2018

Sarian OS: Industrial Routers Security

Kaspersky Lab Security Services, @kl_secservices
Danila Parnishchev, danila.parnishchev at kaspersky.com, @zero_wf

Contents

1 List of Abbreviations

ASN.1	Abstract Syntax Notation One
ASP	Active Server Pages
ATM	Automated Teller Machine
BIOS	Basic Input-Output System
CLI	Command Line Interface
CPU	Central Processing Unit
DNS	Domain Name System
JTAG	Joint Test Action Group
IP	Internet Protocol
MIB	Management Information Base
OS	Operating System
PDU	Protocol Data Unit
PRNG	Pseudorandom Number Generator
RAM	Random Access Memory
RCI	Remote Command Interface
RF	Radio Frequency
RTOS	Real-time Operating System
SDR	Software Defined Radio
SNMP	Simple Network Management Protocol
SNTP	Simple Network Time Protocol
SSH	Secure Shell
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
XML	Extensible Markup Language

2 Introduction

Industrial routers are widely used at factories, power stations, manufacturing automation, secure ATM and other industries to provide secure and stable network connections between different parts of manufacturing infrastructures. In such crucial areas of use, digital security is very important, because the cost of security flaws is usually high.

Currently there are many solutions for secure industrial networks available at the market. Among popular industrial router vendors are Cisco, Moxa, Westermo, B&B and Digi.

This paper is devoted to the security research of industrial routers on the example of Digi wireless routers family. We carried out all the studies described in this document with Digi WR21 router, although most of the results obtained are applicable to other TransPort routers.

The primary aim of this research is to help vendors of industrial routers to improve security of their products. Additionally, this research illustrates some techniques and tools that can be used to perform such kind of studies in the future.

2.1 Previous Research

Currently there is no security research based on industrial routers from Digi wireless routers family available for public. However, there are some recent works on industrial routers of other vendors. One of them is the report from Qualys security specialist called "Westermo MRD-305-DIN, MRD-315, MRD-355 and MRD-455 Multiple Security Vulnerabilities", which is available at the following link:

https://threatprotect.qualys.com/2017/08/29/westermo-mrd-305-din-mrd-315-mrd-355-and-mrd-455-multiple-security-vulnerabilities/

Another work on the subject is the blog post from Cisco Talos Group about security research of Moxa industrial routers, which is available at the following link:

http://blog.talosintelligence.com/2017/04/moxa-box.html

3 Research Target

We chose Digi WR21 industrial router as a target for this research. Figure 1 shows the appearance of this device.

Figure 1 – Digi WR21 industrial router

This router supports not only standard Ethernet, but also wireless networking (3G/4G LTE), and has a wide range of applications. The product page is available at https://www.digi.com/products/cellular-solutions/cellular-routers/digi-transport-wr21. It contains detailed information about the device and its features.

The main reason why we selected Digi WR21 as a research target is that there is no public research of Digi industrial routers now despite the fact that they are widely used in different industries. Figure 2 illustrates the map of different Digi network devices connected to the Internet. Figure 3 shows the map of using Digi Transport® industrial routers connected to the Internet. The map was generated using Shodan service.

According to the maps, there are more than 10000 Digi devices available through the Internet, and at least 576 are Digi Transport® industrial routers.

One more reason why we decided to study the security of the industrial router from Digi Transport family is that all the routers from this family use custom operating system Sarian OS, which is an interesting security research target by itself.

Digi WR21 firmware is available at the product page. At the time of the study, actual firmware version was 5.2.17.12 (March 2017), so all results of this research are applicable to this version of firmware.

Figure 2 – Digi network products usage around the World

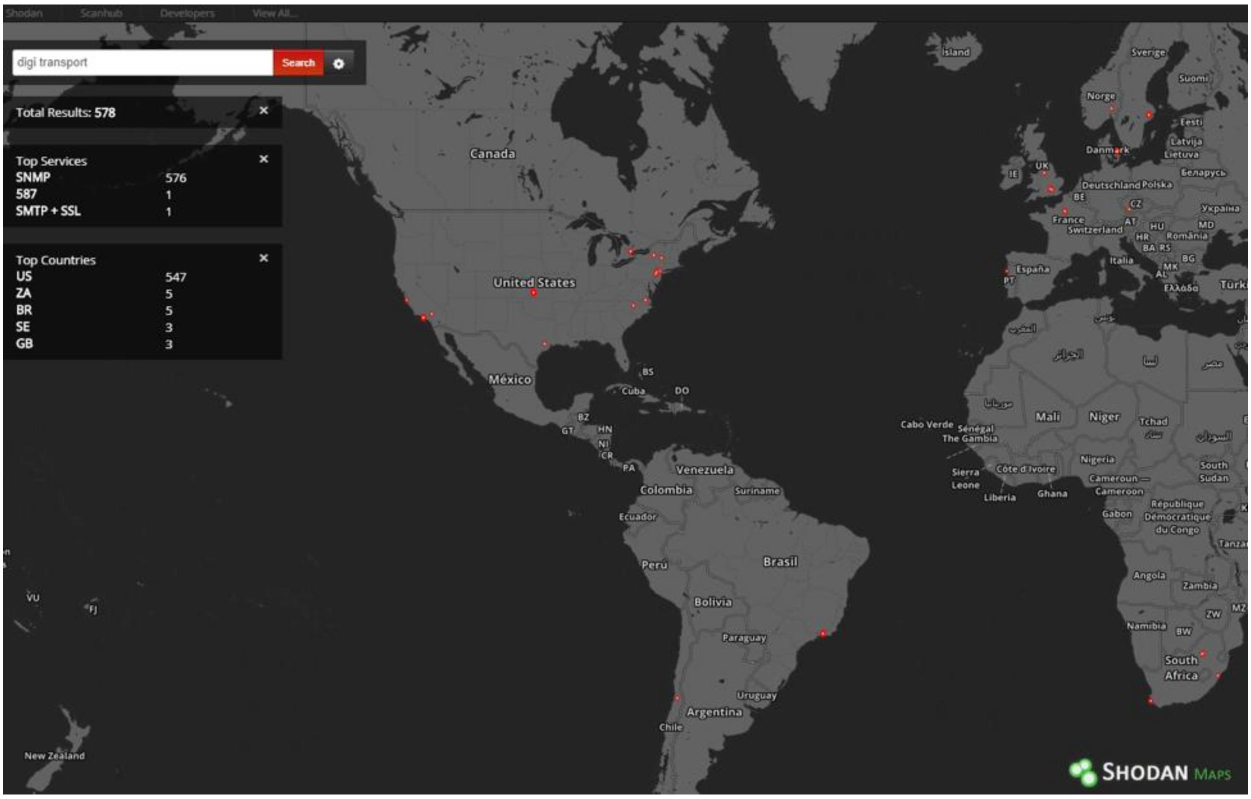


Figure 3 – The map of Digi Transport® devices used around the World (Shodan service info)

4 Digi WR21 Hardware Internals

4.1 Hardware Components

In order to understand on which platform the firmware is executed, it is necessary to observe device's internals.

Figure 4 shows the general view of the router's circuit board.

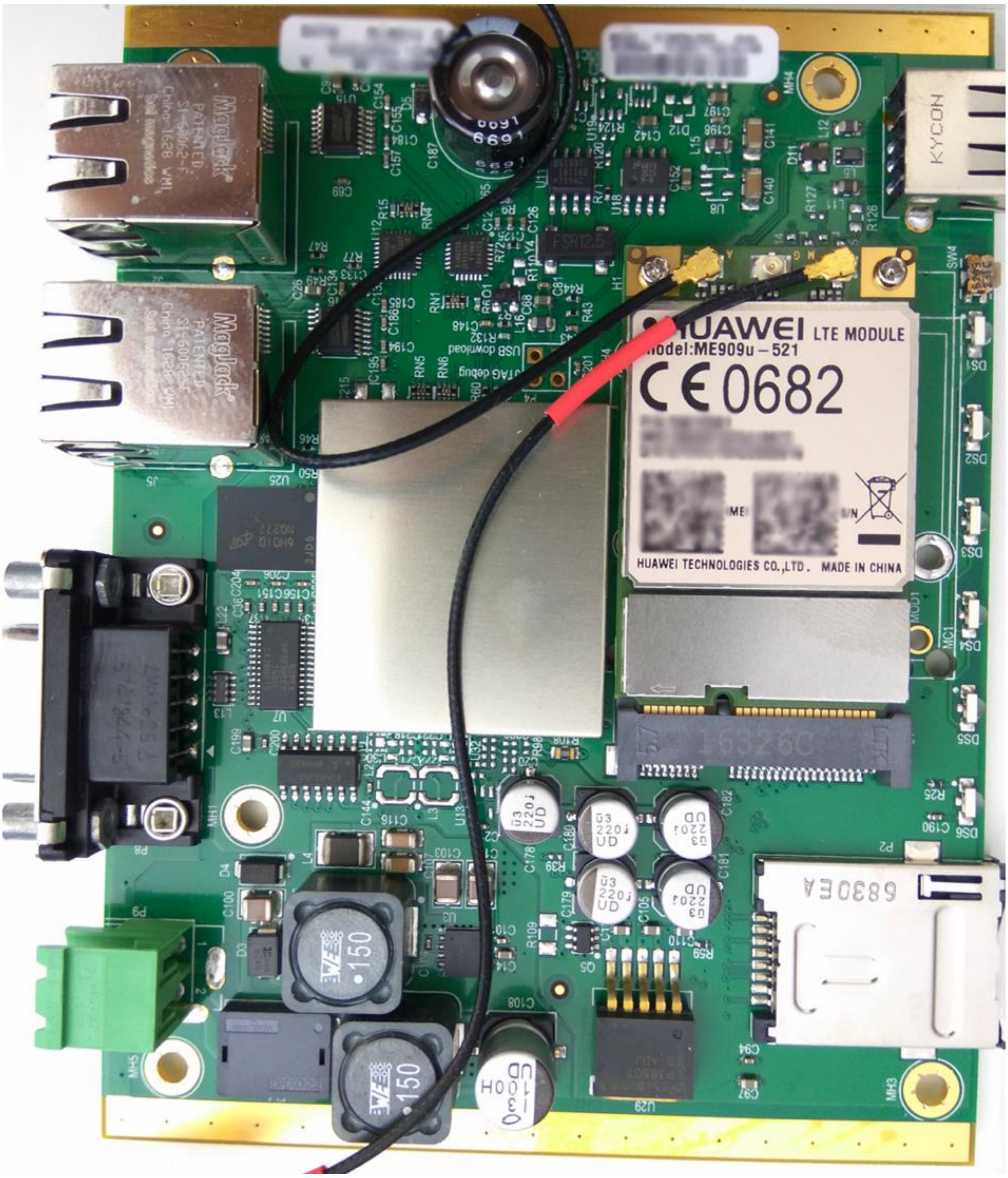


Figure 4 – Digi WR21 device circuit board (general view)

Here we can see the Huawei cellular modem, which is connected to the PCIe extension slot. One more thing to notice is the metal shield that covers part of the board.

Figure 5 illustrates the same board with the shield removed. For completeness, the other side of the board is shown at Figure 6.

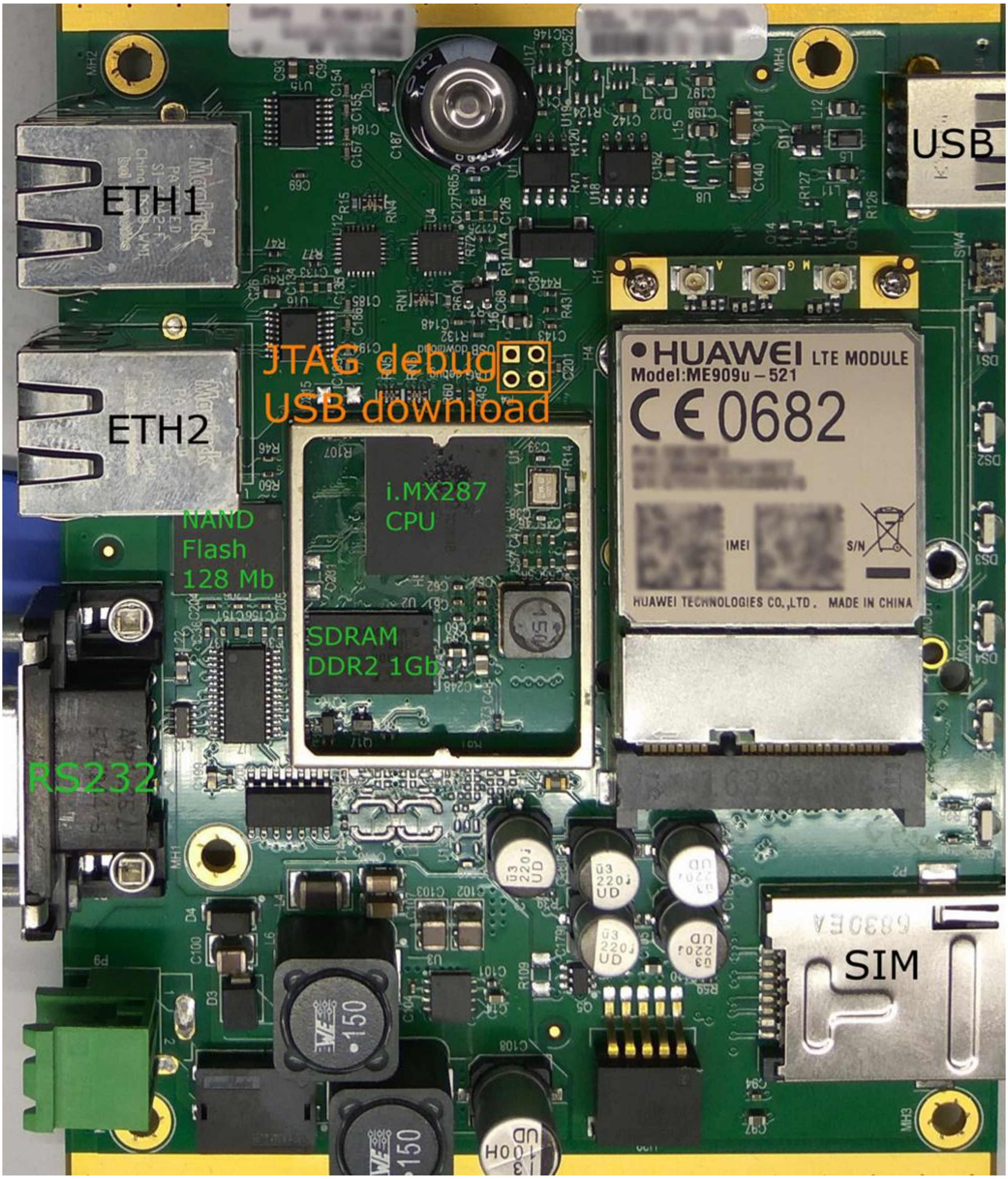


Figure 5 - Digi WR21 device circuit board (side A)

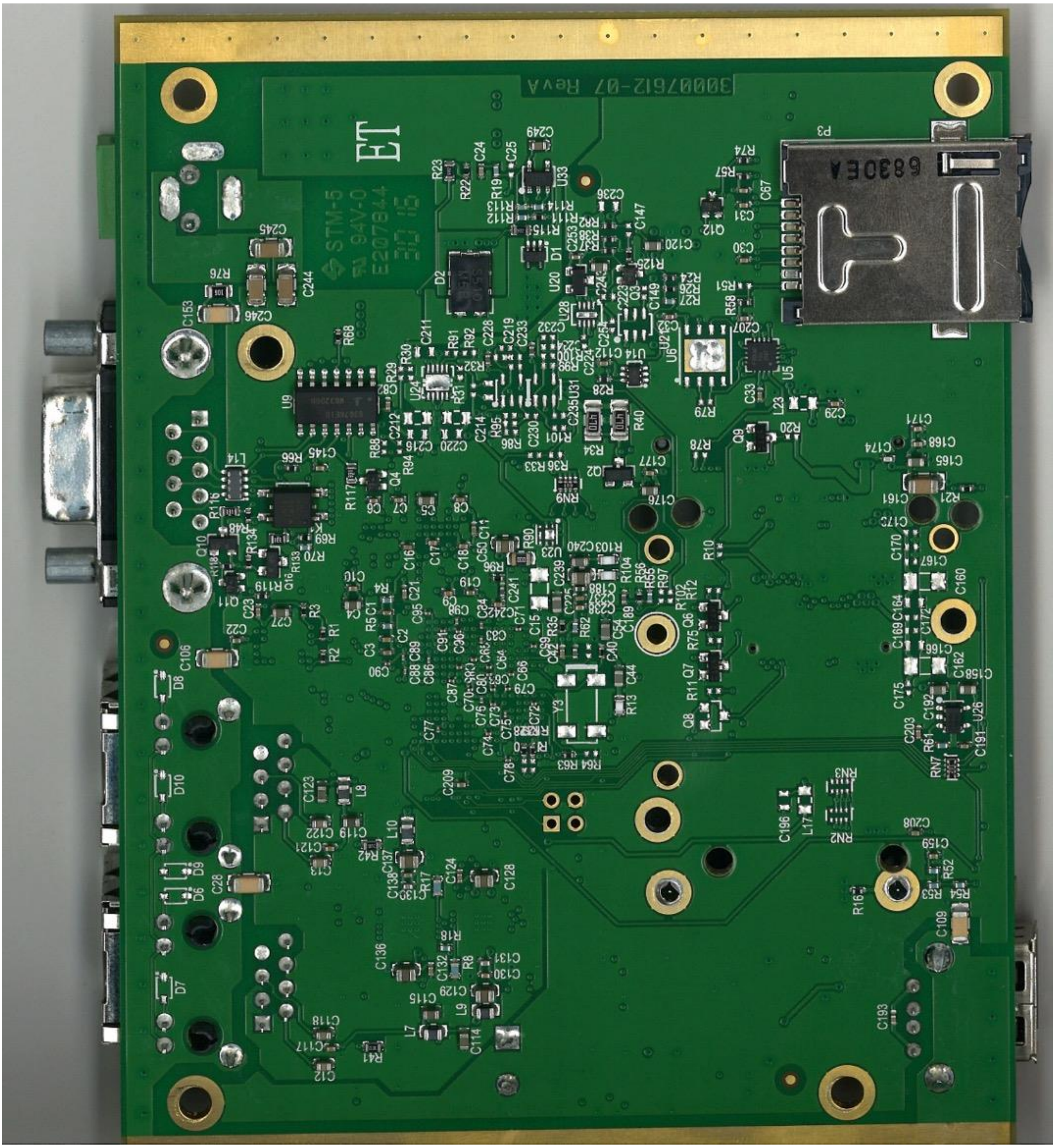


Figure 6 - Digi WR21 device circuit board (side B)

Internally Digi WR21 device consists of the following main parts:

- NXP i.MX28 CPU, which has ARM9 32-bit core;
- Huawei ME909U LTE module;
- ISSI 1 GB DDR2 SDRAM;
- Micron NAND Flash storage memory (128 Mbytes);
- Ethernet, RS232, USB connectors and 2 slots for SIM-cards.

Four contact holes on the board are worth mentioning. These contact holes are marked as “JTAG debug or USB download”. We found that one of the contacts is connected directly to the ground of the board. Remaining three contacts are not enough to function as JTAG, which in general requires at least TDI, TDO, TMS and TCK. Therefore, we assumed that these contacts are intended for downloading firmware into NAND Flash memory using USB bus. This recovery feature is supported by i.MX28 CPU. For more information about this CPU feature, see “i.MX28 Applications Processor Reference Manual”.

