Grandstream VoIP GXP1610 Research

by. Sam Cesario
email: samcesario1@gmail.com

Introduction

This document describes the process of researching the vulnerabilities of a Grandstream GXP1610 VoIP phone. The path taken was two fold, attempting high and low level software methods to gain access. The results of this experiment were a deeper understanding of embedded systems as they apply in real-world applications, a look into firmware images and assembly level instructions, and a greater understanding of the risk faced with embedded systems.

Research

Research was conducted throughout the process and captured below.

Instructions breakdown:

Hint 1: ssh/admin/admin/hlm0mz

- Points to a directory or file in the phone
- Potentially contains the message

Hint 2: Soldering

- Signifies potentially gaining access to the firmware image via a port on the device, possibly a serial connection
- Upon further inspection, a 20 pin output was found on the board
    - The 1st pin and the 10th pin read ~5 volts
    - Pins 2, 4, 6, 8, 12, 14, 16, 18, 20 gave zero volts.
    - Pins 3, 5 , 7, 9, 11, 13, 14, 15, 17, 19 gave millivolt readings.

Hint 3: Reflashing the firmware might destroy the message so do it only if you think it's a good idea

- Flashing the firmware could be done by using the update command and pointing the device to a personal server to pull the firmware.

Hint 4:

K[0.. 3] := { 0xd76aa478, 0xe8c7b756, 0x242070db, 0xclbdceee }
K[4.. 7] := { 0xf57c0faf, 0x4787c62a, 0xa8304613, 0xfd469501 }
K[8..11] := { 0x698098d8, 0x8b44f7af, 0xffff5bb1, 0x895cd7be }
K[12..15] := { 0x6b901122, 0xfd987193, 0xa679438e, 0x49b40821 }

- Could be an address in the firmware of several locations. One location could be the assembly instructions for a root password, the location of the ssh/admin/admin/hlm0mz folder, or a spot in the firmware where the ssh admin menu is pulled up.
- Or this could aid in decrypting the message.

Voiding Warranty: The warranty can be voided if the sticker on the phone screw is broken, or if the firmware is re-flashed.

Phone Information:

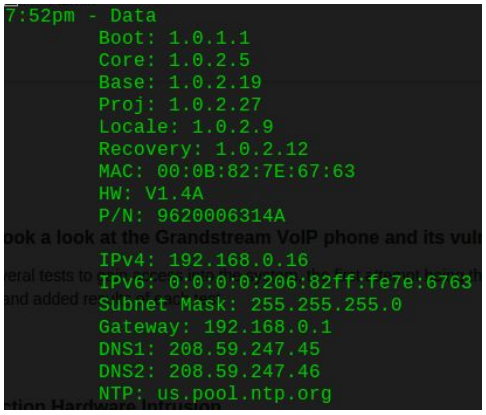

Fig 1. Output of information found on the phone via the on screen menu.

```
PORT 22/tcp open ssh
     80/tcp open http
Running: Linux 2.6.X|3.x
OS CPE: cpe:/o:linux:linux_kernel:2.6 cpe:/o:linux:linux_kernel:3
OS details: Linux 2.6.32 - 3.10
```

Fig 2. NMAP output

- Firmware Grandstream v1.0.2.27
- Dropbear sshd 2013.58. Dropbear is a small ssh client for embedded systems and v0.58 has been documented to have the CVE - 2013 - 4421 vulnerability.
- SoC used is a DVF9918 from DSP Group.
- DVF9918 uses a ARM926 dual core processor.
- DRAM used is from ZENTEL A353

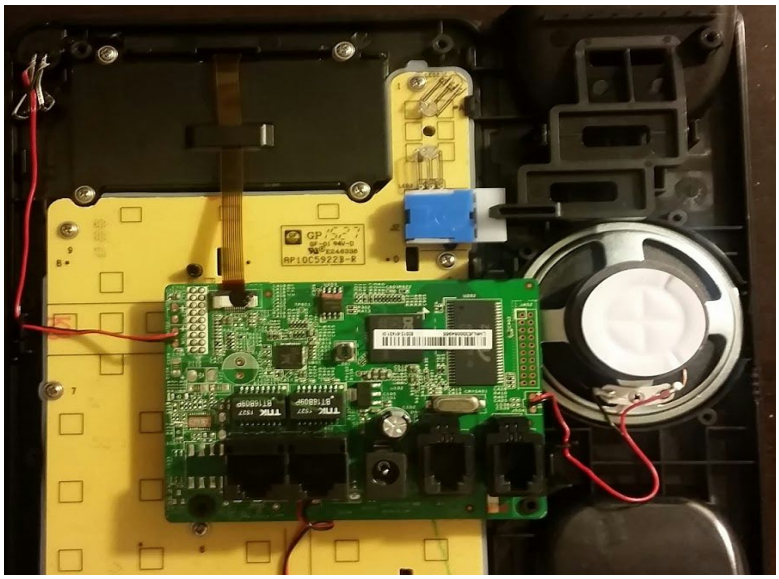

Fig. 3 A second Phone of the same model was purchased to test without destroying the firmware on the research phone.
**Notice the 20 pin connection on the right side of the circuit board.*

Exploits:
Dropbear v0.58 CVE - 2013 - 4421 - *"Error in buf_decompress when handling compressed payloads can be exploited to exhaust available memory resources by sending a specially crafted packet."*

When a packet is sent across ssh it can be compressed to be sent and then uncompressed on the server side. This uncompression in v0.58 can be exploited to overload the buffer and create a DDoS attack potentially allowing access to the system.

