional security implications that could lead to exceptional business impact. Findings are marked as critical severity to communicate an exigent need for immediate remediation. Examples include threats to human safety, permanent loss or compromise of business-critical data, and evidence of prior compromise.
High	Vulnerability introduces significant technical risk to the system that is not contingent on other issues being present to exploit. Examples include creating a breach in the confidentiality or integrity of sensitive business data, customer information, or administrative and user accounts.
Medium	Vulnerability does not in isolation lead directly to the exposure of sensitive business data. However, it can be leveraged in conjunction with another issue to expose business risk. Examples include insecurely storing user credentials, transmitting sensitive data unencrypted, and improper network segmentation.
Low	Vulnerability may result in limited risk or require the presence of multiple additional vulnerabilities to become exploitable. Examples include overly verbose error messages, insecure TLS configurations, and detailed banner information disclosure.
Informational	Finding does not have a direct security impact but represents an opportunity to add an additional layer of security, is a deviation from best practices, or is a security-relevant observation that may lead to exploitable vulnerabilities in the future. Examples include vulnerable yet unused source code and missing HTTP security headers.

APPENDIX B — TEST PLAN

The following section contains the test cases completed for each methodology in scope.

HAA METHODOLOGY

COMPLETED TEST CASES

TEST CASE	DESCRIPTION
Conduct dynamic testing for authentication vulnerabilities	Attempt to misidentify or overwrite existing users through the user registration, password reset, and login features within the application.
Perform dynamic testing for signature and authentication code validation issues	Attempt to alter or forge signed data that the application trusts as genuine.
Perform dynamic testing for function-level authorization controls	Attempt to perform actions or functions with a user or role that should be restricted from those actions and functions.
Conduct dynamic testing for mass assignment	Attempt to find mass assignment vulnerabilities.
Perform dynamic testing for directory traversal	Attempt to manipulate file paths so that they refer to files that are not intended to be accessed.
Conduct dynamic testing for resource-based authorization controls	Attempt to access resources that a user should be restricted from accessing.
Perform dynamic testing for code injection	Attempt to provide malicious input to code being constructed with user-provided content in order to cause the interpreter to execute the provided code.
Perform dynamic testing for command injection	Attempt to edit the contents of command-line calls using special characters inside user-provided parameters within the construction of the command.
Perform dynamic testing for the insecure	Locate places where the system sends sensitive data without proper safeguards.

TEST CASE	DESCRIPTION
transmission of sensitive data	
Conduct dynamic testing for the insecure handling of sensitive data	Attempt to find places where the system mishandles sensitive data, either by sending it to users that should not have that data or by storing it insecurely.
Perform dynamic testing for weak symmetric encryption	Attempt to find issues with symmetric encryption implementation that may cause sensitive data to be disclosed.
Review dependency confusion vulnerabilities	Review application dependencies for potential exploitation through typosquatting or private vs. public repository sourcing.
Perform dynamic testing for file-handling vulnerabilities	Locate any files being mishandled in a manner that allows interaction with the server's filesystem or command execution on the server.
Perform dynamic testing for uncommon injection flaws	Attempt to find XML injection, LDAP injection, NoSQL injection, or expression language injection.
Perform NoSQL injections	Provide malicious input to NoSQL-based queries to perform unintended actions on a NoSQL database or to execute malicious code and unvalidated input within the application itself.
Conduct dynamic testing for insecure file upload	Determine whether uploaded files are loaded into directories that allow them to be interpreted as server-side or client-side code.
Perform dynamic testing for clickjacking	Attempt to find pages that can be rendered in an iframe to trick users into performing actions.
Conduct dynamic testing for information disclosure	Attempt to find responses that contain overly verbose information about the system under review.
Perform dynamic testing for memory management vulnerabilities	Attempt to identify user-supplied inputs that are unsafely loaded into memory or that unsafely reference existing memory.

TEST CASE	DESCRIPTION
Perform dynamic testing for authentication requirements	Evaluate the strength of the password requirements used for password-based logins and determine whether multi-factor authentication is used for sensitive logins.
Perform dynamic testing for session lifecycle issues	Evaluate sessions used by the system to ensure common issues are not present.
Conduct dynamic testing for user enumeration	Attempt to find places to disclose users registered in an application.
Perform forward tracing of security critical functionality in source code	Identify and analyze areas of critical functionality in code. Find and target system-specific goals and areas of high security relevance. Perform manual analysis of the security controls around these areas to ensure adherence to overall security goals.
Perform dynamic testing for object deserialization issues	Identify whether user-supplied inputs are used as serialized objects and sent to an unsafe deserialization routine.
Conduct dynamic testing for known vulnerabilities	Attempt to find known vulnerabilities in components used by the application.
Perform software composition analysis against system dependencies	Perform automated software composition analysis and checking of dependencies against public vulnerability lists. Investigate all findings to determine practical impact against the target system based on the usage and conditions necessary for exploitability.
Complete automated static code analysis of relevant codebases	Perform automated static code analysis for all codebases that are not open source. Manually investigate all findings for true positives and assess specific system risks.
Conduct dynamic testing for cross-origin resource sharing (CORS) issues	Attempt to find a CORS configuration that allows malicious sites to access the application on behalf of the user.
Perform dynamic testing for cross-site request forgery (CSRF)	Attempt to find form submission functionality that fails to verify the source of the submission for a state-changing action.

TEST CASE	DESCRIPTION
Perform dynamic testing for cross-site scripting (XSS) in client-side frameworks	Determine whether insecure functions are used to write user-provided content to the page within client-side frameworks.
Perform dynamic testing for XSS	Attempt to find user-supplied payloads that execute client-side code on other users' browsers.
Perform dynamic testing for lesser-known same-origin policy bypasses	Attempt to find the insecure use of Flash across origins and the insecure use of JSONP.
Conduct dynamic testing for WebSocket hijacking	Attempt to find WebSockets that fail to verify that the WebSocket originated from the site itself.
Conduct dynamic testing for unsafe `postMessage` or event handler use	Determine whether the `postMessage` API is in use or whether event handling allows messages from unintended sources.
Conduct dynamic testing for SQL injection	Attempt to edit the contents of a SQL query by inserting special characters inside user-provided parameters that are used to construct the query.
Perform dynamic testing for server-side request forgery (SSRF)	Attempt to manipulate some or all of the URLs in use by the system.
Perform dynamic testing for server-side template injection	Attempt to specify server-side template code that is evaluated by a template engine in order to execute arbitrary code on the affected system.
Perform SSO testing for SAML, OAuth 2.0, and common integration problems	Attempt to modify responses or exploit implementation failures to obtain unintended access or privilege escalation through SSO environments.
Conduct dynamic testing for XML external entities (XXE)	Attempt to find misconfigured XML parsers that allow the interpretation of XXE.