4.2 Debug Interfaces

According to “i.MX28 Applications Processor Reference Manual”, i.MX28 CPU supports JTAG debugging. Figure 7 shows the ball map for the BGA CPU.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
A	VSS	SSP3_SCK	SSP2_SCK	SSP0_CMD	SSP0_DATA3	SSP0_SCK	VDDIO33	USB1DP	VSS	USB0DM	PSWITCH	XTALI	VDD4P2	RESETN	BATTERY	DCDC_LP	DCDC_GND	A
B	SSP1_SCK	SSP3_MISO	SSP2_MISO	SSP0_DATA7	SSP0_DATA4	SSP0_DATA0	VSS	USB1DM	DEBUG	USB0DP	VSSA2	XTALO	VSSA1	HSADC0	DCDC_BATT	DCDC_VDDA	DCDC_LN1	B
C	SSP1_CMD	SSP3_MOSI	SSP2_MOSI	SSP2_SS0	SSP0_DATA5	SSP0_DATA1	I2C0_SCL	LRADC2	LRADC1	TESTMODE	RTC_XTALO	VDDXTAL	VDDA1	LRADC6	LRADC0	VSS	DCDC_VDDIO	C
D	SSP1_DATA0	SSP3_SS0	SSP2_SS1	SSP2_SS2	SSP0_DATA6	SSP0_DATA2	SPDIF	I2C0_SDA	LRADC3	SSP0_DETECT	RTC_XTALI	JTAG_TMS	LRADC4	JTAG_TRST	LRADC5	VDD1P5	DCDC_VDDD	D
E	SSP1_DATA3	ENET_CLK	ENET0_TX_CLK	ENET0_RX_EN	VSS	VDDIO33	SAIF0_SDATA0	SAIF1_SDATA0	PWM3	PWM4	JTAG_TCK	JTAG_TDI	JTAG_TDO	JTAG_RTCK	VSS	VDDIO33	VDD5V	E
F	ENET0_TXD0	ENET0_TXD1	ENET0_RX_CLK	ENET0_TX_EN	AUART2_TX	AUART2_RX	SAIF0_BITCLK	VDDIO18	VDDIO18	VDDD	VDDD	VDDD	EMI_D14	VSSIO_EMI	EMI_DQM1	VSSIO_EMI	EMI_D15	F
G	ENET0_TXD2	ENET0_TXD3	VDDIO33	ENET0_MDC	AUART0_RX	SAIF0_LRCLK	SAIF0_MCLK	VDDIO18	VDDIO18	VDDD	VDDD	VDDD	VDDIO_EMI	EMI_D10	VDDIO_EMI	EMI_D08	VDDIO_EMI	G
H	ENET0_RXD0	ENET0_RXD1	VSS	ENET0_MDIO	AUART0_TX	AUART2_CTS	AUART2_RTS	VDDIO33	VSS	VSS	VSS	VSS	EMI_D12	VSSIO_EMI	EMI_D09	VSS	EMI_D13	H
J	ENET0_RXD2	ENET0_RXD3	ENET0_CRS	ENET0_COL	AUART1_RTS	AUART0_CTS	AUART0_RTS	VDDIO33	VDDIO33	VDDIO33	VSS	VSS	VDDIO_EMIQ	EMI_D11	VSS	EMI_DQS1N	EMI_DQS1	J
K	LCD_WR_RWN	LCD_D00	LCD_D01	AUART1_TX	AUART1_CTS	AUART3_RTS	PWM0	PWM2	VSS	VSS	VSS	VDDD	EMI_VREF1	EMI_DDR_OPEN	VDDIO_EMIQ	EMI_DQS0N	EMI_DQS0	K
L	LCD_VSYNC	LCD_D02	LCD_D03	AUART1_RX	AUART3_TX	AUART3_CTS	PWM1	GPMI_RDY3	GPMI_RESETN	VSS	VSS	VSSIO_EMI	VDDIO_EMI	EMI_D06	EMI_DDR_OPEN_FB	EMI_CLKN	EMI_CLK	L
M	LCD_HSYNC	LCD_D04	LCD_D05	LCD_RS	AUART3_RX	LCD_RESET	GPMI_CE2N	GPMI_RDY2	GPMI_CE3N	VDDIO_EMI	VDDIO_EMI	VDDIO_EMI	EMI_D01	VSS	EMI_DQM0	VSSIO_EMI	EMI_D07	M
N	LCD_DOTCLK	LCD_D06	VDDIO33	VSS	LCD_ENABLE	GPMI_RDY0	GPMI_CE0N	GPMI_RDY1	GPMI_CE1N	EMI_A14	EMI_A07	EMI_BA2	VDDIO_EMI	EMI_D03	VDDIO_EMI	EMI_D00	VDDIO33_EMI	N
P	LCD_D07	LCD_D08	LCD_D09	LCD_RD_E	LCD_CS	GPMI_ALE	GPMI_CLE	GPMI_WRN	EMI_CE1N	EMI_A09	VDDIO_EMI	EMI_CE0N	EMI_D04	VSSIO_EMI	EMI_D02	VSSIO_EMI	EMI_D05	P
R	LCD_D10	LCD_D11	LCD_D17	LCD_D20	LCD_D23	GPMI_RDN	GPMI_D05	GPMI_D02	EMI_A06	VSSIO_EMI	EMI_A05	VSSIO_EMI	VDDIO_EMI	EMI_VREF0	VDDIO_EMIQ	EMI_RASN	EMI_ODT0	R
T	LCD_D12	LCD_D13	LCD_D16	LCD_D19	LCD_D22	GPMI_D07	GPMI_D04	GPMI_D01	EMI_A13	EMI_A11	EMI_A03	EMI_BA1	EMI_CKE	VSSIO_EMI	EMI_WEN	EMI_BA0	EMI_ODT1	T
U	VSS	LCD_D14	LCD_D15	LCD_D18	LCD_D21	GPMI_D06	GPMI_D03	GPMI_D00	EMI_A08	EMI_A04	EMI_A12	EMI_A01	EMI_A10	EMI_A02	EMI_A00	EMI_CASN	VSSIO_EMI	U
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	

Figure 7 – i.MX287 BGA ball map (the picture was taken from “i.MX28 Applications Processor Reference Manual”)

We unsoldered the CPU chip and connected thin isolated copper wires to contact holes according to “JTAG_TMS” (D12), “JTAG_TRST” (D14), “JTAG_TCK” (E11), “JTAG_TDI” (E12), “JTAG_TDO” (E13) and “JTAG_RTCK” (E14). After that, we soldered the chip back into its place. Figure 8 illustrates the overall result.

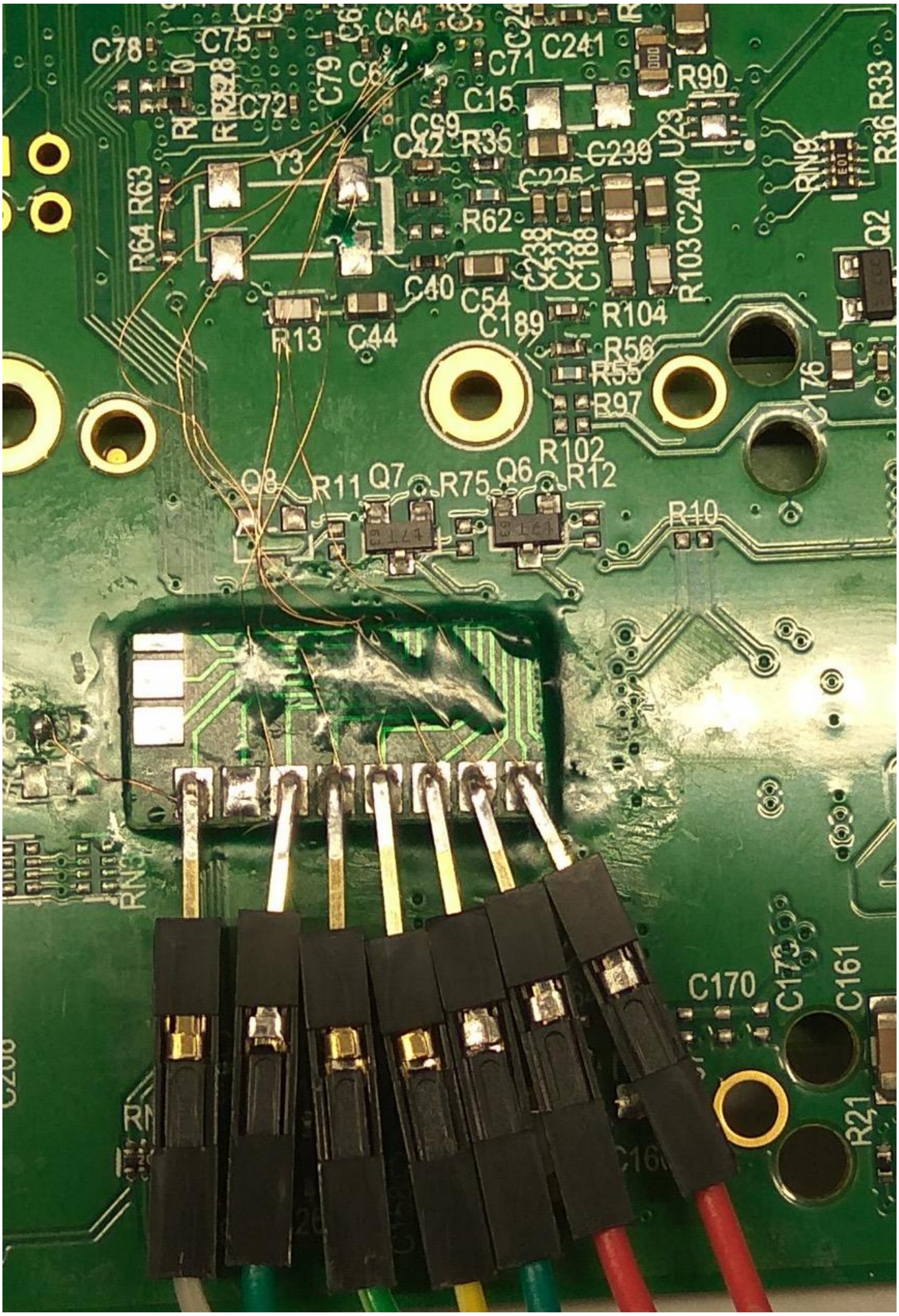


Figure 8 – CPU JTAG

We used Bus Blaster by Dangerous Prototypes as the JTAG adapter Figure 9 shows CPU JTAG pins connected to Bus Blaster.

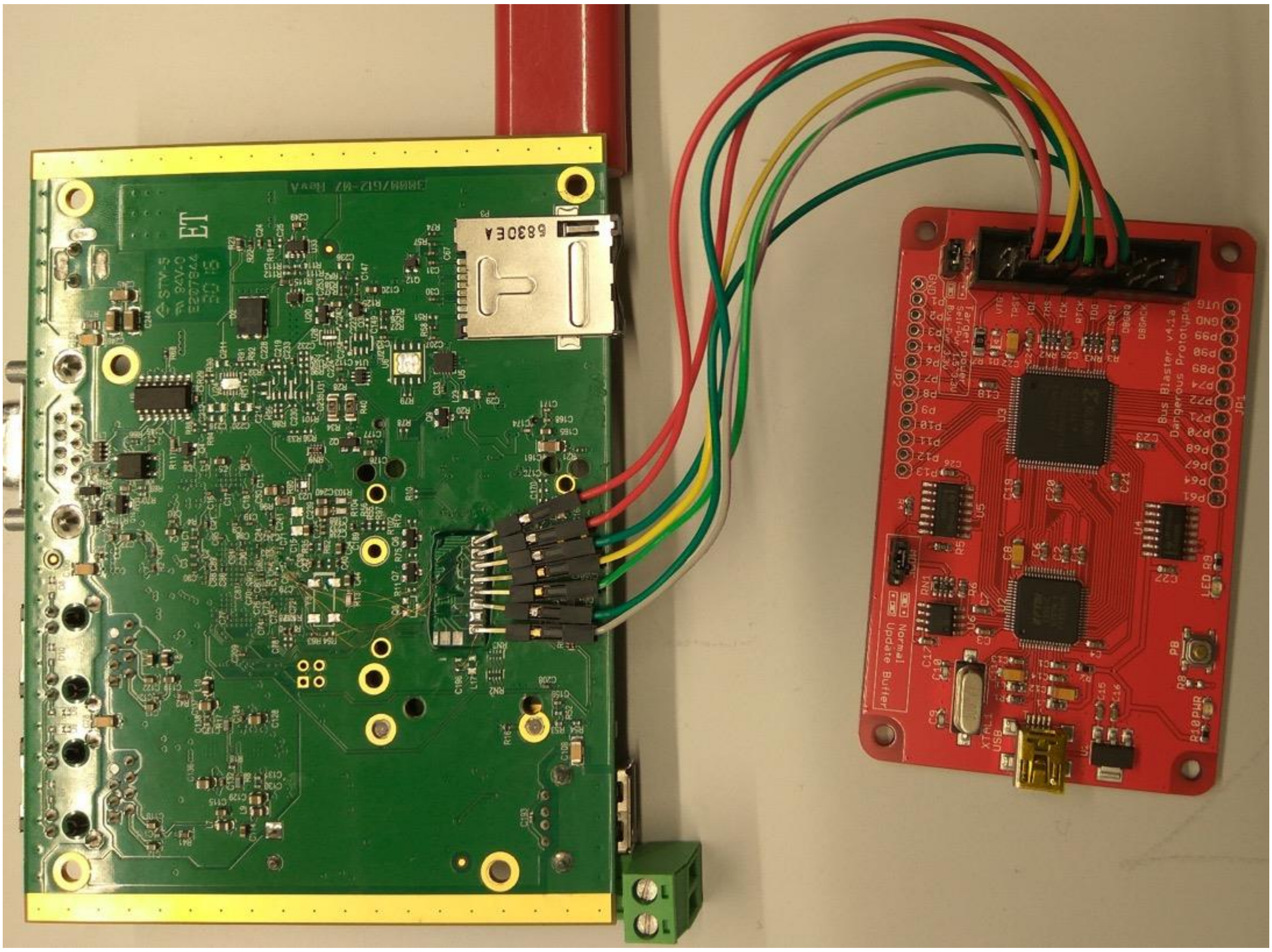


Figure 9 – CPU JTAG pins connected to Bus Blaster

To access Bus Blaster and use it as a JTAG adapter, we used OpenOCD application. This application uses configuration files to connect to a JTAG adapter and start debugging. We had to create only one configuration file *openocd.cfg,* which in our case contained the following lines:

```
source [find interface/ftdi/dp_busblaster.cfg]
source [find target/imx28.cfg]
adapter_khz 400
```

In order to provide OpenOCD this configuration, we can either start it without any parameters from the same folder where the configuration file is saved, or use "-f" option to specify the path to the configuration file.

After a successful start with the configuration file, OpenOCD application automatically discovers the target adapter and starts waiting for connections at TCP ports 3333 (gdb server) and 4444 (OpenOCD command shell).

In order to debug the router's firmware, we used gdb-multiarch utility, which supports ARM architecture. The following commands can be used to connect to OpenOCD gdb server, halt the CPU and start debugging using gdb:

```
gdb-multiarch
(gdb) set architecture arm
(gdb) target remote localhost:3333
(gdb) monitor halt
```

Executing this commands in gdb, we were able to halt the CPU Therefore, we found out that JTAG on the CPU was not disabled. Unfortunately, we were still unable to debug the device, because since approximately 2 seconds after halting it rebooted every time we established the connection between the device and the JTAG adapter. As we figured out later, continuous resets happened because of the embedded watchdog timer. In order to disable the timer and obtain the opportunity to debug the router, we used BIOS command console. For information about disabling the timer, please refer to section 8.3 of this document.

5 Digi WR21 Firmware Structure

Device firmware consists of several files listed in Table 1.

Table 1 – Files included in Digi WR21 firmware

File Name	File Type	File Size	File Content
boot.rom	Binary executable	256 KB	Bootloader code
image	Encrypted and compressed binary executable	~ 4.3 MB	Image of Sarian OS
logcodes.txt	Text file	21 KB	System event codes and their meanings
privpy.enc	Encrypted zip-archive	61 KB	Proprietary Python scripts
python.zip	Zip-archive	~ 1.7 MB	Compiled Python standard modules
wizards.zip	Zip-archive	376 KB	Compiled Python modules, extending device web server functionality
wr21.web	Custom archive	~ 1.5 MB	web server files (ASP and HTML pages, CSS and JavaScript files and pictures)

Actually, the bootloader is called “ARM Sarian BIOS”, so we will use the term “BIOS” further in this research.

CRC16-CCITT is used for the consistency checks of executable firmware files. All CRC16 fields of files mentioned further in this chapter contain CRC16-CCITT checksums. We used the following Python function to calculate CRC16 of firmware files during research:

```
def crc16(data):
      crc16_tab = (
      0x0000, 0x1189, 0x2312, 0x329B, 0x4624, 0x57AD, 0x6536, 0x74BF,
      0x8C48, 0x9DC1, 0xAF5A, 0xBED3, 0xCA6C, 0xDBE5, 0xE97E, 0xF8F7,
      0x1081, 0x0108, 0x3393, 0x221A, 0x56A5, 0x472C, 0x75B7, 0x643E,
      0x9CC9, 0x8D40, 0xBFDB, 0xAE52, 0xDAED, 0xCB64, 0xF9FF, 0xE876,
      0x2102, 0x308B, 0x0210, 0x1399, 0x6726, 0x76AF, 0x4434, 0x55BD,
      0xAD4A, 0xBCC3, 0x8E58, 0x9FD1, 0xEB6E, 0xFAE7, 0xC87C, 0xD9F5,
      0x3183, 0x200A, 0x1291, 0x0318, 0x77A7, 0x662E, 0x54B5, 0x453C,
      0xBDCB, 0xAC42, 0x9ED9, 0x8F50, 0xFBEF, 0xEA66, 0xD8FD, 0xC974,
      0x4204, 0x538D, 0x6116, 0x709F, 0x0420, 0x15A9, 0x2732, 0x36BB,
      0xCE4C, 0xDFC5, 0xED5E, 0xFCD7, 0x8868, 0x99E1, 0xAB7A, 0xBAF3,
      0x5285, 0x430C, 0x7197, 0x601E, 0x14A1, 0x0528, 0x37B3, 0x263A,
      0xDECD, 0xCF44, 0xFDDF, 0xEC56, 0x98E9, 0x8960, 0xBBFB, 0xAA72,
      0x6306, 0x728F, 0x4014, 0x519D, 0x2522, 0x34AB, 0x0630, 0x17B9,
      0xEF4E, 0xFEC7, 0xCC5C, 0xDDD5, 0xA96A, 0xB8E3, 0x8A78, 0x9BF1,
      0x7387, 0x620E, 0x5095, 0x411C, 0x35A3, 0x242A, 0x16B1, 0x0738,
      0xFFCF, 0xEE46, 0xDCDD, 0xCD54, 0xB9EB, 0xA862, 0x9AF9, 0x8B70,
      0x8408, 0x9581, 0xA71A, 0xB693, 0xC22C, 0xD3A5, 0xE13E, 0xF0B7,
      0x0840, 0x19C9, 0x2B52, 0x3ADB, 0x4E64, 0x5FED, 0x6D76, 0x7CFF,
      0x9489, 0x8500, 0xB79B, 0xA612, 0xD2AD, 0xC324, 0xF1BF, 0xE036,
```

```
    0x18C1, 0x0948, 0x3BD3, 0x2A5A, 0x5EE5, 0x4F6C, 0x7DF7, 0x6C7E,
    0xA50A, 0xB483, 0x8618, 0x9791, 0xE32E, 0xF2A7, 0xC03C, 0xD1B5,
    0x2942, 0x38CB, 0x0A50, 0x1BD9, 0x6F66, 0x7EEF, 0x4C74, 0x5DFD,
    0xB58B, 0xA402, 0x9699, 0x8710, 0xF3AF, 0xE226, 0xD0BD, 0xC134,
    0x39C3, 0x284A, 0x1AD1, 0x0B58, 0x7FE7, 0x6E6E, 0x5CF5, 0x4D7C,
    0xC60C, 0xD785, 0xE51E, 0xF497, 0x8028, 0x91A1, 0xA33A, 0xB2B3,
    0x4A44, 0x5BCD, 0x6956, 0x78DF, 0x0C60, 0x1DE9, 0x2F72, 0x3EFB,
    0xD68D, 0xC704, 0xF59F, 0xE416, 0x90A9, 0x8120, 0xB3BB, 0xA232,
    0x5AC5, 0x4B4C, 0x79D7, 0x685E, 0x1CE1, 0x0D68, 0x3FF3, 0x2E7A,
    0xE70E, 0xF687, 0xC41C, 0xD595, 0xA12A, 0xB0A3, 0x8238, 0x93B1,
    0x6B46, 0x7ACF, 0x4854, 0x59DD, 0x2D62, 0x3CEB, 0x0E70, 0x1FF9,
    0xF78F, 0xE606, 0xD49D, 0xC514, 0xB1AB, 0xA022, 0x92B9, 0x8330,
    0x7BC7, 0x6A4E, 0x58D5, 0x495C, 0x3DE3, 0x2C6A, 0x1EF1, 0x0F78,
    )
    fcs = 0xFFFF
    i = 0
    while i < len(data):
        cur = ord(data[i])
        crc_index = (fcs ^ cur) & 0xFF
        fcs = (fcs >> 8) ^ crc16_tab[crc_index]
        i += 1
    return fcs
```

5.1 BIOS Image

Table 2 illustrates the structure of the "boot.rom" file.

