AVA Labs

Avalanche SDK Audit Report

Prepared by: **Halborn**

Date of Engagement: **August 27th, 2021 - September 14th, 2021**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 3

CONTACTS 3

1 EXECUTIVE OVERVIEW 4

1.1 INTRODUCTION 5

1.2 AUDIT SUMMARY 5

1.3 TEST APPROACH & METHODOLOGY 6

RISK METHODOLOGY 6

1.4 SCOPE 8

2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 9

3 FINDINGS & TECH DETAILS 9

3.1 (HAL-01) UNFILTERED DATA CAN LEAD CROSS-SITE SCRIPTING - LOW 11

Description 11

Code Location 11

Risk Level 13

Recommendations 13

References 13

Remediation Plan 13

3.2 (HAL-02) USE OF UNFILTERED HOST AND PROTOCOL - LOW 14

Description 14

Code Location 15

Risk Level 15

Recommendations 16

Remediation Plan 16

3.3 (HAL-03) UNUSED VARIABLES - INFORMATIONAL 17

	Description	17
	Code Location	17
	Risk Level	18
	Recommendations	18
	Remediation Plan	18
4	FUZZ TESTING	19
	Description	20
4.1	PROPERTY BASED COVERAGE-GUIDED FUZZING	20
	Example Fuzzing Test Cases	21
	Fuzzing Progress	23
	Result	23
5	AUTOMATED TESTING	23
	Description	25
5.1	ESLint	26
5.2	Jest	28
5.3	NodeJSScan	29
5.4	LGTM	31

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	09/01/2021	Ataberk Yavuzer
0.9	Document Edits	09/14/2021	Ataberk Yavuzer
1.0	Final Draft	09/16/2021	Gabi Urrutia
1.1	Remediation Plan	09/23/2021	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
Ataberk Yavuzer	Halborn	Ataberk.Yavuzer@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

AVA Labs engaged Halborn to conduct a security assessment on their **Avalanche** Software Development Kit beginning on August 27th and ending on September 14th, 2021.

Avalanche SDK is a JavaScript Library for interfacing with the Avalanche Platform. It allows users to issue commands to the Avalanche node APIs.

The security assessment was scoped to the Github repository of Avalanche SDK. An audit of the security risk and implications regarding the changes introduced by the development team at AVA Labs prior to its production release shortly following the assessments deadline.

Though this security audit's outcome is satisfactory, only the most essential aspects were tested and verified to achieve objectives and deliverable set in the scope due to time and resource constraints. It is essential to note the use of the best practices for secure SDK development.

1.2 AUDIT SUMMARY

The team at Halborn was provided nearly four weeks for the engagement and assigned a full time security engineer to audit the security of the Avalanche SDK. The security engineer is blockchain and smart-contract security experts with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit to achieve the following:

- Ensure that Avalanche SDK functions are intended.
- Identify potential security issues with the Avalanche SDK.

In summary, Halborn identified few security risks that were addressed by AVA Labs Team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of the Avalanche SDK. While manual testing is recommended to uncover flaws in logic, process,and implementation; automated testing techniques help enhance coverage of structures 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:

- Research into architecture and purpose.
- Static Analysis of security for scoped SDK, and imported functions. (nodejsscan, Dependency-Check, eslint, LGTM)
- Manual Assessment for discovering security vulnerabilities on codebase.
- Ensuring correctness of the codebase. (jest)
- Dynamic Analysis on SDK functions and data types.
- Property based coverage-guided fuzzing. (jsfuzz)

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. It's quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that was 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

1.4 SCOPE

IN-SCOPE:

The security assessment was scoped to ava-labs/avalanchejs repository.