Interfaces:

1. JTAG - Joint Test Action Group. Defines debug communication via data lines. Now used for testing and inserting breakpoints and code via data lines.

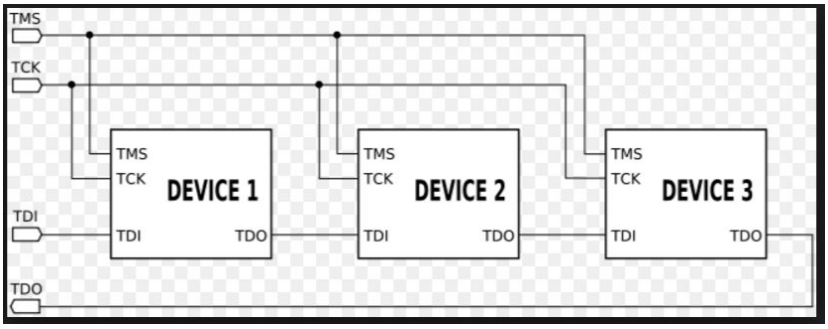

Fig. 4 JTAG daisy chain setup

Procedures

A. CVE- 2013 - 4421 exploit

Steps:

1. Downloading Dropbear v0.58 and v.59.
2. Study packet.c and the vulnerability code changes documented from v0.58 to v0.59.
3. Attempt to recreate a scenario where the buffer would be overloaded.
4. Upload file via scp with payload.

B. Brute Force:

Steps:

1. Connect to ssh via admin.
2. Test various bash commands on the command prompt.
3. Test various logins for ssh using metasploit.
4. Look at HTML requests being sent via the browser access with wireshark.
5. Listen to voicemails, if any for a message.
6. Attempt to access hidden menus via the on phone buttons.

C. Hardware Intrusion:

1. Open up the test phone without voiding the warranty.
2. Prob connections on the phone for outputs with a multimeter.
3. Establish a connection to outputs containing data.
4. Connect the outputs to an Arduino and attempt to view data on computer serial port
5. Attempt to create a JTAG interface to communicate with the device, with a FPGA, Arduino or off the shelf JTAG device.
    a. Create a simulation script to model the output of the phone.
    b. Create a JTAG interface capable of interfacing with data according to the JTAG specifications.
    c. Test the JTAG interface with the simulation inserting breakpoints and viewing output.
    d. Replace the simulation with the phone via Arduino output pins connecting as per Fig 4. above, the phone as device 1 and the Arduino as device 2.
    e. Record the startup output of the firmware.
    f. Record the ssh login output of the firmware.
    g. Decompress the output with binwalk to look for the addresses in the K vectors.
    h. Step through the ssh login attempt and create a jump command to skip loading the on screen menu, dumping into a batch.
    i. Setup a user with all permissions for easier login later.
    j. Access the folder mentioned in the instructions.
6. Debug and Follow the message.

Analysis and Results

The above attempts aim at gaining access into the embedded device. Once root access was granted, the ssh/admin/admin/hlm0mz directory or file would be inspected as will as a root user will be setup to allow for easy access later. The message would be found, decrypted and followed.

Procedure A. outlines attempting the overload the compression buffer of the dropbear sshd client. I was able to extract most of the packet.c information to create a test file. This procedure's end result would have left with a DDoS type attack, and might not have been as useful as other avenues of attack; instead of dumping to a bash shell the phone may just reset and not provided root access.

Procedure B. provided the shortest path to gaining root access. The attempts while not providing root access allowed for discovery of the phone and different avenues of penetration testing in general.

Procedure C. outlines the process of JTAG interfacing with the phone, setting breakpoints and inserting code to prompt a bash screen. I was able to create a simulated ARM instruction sender with a few commands from the decompressed stock firmware image with an Arduino. The simulation sends half the 16-bit instruction on TCK as project time-restrictions prevented me from implementing bit sending. I was able to capture the instructions and the JTAG breakpoint and insertion positions were not coded. An FPGA would have been ideal for the JTAG interface because of its state machine functionality. The FPGA would have been realized in verilog and tested with the phone via an oscilloscope.

Conclusion

In conclusion, Procedure C seemed to be the best avenue of success. The two limiting factors for find the message were time and resources. The code for this project is hosted on bitbucket at https://bitbucket.org/Smores42/rbs/ . Balancing my current job, and also resources at my disposal I feel I was able to learn a great deal about embedded systems and different avenues of penetration testing.