<u>Web Wallet</u>

The audit was conducted by TECHFUND in the first week of November. The code used was made available as a zip file via a private communication channel.

Vulnerabilities

High	1
Medium	1
Low	2
Note	1

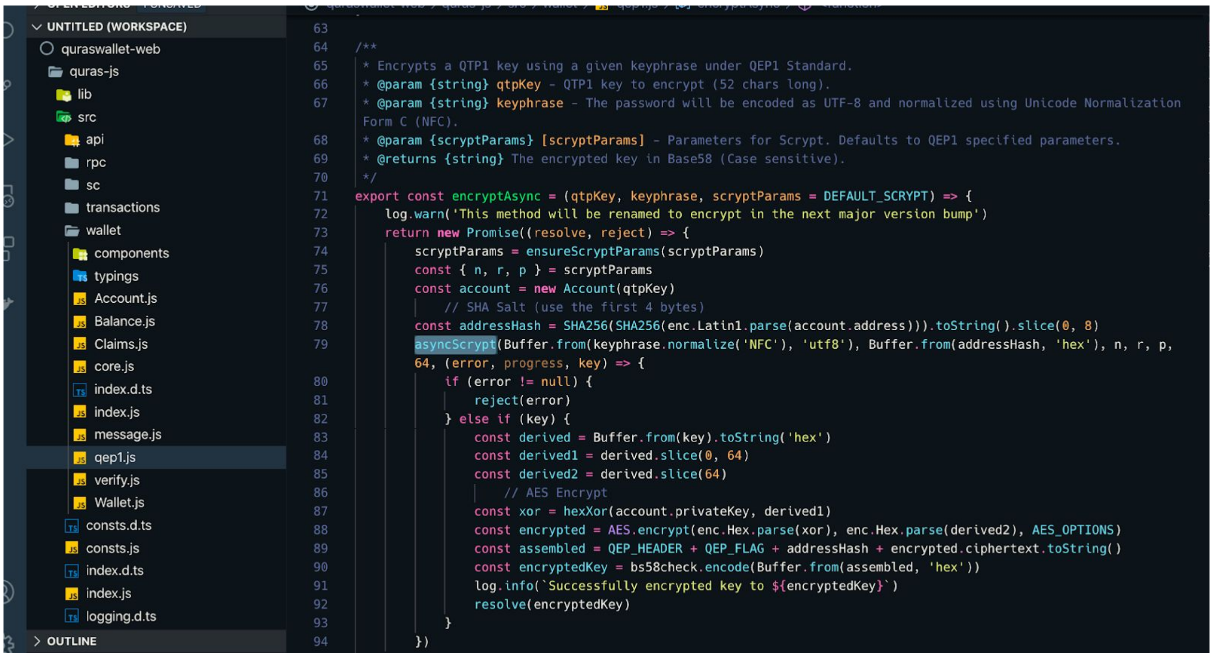
We found above vulnerabilities in the code that have been described below.

1. **Error on passwords with more than 64 bytes**
   HIGH

Quras-js > wallet > Wallet

https://github.com/ricmoo/scrypt-js/issues/11

There is a major issue in the encryption library used,
“When password is more than 64bytes, PBKDF2_HMAC_SHA256_OneIter function runs SHA256 for password.SHA256 expects that the argument is Array but the user uses Buffer.”



```javascript

/**
 * Encrypts a QTP1 key using a given keyphrase under QEP1 Standard.
 * @param {string} qtpKey - QTP1 key to encrypt (52 chars long).
 * @param {string} keyphrase - The password will be encoded as UTF-8 and normalized using Unicode Normalization Form C (NFC).
 * @param {scryptParams} [scryptParams] - Parameters for Scrypt. Defaults to QEP1 specified parameters.
 * @returns {string} The encrypted key in Base58 (Case sensitive).
 */
export const encryptAsync = (qtpKey, keyphrase, scryptParams = DEFAULT_SCRYPT) => {
    log.warn('This method will be renamed to encrypt in the next major version bump')
    return new Promise((resolve, reject) => {
        scryptParams = ensureScryptParams(scryptParams)
        const { n, r, p } = scryptParams
        const account = new Account(qtpKey)
            // SHA Salt (use the first 4 bytes)
        const addressHash = SHA256(SHA256(enc.Latin1.parse(account.address))).toString().slice(0, 8)
        asyncScrypt(Buffer.from(keyphrase.normalize('NFC'), 'utf8'), Buffer.from(addressHash, 'hex'), n, r, p, 64, (error, progress, key) => {
            if (error != null) {
                reject(error)
            } else if (key) {
                const derived = Buffer.from(key).toString('hex')
                const derived1 = derived.slice(0, 64)
                const derived2 = derived.slice(64)
                    // AES Encrypt
                const xor = hexXor(account.privateKey, derived1)
                const encrypted = AES.encrypt(enc.Hex.parse(xor), enc.Hex.parse(derived2), AES_OPTIONS)
                const assembled = QEP_HEADER + QEP_FLAG + addressHash + encrypted.ciphertext.toString()
                const encryptedKey = bs58check.encode(Buffer.from(assembled, 'hex'))
                log.info(`Successfully encrypted key to ${encryptedKey}`)
                resolve(encryptedKey)
            }
        })
```