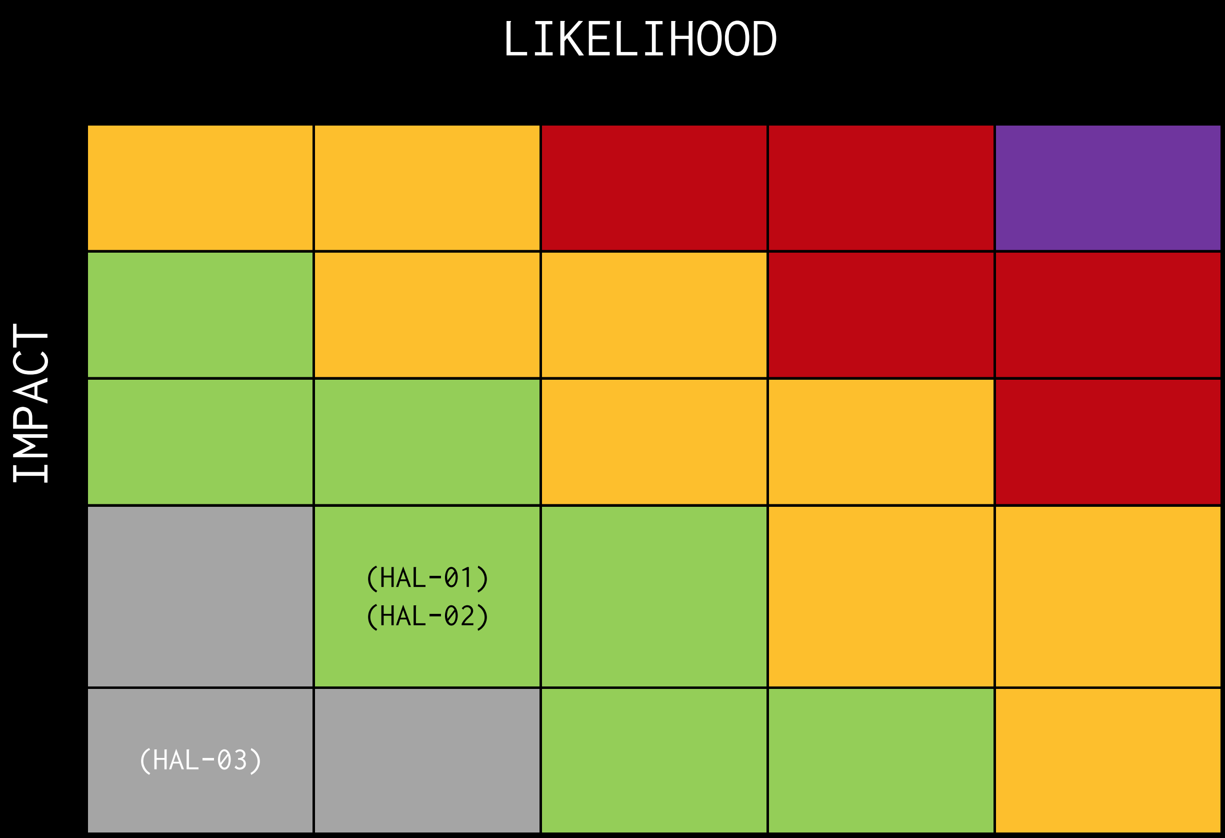
Commit ID: 8eddb4d35a0d7e2504aebb60f6812e378e6558c3

OUT-OF-SCOPE:

External libraries.

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	0	2	1



SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
UNFILTERED DATA CAN LEAD CROSS-SITE SCRIPTING	Low	SOLVED - 09/22/2021
USE OF UNFILTERED HOST AND PROTOCOL	Low	SOLVED - 09/22/2021
UNUSED VARIABLES	Informational	SOLVED - 09/22/2021

FINDINGS & TECH DETAILS

3.1 (HAL-01) UNFILTERED DATA CAN LEAD CROSS-SITE SCRIPTING - LOW

Description:

Cross-Site Scripting (XSS) attacks are a type of injection, in which malicious scripts are injected into otherwise benign and trusted websites. XSS attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. Flaws that allow these attacks to succeed are quite widespread and occur anywhere a web application uses input from a user within the output it generates without validating or encoding it.

During the test, it is seen that there are no input/output filtering on the serialization and deserialization methods. The SDK may deserialize malicious content. There is a possibility to trigger Cross-Site Scripting vulnerability by reflecting deserialized data without filtering.

Code Location:

Listing 1: src/utils/serialization.ts

```
deserialize(fields: object, encoding?: SerializedEncoding) {
   if (typeof fields["_typeName"] !== "string") {
     throw new TypeNameError(
       "Error - Serializable.deserialize: _typeName must be a string, found: " +
         typeof fields["_typeName"]
     )
   }
   if (fields["_typeName"] !== this._typeName) {
     throw new TypeNameError(
       "Error - Serializable.deserialize: _typeName mismatch -- expected: " +
         this._typeName +
         " -- received: " +
         fields["_typeName"]
     )
   }
```

```
    if (
      typeof fields["_typeID"] !== "undefined" &&
      fields["_typeID"] !== null
    ) {
      if (typeof fields["_typeID"] !== "number") {
        throw new TypeIdError(
          "Error - Serializable.deserialize: _typeID must be a number, found: " +
            typeof fields["_typeID"]
        )
      }
      if (fields["_typeID"] !== this._typeID) {
        throw new TypeIdError(
          "Error - Serializable.deserialize: _typeID mismatch -- expected: " +
            this._typeID +
            " -- received: " +
            fields["_typeID"]
        )
      }
    }
    if (
      typeof fields["_codecID"] !== "undefined" &&
      fields["_codecID"] !== null
    ) {
      if (typeof fields["_codecID"] !== "number") {
        throw new CodecIdError(
          "Error - Serializable.deserialize: _codecID must be a number, found: " +
            typeof fields["_codecID"]
        )
      }
      if (fields["_codecID"] !== this._codecID) {
        throw new CodecIdError(
          "Error - Serializable.deserialize: _codecID mismatch -- expected: " +
            this._codecID +
            " -- received: " +
            fields["_codecID"]
        )
      }
    }
  }
```

Risk Level:

Likelihood - 2
Impact - 2

Recommendations:

It is strongly recommended to filter the input and output fields that will be directed to the applications on the SDK. Generally, the process of filtering inputs and outputs is left to developers using the SDK. However, if this filtering is performed primarily on the SDK itself rather than the software using the SDK, this will create less workload for the developers and the usability of the Avalanchejs SDK will increase. Also, DOMPurify the powerful XSS sanitizer library that can be used on all inputs/outputs.

