HALBORN

StaderLabs - Off-chain Security Review

WebApp Pentest

Prepared by: **Halborn**

Date of Engagement: **May 30th, 2023 - July 5th, 2023**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	05/30/2023	Carlos Polop
0.2	Document Updates	06/07/2023	Pau Borda Carrasco
0.3	Draft Review	06/14/2023	Carlos Polop
0.4	Draft Review	06/15/2023	Gabi Urrutia
1.0	Remediation Plan	07/03/2023	Pau Borda Carrasco
1.1	Remediation Plan Review	07/05/2023	Carlos Polop
1.2	Remediation Plan Review	07/05/2023	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Carlos Polop	Halborn	Carlos.Polop@halborn.com
Pau Borda Carrasco	Halborn	Pau.Carrasco@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

StaderLabs engaged Halborn to conduct a security audit on several Off-chain code repositories starting on May 30th, 2023 and ending on July 5th, 2023. The security assessment was scoped to the repositories provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided two weeks for the engagement and assigned a full-time security engineer to audit the security of the code. The security engineer is a penetration testing expert with advanced knowledge in web, recon, API, code review, discovery & infrastructure penetration testing.

The purpose of this audit is to:

- Ensure that the code doesn't contain vulnerabilities of potential security bus, and report the ones found.

In summary, Halborn identified some security risks that were mostly addressed by the StaderLabs team.

Two medium severity issues were identified: lack of resource and rate limiting, and log with low retention. These vulnerabilities could lead to an attacker to potential denial-of-service (DoS) attacks, and could affect StaderLabs team to not be able to investigate incidents in the past due to the low log retention period.

Lastly, five low severity issues were found: no filter for already requested presign msg, lack of API authorization methods, use of outdated packages, dependencies not pinned to exact version, and CORS misconfiguration. While these issues pose a lower risk, they still present potential security concerns.

Overall, it is essential to address these vulnerabilities in order to improve the security posture of the application.

1.3 SCOPE

The provided repositories in scope for this assessment are stader-labs/eth-offchain-infra, stader-labs/ethx-rewards-offchain, and stader-labs/ethx-verification-apis, and their commits are:

Commit ID: a7b990eba438ede76feb4de456638c751879973e
Commit ID: 2d807bf62d88e5dc548fcc8db96100e604e30cf5
Commit ID: 14ae14a7610782175dc69f8203776c5e3707133e

1.4 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual view of the code and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the program audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of programs and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Storing private keys and assets securely.
- Application Logic Flaws.
- Fuzzing of all input parameters.
- Areas where insufficient validation allows for hostile input.
- Research into architecture and purpose.
- Static Analysis of security for scoped program.
- Manual Assessment for discovering security vulnerabilities.
- Known vulnerabilities in 3rd party / OSS dependencies.

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.
1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW

3 - 1 - VERY LOW AND INFORMATIONAL

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	5	4

LIKELIHOOD

IMPACT

		(HAL-01)		
	(HAL-03) (HAL-04)	(HAL-02)		
	(HAL-05) (HAL-06) (HAL-07)			
(HAL-08) (HAL-09) (HAL-10) (HAL-11)				

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
LACK OF RESOURCES AND RATE LIMITING	Medium	SOLVED - 7/3/2023
LOG WITH LOW RETENTION	Medium	SOLVED - 7/3/2023
NO FILTER FOR ALREADY REQUESTED PRESIGN MSG	Low	SOLVED - 7/3/2023
LACK OF API GATEWAY AUTHORIZATION METHODS	Low	RISK ACCEPTED
USE OF OUTDATED PACKAGES	Low	RISK ACCEPTED
DEPENDENCIES NOT PINNED TO EXACT VERSION	Low	SOLVED - 7/3/2023
PERMISSIVE CROSS ORIGIN RESOURCE SHARING (CORS) POLICY	Low	SOLVED - 7/3/2023
OLD NODEJS VERSION IN LAMBDA	Informational	ACKNOWLEDGED
OLD PYTHON VERSION IN LAMBDA	Informational	ACKNOWLEDGED
INSECURE RANDOM NUMBER GENERATOR	Informational	SOLVED - 7/3/2023
PRESENCE OF TODO COMMENTS IN SOURCE CODE	Informational	SOLVED - 7/3/2023

FINDINGS & TECH DETAILS

3.1 (HAL-01) LACK OF RESOURCES AND RATE LIMITING - MEDIUM

Description:

API requests consume resources such as network, CPU, memory, storage, and budget. This vulnerability occurs when too many requests arrive simultaneously, and the API does not have enough compute resources to handle those requests or increases the price of the invoice if using a pay-per-use mode.