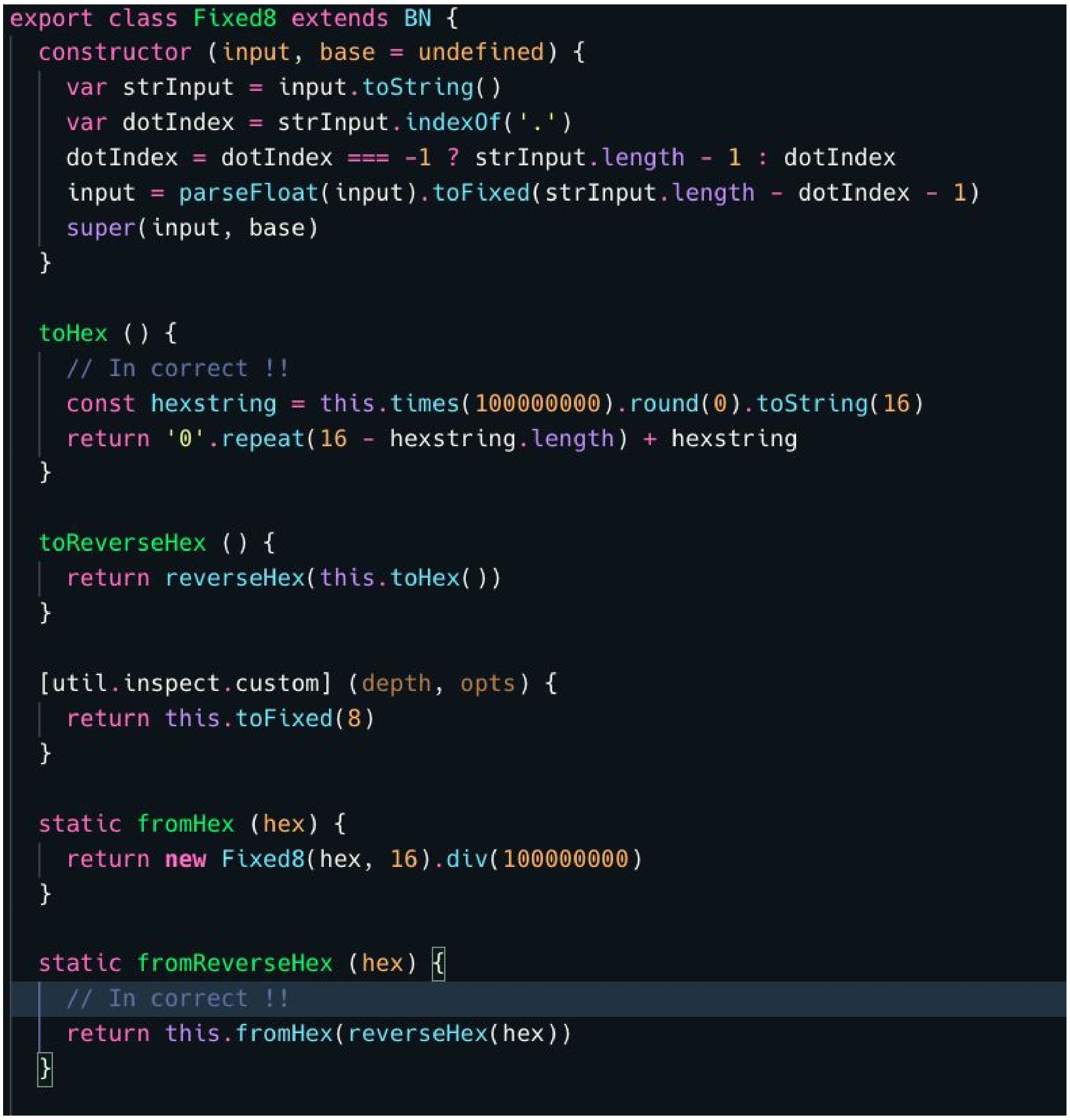
The project depends on an external library for scrypt, although this is not an issue on its own, we would highly recommend to use the inbuilt nodejs Crypto module for scrypt implementation. Or Atleast update it to the latest version as the version used is now depreciated.