Table 2 – BIOS image format

Offset	Size (bytes)	Description
0x00	0x04	BIOS code size
0x04	0x04	CRC-16 checksum of BIOS code
0x08	BIOS code size	BIOS code

BIOS image is neither compressed nor encrypted.

BIOS code is loaded into memory of the device at address 0x40000000. It is for Little Endian 32-bit ARM architecture.

Quick study of BIOS code showed that it supports multitasking. Several tasks with the following names are created during its work:

- TIMER;
- ETH;
- UDP;
- TFTP

BIOS code sets up all the hardware systems, loads an OS and passes control to it.

5.2 Sarian OS Image

5.2.1 Image File Format

Table 3 illustrates the OS image format. We recovered this format from BIOS code part responsible for system boot.

Table 3 – OS image format

Offset	Size (bytes)	Description
0x00	0x02	CRC-16 checksum of the image in compressed and encrypted form
0x02	0x02	CRC-16 checksum of the image in uncompressed and unencrypted form
0x04	0x04	OS loading address
0x08	0x04	The size of the OS image in compressed form
0x0C	0x04	OS image execution start address
0x10	0x0A	Unknown header section
0x1A	0x01	Bit flags, showing whether encryption or compression is used Bit 0 equal to 1 means that the image is encrypted. Bit 4 equal to 1 means that the image is compressed before encryption
0x1B	0x03	Unknown header section
0x1E	0x05	"image" signature
0x23	0x0D	Padding
0x30	0x03	"WW6" signature
0x33	0x4D	Padding
0x70	The size of the OS image in compressed form	OS image (compressed, encrypted)

5.2.2 Image File Encryption and Compression

OS image file ships in compressed and encrypted form. Figure 10 illustrates byte histogram of the image file.

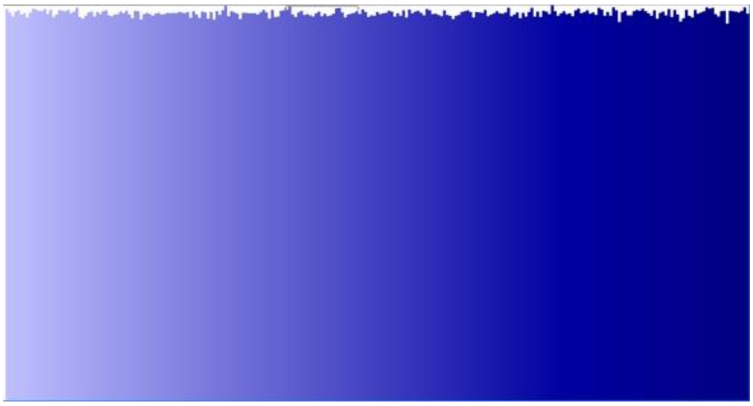


Figure 10 – "image" file histogram

There are two main functions in BIOS code for working with the OS image file:

```
int file_open(char *file_name, char *mode);
int file_read(void *dst, int size, int *p_bytes_read, int fd);
```

The image file, like any other file, must be opened for reading before actual reading process. Mode string is a concatenation of mode flags, like classical mode string of C "fopen" function. Thus, to open a file for reading, "r" mode string is used. If a file that needs to be opened is encrypted and/or compressed, then mode flags "e" and "c" are used (respectively). For example, to open a compressed and encrypted file for reading, the mode string must be equal to "rce" ("read, compressed, encrypted"). File decryption and decompression is made transparently for API users inside the file_read function.

Decryption is a simple XOR of plain OS image data with keystream, produced based on a seed. The seed is hardcoded into BIOS and is equal to 0xAF4.

Firmware signing is not implemented in the device.

Compression algorithm is a variety of LZ77 scheme. We did not study the details of it, because there was no need for doing that in the research. We just used code from BIOS to decompress the image.

We wrote the following script to decrypt and then decompress image files:

```
from array import array
FILE_ENC_SEED_2 = 0xAF4 # the seed for producing XOR keystream
FILE_DECOMPRESS_STATE = 0x00
FILE_DECOMPRESS_FLAGS = 0x00000000
FILE_DEC_ARRAY_INDEX = 0x7EE
def GetGamma(BitCount):
	global FILE_ENC_SEED_2
	R3 = BitCount - 1
	R4 = 0x8081
	R12 = 0x80000000
	R2 = FILE_ENC_SEED_2
	R1 = (1 << (BitCount - 1)) & 0xFFFFFFFF
	R0 = 0
	while R3 != -1:
		R0 = (R0 >> 1) & 0xFFFFFFFF
		if((R2 & 1) != 0):
```

```
                R2 = R2 ^ R4
                R2 = R12 | ((R2 >> 1) & 0xFFFFFFFF)
                R0 = R0 | R1
            else:
                R2 = (R2 >> 1) & 0xFFFFFFFF
            R3 = R3 - 1
        FILE_ENC_SEED_2 = R2
        return R0
def DecryptData(Data):
        Size = len(Data)
        i = 0
        while(Size > 3):
            Gamma = GetGamma(32)
            Data[i] = Data[i] ^ (Gamma & 0xFF)
            Data[i + 1] = Data[i + 1] ^ ((Gamma >> 8) & 0xFF)
            Data[i + 2] = Data[i + 2] ^ ((Gamma >> 16) & 0xFF)
            Data[i + 3] = Data[i + 3] ^ ((Gamma >> 24) & 0xFF)
            i = i + 4
            Size = Size - 4
        while(Size > 0):
            Gamma = GetGamma(8)
            Data[i] = Data[i] ^ (Gamma & 0xFF)
            Size = Size - 1
            i = i + 1
        return Data
def DecryptImage(SrcFileName, DstFileName):
        SrcFile = open(SrcFileName, 'rb')
        DstFile = open(DstFileName, 'wb')
        SrcFile.seek(0, 2)
        Size = SrcFile.tell() - 0x80
        SrcFile.seek(0x80, 0)
        EncryptedData = array('B', SrcFile.read(Size))
        DecryptedData = DecryptData(EncryptedData)
        DstFile.write(DecryptedData)
        SrcFile.close()
        DstFile.close()
def DecompressData(Data):
        global FILE_DECOMPRESS_STATE
        global FILE_DECOMPRESS_FLAGS
        global FILE_DEC_ARRAY_INDEX
        DecompressArray = array('B')
        for i in range(0, 0x800):
            DecompressArray.append(0)
        R8 = len(Data)
        R5 = 0
        DecompressedData = array('B')
        R4 = 0
        while(1):
            R3 = (FILE_DECOMPRESS_FLAGS >> 1) & 0xFFFFFFFF
            FILE_DECOMPRESS_FLAGS = R3
            if((R3 & 0x100) == 0):
                if(R8 == 0):
                    FILE_DECOMPRESS_STATE = 1
                    UncSize = R5
                    return DecompressedData
                R2 = (Data[R4] | 0xFF00) & 0xFFFFFFFF
                R8 = (R8 - 1) & 0xFFFFFFFF
                R4 = (R4 + 1) & 0xFFFFFFFF
                FILE_DECOMPRESS_FLAGS = R2
            if(R8 == 0):
```

```
                FILE_DECOMPRESS_STATE = 2
                UncSize = R5
                return DecompressedData
            R8 = (R8 - 1) & 0xFFFFFFFF
            R2 = (R4 + 1) & 0xFFFFFFFF
            R3 = Data[R4]
            if((FILE_DECOMPRESS_FLAGS & 0x01) != 0):
                DecompressedData.append(R3)
                R5 = (R5 + 1) & 0xFFFFFFFF
                R1 = FILE_DEC_ARRAY_INDEX
                R0 = (R1 + 1) & 0x7FF
                R4 = R2
                DecompressArray[R1] = R3
                FILE_DEC_ARRAY_INDEX = R0
                continue
            R4 = R2
            dword_40029834 = R3
            if(R8 == 0):
                FILE_DECOMPRESS_STATE = 3
                UncSize = R5
                return DecompressedData
            R1 = dword_40029834
            R3 = Data[R4]
            LR = R3 & 0xF0
            LR = R1 | ((LR << 4) & 0xFFFFFFFF)
            R3 = R3 & 0x0F
            R1 = (R4 + 1) & 0xFFFFFFFF
            dword_40029834 = LR
            R3 = (R3 + 2) & 0xFFFFFFFF
            R0 = 0
            while(1):
                LR = (LR + R0) & 0xFFFFFFFF
                R4 = DecompressArray[LR & 0x7FF]
                DecompressedData.append(R4)
                R0 = (R0 + 1) & 0xFFFFFFFF
                DecompressArray[FILE_DEC_ARRAY_INDEX] = R4
                FILE_DEC_ARRAY_INDEX = (FILE_DEC_ARRAY_INDEX + 1) & 0x7FF
                R5 = (R5 + 1) & 0xFFFFFFFF
                if(R3 < R0):
                    break
                LR = dword_40029834
            R8 = (R8 - 1) & 0xFFFFFFFF
            R4 = R1
def DecompressImage(SrcFileName, DstFileName):
    SrcFile = open(SrcFileName, 'rb')
    DstFile = open(DstFileName, 'wb')
    SrcFile.seek(0, 2)
    Size = SrcFile.tell()
    SrcFile.seek(0, 0)
    CompressedData = array('B', SrcFile.read(Size))
    DecompressedData = DecompressData(CompressedData)
    DstFile.write(DecompressedData)
    SrcFile.close()
    DstFile.close()
#usage example
DecryptImage('image', 'image_decrypted')
DecompressImage('image_decrypted', 'image_clear')
```

This script is rather tricky, because it is a “close to assembly” implementation of decryption and decompression algorithms. Alternatively, it is possible to use emulation to obtain clear OS image. For example, Unicorn engine combined with “idaemu” plugin for IDA Pro is suitable for this task.

After decryption and decompression “image” file turns into an OS code image of about 7.5 MB. We took the loading address of this code from the “image” file header (see Table 3). Section 6 of this document contains the analysis of plain OS image.

5.2.3 Hardcoded Secret for Firmware Encryption

Details of the issue were reported to vendor on 14.09.2017. Digi claimed that it is possible to mitigate the issue by means of device hardening.

Firmware for the router is protected by the XOR-based masking algorithm. According to this algorithm, firmware is XOR-ed with the sequence of bytes, produced by simple procedure depending only on the seed value. This value is hard-coded inside the device's boot code (BIOS), and it is the same for all modern firmware versions (5.2.17.12, 5.2.18.3, 5.2.19.6). It equals to 0xAF4. Knowing this secret value allows decrypting device’s firmware, analyzing and modifying it and programming it to the device.

The firmware can be protected from modification with a cryptography signing algorithm. In such case, malicious users will not be able to tamper device's firmware, even after obtaining the highest privilege level in the system.

6 Sarian OS

Sarian OS (Sar/OS, SAROS) is a RTOS developed by Sarian Systems, Ltd.

On April 28, 2008, “Digi International Limited” acquired “Sarian Systems, LTD”. Now Sarian OS is used in Digi products.

Sarian OS is a proprietary software. There is no public information about its internals and API, and there is no research available for this system.

We focused on the sample of Sarian OS included in Digi WR21 device firmware of version 5.2.17.12 (March 8, 2017). Most of the results from this research will be suitable for other versions of the OS included in Sarian and Digi products firmware.

In this section, we will describe Sarian OS internals to understand the environment for running application (HTTP, FTP, SSH, etc.)

6.1 Command Line Interface

6.1.1 General Information

Sarian OS supports command line interface (CLI). For Digi WR21 router this interface is available using SSH or Telnet network connection, serial interface connection and web interface. OS command interface includes many commands. Command help is available after entering “?” in system console. It simply shows the whole list of available commands. CLI allows changing any parameter of the device and monitoring its state.

Command line handlers in the firmware binary are stored as a static array (see Figure 11).

```
OS_DATA:407971B0 CLI_CMD_INFO <aCocoa_0, CLI_cocoa, 3> ; "cocoa"
OS_DATA:407971BC CLI_CMD_INFO <aDhry_0, CLI_dhry, 3> ; "dhry"
OS_DATA:407971C8 CLI_CMD_INFO <aAna, CLI_ana, 5> ; "ana"
OS_DATA:407971D4 CLI_CMD_INFO <aInsana, CLI_insana, 1> ; "insana"
OS_DATA:407971E0 CLI_CMD_INFO <aChannel_cancel_cleanupDBad+0x20, CLI_id, 3> ; "id"
OS_DATA:407971EC CLI_CMD_INFO <aCrc_0+8, CLI_PrintAllCommands, 4> ; "?"
OS_DATA:407971F8 CLI_CMD_INFO <aMem_0, CLI_mem, 3> ; "mem"
OS_DATA:40797204 CLI_CMD_INFO <aDigihw+4, CLI_hw, 3> ; "hw"
OS_DATA:40797210 CLI_CMD_INFO <aChkst, CLI_chkst, 3> ; "chkst"
OS_DATA:4079721C CLI_CMD_INFO <aTasks, CLI_tasks, 3> ; "tasks"
OS_DATA:40797228 CLI_CMD_INFO <aThreads, CLI_threads, 3> ; "threads"
OS_DATA:40797234 CLI_CMD_INFO <aBufs_0, CLI_bufs, 3> ; "bufs"
```

Figure 11 – Command line handlers in the firmware

Each array element is actually a structure CLI_CMD_INFO with the following fields:

```
00000000 CLI_CMD_INFO     struc ; (sizeof=0xC, mappedto_30)
00000000 Name             DCD ? ; name of the command
00000004 Func             DCD ? ; callback function that handles the command
00000008 Access           DCD ? ; minimum access level required to execute the command
0000000C CLI_CMD_INFO     ends
```

Section 6.5 of this document contains the description of access levels for Sarian OS console commands

All CLI callback functions take argument count (argc) as the first parameter, and the array of parameter strings (argv) as the second parameter.

While observing commands available in the OS, we found a stack overflow vulnerability and an authentication weakness. These issues are described in sections 6.1.2 and 6.1.3 respectively.

6.1.2 CVE-2017-XXXX: Stack Overflow in CLI

Severity: Low

CVSS v2 score: 4.6 (AV:L/AC:L/Au:S/C:N/I:N/A:C)

CVE: Not Assigned

Affected products

Digi WR11, WR21, WR31, WR41, and WR44 routers with firmware version less than 5.2.19.11.

Exploitation conditions:

To exploit this vulnerability, a validated user with access level HIGH or SUPER needs to get access to the command line interface (see section 6.5.1 of this document for explanation of user access levels in the system).

Submit date: 27.07.2017

Official patch date: August 2017

Description

CLI (command line interface) of the device supports command «insana». This command takes two string arguments: str1and str2. These arguments are copied to the local stack buffer, which is 64 bytes long. The following code illustrates how "insana" command handler is implemented:

```
int CLI_insana(int argc, char **argv)
{
  char Buffer[64]; // [sp+4h] [bp-44h]

  if ( argc > 2 )
  {
    OS_sprintf(Buffer, "%s %s", argv[1], argv[2]); // overflow can happen here
    sub_40160260("InsAna", Buffer, 0);
  }
  else
  {
    OS_printf("\r\nFormat: insana <string1> <string2>");
  }
  return 0;
}
```

Lengths of str1 and str2 are not checked and can be larger than 64 bytes. In this case stack overflow occurs, allowing changing return address from the local stack frame and executing arbitrary code.

6.1.3 Weak Internal Authentication in CLI

Details of the issue were reported to vendor on 27.07.2017. Digi claimed that it is possible to mitigate the issue by means of device hardening.

System console commands "optson", "optsoff", "sp", "cnt", and "zing" of Digi WR-21 are protected by an additional challenge-response authentication scheme to prevent users from changing some device's parameters or accessing some of its service functionality. The authentication scheme consists of several simple steps:

1. User enters console command he wants to execute in the following form: “<command_name> start”;
2. System takes time stamp and random 4-byte value, computes md5 hash and sends first 4 bytes of it to the user as a challenge;
3. User enters response (which is also 4-byte number) in the following form: “<command_name> <response>”;
4. System calculates response using secret password "rainbow" and compares it to the response sent by the user. If both responses are equal, system grants access to the command for the user.

If user knows the secret password “rainbow”, which is hardcoded in the device's firmware, he can always calculate a valid response code and get access to any protected command.

We used the following simple script to calculate response based on challenge from the device:

```
import sys
import md5
def cli_auth(challenge):
    password = "rainbow"
    m2 = md5.new()
    m2.update(challenge)
    m2.update(password)
    digest = m2.digest()
    response = ord(digest[0])<< 24 | ord(digest[1])<< 16 |
               ord(digest[2])<< 8 | ord(digest[3])
    return response
if(len(sys.argv) < 2):
    print "Please specify challenge as an argument"
    exit()
response = cli_auth(sys.argv[1])
print "Your response is " + str(response)
```

To perform such authentication bypass, user must have the highest access level in the system – level zero (SUPER). See section 6.5.1 of this document for explanation of user access levels in the system.

6.2 Multitasking in Sarian OS

6.2.1 Tasks

Sarian OS is a multitasking system. Each task has its own stack space, reserved exclusively for its purposes.

All tasks are created at system startup and do not ever terminate or start new tasks. Therefore, the task list for the particular system build is fixed. OS image has static task table inside. Table 4 illustrates the format of a task table entry.

Table 4 – Task table entry format

Field	Type	Offset	Size
Task name	char *	0	4
Signal handling routine pointer	Function pointer	4	4
Main routine pointer	Function pointer	8	4
Stack size	Int	0x0C	4
Task ID	Int	0x10	4

Each task has its own unique identifier – task ID. It is a number from 0x00 to 0xFF. It is mostly used by tasks to communicate with each other using system interprocess communication mechanisms. For the description of interprocess communication mechanisms implemented in Sarian OS, please refer to section 6.2.3 of this document.

Task stack size tells the system how much memory it should reserve for the stack of the task.

The system calls the signal handling routine when a signal is passed to the task. For more information about signaling in Sarian OS, please refer to section 6.2.3.1 of this document.

Main routine is an endless loop that performs all actions of the task. It communicates to other tasks, crates new threads, allocates and deallocates dynamic memory buffers and so on.

A task may not have a main routine. In this case, it does not consume processor time and can only handle incoming signals.