An attacker could exploit this vulnerability to overload the API by sending more requests than it can handle. As a result, the API becomes unavailable or unresponsive to new requests.

Code Location:

As shown below, the configuration of the API gateway does not contain any usage plan, which specifies who can access one or more deployed API stages and methods, and the rate at which they can be accessed.

In file eth-offchain-infra/lib/eth-offchain-infra-stack.ts:

Listing 1

```
// ######-START-###### Api gateway ######-START-######
    const api = new aws_apigateway.RestApi(this, `${namingPrefix}Apis`, {
      restApiName: `${namingPrefix}Apis`,
      deployOptions: {
        tracingEnabled: true,
        dataTraceEnabled: true,
        loggingLevel: aws_apigateway.MethodLoggingLevel.INFO,
        metricsEnabled: true,
        // Enable access logging
        accessLogDestination: new aws_apigateway.LogGroupLogDestination(
          logGroup
        ),
```

```
13          accessLogFormat:
14            aws_apigateway.AccessLogFormat.jsonWithStandardFields(),
15        },
16      });
17 // ######-END-###### Api gateway ######-END-######
```

NOTE: The lack of rate limit implementation in the API can cause overloads on the API as well as economic impact since StaderLabs is using an AWS REST API linked to lambdas and dynamoDB pay per use.

CVSS Vector:

- CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:H

Risk Level:

Likelihood - 3
Impact - 4

Recommendation:

This vulnerability is due to the application accepting requests from users at a given time without performing request throttling checks. It is recommended to follow the following best practices:

- Implement a limit on how often a client can call the API within a defined timeframe.
- Notify the client when the limit is exceeded by providing the limit number and the time the limit will be reset.

Remediation Plan:

SOLVED: StaderLabs fixed the issue by implementing `throttlingRateLimit` on the API Gateway configuration. Code shown below:

Listing 2

```
  // ######-START-###### Api gateway ######-START-######
    const api = new aws_apigateway.RestApi(this, `${namingPrefix}
↳ Apis`, {
      restApiName: `${namingPrefix}Apis`,
      deployOptions: {
        tracingEnabled: true,
        dataTraceEnabled: true,
        loggingLevel: aws_apigateway.MethodLoggingLevel.INFO,
        metricsEnabled: true,
        // Enable access logging
        accessLogDestination: new aws_apigateway.
↳ LogGroupLogDestination(
          logGroup
        ),
        accessLogFormat:
          aws_apigateway.AccessLogFormat.jsonWithStandardFields(),
        methodOptions: {
          "*/*": {
            throttlingBurstLimit: 500,
            throttlingRateLimit: 500,
          },
        },
      },
    });
    // ######-END-###### Api gateway ######-END-######
```

References:

CWE-770: Allocation of Resources Without Limits or Throttling

AWS Plan Usage Implementation

3.2 (HAL-02) LOG WITH LOW RETENTION - MEDIUM

Description:

In order for logs to be useful, they need to have a sufficient retention period to be able to investigate incidents in the past. For instance, an attack could happen in January, but it could go unnoticed until May. If the retention period was lower than 5 months, it would be very hard to investigate how the attack was initialized.

One year is a commonly agreed upon standard for long retention, meeting most regulations, including FISMA, HIPAA and PCI DSS. However, this will depend on the kind of log and your risk appetite. It's recommended to retain the logs at least for 90 days.

The following lambdas had a short retention period of 7 days:

- PublicKeyLambda
- SinglePresignLambda
- MultiplePresignLambda
- PresignSubmittedLambda
- PresignsSubmittedLambda
- CreateMerklesForElRewardsLambda
- ReadProofsByOperatorForElRewardsLambda
- InfoByCycleAndPoolId
- ConsensusLayerRewardsDistributorLambda
- DepositEthOverTargetWeightLambda
- ElRewardsVaultDistributorLambda
- UpdatePenaltiesReportedByRatedLambda
- UserRedemptionLambda
- ValidatorBatchDepositLambda
- MarkValidatorReadyToDepositLambda
    - PermissionedMarkValidatorReadyToDepositLambda
    - PermissionlessMarkValidatorReadyToDepositLambda
- NormalExitLambda