2. **Issue in some decodings of Fixed8 String**
   MEDIUM

quras-js > src > utils.js

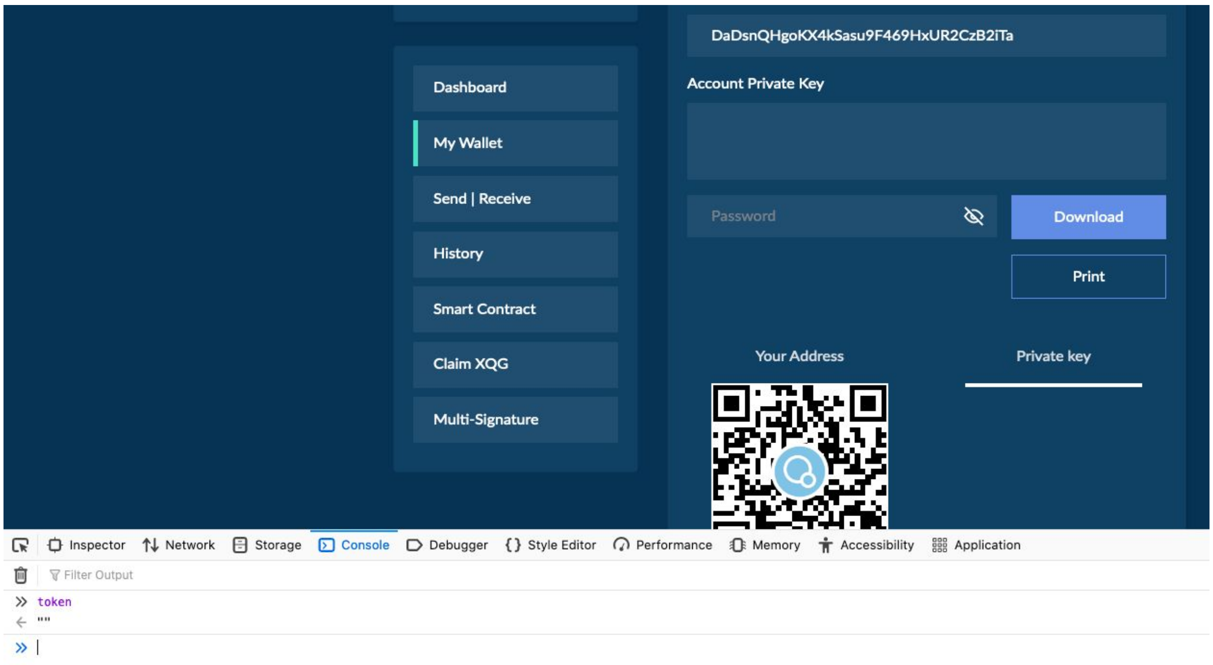
Decoding of fixed8 hex strings will fail for negative input.

Also :
quras.u.fixed82num("ffffffffffffffff") // will fail !