References:

OWASP Cross-Site Scripting
DOMPurify

Remediation Plan:

SOLVED: AVA Labs Team solved the issue by including the DOMPurify library to the mentioned serialization/deserialization functions on this report.

Pull Request #343
https://github.com/ava-labs/avalanchejs/pull/343/files

3.2 (HAL-02) USE OF UNFILTERED HOST AND PROTOCOL - LOW

Description:

During the tests, a function called setAddress, which determines the Base URL, was found on the Avalanche SDK. There are a number of variables that determine the address on this function. These variables are as follows: protocol, host and port. Although, the following line looks secure at first sight and concatenates specified protocol with host:

Listing 2

```
let url: string = `${protocol}://${host}``
```

it is possible to remove the host variable by appending malicious chars to the protocol variable. For example:

Listing 3

```
protocol = 'data:text/html,<script>alert(1)</script>%00';
host = 'avax.network';
```

will be converted into the following item:

Listing 4

```
url = 'data:text/html,<script>alert(1)</script>%00://avax.network';
```

As a result, browsers will parse these URL and malicious content will be triggered.

Additionally, this attack scenario can be achievable with the host parameter and it may lead to possible URL redirects.

Listing 5

```
protocol = 'https';
host = 'www.avax.network:foobar@malicioussite.com'; // malicious part -> :foobar@malicioussite.com
url = 'https://www.avax.network:foobar@malicioussite.com';
```

In this attack scenario, the first part www.avax.network which is actual host was used as username. The colon character (:) has been used to specify there will be basic authentication on URL. The foobar string has been used as password. The at sign (@) also specifies the basic authentication to the following site which is malicioussite.com.

However, Avalanche SDK users will specify this url variable on their projects. Therefore, it will not be possible for any user to manipulate the url value.

Code Location:

Listing 6: src/avalanche.ts (Lines 39,40,43)

```
39 setAddress = (host: string, port: number, protocol: string = "http") => {
40   this.host = host
41   this.port = port
42   this.protocol = protocol
43   let url: string = `${protocol}://${host}`
44   if (port != undefined && typeof port === "number" && port >= 0) {
45     url = `${url}:${port}`
46   }
47   this.url = url
48 }
```

Risk Level:

Likelihood - 2

Impact - 2

Recommendations:

The variables used on this function must filter characters such as @, :, %, < and > before using the data received from the user. In addition, it is recommended to use the **white-listing** method for protocol and host variables.

Remediation Plan:

SOLVED: AVA Labs Team solved the issue by removing symbols on host and protocol variables. Also, the whitelisting methodology has been applied to the protocol variable for forcing to use http or https as protocol.

Pull Request #341
https://github.com/ava-labs/avalanchejs/pull/341/files

3.3 (HAL-03) UNUSED VARIABLES - INFORMATIONAL

Description:

Variables that are declared and not used anywhere in the code are most likely an error due to incomplete refactoring. Such variables take up space in the code and can lead to confusion by readers. Unused variables can cause code clutter in the application.

In the static analysis part of the tests performed, it was determined that there were more than one unused variable on the Avalanche SDK.

Code Location:

Listing 7: src/apis/avm/api.ts (Lines 1104)

```
1102 sourceChain = bintools.cb58Decode(sourceChain)
1103 } else if (!(sourceChain instanceof Buffer)) {
1104 srcChain = bintools.cb58Encode(sourceChain)
```

Listing 8: src/apis/platformvm/api.ts (Lines 1217)

```
1215 sourceChain = bintools.cb58Decode(sourceChain)
1216 } else if (!(sourceChain instanceof Buffer)) {
1217 srcChain = bintools.cb58Encode(sourceChain)
```

Listing 9: src/apis/evm/utxos.ts (Lines 504)

```
503 if (typeof success === "undefined") {
504 outs = aad.getChangeOutputs()
```

Listing 10: src/utils/payload.ts (Lines 46)

```
44 getTypeID(payload: Buffer): number {
45 let offset: number = 0
46 const size: number = bintools.copyFrom(payload, offset, 4).
```

```
readUInt32BE(0)
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendations:

It is recommended to remove unused variables from the code.

Remediation Plan:

SOLVED: AVA Labs Team solved the issue by removing unused variables from the code base.

Pull Request #342
https://github.com/ava-labs/avalanchejs/pull/342/files

FUZZ TESTING

Description:

Fuzzing is a type of automated testing which continuously manipulates inputs to a program to find issues such as panics or bugs. These semi-random data mutations can discover new code coverage that existing unit tests may miss, and uncover edge case bugs which would otherwise go unnoticed. Since fuzzing can reach these edge cases, fuzz testing is particularly valuable for finding security exploits and vulnerabilities. The Halborn team worked in many fuzzing test cases.

4.1 PROPERTY BASED COVERAGE-GUIDED FUZZING

During the fuzzing tests, the Halborn team used the property based fuzzing with jsfuzz tool. Basically, all important functions were tested against crashes. To achieve this, multiple specific test cases were written for multiple .js files. Most important functions on the Avalanche SDK have passed the fuzzing tests.

Example Fuzzing Test Cases:

tests > fuzz_test.js > ...