- ForcedExitLambda
- ExitValidatorEventTrackLambda
- ForceExitValidatorEventTrackLambda
- ValidatorPreDepositedOnBeaconChainEventTrackLambda

Code Location:

In file eth-offchain-infra-/lib/eth-offchain-infra-stack.ts:

Listing 3

```
const getBaseLambdaOptions = (
  env: Record<string, string>
): aws_lambda_nodejs.NodejsFunctionProps => ({
  awsSdkConnectionReuse: true,
  handler: "handler",
  memorySize: 512, // Can be increased to 512/1024 for more memory
↳ ->performance
  // ephemeralStorageSize: Size.gibibytes(1), // /tmp directory in
↳  MiB
  // reservedConcurrentExecutions ?? // Max concurrent executions
  environment: env,
  logRetention: aws_logs.RetentionDays.ONE_WEEK,
  tracing: aws_lambda.Tracing.ACTIVE,
  timeout: Duration.seconds(900),
  retryAttempts: 0,
  runtime: aws_lambda.Runtime.NODEJS_16_X,
});
```

Listing 4

```
// ######-START-###### Log groups ######-START-######
    const logGroup = new aws_logs.LogGroup(this, `${namingPrefix}
↳ LogGroup`, {
      logGroupName: `${namingPrefix}LogGroup`,
      retention: aws_logs.RetentionDays.ONE_WEEK,
    });
    // ######-END-###### Log groups ######-END-######
```

Risk Level:

Likelihood - 3
Impact - 3

Recommendation:

Increase the retention period of the log groups to something acceptable to StaderLabs risk appetite, it is recommended to retain the logs for at least 90 days.

Remediation Plan:

SOLVED: StaderLabs fixed the issue by implementing one year log retention. Code shown below:

In file eth-offchain-infra-/lib/eth-offchain-infra-stack.ts:

Listing 5

```
  environment: env,
  logRetention: aws_logs.RetentionDays.ONE_YEAR,
  tracing: aws_lambda.Tracing.ACTIVE,
  timeout: Duration.seconds(900),
  retryAttempts: 0,
```

Listing 6

```
// ######-START-###### Log groups ######-START-######
    const logGroup = new aws_logs.LogGroup(this, `${namingPrefix}
↳ LogGroup`, {
      logGroupName: `${namingPrefix}LogGroup`,
      retention: aws_logs.RetentionDays.ONE_YEAR,
    });
    // ######-END-###### Log groups ######-END-######
```

References:

- Altering CloudWatch log retention
- Log Retention guidelines

3.3 (HAL-03) NO FILTER FOR ALREADY REQUESTED PRESIGN MSG - LOW

Proof-of-concept:

The implementation of the validation and storage of presign message workflow works as below:

1. The node will start the process by calling an API endpoint, sending the validator pub key and a msg.
2. The lambda SinglePresignLambda or MultiplePresignLambda will be triggered in order to validate & save presign messages to the database.
3. The lambda will validate the presign message.
    - Will decrypt signature.
    - Verify that the validator & related info is valid.
    - Verify that the signature is valid by calling another API (ethx-verification-apis).
    - Save the presign message to the database.
4. When storing the information in the Dynamo DB, a PUT method is being used, which means that the operation will be a write or an overwrite (if the PK-SK exists).
5. Since the lambda is not checking if the information is already stored and the API, and is not restricted by origin or by throttle/rate limit, an attacker could initiate the node, start the process, get the value and update the value infinitely causing a cost over the AWS infrastructure involved.
6. It is true that the node will call PresignSubmittedLambda or PresignsSubmittedLambda to check if the message is already submitted, and the API will return a boolean. If the boolean is false, the node will not engage the save presign message workflow. However, the call can be done independently.