CLI command “tasks” shows the whole list of tasks running in the system. The example output of this command is shown below:

```
NAME         PRI  STATUS      PC        SP       DELAY    MSG_Q
----------   ---  -------  --------  --------   -----    -----
DEBLOG         1  RDY-D    40134f00  40ba4890       4        0
USBISR         2  Q        40149ee8  40ba8780       0        0
USBHOST        3  Q        40149ee8  40ba7778       0        0
Ethernet      13  Q        40149ee8  40ba5f50       0        0
V120          15  Q        40149ee8  40bafe10       0        0
LAPB Link     63  Q        40149ee8  40bacf20       0        0
X25 Layer     66  Q        40149ee8  40badb90       0        0
X25 Switch    68  Q        40149ee8  40baf4a8       0        0
```

```
PPPOE         71  Q         40149ee8  40bc5f70      0        0
BRIDGE        72  Q         40149ee8  40bc67c8      0        0
MULTITX      101  Q         40149ee8  40bac5a8      0        0
REALPORT     102  Q         40149ee8  40bab5e0      0        0
SCRIBATSK    103  Q         40149ee8  40bb3a60      0        0
Modem Call   109  Q         40149ee8  40bc7348      0        0
L2TP         112  Q         40149ee8  40ba90b0      0        0
Power Cont   113  Q         40149ee8  40bcde90      0        0
Tunnel       114  Q         40149ee8  40bd4330      0        0
PPP          115  Q         40149ee8  40bbb318      0        0
QOS          117  Q         40149ee8  40bcd6a8      0        0
TCP          118  Q         40149ee8  40bb82c8     10        0
TCP Utilit   120  Q         40149ee8  40bba2d0      5        0
PANS         123  Q         40149ee8  40bc4ff8      0        0
RADIUS Cli   124  Q         40149ee8  40bce540      0        0
X25 PAD      125  Q         40149ee8  40bae818      0        0
MODBUS       126  Q         40149ee8  40baacd8      0        0
TPAD         127  Q         40149ee8  40bb0768      0        0
FTPCLI       128  Q         40149ee8  40bc01c0      0        0
GPS          129  Q         40149ee8  40bb1098      0        0
SSLCLI       133  Q         40149ee8  40bcaae0      0        0
OSPF         134  Q         40149ee8  40bcba88      0        0
BGP          135  Q         40149ee8  40bcc958      0        0
PPTP         137  Q         40149ee8  40ba99f8      0        0
TACPLUS      138  Q         40428670  40baa3b0      0        0
OpenVPN      139  Q         40149ee8  40bd53a0      0        0
SSH client   141  Q         40149ee8  40bd14e8      0        0
TELITUPD     142  Q         40149ee8  40bb1a18      0        0
SSH server   189  Q         40149ee8  40bcf5e0      0        0
CMD          190  RDY       4015171c  40bca2a0      0        0
FTP          191  Q         40149ee8  40bc11b0      0        0
SMTP         192  Q         40149ee8  40bbf138      0        0
WEB          193  Q         40149ee8  40bbd290      2        0
Async        195  Q         40149ee8  40ba5010      0        0
SCP          196  Q         40149ee8  40bd0568      0        0
CERT         197  Q         40149ee8  40bd2498      0        0
TEMPLOG      198  Q         40149ee8  40bd62c8      0        0
QDL          202  Q         40149ee8  40bd7248      0        0
IDIGISMSD    203  Q         40149ee8  40bb7338      0        0
CloudConne   204  Q         40149ee8  40bd8298      1        0
HealthMetr   205  Q         40149ee8  40bd9240      1        0
BASTSK       241  Q         40149ee8  40bb5a60      0        0
Timer Enti   242  RDY       40149ee8  40ba67f8      0        1
IKE          243  Q         40149ee8  40bc40b8      0        0
DH           245  Q         40149ee8  40bc2938      0        0
FLASH        246  Q         40149ee8  40bc8330      0        0
LOWPRIO      247  Q         40149ee8  40bd3480      0        0
PYTHON       248  Q         40149ee8  40bb63b0      0        0
IDLE         254  RDY       e59f3024  40823f38      0        0
```

Sarian OS has modular structure. The kernel of the system implements only basic functionality for handling tasks, threads and their communications to each other. All other systems, such as file system and networking, are implemented as separate tasks in the system.

6.2.2 Threads

Each task can create one or more child threads. Structures with information about all child threads of a particular task are stored in a list.

For example, threads can be used by some network tasks to handle each client connection separately.

6.2.3 Interprocess Communication

There are two mechanisms in Sarian OS available for the interprocess communication: messages and signals.

6.2.3.1 Signals

Each task has a signal handling routine, which is called by pointer from system task table when a signal is sent to the task. Signal consists of a numeric code and an optional binary payload of arbitrary format. Signal handling routine typically contains a switch expression that checks the signal code and handles it appropriately. Signals are blocking. If a task sends a signal, its execution will continue only after the signal will have been received and properly handled.

Signals can be broadcast. In this case, system calls every signal handler from system task table until it reaches the end of table or until one of the handlers returns result value other than -1.

6.2.3.2 Messages

Tasks can send messages to each other. A message contains message code, destination and source task IDs and an optional payload of arbitrary format.

Every task that has a main routine also has message queue. The system places messages sent to this task to this queue. Main routine of the task waits for incoming messages and handles them in an endless loop.

Messages are non-blocking. Main task routine that sends a message continues to execute immediately.

Sarian OS has a pool of free messages. When a task wants to send a message, it allocates one of free messages from that queue, fills it with data, sender and receiver ID, and sends it. The receiving task handles data in the message and returns it to the queue of free messages.

A message may contain a pointer to a payload. Special objects called system buffers are used as containers for payload of messages. They are allocated and freed by tasks when there is a need to transfer large payload with a message. A small payload can be placed into a message body.

6.3 File System

File system of Digi WR21 device is located in flash memory. Most of file system's functionality is implemented in "FLASH" task.

File system supports three mounting points:

- "/" – root directory of embedded file system;

- “U” – mounting point for removable USB flash devices (Digi routers have USB ports for connecting USB memory devices). It supports memory sticks formatted as FAT32 or FAT16 volumes;
- “S” – mounting point for SD-MMC memory cards.

Root directory has only one subdirectory “/user”. There is no functionality for deleting or creating directories in file system.

Sarian OS API for working with files is similar to the API used in C language standard library. Table 5 contains main functions for working with files in Sarian OS.

Table 5 – Main functions for working with files in Sarian OS

Function Prototype	Function Description
int OS_file_open(char *name, char *mode)	Open file in selected mode.
int OS_file_close(int fd)	Close previously opened file
int OS_file_read(void *buf, int count, int *bytes_read, int fd)	Read previously opened file
int OS_file_write(void *buf, int count, int *bytes_written, int fd)	Write previously opened file

CLI of a Digi WR router has some commands for managing files, for example:

- “type” – prints file contents on the screen;
- “dir” – shows files in current directory;
- “cd” – change current directory;
- “del” – delete selected file.

All files whose names begin with prefix “priv” are considered protected from reading. Such files can be read only if opened with mode flag “k”. However, the system does not provide an interface for reading files with arbitrary mode flags, so only the system can read files starting with “priv” prefix. By default, router stores private SSH and SSL keys in such files protected from reading.

6.4 Networking

Sarian OS has a socket-based API for network communications. The system supports TCP and UDP sockets.

TCP task of the system handles network communications, and TCPUTILS task handles most of service protocols like SNMP, SNTP and DNS. This task handles appropriate sockets for these standard services.

6.5 Users and Access Rights

6.5.1 Access rights

Sarian OS supports up to 15 users. Each user has an access level. The following access levels exist in the system:

- 0 – SUPER;
- 1 – HIGH;
- 2 – MEDIUM;
- 3 – LOW;
- 4 – NONE;
- 5 – W-HIGH R-LOW;
- 6 – W-HIGH R-MED;
- 7 – PARAMETER;
- 8 – READ-ONLY.

CLI command “user” can be used to show users registered in the system and change their passwords and access levels.

6.5.2 Protected System Resources

User access rights are mostly used for differentiation of access to CLI commands. When a user accesses system console providing username and password, the system welcomes him and prints his access level on the screen. Each CLI command has minimum access level required for its execution. If user access level is lower than CLI command’s minimum access level, user cannot execute such command. Access level zero (“SUPER”) provides access to the full set of CLI commands.

File system also supports some restrictions for accessing files. Only users with “SUPER” access level can read some files, for example, “pwds.da0” file.

Different tasks implementing network services can also use authentication mechanism provided by the system to distinguish user access to services like FTP, SSH, web server and so on.

6.5.3 Usernames and Passwords

Usernames in Sarian OS are case-insensitive. They are stored in the configuration file of the device “config.da0”.

Passwords are stored in “pwds.da0” file in encrypted and base64-encoded form. Here is the example of pwds.da0 file contents:

```
config last_saved_safe "16:10:00, 10 Aug 2017"
config last_saved_safe_changes "1"
config last_saved_safe_user "WEB 20"
addp 0 epassword "KD5lSVJDVVg="
snmpuser 0 eCommunity "KCp0VkxP"
user 1 epassword "KD5lSVJDVVg="
user 2 epassword "LSxzSBc="
```

There are two encryption modes for user passwords:

1. XOR with the keystream hardcoded in the firmware;

2. Encrypting with AES-192.

In the first mode, passwords are XOR-ed against hardcoded gamma shown at Figure 12. For long passwords, the keystream is used in cycle.

```
OS_DATA:4079419B                           DCB    0
OS_DATA:4079419C        PWD_XOR_KEY        DCB 0x58, 0x5F, 0x16, 0x3A, 0x25, 0x2C, 0x27, 0x3C, 0x32
OS_DATA:4079419C                                                   ; DATA XREF: PasswordXOR+C↑o
OS_DATA:4079419C                                                   ; PasswordXOR:loc_40127A64↑r ...
OS_DATA:4079419C                           DCB 0x3D, 0x21, 0x7D, 0x36, 0x7C, 0x75, 0x64, 0x6C, 0x6C
OS_DATA:4079419C                           DCB 0x6C, 0x75, 0x37, 0x2C, 0x75, 6, 0x34, 0x27, 0x3C
OS_DATA:4079419C                           DCB 0x34, 0x3B, 0x75, 6, 0x2C, 0x26, 0x21, 0x30, 0x38
OS_DATA:4079419C                           DCB 0x26, 0x75, 0x78, 0x75, 0x14, 0x39, 0x39, 0x75, 7
OS_DATA:4079419C                           DCB 0x3C, 0x32, 0x3D, 0x21, 0x26, 0x75, 7, 0x30, 0x26
OS_DATA:4079419C                           DCB 0x30, 0x27, 0x23, 0x30, 0x31, 0x58, 0x5F
```

Figure 12 – Key stream for password protection is hardcoded in the firmware

For the second protection mode, the encryption scheme can be described using the following pseudocode:

```
void Get_BIOS_Secret(char *secret, int BitCount = 192)
{
      char random_data[16];
      OS_rand(random_data, 8);
      OS_rand(random_data + 8, 8);
      char serial_number[4]; //device serial number, 4 bytes
      char mac_address[6]; //device MAC address
      char hw_revision[6]; //device hardware revision, 6 bytes
      char bios_data[0x10];
      memcpy(bios_data, mac_address, 6);
      memcpy(bios_data + 6, serial_number, 4);
      memcpy(bios_data + 10, hw_revision, 6);

      random_data_hash = sha1(random_data, 0x10);
      random_data_hash_2 = sha1(random_data_hash, 0x14);
      bios_hash = sha1([bios_data, random_data_hash_2], 0x24);
      for(int i = 0; i < 4; i++)
          result[i] = bios_hash[i];
      for(int i = 0; i < 0x10; i++)
          result[i + 4] = random_data_hash[i] ^ random_data_hash_2[4 + i];
      for(int i = 0; i < 4; i++)
          result[0x14 + i] = random_data_hash[0x10 + i];
}
char aes_key[0x18];
char expanded_key[...];
char aes_iv[16];
char epasswd[...];
char dst[...];
Get_BIOS_Secret(aes_key, 192);
OS_rand(iv, 16);
aesEncrypt(password, epasswd + 16, password_len, aes_key, aes_iv, 1);
memcpy(epasswd, aes_iv, 16);
dst[0] = '.';
dst[1] = '0';
dst[2] = '5';
dst[3] = ';';
Base64Encode(dst + 4, epasswd, length);
//dst contains encrypted password string
```

As one can see, passwords are encrypted with AES with the length of the key equal to 192 bits and the block size equal to 128 bits. After encryption, the concatenation [encrypted_password, initial_vector] is encoded with base64 and stored in “pwds.da0” file.

The 192-bit secret key is generated using a 16-byte pseudo-random sequence and some constant information about the device, which is actually concatenation of device MAC address, serial number and hardware version. Once generated, it is stored in the device’s flash memory and used for encrypting and decrypting passwords.

To decrypt passwords protected by the second encryption scheme, it is needed to guess a pseudo-random 16-byte value, or read the AES-key somehow from the device’s flash memory. It is stored in a raw mode, outside of file system, in the 512-th page of NAND memory. Reading the secret key is possible using the BIOS console, described in section 8 of this document. Fortuna algorithm is used to generate pseudo-random values in the system. The implementation of Fortuna in Sarian OS showed good properties of the output stream. See section 6.7 for the details.

Users of the router can select the encryption mode for their passwords in the device’s configuration. The first mode is the default option for the router.

Security policies for user passwords are not implemented.

6.5.4 User Authentication

Regardless of chosen protection scheme for passwords, usernames, passwords and access levels of all users registered in the system are always stored in plaintext in RAM of the device. Usernames, passwords and access levels in RAM are used to distinguish users' access to CLI, files and services like FTP, WEB, SSH, Telnet and so on. Checking user’s password equals to simple comparing of strings. Encryption and decryption of passwords are used only when passwords are stored to “pwds.da0” file in file system or loaded from it to RAM.

In addition to the standard system authentication, Digi WR routers support authentication through TACPLUS server or RADIUS server. Server-side authentication can replace system authentication. To enable this kind of authentication, a router must be configured accordingly. TACPLUS and RADIUS tasks implement server-side authentication in the router’s system.

6.5.5 Weak Password Requirements

Details of the issue were reported to vendor on 14.09.2017. Digi claimed that it is possible to mitigate the issue by means of device hardening.

Operating system that runs on the router does not include any password policy, so users of the router can have arbitrary passwords, including weak and well-known passwords. Such passwords sometimes can be easily guessed by attackers using special tools for brute-forcing credentials.

To improve router's security and make it significantly harder for attackers to guess passwords using brute-force attacks, strong passwords usage should be enforced. A password strength policy should contain the following attributes:

1. Minimum and maximum length;
2. Require mixed character sets (alpha, numeric, special, mixed case);
3. Do not contain user name;
4. Expiration;
5. No password reuse.

6.5.6 Hardcoded Secret for Credentials Encryption

Details of the issue were reported to vendor on 14.09.2017. Digi claimed that it is possible to mitigate the issue by means of device hardening.

Account passwords in the router's system are stored in the file "pwds.da0" in one of two forms:

- XOR-ed with hardcoded in the device's firmware key, which is the default option;
- Encrypted with AES-192 using a secret key, which is stored inside flash memory in raw format outside of the device's file system.

In any case, passwords are encoded using base-64 after encryption.

File "pwds.da0" can be accessed by any system user with the lowest access level through RCI service. This fact will be covered in details in section 9.5.4.2 of this document. In addition to RCI service, other ways may exist to obtain this file. If the first password protection option is used, then user passwords from the file can be easily decrypted.

The following Python script illustrates how to decode user passwords, if XOR-based protection is used:

```
import base64
import array
def pwd_decrypt_xor(base64_pass):
	key =  [0x58, 0x5F, 0x16, 0x3A, 0x25, 0x2C, 0x27, 0x3C,
		0x32, 0x3D, 0x21, 0x7D, 0x36, 0x7C, 0x75, 0x64,
		0x6C, 0x6C, 0x6C, 0x75, 0x37, 0x2C, 0x75, 0x06,
		0x34, 0x27, 0x3C, 0x34, 0x3B, 0x75, 0x06, 0x2C,
		0x26, 0x21, 0x30, 0x38, 0x26, 0x75, 0x78, 0x75,
		0x14, 0x39, 0x39, 0x75, 0x07, 0x3C, 0x32, 0x3D,
		0x21, 0x26, 0x75, 0x07, 0x30, 0x26, 0x30, 0x27,
		0x23, 0x30, 0x31, 0x58, 0x5F, 0x00, 0x00, 0x00]
	xor_pass = array.array("b", base64.b64decode(base64_pass))
	for i in range(0, len(xor_pass)):
		xor_pass[i] ^= key[i % (len(key) - 3)]
	return xor_pass.tostring()
```

AES encryption is the default choice for user who configures the device with "Quick Start Wizard", but there is an option to skip the wizard and do the setup manually. In that case, the weak password encryption scheme is the default option.

6.5.7 Permanent Storage of Credentials in RAM

Details of the issue were reported to vendor on 14.09.2017. Digi claimed that it is possible to mitigate the issue by means of device hardening.

Regardless of protection scheme used, user names and passwords are permanently stored in device's RAM in plain text. Figure 13 illustrates this fact.

```
40856f00 00 00 00 00 00 00 00 00  af 31 0c 00 75 73 65 72   |.........1..user|
40856f10 6e 61 6d 65 00 00 00 00  00 00 00 00 00 00 00 00   |name............|
40856f20 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40856f30 00 00 00 00 00 70 61 73  73 77 6f 72 64 00 00 00   |.....password...|
40856f40 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40856f50 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40856f60 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40856f70 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40856f80 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40856f90 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40856fa0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40856fb0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40856fc0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40856fd0 00 00 00 00 00 00 00 00  00 01 00 00 01 00 00 00   |................|
40856fe0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40856ff0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857000 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857010 00 00 00 00 00 00 00 00  75 73 65 72 32 00 00 00   |........user2...|
40857020 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857030 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857040 00 54 6f 70 53 65 63 72  65 74 50 61 73 73 77 6f   |.TopSecretPasswo|
40857050 72 64 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |rd..............|
40857060 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857070 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857080 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857090 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
408570a0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
408570b0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
408570c0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
408570d0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
408570e0 00 00 00 00 00 01 00 00  01 00 00 00 00 00 00 00   |................|
408570f0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857100 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857110 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857120 00 00 00 00 75 73 65 72  33 00 00 00 00 00 00 00   |....user3.......|
40857130 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857140 00 00 00 00 00 00 00 00  00 00 00 00 00 59 6f 75   |.............You|
40857150 57 69 6c 6c 4e 65 76 65  72 47 75 65 73 73 40 26   |WillNeverGuess@&|
40857160 23 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |#...............|
40857170 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
```

Figure 13 – Dump of RAM memory with system usernames and passwords

Note: the way of reading (writing) data from (to) RAM memory will be discussed in section 8 of the document.

This way of storing credentials may potentially lead to password encryption bypass.

6.6 Memory Management

6.6.1 RAM and NVRAM

Digi WR21 firmware uses two global memory pools:

1. RAM global pool – starts right after code area (for Digi WR21 with firmware version 5.2.17.12 it is 0x40c58400) and ends at address 0x47FE0000 ~ 115 Mbytes;
2. NVRAM global pool – starts at 0x47FE0004 and ends at 0x48000000 ~ 128 Kbytes.

The system initializes these pools at startup. Memory from these pools is allocated using the following system function:

```
//Allocates dwCount * 4 bytes from RAM or NVRAM depending on memType value
OS_dw_alloc(int memType, unsigned int dwCount)
```

Memory from global pools is allocated permanently. There is no way to free once allocated memory buffer from one of these pools.

6.6.2 General Dynamic Memory Pool

The system allocates large dynamic pool from RAM for use by tasks. Any task can allocate memory in it, use it and then free it. The system provides API for working with this pool. Table 6 contains the description of main API functions.

Table 6 – System API for working with dynamic memory

Prototype	Description
void *OS_smalloc(int bytes)	Allocate memory and initializes it with zeroes
void OS_sfree(void *buf)	Free previously allocated memory
void * OS_realloc(void *buf, int newSize)	Reallocates previously allocated buffer.

All buffers are 8 byte aligned. Before each allocated buffer there is 8-byte header containing size of the buffer and other maintenance data for system memory manager.

6.6.3 Special Pools

Tasks can allocate permanent memory buffers using *OS_dw_alloc* system call. Therefore, it is possible for a task to create its own heap. Some tasks in Digi WR21 firmware, like USB handling task and Python interpreter task use this opportunity and work with their own heaps.

6.7 Built-in System PRNG

There are two PRNG implemented in Sarian OS – simple linear congruential generator and Fortuna generator. By default, the Fortuna generator is used.

Fortuna is a cryptographically secure PRNG devised by Bruce Schneier and Niels Ferguson and published in 2003.

Random numbers are obtained from the system using *OS_rand(int byteCount)* API call.

Built-in Python interpreter implements *os.urandom()* function, which internally calls *OS_rand()* function. Therefore, it is not difficult to generate pseudorandom sequence of any length on the device using Python. It is then possible save it to file on the device, copy it to PC and analyze quality of random data with standard PRNG tests.

In this research, we generated the pseudorandom sequence of eight Mbytes on the device. Then we tested it using free "ENT" program available at the following link: http://www.fourmilab.ch/random/.

Figure 14 illustrates the result of our test.

```
C:\Tools\random>ent.exe rand.bin
Entropy = 7.999980 bits per byte.

Optimum compression would reduce the size
of this 8388608 byte file by 0 percent.

Chi square distribution for 8388608 samples is 236.39, and randomly
would exceed this value 79.25 percent of the times.

Arithmetic mean value of data bytes is 127.4819 (127.5 = random).
Monte Carlo value for Pi is 3.138834748 (error 0.09 percent).
Serial correlation coefficient is 0.000155 (totally uncorrelated = 0.0).
```

Figure 14 – Testing built-in system PRNG

As one can see, the quality of pseudorandom data is high. Therefore, it can be concluded that Fortuna implementation in Sarian OS provides a good level of security for generating cryptographic keys used in the system.

6.8 System Log

6.8.1 System Log Messages

System logs on the device are stored in the file "eventlog.txt". Sarian OS has an API function *OS_Log(int event, int reason, ...)*. File "logcodes.txt", included in device's firmware, contains all possible event and reason codes passed to *OS_Log()* function.