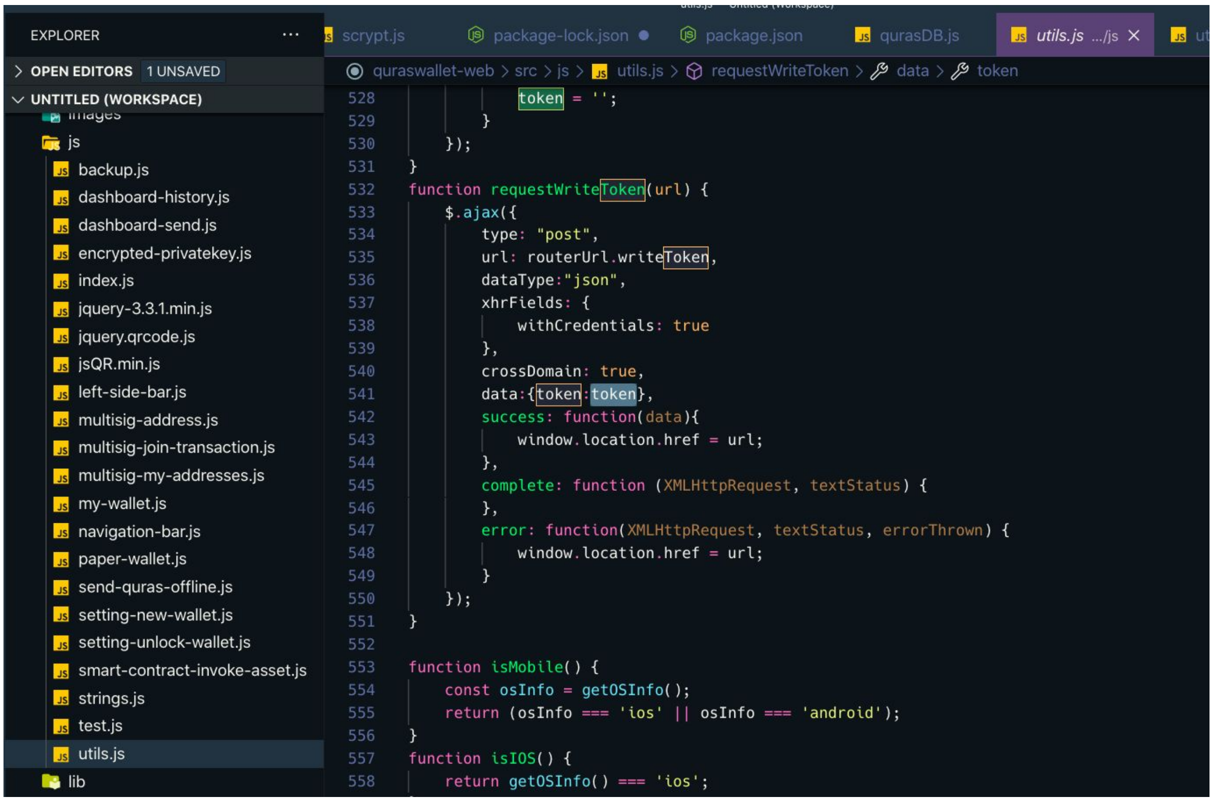
```
export class Fixed8 extends BN {
  constructor (input, base = undefined) {
    var strInput = input.toString()
    var dotIndex = strInput.indexOf('.')
    dotIndex = dotIndex === -1 ? strInput.length - 1 : dotIndex
    input = parseFloat(input).toFixed(strInput.length - dotIndex - 1)
    super(input, base)
  }

  toHex () {
    // In correct !!
    const hexstring = this.times(100000000).round(0).toString(16)
    return '0'.repeat(16 - hexstring.length) + hexstring
  }

  toReverseHex () {
    return reverseHex(this.toHex())
  }

  [util.inspect.custom] (depth, opts) {
    return this.toFixed(8)
  }

  static fromHex (hex) {
    return new Fixed8(hex, 16).div(100000000)
  }

  static fromReverseHex (hex) {
    // In correct !!
    return this.fromHex(reverseHex(hex))
  }
```