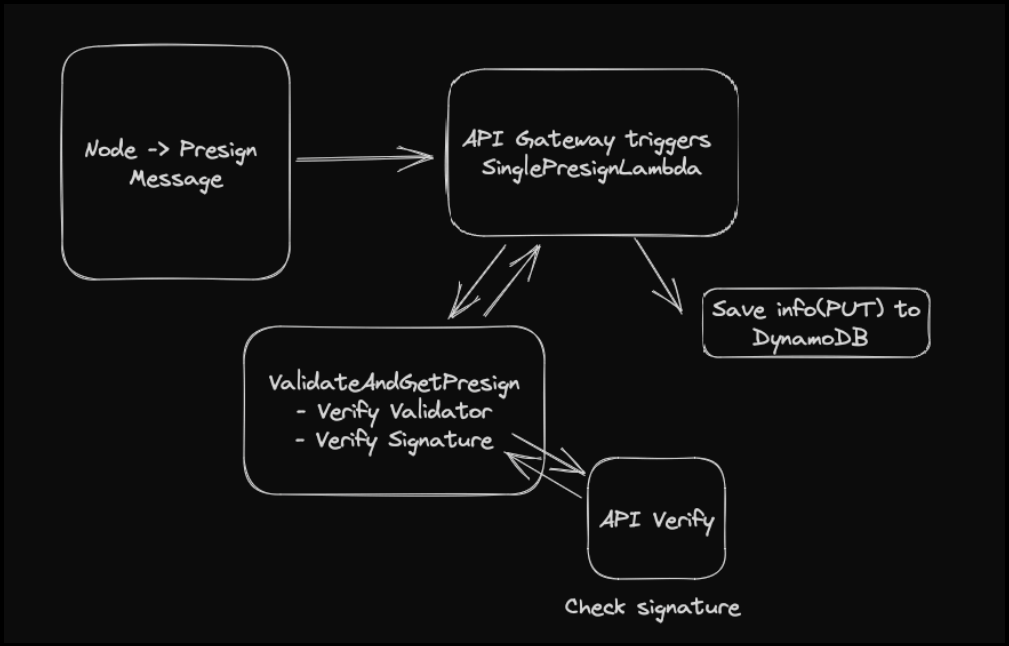

Figure 1: Workflow

CVSS Vector:

- CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:H

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

The recommended solution is:
- Check if the message exists in the DynamoDB before the storage. The code in PresignSubmittedLambda and PresignsSubmittedLambda do this check, the check should be in SinglePresignLambda and MultiplePresignLambda before starting the process of validation and storage of presign message.

Remediation Plan:

SOLVED: StaderLabs fixed the issue by implementing a filter before storing a presign msg to ensure that has not been requested before. Code shown below:

In file src/api/presign/savePresigns.ts:

Listing 7

```
1  const store = getPresignMsgStore();
2
3      const inputPresignMsg = presignMessageSchema.parse(JSON.parse(
↳ body));
4      const alreadyExists = await store.getById(
5        inputPresignMsg.validatorPublicKey
6      );
7      if (alreadyExists) {
8        return {
9          statusCode: 400,
10         body: JSON.stringify({
11           success: false,
12           error: `Presign message for validator ${inputPresignMsg.
↳ validatorPublicKey} already exists`,
13         }),
14       };
15     }
```

Listing 8

```
1        const alreadyExists = await store.getById(msg.
↳ validatorPublicKey);
2       if (alreadyExists) {
3         responses[msg.validatorPublicKey] = {
4           success: false,
5           error: `Presign message for validator ${msg.
↳ validatorPublicKey} already exists`,
6         };
7         continue;
8       }
```

3.4 (HAL-04) LACK OF API GATEWAY AUTHORIZATION METHODS - LOW

Description:

Authorization is not enabled for the API Gateway methods. This could allow unexpected access to unauthorized attackers or malicious insiders.

Code Location:

As shown below, the configuration of the API gateway does not contain any API KEY usage or authorization method, which specifies who can access one or more deployed API stages and methods.

In file eth-offchain-infra/lib/eth-offchain-infra-stack.ts:

Listing 9

```
// ######-START-###### Api gateway ######-START-######
    const api = new aws_apigateway.RestApi(this, `${namingPrefix}Apis`, {
      restApiName: `${namingPrefix}Apis`,
      deployOptions: {
        tracingEnabled: true,
        dataTraceEnabled: true,
        loggingLevel: aws_apigateway.MethodLoggingLevel.INFO,
        metricsEnabled: true,
        // Enable access logging
        accessLogDestination: new aws_apigateway.LogGroupLogDestination(
          logGroup
        ),
        accessLogFormat:
          aws_apigateway.AccessLogFormat.jsonWithStandardFields(),
      },
    });
// ######-END-###### Api gateway ######-END-######
```

Note: The lambda CreateMerklesForElRewardsLambda in the create.ts file,

which is the code that the lambda uses, it checks for an accessKey to create a Merkle tree for the rewards of a cycle.

CVSS Vector:

- CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:H

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

Use authorization method or require API Key.