6.8.2 Debug Messages

Tasks in the system can produce debug messages, if debug output is turned on. To turn on the debug output the following console commands can be used:

- debug 0 – enable debug messages and display them in the console available by device's serial port;
- debug T – enable debug messages and display them in the console available by device's Telnet service.

A console used as debug message output can still be used to enter commands. Therefore, there is no need to use two connections to the device by different channels for entering commands and viewing the debug output.

Some services of a router have additional debug parameters that must be switched on to start generating debug output for a specific service.

7 Built-in Python

7.1 Built-in Functions and External Library Modules

Digi WR21 firmware contains version 2.6.1 embedded Python interpreter. Its main purpose is to provide users a way to extend router's functionality.

Library modules for Python are included in device's firmware update pack as "python.zip" archive. This archive contains compiled Python modules only, but it is easy to decompile them using any of Python decompilers available in Internet.

Latest information about using Python in Digi devices is available at "Digi Python Wiki Archive" (http://cms.digi.com/resources/documentation/digidocs/90001537/).

7.2 Wizards

Wizards are Python modules that extend web server functionality of Digi device. They provide a simple way of configuring a Digi device for typical use cases through an embedded web server.

Wizard modules are placed in the "wizards.zip" file that is included in a device's firmware. They register callback routines in the device's web server. All web requests not handled by the web server are passed to Python task that invokes wizards to handle the request.

8 BIOS Console

During the research, it turned out that the Sarian BIOS implements a user console with rather interesting abilities. In this section of the document, we will describe some of these abilities.

8.1 Entering BIOS Console

To access the Sarian BIOS console one needs to connect to the device using the serial port and a terminal program like "minicom" on Linux or "PuTTY" on Windows. Default serial parameters are 115200 8N1. Then, right after applying power to the device or initiating a reboot, one should type acute symbol ("`") without spaces or new lines. After that, the following message appears in the console:

```
ARM Sarian Bios Ver 7.59u
11:44:43, 28 Jul 2017
?>
```

The next step is providing password for entering BIOS. The password is hardcoded in BIOS and is equal to "ytrewq".

After providing correct password and entering BIOS, it is possible to use the following console commands:

```
adump    <addr> [<len>]             - dump memory in ascii strings
attrib   <file> ro|rw               - modify file attributes
blank    <addr> <len>               - check if memory blank
mmu      on|off                     - MMU control
dcache   flush|inval <addr> [<len>]- data cache ops
cache    I|D|ID on|off              - cache control
copy     <from> <to>                - copy a flash file
compare  <addr1> <addr2> <len>      - compare memory range
console  <speed> <DPS>              - set console port parameters
crc      <addr> <len>               - calc crc on memory range
dmpreg                              - dump regs from last illegal exception
dump     <addr> [<len> [b|w|l]]     - display memory contents
db       <addr> [<len>]             - byte display of memory
defrag                              - defragment flash memory
delete   <file>                     - delete a flash file
dir                                 - directory of flash
disassem <addr> [<len>]             - disassemble memory
dhry     <loops>                    - dhrystone test
dl       <addr> [<len>]             - long display of memory
dw       <addr> [<len>]             - word display of memory
edit     <addr> [b|w|l [<value>]]   - edit memory
exit                                - exit monitor
erase    <addr>                     - erase flash segment
eraseall                            - erase entire flash
fill     <start> <len> <val>        - fill a range of memory
free                                - show free flash space
scan                                - scans flash dir for errors
go       [<addr>]                   - execute from address
info                                - display compilation settings
inum     [<num>]                    - view/set image boot number
partnum  [<num>]                    - view/set part number
hwnum    [<num>]                    - view/set hardware revision
```

```
hwver    [<ver>]                    - view/set hardware version
km                                  - invalidate the dir mirror
led      <lednum> on|off|red|...    - write to the leds
look     <startaddr>                - monitor value of addresses
map                                 - map of flash usage
mcopy    <src> <dest> <len>         - copy memory range
move     <src> <dest> <len>         - read flash to RAM
query                               - flash fragmentation query
rename   <from> <to>                - rename a flash file
reset                               - hard reset
puk      <puk> [<PUK>]              - PUK
sn       <sn> [<MAC0> [<MAC1>] ...]- serial nb & MAC addrs
sn2      <sn2>                      - serial 2
tempcal  [<offset>]                 - view/set temperature calibration
time     [hh mm ss dd mm yyyy]      - set time and date
touch                               - update file time/date
type     <filename> [pause]         - dump file contents in ascii
upload   all                        - upload flash/memory contents
verbose                             - toggle verbose mode
wipe                                - wipe free flash to 0xff
write    <src> <dest> <len>         - write data to flash
xm       [<dest addr>]              - xmodem bios update
xma                                 - xmodem full system update
xmboot                              - xmodem boot rom code to boot flash
xmodem   <addr>                     - start xmodem download
xmodemu  <hex-addr> <hex-size>      - start xmodem upload
?        [<search string>]          - display help
tftp     <file> <from ip> <our ip> { tftp port }- TFTP a file
tftps    <file> <from ip> <our ip> { tftp port }- TFTP send a file
bufs                                - bufs
ethprt   <port>                     - set default eth port (0=any)
ethspd   [0|10|100]                 - set/show eth speed (0=auto)
syoff                               - disable SYNC ports until next reboot
adsloff                             - disable ADSL port until next reboot
bist     ...                        - initiate built-in self-tests
bcb                                 - write boot control block(s)
nand                                - show nand flash info
pbutton  <period> <trans> <t'out>   - monitor pushbutton transitions
pbcount  [<reset>]                  - count pushbutton transitions
i2cwrite <hexaddr> <len> <hexdata> - write to I2C device
i2cread  <hexaddr> <len>            - read from I2C device
```

BIOS console allows controlling device hardware, writing code to RAM and passing control to it, making changes to file system, reading and writing flash memory in raw mode and so on. Reading and writing flash memory is performed using “nand” command with subcommand “read” and “write” respectively.

8.2 NAND Dump

It is possible to create a dump of the entire flash memory (128 Mbytes) of the device. NAND dump can serve as a backup if something goes wrong with the device. It can be done using the following console command:

```
tftps .all [our_ip_address] [digi_ip_address]
```

This command assigns “digi_ip_address” to the router and starts TFTP file transfer from it to the specified “our_ip_address”. The router compresses data before sending it to the network.

8.3 Disabling the Watchdog Timer

While solving the problem of continuous resets of the router while trying to debug through JTAG interface, which was mentioned in section 4.2 of the document, we found an interesting string in BIOS image (see Figure 15):

```
v8 = sub_40001D28(v5, HW_CLKCTRL_HBUS & 0x1F);
BIOS_printf("clk_p (ARM core):     %ld MHz\r\n", v5);
BIOS_printf("clk_h (AHB/APBH):     %ld MHz\r\n", v8);
BIOS_printf("clk_emi (DDR):        %ld MHz\r\n", v7);
v9 = sub_400051B4();
BIOS_printf("Boot port:            %d\r\n", v9);
if ( *(_DWORD *)&byte_40028FD8 )
  v10 = "external";
else
  v10 = "internal";
BIOS_printf("Async clock:          %s", v10);
if ( *(_DWORD *)&byte_40028FD8 )
  BIOS_printf(" (%d Hz)\r\n");
else
  BIOS_printf("\r\n");
BIOS_printf("H/W Watchdog enabled: %s\r\n", "yes");
if ( byte_40028FDC )
  v11 = "yes";
else
  v11 = "no";
BIOS_printf("RS485 detected:       %s\r\n", v11);
BIOS_printf("Bios load address     %p\r\n", 0x40000000);
BIOS_printf("User files active     %s\r\n", "yes");
BIOS_printf("Size of FLASHSECT     %d\r\n", 12);
BIOS_printf("Size of FLASHFILE     %d\r\n", 36);
BIOS_printf("Size of FLASHDIR      %d\r\n", FLASHDIR_SIZE);
BIOS_printf("Flash dir reserved    %d\r\n", FLASHDIR_RSVD);
BIOS_printf("Size of FLASH mem     %d\r\n", (FLASH_SIZE - 0x100000) >> 14 << 14);
BIOS_printf("Size of FLASH device  %d\r\n", FLASH_SIZE);
BIOS_printf("Flash block size      %d\r\n", FLASH_BLOCK_SIZE << 10);
BIOS_printf("Max FLASH files       %d\r\n", FLASH_MAX_FILES);
BIOS_printf("Size of SDRAM         %d\r\n", SDRAM_SIZE);
```

Figure 15 – BIOS code prints hardware configuration parameters to the screen

We checked the i.MX287 CPU reference and figured out that bit 4 of CPU register HW_RTC_CTRL is responsible for enabling and disabling the watchdog timer. The address of HW_RTC_CTRL inside the CPU address space is 0x80056000. In order to clear the bit, we need to write "1" to the forth bit of four-byte register at address 0x80056008 (HW_RTC_CTRL_CLR), which is actually an interface for clearing bits in HW_RTC_CTRL:

```
?>
>>dump 80056000 4
80056000 08 00 00 10
>>edit 80056008 l 00000010
>>dump 80056000 4
80056000 08 00 00 00
>>exit
Boot bios active!
```

After this action, the device stopped resetting after halting by JTAG debugger, and we gained the ability to debug the firmware.

8.4 Adding Memory Reading and Writing Capabilities to Sarian OS Console

BIOS CLI handlers in the firmware binary are organized as an array (see Figure 16).

```
BIOS_DATA:400282B0 BIOS_CLI_CMD_INFO <BIOS_CLI_atrib, aAttrib, 3, 3, aModifyFileAttr, \
BIOS_DATA:400282B0                    aFileRoRw>
BIOS_DATA:400282B0 BIOS_CLI_CMD_INFO <BIOS_CLI_blank, aCheckIfMemoryB+0x10, 3, 3, \
BIOS_DATA:400282B0                    aCheckIfMemoryB, aAddrLen_0>
BIOS_DATA:400282B0 BIOS_CLI_CMD_INFO <BIOS_CLI_mmu, aMmu, 2, 2, aMmuControl, aOnOff>
BIOS_DATA:400282B0 BIOS_CLI_CMD_INFO <BIOS_CLI_dcache, aDcache, 3, 4, aDataCacheOps, \
BIOS_DATA:400282B0                    aFlushInvalAddr>
BIOS_DATA:400282B0 BIOS_CLI_CMD_INFO <BIOS_CLI_cache, aCache, 3, 3, aCacheControl, \
BIOS_DATA:400282B0                    aIDIdOnOff>
BIOS_DATA:400282B0 BIOS_CLI_CMD_INFO <BIOS_CLI_copy, aCopy, 3, 4, aCopyAFlashFile, \
BIOS_DATA:400282B0                    aFromTo>
BIOS_DATA:400282B0 BIOS_CLI_CMD_INFO <BIOS_CLI_compare, aCompare, 4, 4, aCompareMemoryR, \
BIOS_DATA:400282B0                    aAddr1Addr2Len>
BIOS_DATA:400282B0 BIOS_CLI_CMD_INFO <BIOS_CLI_console, aConsole, 3, 3, aSetConsolePort, \
BIOS_DATA:400282B0                    aSpeedDps>
```

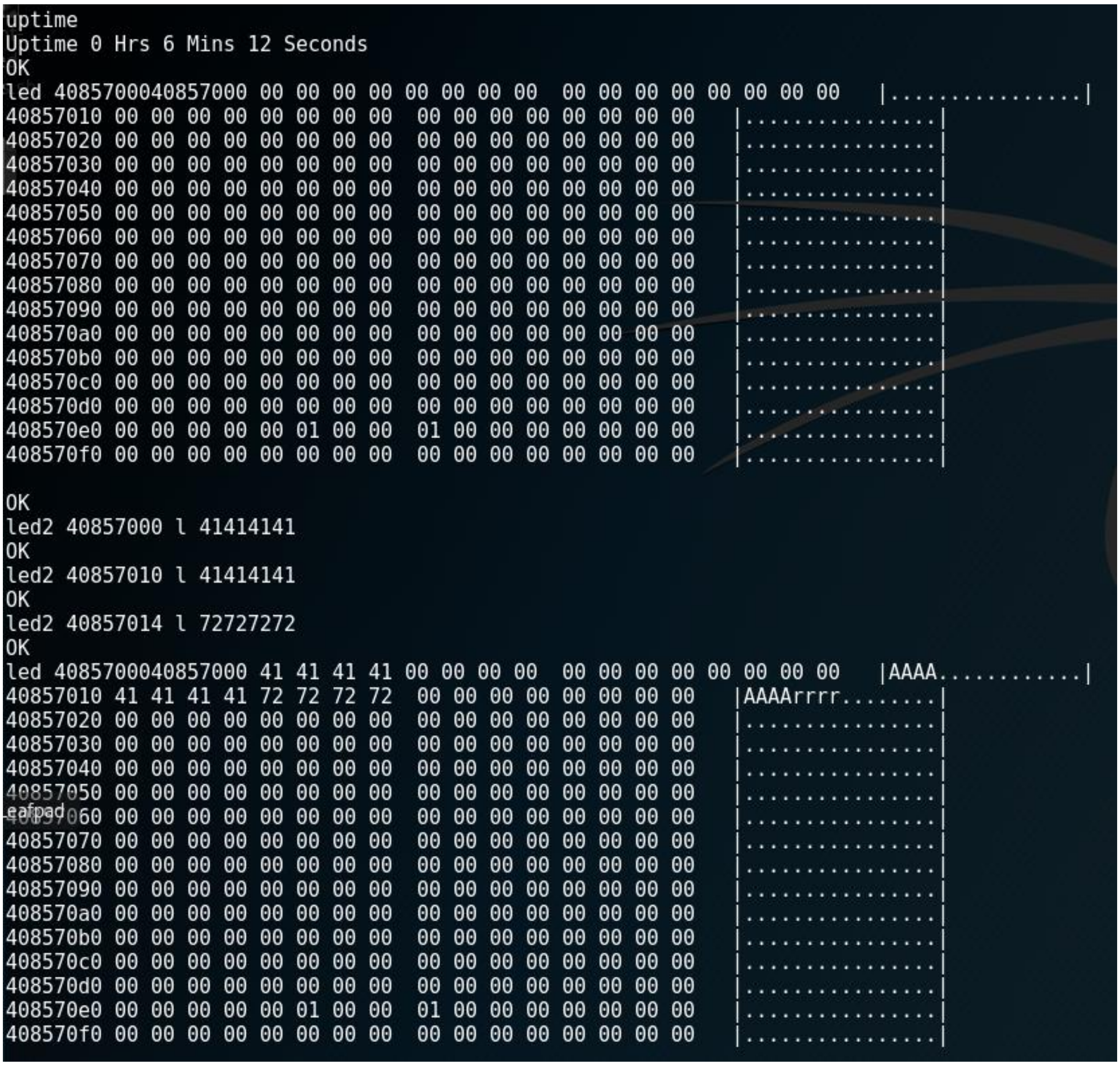
Figure 16 – BIOS command line handlers in firmware

Each array element is a structure BIOS_CLI_CMD_INFO with the following fields:

```
00000000 BIOS_CLI_CMD_INFO struc ; (sizeof=0x18, mappedto_131)
00000000 Function         DCD ? ; callback function that handles the command
00000004 Name             DCD ? ; name of the command
00000008 field_8          DCD ?
0000000C field_C          DCD ?
00000010 Help             DCD ?
00000014 HelpParams       DCD ?
00000018 BIOS_CLI_CMD_INFO ends
```

The most important thing here is the fact that all CLI callback functions take argument count (argc) as the first parameter, and the array of parameter strings (argv) as the second parameter. Therefore, callback function prototype used in BIOS is similar to that used in OS.

BIOS console allows writing code to RAM and executing it. This fact in conjunction with the lack of verification of firmware authenticity (see section 5.2.2 of this document for explanation of this fact) allows loading a modified OS image to RAM and starting its execution using the BIOS console. In this research, we used it to add memory reading and writing console commands to Sarian OS. We modified callback pointers of unused commands “led” and ”led2” to point to “dump” and “edit” BIOS handlers respectively. This allowed us to watch RAM memory contents of working Sarian OS. Figure 17 demonstrates how memory reading and writing works in the system after our patch.



```
uptime
Uptime 0 Hrs 6 Mins 12 Seconds
OK
led 4085700040857000 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857010 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857020 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857030 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857040 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857050 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857060 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857070 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857080 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857090 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
408570a0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
408570b0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
408570c0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
408570d0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
408570e0 00 00 00 00 00 01 00 00  01 00 00 00 00 00 00 00   |................|
408570f0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|

OK
led2 40857000 l 41414141
OK
led2 40857010 l 41414141
OK
led2 40857014 l 72727272
OK
led 4085700040857000 41 41 41 41 00 00 00 00  00 00 00 00 00 00 00 00   |AAAA............|
40857010 41 41 41 41 72 72 72 72  00 00 00 00 00 00 00 00   |AAAArrrr........|
40857020 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857030 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857040 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857050 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857060 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857070 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857080 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
40857090 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
408570a0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
408570b0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
408570c0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
408570d0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
408570e0 00 00 00 00 00 01 00 00  01 00 00 00 00 00 00 00   |................|
408570f0 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   |................|
```

Figure 17 – Reading and writing memory in OS CLI

9 Network Services

As any industrial routers, Digi WR router provides many different network services. Security and safety of these services play a crucial role in the overall security of a router. Therefore, in this research we studied most of available services in order to find potential weaknesses and vulnerabilities. This section of the document contains results of analyzing network services of the device. The analysis of services is based on the knowledge of OS internals documented in section 6 of this paper.

9.1 List of Network Services

To verify results of network scanner, one can use console commands “gpstat”, “socks” and “socks udp” on the device side. These commands show the list of TCP and UDP sockets opened on the device. “gpstat” command shows general-purpose sockets available in the system.

The list of services is presented in the table below. Further document describes selective

analysis of services.

Table 7 – List of network services available on the device

Port	Service	Comment
TCP 21 (8021)	FTP	Provides remote access to file system of the device
TCP 22 (8022)	SSH	Provides secure remote access to system console of the device
TCP 23 (8023)	Telnet	Provides remote access to system console of the device
TCP 80 (8080)	HTTP	“GoAhead” embedded web server
TCP 443	HTTPS	“GoAhead” embedded web server. By default, only HTTP service is active. But it is possible to turn on HTPS in the device settings
TCP 4000 (12000)	ASY 0	If a device is connected to the serial interface of the router, then connection to this network port initiates data forwarding from the serial device to network and from network to the device. For example, if a terminal program is connected to the serial port of the rooter and a network client is connected to TCP port 4000 of the router. Then all data sent by client will be printed in the terminal screen, and all data entered in the terminal will be sent to the client by network.
TCP 4001 (12001)	ASY 1	-
TCP 4002 (12002)	ASY 2	After connection the router sends string “reserved for use by modem” and immediately closes the connection
TCP 4003 (12003)	ASY 3	-

TCP 4004 (12004)	ASY 4	After connection the router sends string “reserved for use by modem” and immediately closes the connection
TCP 4005 (12005)	ASY 5	This service provides access to AT command interface of Huawei modem
TCP 4006 (12006)	ASY 6	-
TCP 4007 (12007)	ASY 7	-
TCP 4008 (12008)	ASY 8	-
TCP 4009 (12009)	ASY 9	-
UDP 53	DNS	-
UDP 67	DHCP Server	-
UDP 161	SNMP	-
UDP 500	IKE	This protocol is used by IPsec VPN
UDP 2362	ADDP	Proprietary Digi protocol for discovering Digi devices in the network and configuring them automatically.
UDP 4052	Backup IP Service	Proprietary Digi protocol for exchanging data about availability of different hosts in the network between Digi routers
UDP 4500	IPsec	IPsec NAT Traversal (VPN)

We enumerated services on the device from Ethernet network and from cellular network to confirm that they are identical.

9.2 FTP Service

9.2.1 General Description

FTP service allows browsing, renaming, copying and deleting files in file system of the device. To access this service, authentication is required. Only Ftp commands "USER", "PASS", "HELP" and "QUIT" are available before authentication.

Anonymous user is disabled by default, and can be turned on manually.

FTP service in the system is implemented by FTP task.

9.2.2 CVE-2017-XXXX: Stack frame corruption in FTP service

Severity: Medium to low

CVSS v2 score: 6.8 (AV:N/AC:L/Au:S/C:N/I:N/A:C)

CVE: Not Assigned

Affected products

Digi WR11, WR21, WR31, WR41, and WR44 routers with firmware version less than 5.2.19.6.