```javascript
const avalanchejs = require('avalanche');

// const binTools = avalanchejs.BinTools.getInstance()

const myNetworkID = 12345
const avalanche = new avalanchejs.Avalanche("localhost", 9650, "http", myNetworkID)
const xchain = avalanche.XChain()
const myKeychain = xchain.keyChain()


function fuzz(buf) {
    try {
        (myKeychain.importKey(buf))
    } catch (e) {
        console.log(e)
        if (e.message.indexOf('false') !== -1) {

        }
        else {
            throw e;
        }
    }
}

module.exports = {
    fuzz
};

```

tests > fuzz_test2.js > fuzz

```javascript
const avalanchejs = require('avalanche');

const binTools = avalanchejs.BinTools.getInstance()

const myNetworkID = 12345
const avalanche = new avalanchejs.Avalanche("localhost", 9650, "http", myNetworkID)
const xchain = avalanche.XChain()
const myKeychain = xchain.keyChain()

function fuzz(buf) {
    try {
        binTools.cb58Decode(buf)
        //(myKeychain.getAddresses(`X-local${buf}`))
    } catch (e) {
        if (e.message.indexOf('ChecksumError:') !== -1
        || e.message.indexOf('invalid checksum')) {}
        else {
            throw e;
        }
    }
}

module.exports = {
    fuzz
};

```

FUZZ TESTING

tests > JS fuzz_test3.js > fuzz

```
  outputs.push(secpMintOutput)

  const unsignedTx = await xchain.buildCreateAssetTx(
    utxoSet,
    xAddressStrings,
    xAddressStrings,
    initialStates,
    fuzzingBuf, //assetName
    fuzzingBuf,     //assetSymbol
    denomination,
    outputs
  )
  const tx = unsignedTx.sign(xKeychain)
  const txid = await xchain.issueTx(tx)
  console.log(`Success! TXID: ${txid}`)
}

async function fuzz(buf){
    try{
        await resolver(buf)
    }
    catch(e){
        //console.log(e)
        if ((e.message.indexOf('TypeError:') !== -1)
        || (e.message.indexOf("Cannot read property") !== -1)
        || (e.message.indexOf('name is too short') !== -1)
        || (e.message.indexOf('Symbols may not exceed length of 4') !== -1)) {}
        else {
            throw e;
            }
        }
}
```

FUZZ TESTING

Fuzzing Progress:

```
#73 PULSE     cov: 14907 corp: 3 exec/s: 35 rss: 264.64 MB
#190 PULSE     cov: 14907 corp: 3 exec/s: 38 rss: 270.83 MB
#319 PULSE     cov: 14907 corp: 3 exec/s: 42 rss: 272.17 MB
#448 PULSE     cov: 14907 corp: 3 exec/s: 42 rss: 275.14 MB
#536 PULSE     cov: 14907 corp: 3 exec/s: 29 rss: 276.87 MB
#642 PULSE     cov: 14907 corp: 3 exec/s: 35 rss: 281.08 MB
#791 PULSE     cov: 14907 corp: 3 exec/s: 49 rss: 284.23 MB
#926 PULSE     cov: 14907 corp: 3 exec/s: 44 rss: 288.51 MB
#1058 PULSE     cov: 14907 corp: 3 exec/s: 44 rss: 287.74 MB
#1193 PULSE     cov: 14907 corp: 3 exec/s: 45 rss: 292.14 MB
#1317 PULSE     cov: 14907 corp: 3 exec/s: 41 rss: 300.37 MB
#1454 PULSE     cov: 14907 corp: 3 exec/s: 45 rss: 302.54 MB
#1602 PULSE     cov: 14907 corp: 3 exec/s: 49 rss: 255.3 MB
#1734 PULSE     cov: 14907 corp: 3 exec/s: 43 rss: 234.87 MB
#1850 PULSE     cov: 14907 corp: 3 exec/s: 38 rss: 234.96 MB
#1976 PULSE     cov: 14907 corp: 3 exec/s: 41 rss: 223.06 MB
#2103 PULSE     cov: 14907 corp: 3 exec/s: 42 rss: 216.63 MB
#2229 PULSE     cov: 14907 corp: 3 exec/s: 41 rss: 217.92 MB
#2366 PULSE     cov: 14907 corp: 3 exec/s: 45 rss: 217.76 MB
#2498 PULSE     cov: 14907 corp: 3 exec/s: 43 rss: 217.78 MB
#2655 PULSE     cov: 14907 corp: 3 exec/s: 52 rss: 217.9 MB
#2798 PULSE     cov: 14907 corp: 3 exec/s: 47 rss: 219.64 MB
#2948 PULSE     cov: 14907 corp: 3 exec/s: 49 rss: 219.82 MB
#3075 PULSE     cov: 14907 corp: 3 exec/s: 42 rss: 220.78 MB
#3206 PULSE     cov: 14907 corp: 3 exec/s: 43 rss: 221.17 MB
#3333 PULSE     cov: 14907 corp: 3 exec/s: 42 rss: 220.03 MB
#3462 PULSE     cov: 14907 corp: 3 exec/s: 42 rss: 220.67 MB
#3589 PULSE     cov: 14907 corp: 3 exec/s: 42 rss: 220.36 MB
#3720 PULSE     cov: 14907 corp: 3 exec/s: 43 rss: 224.15 MB
#3857 PULSE     cov: 14907 corp: 3 exec/s: 45 rss: 225.08 MB
#4004 PULSE     cov: 14907 corp: 3 exec/s: 49 rss: 224.72 MB
#4151 PULSE     cov: 14907 corp: 3 exec/s: 49 rss: 223.68 MB
#4294 PULSE     cov: 14907 corp: 3 exec/s: 47 rss: 225.05 MB
#4426 PULSE     cov: 14907 corp: 3 exec/s: 43 rss: 226.62 MB
#4529 PULSE     cov: 14907 corp: 3 exec/s: 34 rss: 226.08 MB
#4645 PULSE     cov: 14907 corp: 3 exec/s: 38 rss: 228.09 MB
#4788 PULSE     cov: 14907 corp: 3 exec/s: 47 rss: 226.07 MB
#4905 PULSE     cov: 14907 corp: 3 exec/s: 38 rss: 226.54 MB
#5031 PULSE     cov: 14907 corp: 3 exec/s: 41 rss: 230.04 MB
#5152 PULSE     cov: 14907 corp: 3 exec/s: 40 rss: 232.42 MB
#5286 PULSE     cov: 14907 corp: 3 exec/s: 44 rss: 232.35 MB
#5418 PULSE     cov: 14907 corp: 3 exec/s: 43 rss: 234.83 MB
#5546 PULSE     cov: 14907 corp: 3 exec/s: 42 rss: 232.5 MB
#5676 PULSE     cov: 14907 corp: 3 exec/s: 43 rss: 233.56 MB
#5803 PULSE     cov: 14907 corp: 3 exec/s: 42 rss: 238.67 MB
#5929 PULSE     cov: 14907 corp: 3 exec/s: 41 rss: 240.16 MB
#6064 PULSE     cov: 14907 corp: 3 exec/s: 44 rss: 240.42 MB
#6210 PULSE     cov: 14907 corp: 3 exec/s: 48 rss: 236.31 MB
#6336 PULSE     cov: 14907 corp: 3 exec/s: 41 rss: 238.62 MB
#6458 PULSE     cov: 14907 corp: 3 exec/s: 40 rss: 234.24 MB
#6581 PULSE     cov: 14907 corp: 3 exec/s: 41 rss: 230.55 MB
```

Result:

As a result of the fuzzing tests performed at the input points of the SDK, it has been seen that **the application is safe against many fuzzing tests**. During the fuzzing process, **no crash has been happened**.

AUTOMATED TESTING

Description:

Halborn used automated testing techniques to enhance coverage of certain areas of the scoped items. Among the tools used ESLint, NodeJSScan, Jest and LGTM. ESLint tries to statically analyzes the code to quickly find problems related with JavaScript projects. NodeJSScan contains multiple security assessment tools in it such as Dependency-Scan of OWASP. Jest tool is already appended by AVA Labs Team on the project to state if functions working proper. Lastly, LGTM analyzes the entire history of a project, so you can see how your alerts have changed over time, and which specific events or commits had the biggest impact on your code quality. Also, it checks the codebase to find security vulnerabilities. After Halborn verified the codebase in the repository and was able to work on it correctly, all of these tools were run on the all-scoped structures. These tools can statically verify security related issues across the entire codebase.

5.1 ESLint

According to the following screenshots, there could be multiple improvements on the codebase. It has been decided to not to put these issues on the report in details because these issues do not pose any security risk.

These improvements are listed below:

1. Removing Unused Variables
2. Adding return type to functions
3. Changing “any” keyword to specific data type

```
/home/ziion/Desktop/clients/AVA/files/avalanchejs/src/apis/avm/api.ts
    93:30  warning  Generic Object Injection Sink                    security/detect-object-injection
    95:32  warning  Generic Object Injection Sink                    security/detect-object-injection
   135:14  warning  Generic Object Injection Sink                    security/detect-object-injection
   137:27  warning  Generic Object Injection Sink                    security/detect-object-injection
  1820:20  warning  Generic Object Injection Sink                    security/detect-object-injection
  1822:38  warning  Generic Object Injection Sink                    security/detect-object-injection
  1829:22  warning  Generic Object Injection Sink                    security/detect-object-injection
  1834:15  warning  Generic Object Injection Sink                    security/detect-object-injection
  1861:54  warning  Generic Object Injection Sink                    security/detect-object-injection
  1862:25  warning  Variable Assigned to Object Injection Sink       security/detect-object-injection
  1862:25  warning  Generic Object Injection Sink                    security/detect-object-injection