Remediation Plan:

RISK ACCEPTED: StaderLabs accepted the risk of the issue. StaderLabs does not discard the possibility of fixing it in a future release.

References:

AWS API KEY implementation
API Gatweay Authorizers

3.5 (HAL-05) USE OF OUTDATED PACKAGES - LOW

Description:

The components of the platform use third-party dependencies to delegate handling of different kinds of operations. However, the dependencies bring forth an expected downside where the security posture of the real application is now resting on them. The in-scope repository contained some outdated packages, some with publicly disclosed vulnerabilities of moderate risk. Additionally, some of these packages were not maintained anymore, therefore should any new vulnerability be found, they might not receive the needed security patches.

Evidences:

```
pauborda@HAL-LT128-MBP eth-offchain-infra-a7b990eba438ede76feb4de456638c751879973e % npm audit
# npm audit report

fast-xml-parser  <4.2.4
Severity: high
fast-xml-parser vulnerable to Regex Injection via Doctype Entities - https://github.com/advisories/GHSA-6w63-h3fj-q4vw
fix available via `npm audit fix`
node_modules/fast-xml-parser
  @aws-sdk/client-sts  <=3.347.0
  Depends on vulnerable versions of fast-xml-parser
  node_modules/@aws-sdk/client-sts
    @aws-sdk/client-dynamodb  3.12.0 - 3.347.0
    Depends on vulnerable versions of @aws-sdk/client-sts
    node_modules/@aws-sdk/client-dynamodb
    @aws-sdk/client-secrets-manager  3.12.0 - 3.347.0
    Depends on vulnerable versions of @aws-sdk/client-sts
    node_modules/@aws-sdk/client-secrets-manager
    @aws-sdk/client-ssm  3.12.0 - 3.347.0
    Depends on vulnerable versions of @aws-sdk/client-sts
    node_modules/@aws-sdk/client-ssm

5 high severity vulnerabilities

pauborda@HAL-LT128-MBP eth-offchain-infra-a7b990eba438ede76feb4de456638c751879973e %
```

Figure 2: Npm Audit Vulnerabilities

CVSS Vector:

- CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:C/C:L/I:N/A:L

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

It is highly recommended to perform automated analysis of dependencies from the birth of the project to ensure they do not contain any security issues. The developers need to be aware of it and apply the required mitigation measures to secure the affected components.

Remediation Plan:

RISK ACCEPTED: StaderLabs accepted the risk of the issue. StaderLabs is using packages that they have used in production over the last one year and are stable. However, StaderLabs is planning to upgrade later next year.

3.6 (HAL-06) DEPENDENCIES NOT PINNED TO EXACT VERSION - LOW

Description:

The application contained some dependencies that were not pinned to an exact version, but were set to a supported version (x.x.x). This could potentially allow dependency attacks.

Code location:

Dependencies eth-infra-offchain:

Listing 10

```
"dependencies": {
  "@aws-sdk/client-dynamodb": "^3.44.0",
  "@aws-sdk/client-secrets-manager": "^3.171.0",
  "@aws-sdk/client-ssm": "^3.186.0",
  "@aws-sdk/lib-dynamodb": "^3.44.0",
  "aws-cdk-lib": "2.37.1",
  "aws-xray-sdk-core": "^3.3.4",
  "constructs": "^10.0.0",
  "ethers": "^5.7.2",
  "keccak256": "^1.0.6",
  "merkletreejs": "^0.3.9",
  "node-fetch": "^3.3.0",
  "serialize-error": "^11.0.0",
  "source-map-support": "^0.5.16",
  "zod": "^3.21.2"
}
```

Dependencies ethx-rewards-offchain:

Listing 11

```
"dependencies": {
        "@lodestar/api": "^1.5.1",
        "@lodestar/config": "^1.5.1",
```

```
        "@openzeppelin/merkle-tree": "^1.0.4",
        "bignumber.js": "^9.1.1",
        "bls-eth-wasm": "^1.0.6",
        "cross-fetch": "^3.1.5",
        "dotenv": "^16.0.3",
        "ethers": "^5.7.2",
        "isomorphic-fetch": "^3.0.0",
        "keccak256": "^1.0.6",
        "merkletreejs": "^0.3.9"
      },
```

Dependencies ethx-verification-apis:

Listing 12

```
aws-cdk-lib>=2.13.0
constructs>=10.0.0,<11.0.0
aws-cdk.aws-apigatewayv2-alpha
aws-cdk.aws-apigatewayv2-integrations-alpha
aws-cdk.aws-lambda-python-alpha
```

CVSS Vector:

- CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

Pinning dependencies to an exact version (=x.x.x) can reduce the chance of inadvertently introducing a malicious version of a dependency in the future.