3. Token is not set properly and leads to failed API calls LOW



DaDsnQHgoKX4kSasu9F469HxUR2CzB2iTa

```
>> token
<- ""
>>
```



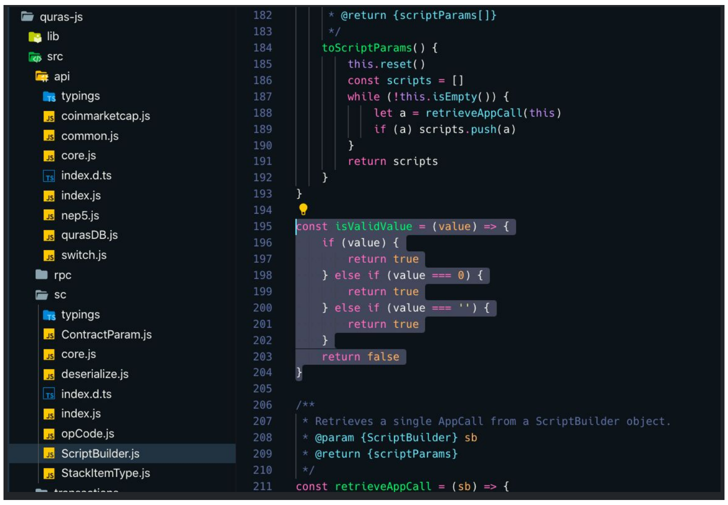
```
                token = '';
            }
        });
    }
    function requestWriteToken(url) {
        $.ajax({
            type: "post",
            url: routerUrl.writeToken,
            dataType:"json",
            xhrFields: {
                withCredentials: true
            },
            crossDomain: true,
            data:{token:token},
            success: function(data){
                window.location.href = url;
            },
            complete: function (XMLHttpRequest, textStatus) {
            },
            error: function(XMLHttpRequest, textStatus, errorThrown) {
                window.location.href = url;
            }
        });
    }

    function isMobile() {
        const osInfo = getOSInfo();
        return (osInfo === 'ios' || osInfo === 'android');
    }
    function isIOS() {
        return getOSInfo() === 'ios';
```

4. False is not a valid implementation in Contract Params
   LOW

In script builder the false value should be treated as a valid value, currently it is not being handled. Hence if a “false” is passed in Script Builder we get “false” for a correct value.

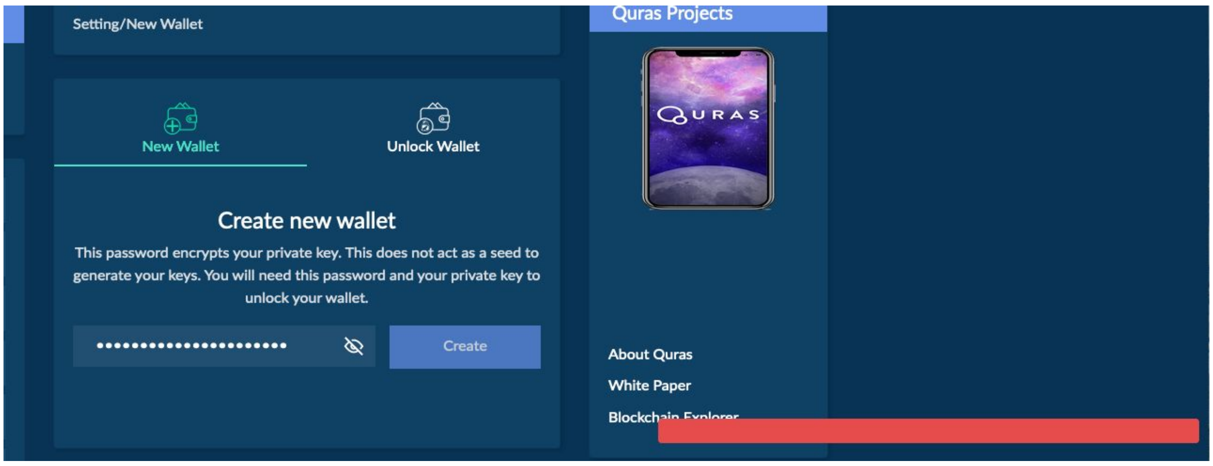
Solution :

else if(value === false) return true;



```
    * @return {scriptParams[]}
    */
  toScriptParams() {
    this.reset()
    const scripts = []
    while (!this.isEmpty()) {
      let a = retrieveAppCall(this)
      if (a) scripts.push(a)
    }
    return scripts
  }
}

const isValidValue = (value) => {
  if (value) {
    return true
  } else if (value === 0) {
    return true
  } else if (value === '') {
    return true
  }
  return false
}

/**
 * Retrieves a single AppCall from a ScriptBuilder object.
 * @param {ScriptBuilder} sb
 * @return {scriptParams}
 */
const retrieveAppCall = (sb) => {
```

5. Create wallet fails for long password inputs without throwing any error
   NOTE



Create new wallet

This password encrypts your private key. This does not act as a seed to generate your keys. You will need this password and your private key to unlock your wallet.

••••••••••••••••••••••

[redacted]