/home/ziion/Desktop/clients/AVA/files/avalanchejs/src/apis/avm/basetx.ts
  161:9   warning  Generic Object Injection Sink  security/detect-object-injection
  163:33  warning  Generic Object Injection Sink  security/detect-object-injection
  165:44  warning  Generic Object Injection Sink  security/detect-object-injection

/home/ziion/Desktop/clients/AVA/files/avalanchejs/src/apis/avm/exporttx.ts
  118:10  warning  Generic Object Injection Sink  security/detect-object-injection
  175:17  warning  Generic Object Injection Sink  security/detect-object-injection
  214:15  warning  Generic Object Injection Sink  security/detect-object-injection

/home/ziion/Desktop/clients/AVA/files/avalanchejs/src/apis/avm/importtx.ts
  148:17  warning  Generic Object Injection Sink  security/detect-object-injection
  181:9   warning  Generic Object Injection Sink  security/detect-object-injection
  183:33  warning  Generic Object Injection Sink  security/detect-object-injection
  185:44  warning  Generic Object Injection Sink  security/detect-object-injection
  220:15  warning  Generic Object Injection Sink  security/detect-object-injection

/home/ziion/Desktop/clients/AVA/files/avalanchejs/src/apis/avm/initialstates.ts
   28:7   warning  Generic Object Injection Sink               security/detect-object-injection
   28:23  warning  Generic Object Injection Sink               security/detect-object-injection
   41:7   warning  Generic Object Injection Sink               security/detect-object-injection
   41:22  warning  Generic Object Injection Sink               security/detect-object-injection
   59:7   warning  Generic Object Injection Sink               security/detect-object-injection
   61:5   warning  Generic Object Injection Sink               security/detect-object-injection
   73:7   warning  Generic Object Injection Sink               security/detect-object-injection
   84:9   warning  Generic Object Injection Sink               security/detect-object-injection
  100:28  warning  Variable Assigned to Object Injection Sink  security/detect-object-injection
  104:28  warning  Generic Object Injection Sink               security/detect-object-injection
  110:31  warning  Generic Object Injection Sink               security/detect-object-injection
  112:19  warning  Generic Object Injection Sink               security/detect-object-injection

/home/ziion/Desktop/clients/AVA/files/avalanchejs/src/apis/avm/keychain.ts
  144:20  warning  Generic Object Injection Sink  security/detect-object-injection
  152:20  warning  Generic Object Injection Sink  security/detect-object-injection
```

Figure 1: ESLint Results - 1

```
/home/ziion/Desktop/clients/AVA/files/avalanchejs/src/apis/avm/utxos.ts
  155:7   warning  Generic Object Injection Sink                security/detect-object-injection
  156:7   warning  Generic Object Injection Sink                security/detect-object-injection
  156:40  warning  Function Call Object Injection Sink          security/detect-object-injection
  167:26  warning  Generic Object Injection Sink                security/detect-object-injection
  174:9   warning  Generic Object Injection Sink                security/detect-object-injection
  175:11  warning  Function Call Object Injection Sink          security/detect-object-injection
  175:11  warning  Generic Object Injection Sink                security/detect-object-injection
  181:7   warning  Generic Object Injection Sink                security/detect-object-injection
  232:23  warning  Variable Assigned to Object Injection Sink   security/detect-object-injection
  243:11  warning  Generic Object Injection Sink                security/detect-object-injection
  257:52  warning  Function Call Object Injection Sink          security/detect-object-injection
  262:41  warning  Generic Object Injection Sink                security/detect-object-injection
  265:52  warning  Function Call Object Injection Sink          security/detect-object-injection
  294:32  warning  Generic Object Injection Sink                security/detect-object-injection
  295:26  warning  Generic Object Injection Sink                security/detect-object-injection
  298:11  warning  Function Call Object Injection Sink          security/detect-object-injection
  305:11  warning  Generic Object Injection Sink                security/detect-object-injection
  310:26  warning  Generic Object Injection Sink                security/detect-object-injection
  313:11  warning  Function Call Object Injection Sink          security/detect-object-injection
  318:11  warning  Generic Object Injection Sink                security/detect-object-injection
  475:13  warning  Generic Object Injection Sink                security/detect-object-injection
  476:34  warning  Function Call Object Injection Sink          security/detect-object-injection
  567:43  warning  Function Call Object Injection Sink          security/detect-object-injection
  574:35  warning  Function Call Object Injection Sink          security/detect-object-injection
  649:9   warning  Generic Object Injection Sink                security/detect-object-injection
  651:9   warning  Generic Object Injection Sink                security/detect-object-injection
  732:37  warning  Function Call Object Injection Sink          security/detect-object-injection
  738:33  warning  Function Call Object Injection Sink          security/detect-object-injection
  745:47  warning  Function Call Object Injection Sink          security/detect-object-injection
  819:39  warning  Function Call Object Injection Sink          security/detect-object-injection
  834:47  warning  Function Call Object Injection Sink          security/detect-object-injection
  839:45  warning  Generic Object Injection Sink                security/detect-object-injection
  842:33  warning  Function Call Object Injection Sink          security/detect-object-injection
  847:10  warning  Generic Object Injection Sink                security/detect-object-injection
  908:26  warning  Variable Assigned to Object Injection Sink   security/detect-object-injection
  942:50  warning  Function Call Object Injection Sink          security/detect-object-injection
  947:37  warning  Generic Object Injection Sink                security/detect-object-injection
  950:48  warning  Function Call Object Injection Sink          security/detect-object-injection
```

Figure 2: ESLint Results - 2