Remediation Plan:

SOLVED: StaderLabs solved the issue by pinning all dependencies to exact versions. Code shown below:

Dependencies eth-infra-offchain:

Listing 13

```
"devDependencies": {
  "@typechain/ethers-v5": "10.2.0",
  "@types/aws-lambda": "8.10.111",
  "@types/jest": "26.0.24",
  "@types/node": "16.18.14",
  "aws-cdk-lib": "2.37.1",
  "constructs": "10.1.272",
  "jest": "26.6.3",
  "ts-jest": "26.5.6",
  "ts-node": "10.9.1",
  "typechain": "8.1.1",
  "typescript": "4.9.5"
},
"dependencies": {
  "@aws-sdk/client-dynamodb": "3.287.0",
  "@aws-sdk/client-kms": "3.354.0",
  "@aws-sdk/client-secrets-manager": "3.287.0",
  "@aws-sdk/client-ssm": "3.287.0",
  "@aws-sdk/lib-dynamodb": "3.287.0",
  "aws-cdk-lib": "2.37.1",
  "aws-xray-sdk-core": "3.4.1",
  "constructs": "10.1.272",
  "ethers": "5.7.2",
  "keccak256": "1.0.6",
  "merkletreejs": "0.3.9",
  "node-fetch": "3.3.0",
  "serialize-error": "11.0.0",
  "source-map-support": "0.5.21",
  "zod": "3.21.4"
}
```

Dependencies ethx-rewards-offchain:

Listing 14

```
"dependencies": {
    "bignumber.js": "9.1.1",
    "cross-fetch": "3.1.5",
    "dotenv": "16.0.3",
    "ethers": "5.7.2",
    "isomorphic-fetch": "3.0.0",
    "keccak256": "1.0.6",
    "merkletreejs": "0.3.9"
  },
  "devDependencies": {
    "@babel/helper-compilation-targets": "7.21.4",
    "@trivago/prettier-plugin-sort-imports": "4.1.1",
    "@typechain/ethers-v5": "10.2.0",
    "@types/isomorphic-fetch": "0.0.36",
    "@types/node": "14.18.38",
    "@typescript-eslint/eslint-plugin": "5.55.0",
    "@typescript-eslint/parser": "5.55.0",
    "eslint": "8.22.0",
    "eslint-config-airbnb-base": "15.0.0",
    "eslint-config-prettier": "8.7.0",
    "eslint-formatter-table": "7.32.1",
    "eslint-plugin-import": "2.27.5",
    "eslint-plugin-prettier": "4.2.1",
    "nodemon": "2.0.22",
    "prettier": "2.8.4",
    "typechain": "8.1.1",
    "typescript": "4.9.5"
  }
}
```

Dependencies ethx-verification-apis:

Listing 15

```
aws-cdk-lib==2.69.0
constructs==10.1.278
aws-cdk.aws-apigatewayv2-alpha==2.69.0a0
aws-cdk.aws-apigatewayv2-integrations-alpha==2.69.0a0
aws-cdk.aws-lambda-python-alpha==2.69.0a0
```

3.7 (HAL-07) PERMISSIVE CROSS ORIGIN RESOURCE SHARING (CORS) POLICY - LOW

Description:

The application seems to have a wildcard origin CORS configuration with authorization header allowed, which is very permissive. This feature primarily restricts reading information from another origin, where an origin is defined as a combination of protocol, port, and domain name.

Code Location:

In file eth-offchain-infra/src/constants/apiHeaders.ts:

Listing 16

```
/**
 * API_HEADERS for cors requests
 */
const API_HEADERS = {
  "Access-Control-Allow-Origin": "*",
  "Content-Type": "application/json",
  "Access-Control-Allow-Method": "GET,POST,OPTIONS",
};

export default API_HEADERS;
```

In file ethx-verification-apis/app.py:

Listing 17

```
# Create the HTTP API with CORS
        http_api = _apigw.HttpApi(
            self, "EthxVerificationApigw",
            cors_preflight=_apigw.CorsPreflightOptions(
                allow_methods=[_apigw.CorsHttpMethod.GET],
                allow_origins=["*"],
```

```
7                 max_age=Duration.days(10),
8             )
9         )
```

CVSS Score:

- CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:N/I:L/A:N

Risk Level:

Likelihood - 2
Impact - 2

Recommendation:

If the application does not rely on cross-origin requests, the CORS policy should be removed. This will result in the default Same-Origin Policy being enforced, preventing CSRF and similar attacks from reading sensitive response data. If the application requires cross-origin requests to function, the CORS policy should be carefully defined to only permit resource sharing with trusted origins.