Exploitation conditions:

To exploit the vulnerability, an attacker must either have user credentials for the router, or use anonymous access, which is disabled by default in the configuration of the router.

Submit date: 27.07.2017

Official patch date: August 2017

Description

FTP command “TYPE” with an argument other than B, b, I, i, L, l, causes the router to crash. It happens because the following code is used to send an error string to the client:

```
ftpSend(socket_id, "501 Unknown type \"%s\"\r\n")
```

As one can see, there must be a third parameter in this call, but it does not exist. As a result, the stack corrupts after this call and the system crashes.

The “TYPE” command is available without user authentication only if anonymous access is allowed, which is not true by default. Therefore, the severity of this bug can be considered “medium to low”.

9.3 SSH Server

9.3.1 Turning On SSH Server Debug Messages

Digi SSH server provides a lot of helpful debug output. To turn it on, one should execute the following command sequence in system console:

```
ssh 0 debug ON
debug 0
```

The first command activates debug output from SSH server, and the second one enables debug output from any source to serial port. To view debug messages, one should connect a terminal to serial port of the device.

9.3.2 Key Storage Security

SSH server private key is stored as a file in file system of the router. By default, it is stored in the file “privSSH.pem”.

Files whose name starts from “priv” prefix are not available for reading neither in the system nor in BIOS. This mechanism protects secret keys from compromising. It is based on the simple name check in *file_open(char *filename, char *mode)* function. Firmware contains two hardcoded lists of prefixes that are protected from reading by users. If the file name starts from one of prefixes from the first list, then *file_open()* function expects that there is “k” flag in *mode* string. If it is not true, the function returns error. The same principle works for prefixes from the second list except that the “p” flag is required.

Firmware v. 5.2.17.12 contains only “priv” prefix in the first list and “pwds.” prefix in the second list.

Figure 18 shows the code that checks prefixes.

```
if ( !isWrite )                          // No 'w' in file open mode str
{
  if ( modeFlags )                       // True if 'r', 'p' or 'k' in mode str
  {
    if ( (v11 = BIOS_PrivFileCheck(fileName), v12 = BIOS_PwdsFileCheck(fileName), !(v12 | v11))
      || v11 && modeFlags & 2            // Set if 'k' in mode str ("key")
      || v12 && modeFlags & 4 )          // Set if 'p' in mode str ("password")
    {
      result = 0;                        // We want to get here
      dword_4006F744 = 0;
      return result;
    }
  }
  return -1;
}
```

Figure 18 – The code for checking file prefixes

Functions *BIOS_SecureFileCheck()* and *BIOS_PwdsFileCheck()* differ only in prefix lists used, therefore we show only one of them as an example at Figure 19.

```
signed int __fastcall BIOS_PrivFileCheck(char *fileName)
{
  char *curPref; // r4 MAPDST
  char **prefs; // r5
  int v5; // r0

  curPref = KEY_FILE_PREFIXES;
  if ( !KEY_FILE_PREFIXES )
    return 0;
  prefs = &KEY_FILE_PREFIXES;
  while ( 1 )
  {
    v5 = strlen(curPref);
    if ( !BIOS_strcmp_lower(curPref, fileName, v5) )
      break;
    curPref = prefs[1];
    ++prefs;
    if ( !curPref )
      return 0;
  }
  return 1;
}
```

Figure 19 – BIOS_SecureFileCheck() function

Figure 20 shows prefixes used for comparison.

```
BIOS_DATA:400281D4 KEY_FILE_PREFIXES DCD aPriv           ; DATA XREF: BIOS_PrivFileCheck+C↑r
BIOS_DATA:400281D4                                       ; "priv"
BIOS_DATA:400281D8 dword_400281D8  DCD 0                 ; DATA XREF: BIOS_PrivFileCheck:loc_4000CED4↑r
BIOS_DATA:400281DC PWDS_FILE_PREFIXES DCD aPwds_         ; DATA XREF: BIOS_PwdsFileCheck+C↑r
BIOS_DATA:400281DC                                       ; "pwds."
BIOS_DATA:400281E0 dword_400281E0  DCD 0                 ; DATA XREF: BIOS_PwdsFileCheck:loc_4000CF3C↑r
```

Figure 20 – Secure file prefix tables

To read these files, we filled prefix tables with zeroes using “edit” console command from the BIOS. After that, we were able to read the key file, as shown at Figure 21.

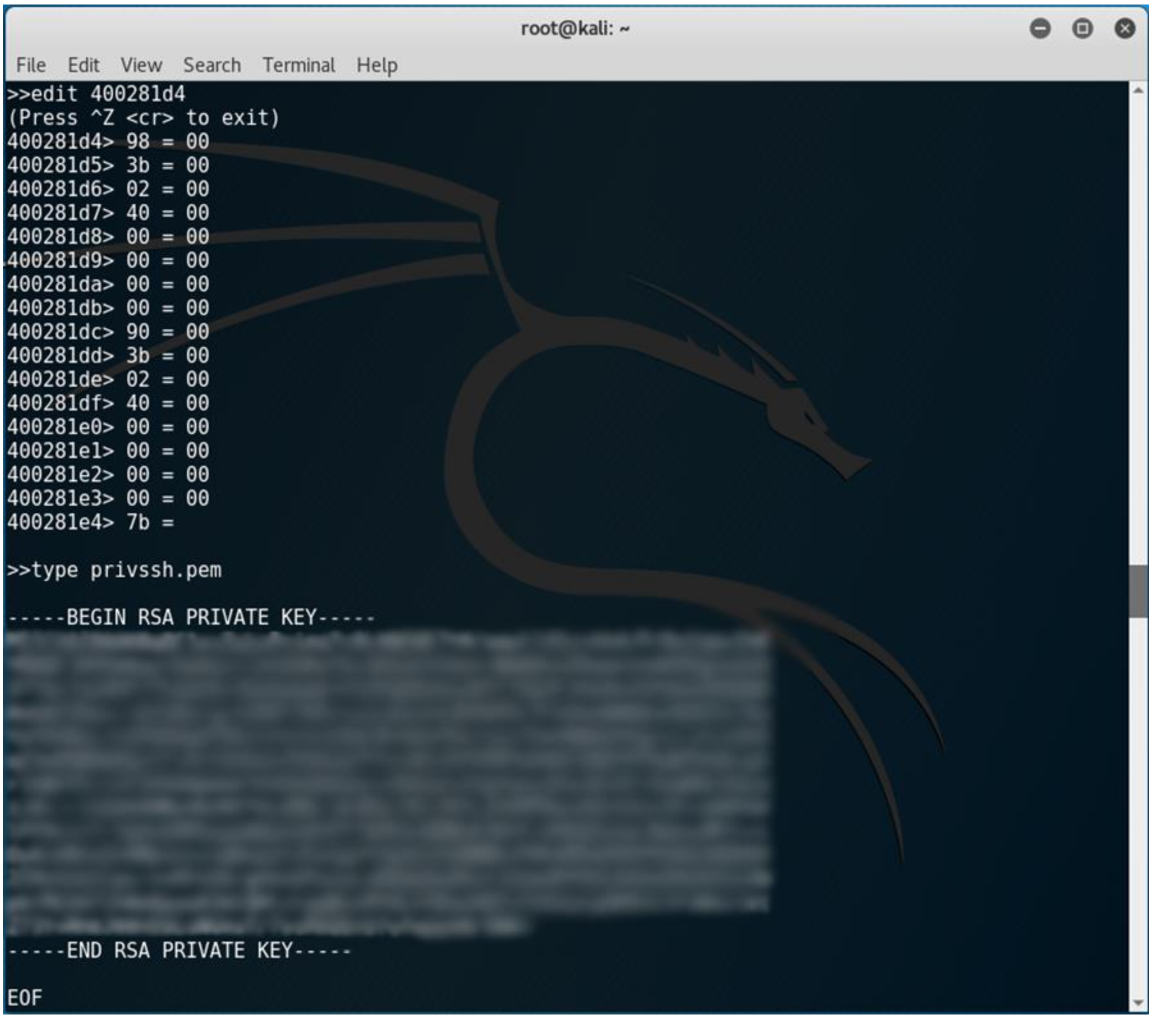


Figure 21 – Reading private SSH server key

User can generate secret SSH key either by third-party tools or by “genkey” system console command in Sarian OS.

9.4 Telnet Service

Telnet service is used in Digi routers to access the system console. There is no difference in using Telnet, SSH or serial connection to the router except for one thing: serial connections can be configured to allow user access to the system console without any authentication with super user permissions, while remote network services always require user authentication.

9.5 Web Service

9.5.1 General Description

Digi routers have embedded web server for managing, updating firmware, configuring and monitoring purposes. It is based on GoAhead library. The exact version of GoAhead used is unknown. However, we found out that the embedded web server on the device contains an old vulnerability CVE-2002-1603. This vulnerability allows remote attackers to obtain the source code of ASP files via a URL terminated with a /, \, %2f (encoded /), %20 (encoded space), or %00 (encoded null) character, which returns the ASP source code unparsed. The highest affected GoAhead server version for this vulnerability is 2.1.7. Therefore, version of GoAhead server used by Digi developers is 2.1.7 (released in 2002) or older.

Embedded web server supports HTTP and HTTPS protocols. By default, only HTTP is used. For instructions how to configure web server for using HTTPS, see “Digi TransPort® WR Routers User Guide”, section “Network Services Page” at page 345.

User authentication is required to access web server of a Digi device.

9.5.2 Web Server Files

Web server files are stored inside the archive “wr21.web” in file system of the device. The archive has custom format. It is possible to study it and unpack files; however, we found it easier to get server files using browser plugins for saving web pages. We used free ScrapBox plugin for Firefox. As web server uses dynamic ASP-files, we also used CVE-2002-1603 to download ASP-files in an unparsed form.

Web server includes the following files:

- JavaScript code files;
- HTML and CSS files;
- Pictures;
- ASP dynamic files.

GoAhead Web server supports registering C-callbacks for different commands. When the server meets such command in an ASP file, it calls the corresponding callback, allowing it to fill requested web page by dynamic information.

Before authentication, users can only access picture files and “login.asp” page.

9.5.3 Service Analysis

GoAhead web server provides API for registering client request handlers. Table 8 contains the list of main GoAhead API functions we need to find out what requests can be handled by server. It also contains some major information about these functions and their parameters. For complete description of these functions, please refer to GoAhead 2.1.7 documentation.

Table 8 – Main GoAhead API functions used in Digi embedded web server

Function Name	Parameters	Description
websUrlHandlerDefine	urlPrefix webDir arg handler flags	Registers a URL handler. Registered handler is called for all URL requests starting with urlPrefix. However, longer prefix means higher handler priority. Handlers with high priority are called first. Supported flags: • WEBS_HANDLER_FIRST (1) – handler must be called first • WEBS_HANDLER_LAST (2) – handler must be called last • WEBS_HANDLER_CSRF (4) – handler requires CSRF token (for POST requests)
websAspDefine	name function	Binds an ASP name to a C procedure. ASP pages of Digi web server contain static data and ASP names. When an ASP-page is requested, ASP parser of the server finds all ASP names on requested ASP page and calls corresponding C-procedures that fill dynamic part of the page.
websFormDefine	name function	Registers GoForm handler. This mechanism is mostly used in Digi web server to handle POST requests.

We found API functions from Table 8 in Digi firmware, traced all places where these functions were called, and obtained the list of URL paths handled by the server. Table 9 contains this list.

Table 9 – URL handlers in Digi web server

Name	URL	Flags	Description
websUrlHandler_security	Any	WEBS_HANDLER_FIRST	This handler is always called first for every request. It checks whether the client is authenticated or not, and allows or denies access to server pages accordingly. Before authentication, only server pictures and “login.asp” page are available.
websUrlHandler_default_first	Any	0	Called after websUrlHandler_security handler. Checks if URL is valid. Calls ASP request handler if ASP page was requested.

websUrlHandler_default_last	Any	WEBS_HANDLER_LAST	Checks whether the request was not handled by previous handlers. If so, it calls Python web server extensions or handling.
websUrlHandler_home	/	0	Default page handler
websUrlHandler_cmdexec	/cmdexec	WEBS_HANDLER_CSRF	Handler for POST requests that execute system console commands
websUrlHandler_configs	/configs		Handler for downloading router configuration files
websUrlHandler_fwupd	/fwupd		Handler for updating firmware of the device
websUrlHandler_UE_rci	/UE/rci		RCI handler. This service is described in Section 9.5.4
websUrlHandler_uploadfile	/uploadfile		Handler for uploading files to the device
websUrlHandler_uploadkey	/uploadkey		Handler for uploading keys to the device
websUrlHandler_goform	/goform	WEBS_HANDLER_CSRF	Goform request handler
websUrlHandler_goform	/insecure	WEBS_HANDLER_CSRF	

9.5.4 Remote Command Interface (RCI)

9.5.4.1 Service Description

Remote Command Interface (RCI) is a command interface that allows managing Digi router using HTTP or HTTPS protocol. Commands are sent to the device as HTTP POST request payloads with URL “/UE/rci”. The device sends the result of executing commands to remote client using standard POST replies. websUrlHandler_UE_rci handler is responsible for executing RCI requests. websUrlHandler_security handler has special check for “UE/rci” URLs. RCI POST request must contain basic HTTP authentication data. Authentication data must be provided in every RCI request. The digest authentication is not supported.

Example of RCI request to Digi router is shown below:

```
POST /UE/rci HTTP/ 1.1
Content-Type: text/html
Authorization: dXNlcm5hbWU6cGFzc3dvcmQ=
Host: 192.168.1.10
User-Agent: Mozilla/ 5.0 (Windows NT 10.0; WOW64; rv: 54.0 ) Gecko/ 20100101 Firefox/
54.0
Accept: text/html,application/xhtml+xml,application/xml;q= 0.9 ,*/*;q= 0.8
Accept-Language: ru-RU,ru;q= 0.8,en-US;q= 0.5 ,en;q= 0.3
Content-Length:  62
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age= 0
<rci_request version= "1.1" ><query_setting/></rci_request>
```

RCI service uses XML format to form requests. Full specification of RCI command interface is available in “Digi Remote Command Interface (RCI) Reference”.

RCI allows configuring device, reading and writing files to file system of the device, rolling back to default configuration and rebooting the device.

9.5.4.2 Privilege Escalation via RCI

Details of the issue were reported to vendor on 14.09.2017. Digi claimed that it is possible to mitigate the issue by means of device hardening.

Affected products

Digi WR11, WR21, WR31, WR41, and WR44 routers

Exploitation conditions:

To exploit this vulnerability, an attacker must have an account on the device with an arbitrary access level. The lowest access level is enough.

Description:

Consider there is a user with username "user3" and password "user3pwd", and he's access level is "low". Figure 22 shows creation of user “user3” using web interface of the router:

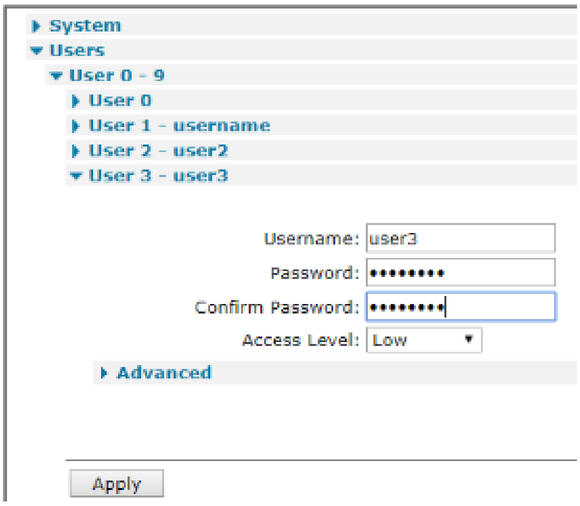


Figure 22 – Creating user with low access level

User "user3" with low access level cannot use system console command "user" through Telnet or SSH. Figure 23 illustrates an unsuccessful attempt to execute this command using telnet:

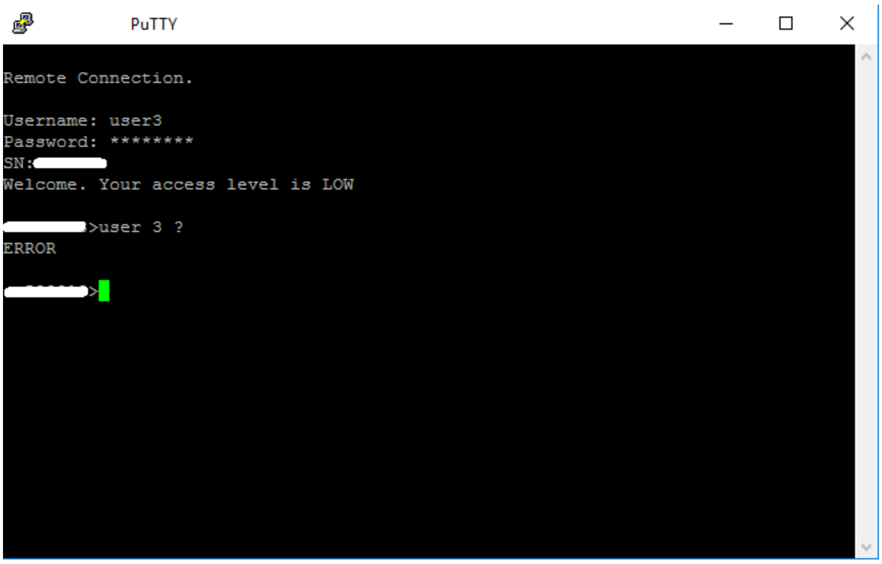


Figure 23 – Attempt to execute “user” console command with low privilege level using Telnet

However, using RCI, user "user3" can access this command, as shown at Figure 24.

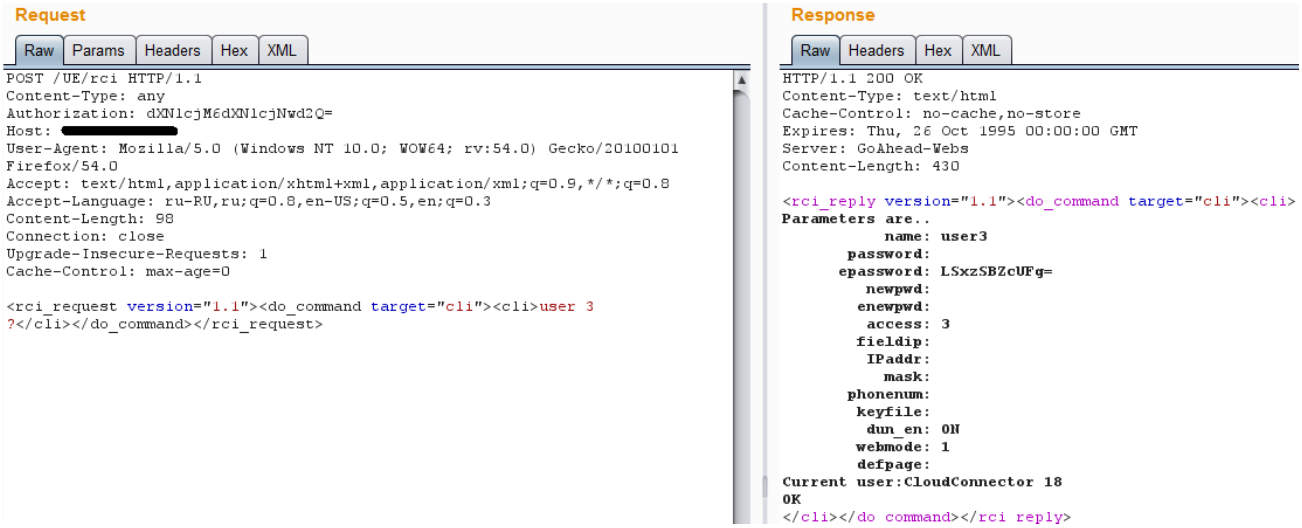
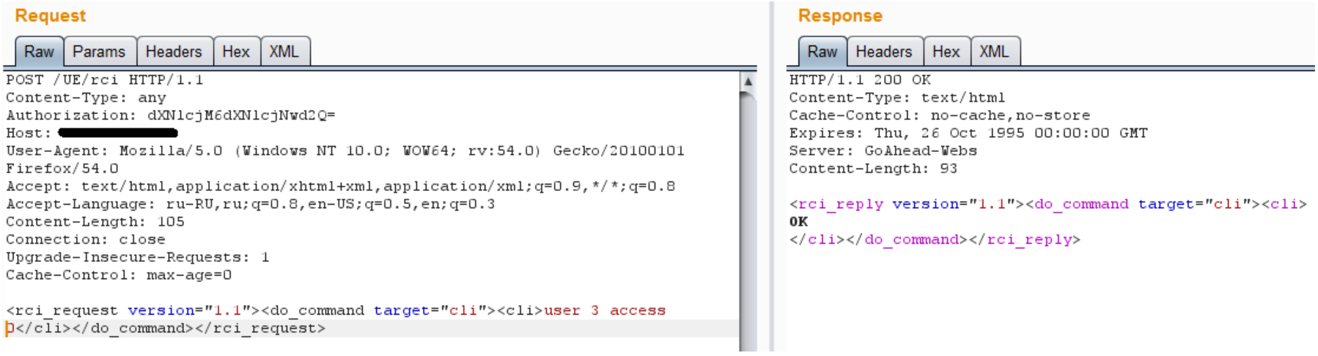


Figure 24 – Executing “user” console command with low privilege level using RCI

Note: in "Authorization" field of HTTP header there is Base64 encoded string "user3:user3pwd".