```
/home/ziion/Desktop/clients/AVA/files/avalanchejs/src/index.ts
  125:20  warning  Generic Object Injection Sink  security/detect-object-injection
  136:20  warning  Generic Object Injection Sink  security/detect-object-injection

/home/ziion/Desktop/clients/AVA/files/avalanchejs/src/utils/base58.ts
  123:11  warning  Generic Object Injection Sink  security/detect-object-injection

/home/ziion/Desktop/clients/AVA/files/avalanchejs/src/utils/bintools.ts
  203:7   warning  Generic Object Injection Sink  security/detect-object-injection
  203:17  warning  Generic Object Injection Sink  security/detect-object-injection
  216:7   warning  Generic Object Injection Sink  security/detect-object-injection
  216:16  warning  Generic Object Injection Sink  security/detect-object-injection

/home/ziion/Desktop/clients/AVA/files/avalanchejs/src/utils/helperfunctions.ts
  25:12  warning  Generic Object Injection Sink  security/detect-object-injection
  27:12  warning  Generic Object Injection Sink  security/detect-object-injection

/home/ziion/Desktop/clients/AVA/files/avalanchejs/src/utils/mnemonic.ts
  40:14  warning  Generic Object Injection Sink  security/detect-object-injection

/home/ziion/Desktop/clients/AVA/files/avalanchejs/src/utils/payload.ts
  65:12  warning  Generic Object Injection Sink  security/detect-object-injection

✖ 251 problems (0 errors, 251 warnings)

Done in 11.09s.
```

Figure 3: ESLint Results - 3

AUTOMATED TESTING

5.2 Jest

As a result of the tests carried out with **Jest**, it has been decided the project has average code coverage. Also, completed tests have shown that some functions run slower than others.

```
PASS  tests/apis/evm/outputs.test.ts
  Inputs
    ✓ EVMOutput comparator (11 ms)

PASS  tests/apis/platformvm/outputs.test.ts
  Outputs
    SECPTransferOutput
      ✓ SelectOutputClass (22 ms)
      ✓ comparator (1 ms)
      ✓ SECPTransferOutput (16 ms)

PASS  tests/apis/avm/types.test.ts
  UnixNow
    ✓ Does it return the right time? (1 ms)

PASS  tests/apis/avm/genesisasset.test.ts
  AVM
    ✓ GenesisAsset (8 ms)

PASS  tests/apis/auth/api.test.ts
  Auth
    ✓ newToken (1 ms)
    ✓ revokeToken (1 ms)
    ✓ changePassword (1 ms)

PASS  tests/apis/avm/genesisdata.test.ts
  AVM
    ✓ GenesisData (3 ms)

PASS  tests/apis/metrics/api.test.ts
  Metrics
    ✓ getMetrics (3 ms)

PASS  tests/apis/health/api.test.ts
  Health
    ✓ getLiveness  (2 ms)

PASS  tests/utils/pubsub.test.ts
  PubSub
    ✓ newSet (1 ms)
    ✓ newBloom
    ✓ addAddresses (1 ms)

PASS  tests/utils/db.test.ts
  DB
    ✓ instantiate singletone (1 ms)