Remediation Plan:

SOLVED: StaderLabs solved the issue by removing the CORS configuration. Code shown below:

In file eth-offchain-infra/src/constants/apiHeaders.ts the CORS configuration has been removed.

In file ethx-verification-apis/app.pythe CORS configuration has been removed.

Listing 18

```
# Create the HTTP API with CORS
        http_api = _apigw.HttpApi(
            self, "EthxVerificationApigw",
        )
```

3.8 (HAL-08) OLD NODEJS VERSION IN LAMBDA - INFORMATIONAL

Description:

A review of the function `getBaseLambdaOptions` in `/lib/eth-offchain-infra-stack.ts` file showed that an outdated version of **NodeJS** is being used. The latest iteration of the language includes many stability and feature improvements which may allow developers to implement more consistent, readable and secure code.

Code location:

Listing 19

```
const getBaseLambdaOptions = (
  env: Record<string, string>
): aws_lambda_nodejs.NodejsFunctionProps => ({
  awsSdkConnectionReuse: true,
  handler: "handler",
  memorySize: 512, // Can be increased to 512/1024 for more memory
↳ ->performance
  // ephemeralStorageSize: Size.gibibytes(1), // /tmp directory in
↳  MiB
  // reservedConcurrentExecutions ?? // Max concurrent executions
  environment: env,
  logRetention: aws_logs.RetentionDays.ONE_WEEK,
  tracing: aws_lambda.Tracing.ACTIVE,
  timeout: Duration.seconds(900),
  retryAttempts: 0,
  runtime: aws_lambda.Runtime.NODEJS_16_X,
});
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Halborn recommends that the StaderLabs team reviews the functionality introduced within the **NodeJs 18** of the language, if compatible, upgrade to the latest version.

Remediation Plan:

ACKNOWLEDGED: StaderLabs acknowledged the issue. StaderLabs does not discard the possibility of fixing it in a future release.

3.9 (HAL-09) OLD PYTHON VERSION IN LAMBDA - INFORMATIONAL

Description:

A review of ethx-verification-apis-/verify/template.yaml file showed that an outdated version of **Python** was being used. The latest iteration of the language includes many stability and feature improvements which may allow developers to implement more consistent, readable and secure code.

Code Location:

Listing 20

```
1  AWSTemplateFormatVersion: '2010-09-09'
2  Transform: AWS::Serverless-2016-10-31
3  Description: >
4    python3.9
5    Verify signature API
6  Globals:
7    Function:
8      Timeout: 900
9
10 Resources:
11   StartFunction:
12     Type: AWS::Serverless::Function
13     Properties:
14       PackageType: Image
15       Architectures:
16         - x86_64
17     Metadata:
18       Dockerfile: Dockerfile
19       DockerContext: .
20       DockerTag: python3.9-v1
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Halborn recommends that the StaderLabs team review the functionality introduced within the **Python** of the language. If compatible, upgrade to the latest version.

Remediation Plan:

ACKNOWLEDGED: StaderLabs acknowledged the issue. StaderLabs does not discard the possibility of implementing it in a future release.

3.10 (HAL-10) INSECURE RANDOM NUMBER GENERATOR - INFORMATIONAL

Description:

The function math.random made use of the math/rand package to generate random unsigned integers. This library is considered insecure due to weak random numbers generation.

Code Location:

In file ethx-rewards-offchain/src/utils/randomNumber.ts:

Listing 21

```
/**
 *
 * @param from - The lower bound of the range
 * @param to - The upper bound of the range
 * @returns - A random number between the specified range
 */
export function randomNumber(from: number, to: number): number {
  if (from >= to) {
    throw new Error("'from' value must be smaller than 'to' value.
");
  }

  const range = to - from + 1;
  return Math.floor(Math.random() * range) + from;
}
```

In file /ethx-rewards-offchain/src/withdraw-validator-list/permissionedValidatorsToExit.ts:

Listing 22

```
const randomOperatorIndex = Math.floor(
    Math.random() * operatorVsValidatorsList.length
  );
```

Recommendation:

It is recommended to make use of the crypto/rand package instead.