User can also change his access level to zero ("super user level"), as one can see at Figure 25.

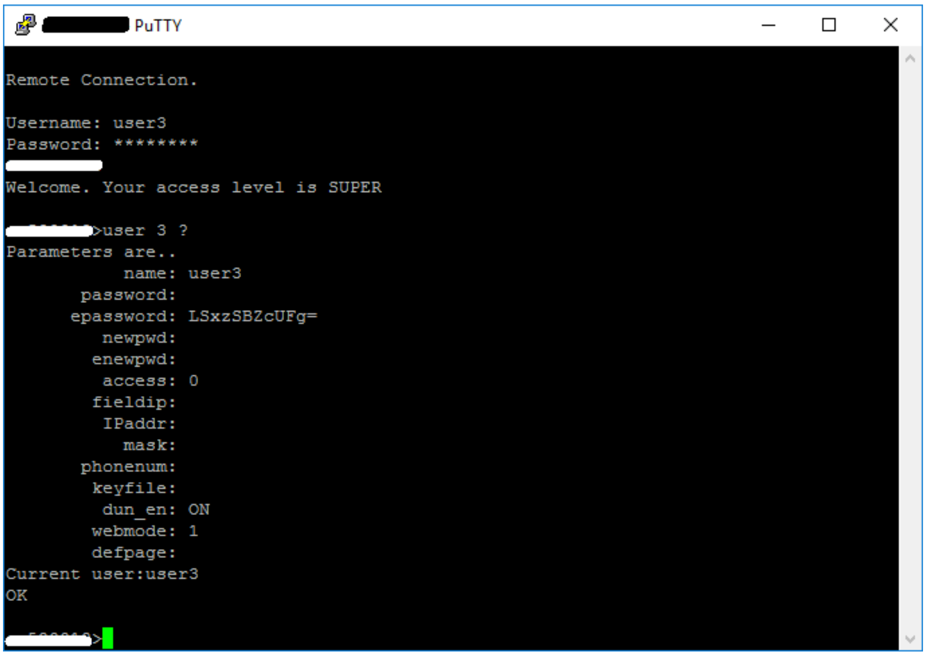


```
Request
Raw | Params | Headers | Hex | XML
POST /UE/rci HTTP/1.1
Content-Type: any
Authorization: dXNlcjM6dXNlcjNwd2Q=
Host: [redacted]
User-Agent: Mozilla/5.0 (Windows NT 10.0; WOW64; rv:54.0) Gecko/20100101
Firefox/54.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: ru-RU,ru;q=0.8,en-US;q=0.5,en;q=0.3
Content-Length: 105
Connection: close
Upgrade-Insecure-Requests: 1
Cache-Control: max-age=0

<rci_request version="1.1"><do_command target="cli"><cli>user 3 access
0</cli></do_command></rci_request>
```

```
Response
Raw | Headers | Hex | XML
HTTP/1.1 200 OK
Content-Type: text/html
Cache-Control: no-cache,no-store
Expires: Thu, 26 Oct 1995 00:00:00 GMT
Server: GoAhead-Webs
Content-Length: 93

<rci_reply version="1.1"><do_command target="cli"><cli>
OK
</cli></do_command></rci_reply>
```

Figure 25 – Escalating user privileges via RCI

Now if user 3 logs in to the system using telnet or SSH, his access level will be high enough to execute "user" command. Figure 26 shows successful attempt of executing “user” command in telnet session:



```
[redacted] PuTTY

Remote Connection.

Username: user3
Password: ********
[redacted]
Welcome. Your access level is SUPER

[redacted]>user 3 ?
Parameters are..
           name: user3
       password:
      epassword: LSxzSBZcUFg=
         newpwd:
        enewpwd:
         access: 0
        fieldip:
         IPaddr:
           mask:
       phonenum:
        keyfile:
         dun_en: ON
        webmode: 1
        defpage:
Current user:user3
OK

[redacted]>
```

Figure 26 – Executing “user” console command with new privilege level using Telnet

As we can see in this example, there is a privilege escalation vulnerability in RCI protocol, which allows raising system user's access level from arbitrary value to zero, which is the highest access level in the router's operating system.

Privilege escalation vulnerability exists because RCI users execute system console commands on behalf of internal user "CloudConnector", no matter what kind of credentials were specified in the HTTP header.

We submitted this information to the vendor. They did not consider this issue as a vulnerability and did not patch it.

9.6 SNMP Service

9.6.1 General Information

SNMP protocol in Digi router is handled by special thread OS_Thread_SNMP, created by TCPUTILS task. By default, SNMP protocol works at UDP port 161. TCPUTILS task receives all SNMP network packets and puts them to the queue. Thread OS_Thread_SNMP handles all the packets in this queue as they arrive.

All three existing SNMP protocol versions are supported and handled by OS_Thread_SNMP. Protocol version identifier is included in every SNMP packet according to SNMP specification.

Further information in this section is based on the following standards:

1. Rose, M. and K. McCloghrie "Structure and Identification of Management Information for TCP/IP-based Internets", RFC 1155, May 1990, <https://www.ietf.org/rfc/rfc1155.txt>
2. Rose, M. and K. McCloghrie "Management Information Base for Network Management of TCP/IP-based internets", RFC 1156, May 1990, <https://www.ietf.org/rfc/rfc1156.txt>
3. Case, J., Fedor, M., Schoffstall, M. and J. Davin, "Simple Network Management Protocol", RFC 1157, May 1990, <https://www.ietf.org/rfc/rfc1157.txt>
4. Case, J., MCcloghrie, K., Rose, M. and S. Waldbusser, "Introduction to Community-based SNMPv2", RFC 1901, January 1996, <https://www.ietf.org/rfc/rfc1901.txt>
5. Case, J., MCcloghrie, K., Rose, M. and S. Waldbusser, "Coexistence between Version 1 and Version 2 of the Internet-standard Network Management Framework", RFC 1905, January 1996, <https://www.ietf.org/rfc/rfc1905.txt>
6. Harrington, D., Presuhn, R. and B. Wijnen "An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks", RFC 3411, December 2002, <https://www.ietf.org/rfc/rfc3411.txt>
7. Presuhn, R., Case, J., MCcloghrie, K., Rose, M. and S. Waldbusser "Management Information Base (MIB) for the Simple Network Management Protocol (SNMP)", RFC 3418, December 2002, <https://www.ietf.org/rfc/rfc3418.txt>

Please refer to this documents for comprehensive information about SNMP protocols.

9.6.2 SNMP v3 Authentication

SNMP service in Digi router supports standard algorithms of authentication and checking message integrity using hash functions MD5, SHA1 and SHA256. There are up to 10 users supported for SNMP protocol. These users are not the same as system users, they exist separately and used only for authentication in SNMP service. Therefore, SNMP service does no use Sarian OS authentication. SNMP users can be added or modified using OS console command "snmpusers" or web server interface.

Each user has secret password for authentication and secret password for encrypting packets.

Secret password for authentication is used to generate authentication key. In turn, this key is used to generate message digests of SNMP packets in HMAC mode implemented in openssl library. The algorithm of producing authentication key from authentication password in Digi router is standard for SNMP v3:

1. User password is copied certain amount of times to fill 1Mbyte buffer;
2. The buffer is hashed using MD5, SHA1 or SHA256: sum = hash(buffer);
3. auth_key = hash(sum + engine_id + sum), where “+” sign means concatenation. engine_id constant can be obtained from the device using special SNMP v3 request according to SNMP specification.

9.6.3 SNMP v3 Encryption

PDU encryption can be performed using DES or AES. An encryption key is generated from a user encryption password. First, the password is hashed like the authentication password. Then the hash and a random value (4 bytes long for DES and 8 bytes long for AES) are turned into an expanded encryption key using standard DES or AES key expanding procedures.

9.6.4 SNMP MIB Tree

Digi router stores SNMP objects in a tree. Figure 27 shows a part of this tree as an example. Additionally, information about supported objects is stored in file “wr21-U.mib”. Users can download this file and use it with any SNMP client tool to control the router by SNMP protocol.

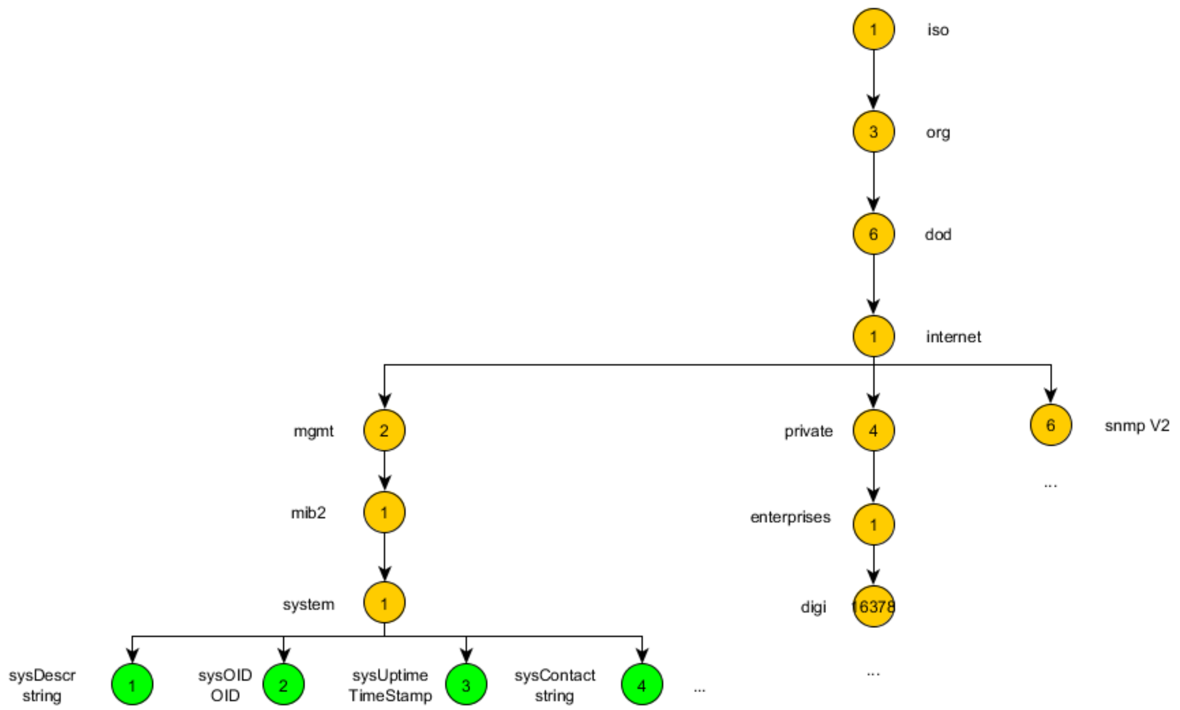


Figure 27 – SNMP MIB-tree of Digi device

9.6.5 CVE-2017-XXXX: Router Denial of Service

Severity: Medium

CVE: Not Assigned

Affected products

Digi WR11, WR21, WR31, WR41, and WR44 routers with firmware version less than 5.2.19.11.

Exploitation conditions:

To exploit this vulnerability, an attacker only needs a network connection with a router with an active SNMP service.

Submit date: 27.07.2017

Official patch date: October 2017

Description

SNMP packets of any protocol version contain octet strings. SNMP v1 and v2 packets include community octet string and SNMP v3 packets include secure username octet. These strings are parsed using snmpDecodeOctetString function:

```
int snmpDecodeOctetString(char **ptr, char *pStrOut, int maxLen)
{
  char FieldType; // r6
  int result; // r0
  int Length; // r5

  FieldType = **ptr; //For octet strings FieldType = 4
  if ( (FieldType & 0xFFFFFFDF) != 4 )
    return -1;
  ++*ptr;
  result = snmpGetASN1Length(ptr);
  Length = result;
  if ( result < 0 )
    return -1;
  if ( result )
  {
    if ( FieldType & 0x20 )
    {
      OS_DebugLog("\r\nSNMP: Constructed OctetString not supported yet");
    }
    else
    {
      if ( result > maxLen )
      {
        OS_DebugLog("\r\nSNMP: asn1DecodeOctetString - Truncating OctetString");
        Length = maxLen;
      }
      OS_memset(pStrOut, 0, Length);
      OS_memcpy(pStrOut, *ptr, Length);
    }
    result = Length;
```

```
    *ptr += Length;
  }
  return result;
}
```

As one can see from the listing, if object type is equal to 0x24, then snmpDecodeOctetString function does not check string length against maxLen and just returns it to the caller as is. Therefore, this unction is dangerous. It is used for parsing all octet strings in all three SNMP versions. The SNMP v1 handler part using this function is shown below:

```
00000000 asn_string      struc ; (sizeof=0x104, mappedto_67)
00000000 buf             DCB 256 dup(?)
00000100 len             DCD ?
00000104 asn_string      ends

int snmpV1PrepareDataElements(char *ptr, int PacketLength, asn_string *communityString,
…)
{
  int CommunityStrLen; // r0
  int Length; // [sp+4h] [bp-64h]
  int Version; // [sp+8h] [bp-60h]
  …

  snmpGetSequenceLength(&ptr, &Length);
  snmpDecodeInteger(&ptr, &Version);
  CommunityStrLen = snmpDecodeOctetString(&ptr, communityString->buf, 255);
  communityString->len = CommunityStrLen;
  if ( CommunityStrLen < 0 )
  {
    OS_DebugLog("\r\nSNMP: v1PrepareDataElements - Failed to decode Community");
    return -4;
  }
  communityString->buf[CommunityStrLen] = 0;
  …
}
```

Handler writes zero to arbitrary memory address >= communityString pointer value. Writing to any address smaller than communityString pointer value is not allowed because there is a check that CommunityStrLen is greater than or equal to zero. communityString is actually a constant pointer that points to an address in OS_BSS zone of the memory. We did not find anything upper than this address in the memory that could be zeroed to cause something more serious than an ordinary crash. Therefore, we consider this vulnerability a denial of service one.

Similar code is used when parsing SNMP v2 community string and SNMPv3 context name field.

9.6.6 CVE-2017-XXXX: Stack Overflow in SNMP Service

Severity: High

CVSS v2 score: 7.6 (AV:N/AC:H/Au:N/C:C/I:C/A:C)

CVE: Not Assigned

Affected products

Digi WR11, WR21, WR31, WR41, and WR44 routers with firmware version less than 5.2.19.11.

Exploitation conditions:

To exploit this vulnerability, an attacker only needs a network connection with a router with an active SNMP service of versions 1 or 2.

Submit date: 27.07.2017

Official patch date: October 2017

Description

SNMP PDU contains a variable binding list. Each entry in this list is a pair “object identifier – value”. The type of object’s value can be “Object Identifier”. If we provide the pair “object identifier” – “value” in variable binding section of the packet, and value type is “object identifier”, it is possible to perform stack overflow and execute arbitrary code in the system. This vulnerability is more complex than previous one, so we break it into three steps.

9.6.6.1 Stack Overflow in snmpParseVariableBindings Function

The function snmpParseVariableBindings is used for parsing variable binding list of SNMP packets. It is shown in the following listing:

```
00000000 SNMP_OID        struc ; (sizeof=0x204, mappedto_81)
00000000 Length          DCD ?
00000004 Value           DCD 128 dup(?)
00000204 SNMP_OID        ends00000000 ;
----------------------------------------------------------------------------
00000000 SNMP_DATA_ELEMENTS struc ; (sizeof=0x238, mappedto_77)
00000000 snmpVersion     DCB ?
00000001 PDU_Type        DCB ?
00000002 field_2         DCB ?
00000003 field_3         DCB ?
00000004 RequestID       DCD ?
00000008 ErrorStatus     DCD ?
0000000C ErrorIndex      DCD ?
00000010 nonRepeaters    DCD ?
00000014 maxRepetitions  DCD ?
…
0000022C pPrevBinding    DCD ?
00000230 pNextBinding    DCD ?
00000234 nBindings       DCD ?
00000238 SNMP_DATA_ELEMENTS ends

void snmpCopyOID(SNMP_OID *pDstOID, SNMP_OID *pSrcOID)
{
  int oidSize; // r5
  int offset; // r3
  int i; // r2

  pDstOID->Length = pSrcOID->Length;
  oidSize = pSrcOID->Length;
```

```
  if ( pSrcOID->Length > 0 )
  {
    offset = 0;
    i = 0;
    do
    {
      ++i;
      pDstOID->Value[offset] = pSrcOID->Value[offset];
      ++offset;
    }
    while ( i != oidSize );
  }
}

int snmpParseVariableBindings(char *pCurPointer, char *EndPointer, SNMP_DATA_ELEMENTS
*pDataElements)
{
  bool v5; // zf
  SNMP_OID *v6; // r1
  _DWORD *a6; // r5
  int Length; // r12
  int ValueType; // [sp+14h] [bp-324h]
  int VarbindLength; // [sp+18h] [bp-320h]
  char VarValueBuf[256]; // [sp+1Ch] [bp-31Ch]
  SNMP_OID oidBuffer; // [sp+11Ch] [bp-21Ch]

  do
  {
    if ( pCurPointer >= EndPointer )
      return 0;
    v5 = snmpGetSequenceLength(&pCurPointer, &VarbindLength) == 0;
    v6 = &oidBuffer;
    if ( !v5 || (a6 = (_DWORD *)snmpDecodeOID(&pCurPointer, &oidBuffer), v6 = (SNMP_OID
*)&ValueType, a6) )
    {
      OS_DebugLog("\r\nSNMP: parseVariableBindings - Failed to decode OID", v6);
      return -4;
    }
    Length = snmpDecodeObjectSyntax(&pCurPointer, &ValueType, VarValueBuf, 255);
    VarbindLength = Length;
    if ( Length < 0 )
    {
      OS_DebugLog("\r\nSNMP: parseVariableBindings - Failed to decode Object Syntax",
&oidBuffer);
      return -4;
    }
  }
  while ( !snmpAddVariableBinding(pDataElements, &oidBuffer, ValueType, VarValueBuf,
Length, a6) );
  OS_DebugLog("\r\nSNMP: parseVariableBindings - addVariableBinding failed");
  return -8;
}
```

As one can see, snmpDecodeObjectSyntax is used to decode variable value. The listing of its code is shown below:

```
int snmpDecodeObjectSyntax(char **pCurPtr, char *pObjectType, char *pValueBuf, int
BufSize)
{
```

```
  unsigned int Type; // r12
  int result; // r0

  Type = **pCurPtr;
  if ( Type == 5 )
  {
    *pObjectType = 5;
    result = snmpDecodeNull(pCurPtr);
  }
  else if ( Type <= 5 )
  {
    if ( Type == 2 )
    {
      *pObjectType = 2;
      result = snmpDecodeNumber(2, pCurPtr, (int *)pValueBuf);
    }
    else
    {
      if ( Type != 4 )
        goto ERROR;
      *pObjectType = 4;
      result = snmpDecodeOctetString((char **)pCurPtr, pValueBuf, BufSize);
    }
  }
  else if ( Type == 0x41 || Type == 0x43 )
  {
    *pObjectType = Type;
    result = snmpDecodeCounter(Type, pCurPtr, (int *)pValueBuf);
  }
  else
  {
    if ( Type != 6 )
    {
ERROR:
      OS_DebugLog("\r\nSNMP: asn1DecodeObjectSyntax - Unsupported syntax type (%d)",
**pCurPtr, pValueBuf);
      return -1;
    }
    *pObjectType = 6;
    result = snmpDecodeOID(pCurPtr, (SNMP_OID *)pValueBuf);
  }
  return result;
}
```

snmpDecodeOID function does not use BufSize. The BufSize is equal to 255, while maximum OID length in snmpDecodeOID function is 128 dwords (each dword is 4 bytes long).