Test Suites: 36 passed, 36 total
Tests:       470 passed, 470 total
Snapshots:   0 total
Time:        150.405 s
Ran all test suites.
```

Figure 4: Jest

5.3 NodeJSScan

As a result of the scans completed with NodeJSScan, many tests were carried out and some issue outputs were produced as a result of these tests. These generated issues were analyzed manually. It has been decided that these issues are not real security vulnerabilities and they do not risk security of the Wallet SDK. As one of the important security assessments of NodeJSScan that is OWASP Dependency-Scan, it is seen that used libraries are safe.

Figure 5: NodeJSScan Results

AUTOMATED TESTING

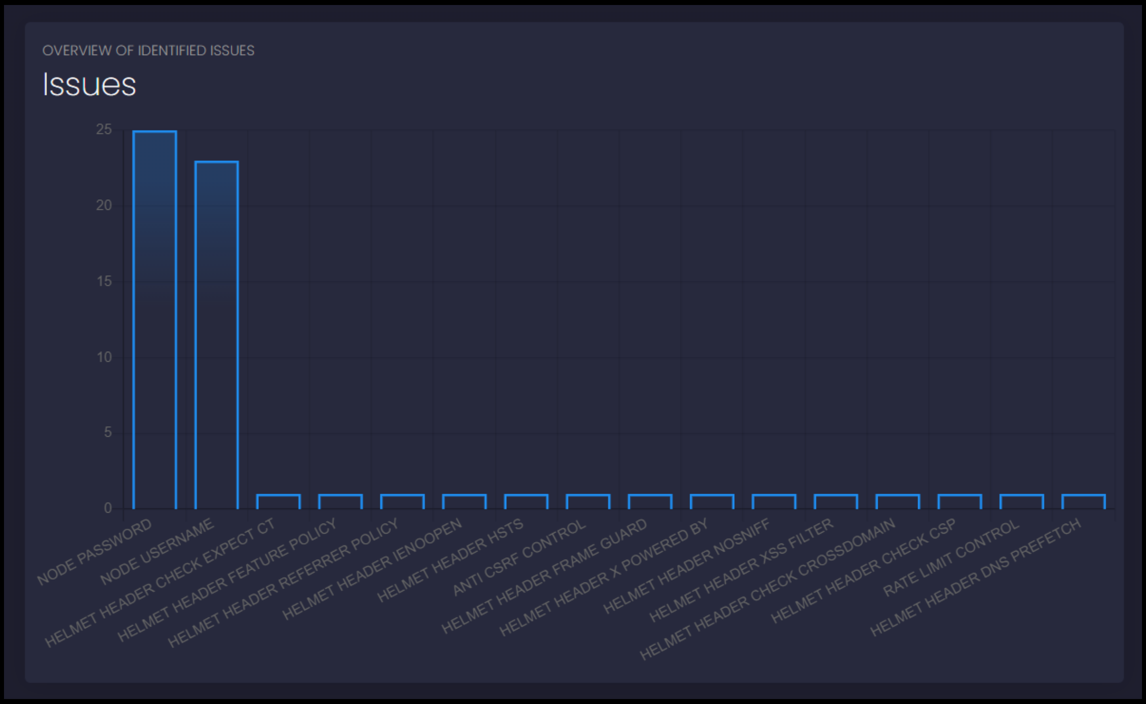


Figure 6: NodeJSScan Results

5.4 LGTM

The project has been security scanned with LGTM and results similar to ESLint outputs have been achieved. As a result of the scans with this tool, no security vulnerabilities were found. During the tests with this tool, only **Unused Variables** findings were reached. The reason for scanning the project publicly with LGTM is that the Github repository is a public repository.

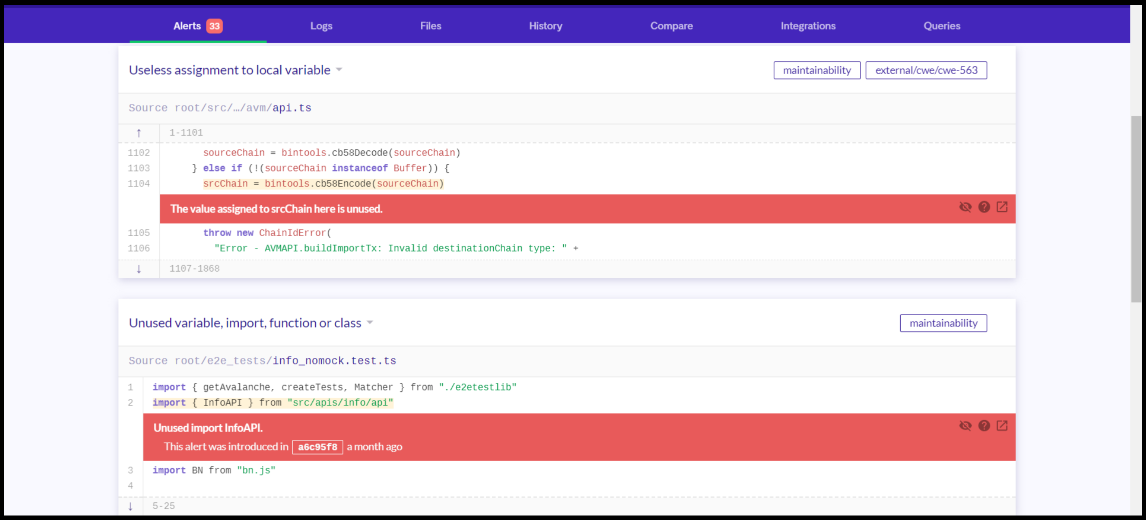


Figure 7: LGTM Scan

Scan results can be accessed via the link below:

Scan Results - LGTM

AUTOMATED TESTING

THANK YOU FOR CHOOSING

HALBORN