Remediation Plan:

SOLVED: StaderLabs solved the issue by implementing crypto module instead of the random module.

In file ethx-rewards-offchain/src/utils/randomNumber.ts:

Listing 23

```
export function mathRandom() {
  return crypto.getRandomValues(new Uint32Array(1))[0] / 2 ** 32;
}
...
  const range = to - from + 1;
  return Math.floor(mathRandom() * range) + from;
}
```

In file /ethx-rewards-offchain/src/withdraw-validator-list/permissionedValidatorsToExit.ts:

Listing 24

```
const randomOperatorIndex = Math.floor(
    mathRandom() * operatorVsValidatorsList.length
  );
```

3.11 (HAL-11) PRESENCE OF TODO COMMENTS IN SOURCE CODE - INFORMATIONAL

Description:

Open To-Dos could point to architecture or programming issues that still need to be resolved. These kinds of comments may also indicate areas of complexity or confusion for developers, which could provide value and insight to an attacker. Known weaknesses within the application requiring further development would be prime targets.

Evidences:

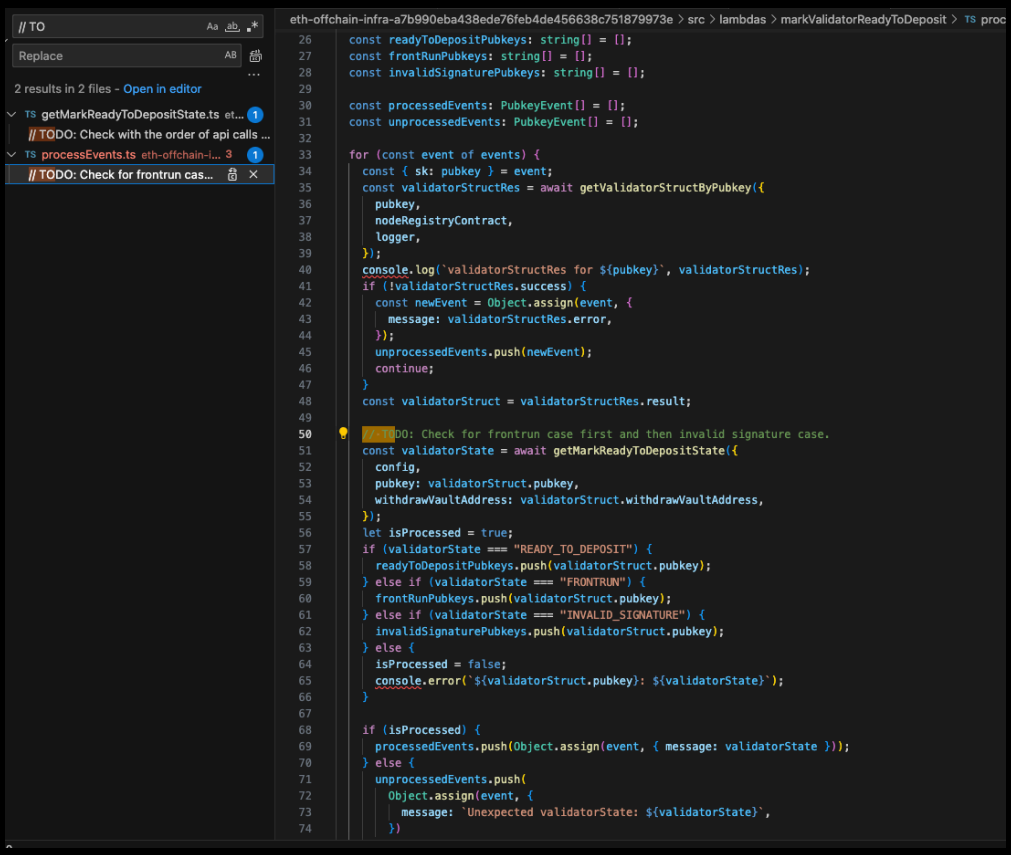

Figure 3: List of files containing “TODO” in the source code.

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

Consider resolving the TODOs before deploying code to a production context. Use an independent issue tracker or other project management software to track development tasks.

Remediation Plan:

SOLVED: StaderLabs solved the issue by removing the TODOs comments.

THANK YOU FOR CHOOSING

// HALBORN