The stack of snmpParseVariableBindings is shown below:

```
-00000338 ValueSize       DCD ?
-00000334 a6              DCD ?                    ; offset
-00000330 var_330         DCD ?
-0000032C var_32C         DCD ?
-00000328 var_328         DCD ?
-00000324 ValueType       DCD ?
-00000320 VarbindLength   DCD ?
-0000031C VarValueBuf     DCB 256 dup(?)
-0000021C oidBuffer       SNMP_OID ?
-00000018 var_18          DCD ?
```

```
-00000014 var_14         DCD ?
-00000010 var_10         DCD ?
-0000000C var_C          DCD ?
-00000008 var_8          DCD ?
-00000004 var_4          DCD ?
+00000000 ; end of stack variables
```

It is possible to overwrite oidBuffer.Length filed in snmpParseVariableBindings function. However, the length of OID in VarValueBuf is not enough to overwrite the entire oidBuffer. Figure 28 demonstrates stack overflow in snmpParseVariableBindings in more clear way.

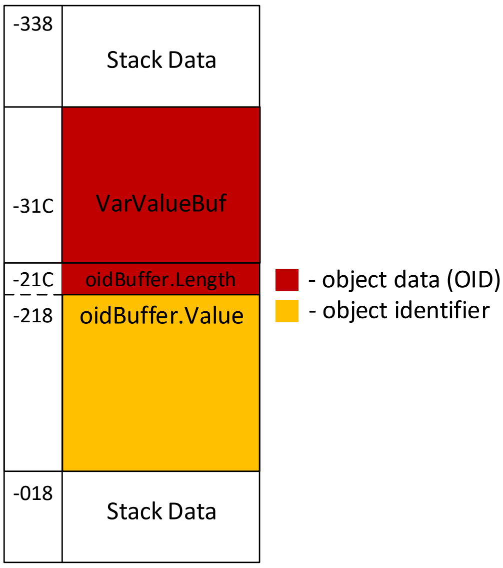


Figure 28 – Overwriting oidBuffer.Length value in snmpParseVariableBindings function

9.6.6.2 Large Buffer Moves to Heap

After parsing a variable snmpParseVariableBindings calls snmpAddVariableBinding to add it to the list of type SNMP_DATA_ELEMENTS. The most important part of the listing of snmpAddVariableBinding function is shown below:

```
00000000 SNMP_VAR_BIND   struc ; (sizeof=0x428, mappedto_78)
00000000 OID             SNMP_OID ?
00000204 field_204       DCD ?
00000208 ValueType       DCD ?
0000020C field_20C       DCD ?
00000210 Value           DCB 256 dup(?)
00000310 ValueSize       DCD ?
…
00000418 One             DCB ?
00000419 field_419       DCB ?
0000041A field_41A       DCB ?
0000041B field_41B       DCB ?
0000041C pPrevBind       DCD ?                    ; offset
00000420 pNextBind       DCD ?                    ; offset
00000424 field_424       DCD ?
00000428 SNMP_VAR_BIND   ends
int  snmpAddVariableBinding(SNMP_DATA_ELEMENTS  *pDataElements,  SNMP_OID  *pOID,  int
ValueType, char *ValueBuf, int ValueSize, _DWORD *a6)
{
  SNMP_VAR_BIND *pVarBind; // r0 MAPDST
```

```
  int v12; // r3
  SNMP_VAR_BIND *pNextBind; // r3
  unsigned __int8 result; // r0
  bool v15; // zf
  int v16; // r3

  if ( a6 )
  {
    *a6 = 0;
    pVarBind = (SNMP_VAR_BIND *)OS_smalloc(0x428);
    if ( pVarBind )
    {
      *a6 = pVarBind;
      goto LABEL_5;
    }
MALLOC_FAILED:
    OS_DebugLog("\r\nSNMP: addVariableBinding - smalloc failed");
    return -8;
  }
  pVarBind = (SNMP_VAR_BIND *)OS_smalloc(0x428);
  if ( !pVarBind )
    goto MALLOC_FAILED;
LABEL_5:
  OS_memset(pVarBind, 0, 0x428);
  pVarBind->One = 1;
  snmpCopyOID(&pVarBind->OID, pOID);
  pVarBind->ValueType = ValueType;
  switch ( ValueType )
  {
    case 1:
    …
    case 6:
      snmpCopyOID((SNMP_OID *)pVarBind->Value, (SNMP_OID *)ValueBuf);
      goto BIND;
    … //Code for adding SNMP_VAR_BIND element to SNMP_DATA_ELEMENTS list
```

In snmpAddVariableBinding function the 0x428 bytes long buffer is allocated from general dynamic memory pool. Then OID with our arbitrary length is copied to this buffer. In this step, it is important not to overflow the buffer on heap, because it can cause undefined behavior. Figure 29 illustrates the second step of the vulnerability.

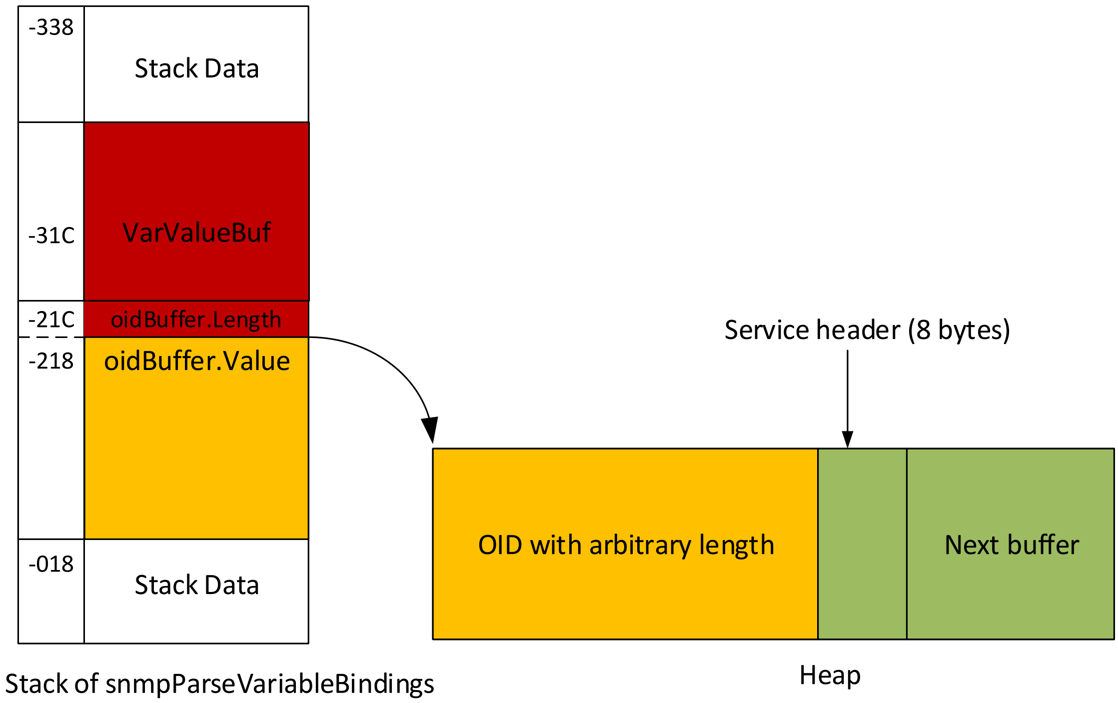


Figure 29 –OID with arbitrary length is copied from stack to heap

9.6.6.3 One More Stack Overflow

Next function that works with OID copied into heap buffer is snmpGetRequest. It handles GET requests of SNMP protocol. The listing of this function is shown below:

```
int __fastcall snmpGetRequest(SNMP_DATA_ELEMENTS *pDataElements, int v3SecurityModel,
asn_string *CommunityStr, int a4, int a5, SNMP_DATA_ELEMENTS *AnswerDE)
{
  SNMP_VAR_BIND *CurBind; // r4
  int RequestID; // r3
  int ObjNumber; // r7
  SNMP_OID_NODE *pNode; // r5
  int v14; // ST0C_4
  char v15; // r6
  int ValueSize; // [sp+18h] [bp-330h]
  char ValueBuf[256]; // [sp+1Ch] [bp-32Ch]
  SNMP_OID_TAIL oidTail; // [sp+11Ch] [bp-22Ch]

  CurBind = pDataElements->pPrevBinding;
  OS_memset(AnswerDE, 0, 0x238);
  RequestID = pDataElements->RequestID;
  ObjNumber = 0;
  AnswerDE->PDU_Type = GET_RESPONSE;
  AnswerDE->RequestID = RequestID;
  AnswerDE->ErrorStatus = 0;
  for ( AnswerDE->ErrorIndex = 0; CurBind; ++dword_40C1D52C )
  {
    pNode = snmpMIB_FindNode(CurBind, &oidTail);
    ++ObjNumber;
    if ( pNode )
    {
      if ( !snmpCheckAccess(pDataElements->PDU_Type, v3SecurityModel, CommunityStr, a4,
(char *)a5, pNode) )
      {
```

```
        v15 = 0;
        snmpSetError(pDataElements, AnswerDE, NO_ACCESS, ObjNumber);
        return v15;
      }
      OS_memset(ValueBuf, 0, 256);
      if ( snmpGetMibNodeData(pNode, &oidTail, ValueBuf, &ValueSize) < 0 )
      {
        ++dword_40C1D528;
        snmpSetError(pDataElements, AnswerDE, GENERAL_ERROR, ObjNumber);
        return 0;
      }
      if  (  snmpAddVariableBinding(AnswerDE,  &CurBind->OID,  pNode->Type,  ValueBuf,
ValueSize, 0) )
        goto LABEL_10;
    }
    else
    {
      v14 = (unsigned __int8)pDataElements->snmpVersion;
      ++dword_40C1D54C;
      if ( !v14 )
      {
        snmpSetError(pDataElements, AnswerDE, OBJ_NOT_FOUND, ObjNumber);
        return 0;
      }
      if ( snmpAddVariableBinding(AnswerDE, &CurBind->OID, 0x80, 0, ValueSize, 0) )
      {
LABEL_10:
        OS_DebugLog("\r\ngetRequest: addVariableBinding failed");
        return -8;
      }
    }
    CurBind = CurBind->pPrevBind;
  }
  return 0;
}
```

This function uses snmpMIB_FindNode to find the requested node in the object tree by OID. Its listing is shown below:

```
SNMP_OID_NODE *snmpMIB_FindNode(SNMP_VAR_BIND *Bind, SNMP_OID_TAIL *oidTail)
{
  SNMP_VAR_BIND *v2; // r5
  int Depth; // r1
  int NewLength; // r12
  bool v6; // zf
  int v7; // r12
  int v8; // r1
  unsigned int i; // r3

  v2 = Bind;
  if ( Bind )
  {
    Bind = (SNMP_VAR_BIND *)snmpFindOidNode(&Bind->OID);
    if ( Bind )
    {
      Depth = Bind->OID.Value[10];
      NewLength = v2->OID.Length - Depth;
      v6 = v2->OID.Length == Depth;
      oidTail->Length = NewLength;
```

```
      if ( NewLength >= 0 && !v6 )
      {
        v7 = 4 * NewLength;
        v8 = (int)v2 + 4 * Depth;
        i = 0;
        do
        {
          oidTail->Value[i / 4] = *(_DWORD *)(v8 + i + 4);
          i += 4;
        }
        while ( i != v7 );
      }
    }
  }
  return (SNMP_OID_NODE *)Bind;
}
```

We do not show the listing of snmpFindOidNode because this function is rather simple: it just passes through the tree until it meets a leave and then copies all OID part left unparsed to oidTail buffer on the stack of snmpGetRequest function. For example, if we provided OID “1.3.6.1.2.1.1.1.2.3.4.5.6”, then “2.3.4.5.6” will be copied to oidTail (see Figure 27 for Digi SNMP object tree).

Stack of snmpGetRequest function is shown below:

```
-00000348 ValueSize       DCD ?
-00000344 a6              DCD ?                   ; offset
-00000340                 DCB ? ; undefined
-0000033F                 DCB ? ; undefined
-0000033E                 DCB ? ; undefined
-0000033D                 DCB ? ; undefined
-0000033C var_33C         DCD ?
-00000338                 DCB ? ; undefined
-00000337                 DCB ? ; undefined
-00000336                 DCB ? ; undefined
-00000335                 DCB ? ; undefined
-00000334 var_334         DCD ?
-00000330 var_330         DCD ?
-0000032C ValueBuf        DCB 256 dup(?)
-0000022C oidTail         SNMP_OID_TAIL ?
-00000028 var_28          DCD ?
-00000024 R4_saved        DCD ?
-00000020 R5_saved        DCD ?
-0000001C R6_saved        DCD ?
-00000018 R7_saved        DCD ?
-00000014 R8_saved        DCD ?
-00000010 R9_saved        DCD ?
-0000000C R10_saved       DCD ?
-00000008 R11_saved       DCD ?
-00000004 LR              DCD ?
+00000000 a5              DCD ?
+00000004 AnswerDE        DCD ?                   ; offset
+00000008
+00000008; end of stack variables
```

Actual length of OID tail is not checked in snmpMIB_FindNode, therefore, if it is long enough, it can overflow oidTail buffer on the stack and change saved registers values and LR to arbitrary values, causing arbitrary code execution. Figure 30 shows the full diagram of the vulnerability.

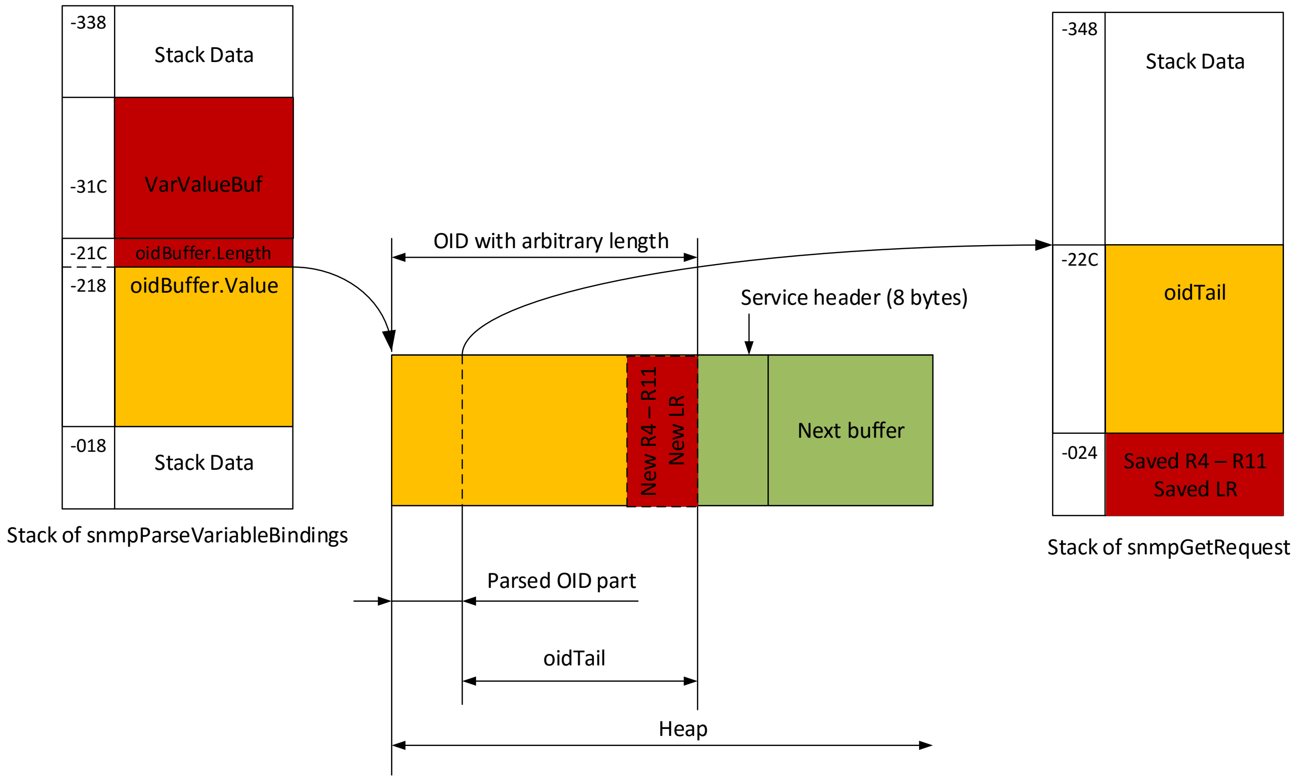


Figure 30 – Full vulnerability diagram

For demonstration purposes, we developed a code that adds a new user with name and password equal to “1” with super user privileges by modifying entries in the user table, which is stored in plain text in the RAM of the device. We have already mentioned this table while talking about permanent credentials storage in RAM weakness (see section 6.5.3 of this document for details).

For debugging our code, we used the JTAG interface we found during hardware revision (see section 4.2 for details about connection to the JTAG interface of the CPU, and section 8.3 for information about deactivating watchdog timer, which prevents debugging by resetting the board of the router).

By sending a special-crafted SNMP packet with this code as payload, we were able to add a temporary user with known credentials to the system. By temporary, we mean an account that disappears after reset of the router, because it is stored only in the RAM of the device. Thus, exploiting this vulnerability for testing purposes on our own router we obtained a remote shell to the device through SSH, telnet or RCI depending on what is available.

9.7 ADDP

ADDP is a proprietary Digi protocol for discovering and configuring Digi routers. By default, this protocol is deactivated. System console command "addp" allows configuring this interface.

Instruments for working with this protocol are available at GitHub:

https://github.com/christophgysin/addp

There is also a research of this protocol available at:

http://qbeukes.blogspot.nl/2009/11/advanced-digi-discovery-protocol_21.html

In our research we decided to focus on other protocols and did not studied the implementation of the protocol in depth, as it was inactive by default. Despite that, ADDP can be a good target for future security research.

9.8 Backup IP Service

9.8.1 General Information

Digi routers use a table where all crucial network nodes are stored. For each node, there is a backup IP address. If router fails to establish a connection to one of such nodes, it uses the backup IP address instead of the primary IP address to talk to this node. Additionally, Digi router can send information about unavailability of a node using Backup IP Service. UDP port 4052 is used to send such information.

Backup address table is configured using web interface of the device (section Configuration > Network > Advanced Network Settings).

UDP port 4052 is handled by TCPUTILS task in the system. The protocol of sharing information about availability of network hosts is proprietary. During the research, we checked procedures that handle incoming packets of backup IP service and did not find vulnerabilities there. In addition to that, the service is deactivated until a user of a router fills the table of backup IP addresses. Finally, this protocol does not allow sending new IP addresses for network hosts to routers, so substitution of legitimate host address to an address controlled by a malefactor is likely not possible. According to the official documentation (see "Digi TransPort® WR Routers User Manual", section "Backup IP addresses"), devices that receive the IP address available/unavailable messages search their own backup IP address tables for the IP addresses indicated, and tag those addresses as available/unavailable as appropriate. As a result, we did not pay much attention to this service. The next section of this document provides protocol packet format we discovered while checking handling procedures.

9.8.2 Packet Format

Table 10 describes backup IP service packet format. All fields in the service are big endian.

Table 10 – Backup IP service packet format

Offset	Size (bytes)	Description
0x00	1	Not used
0x01	2	CRC-16 checksum of the packet. This field is set to zero when calculating checksum.
0x03	2	Full packet size. Must be greater than or equal to 5 bytes
0x05	PacketSize-5	Command list

Each command from command list has format shown in Table 11.

Table 11 – Backup IP service command format

Offset	Size (bytes)	Description
0x00	2	Command code (1 or 2)
0x02	2	Size of the command
0x04	4	IP address of the host whose availability has changed

9.9 Modem AT Commands

While studying network services of the router we discovered that TCP connections on port 4005 of the router allowed sending AT-commands to the built-in modem Huawei ME909u-521 without any authentication. We tried to roll back to default settings of the router but this network service still worked after that. It could still have been a misconfiguration, although we did not find any description of this service in the official “Digi TransPort® WR Routers User Guide”.

Using this interface, we were able to find out modem firmware version (see Figure 31). This is the last firmware version currently available.

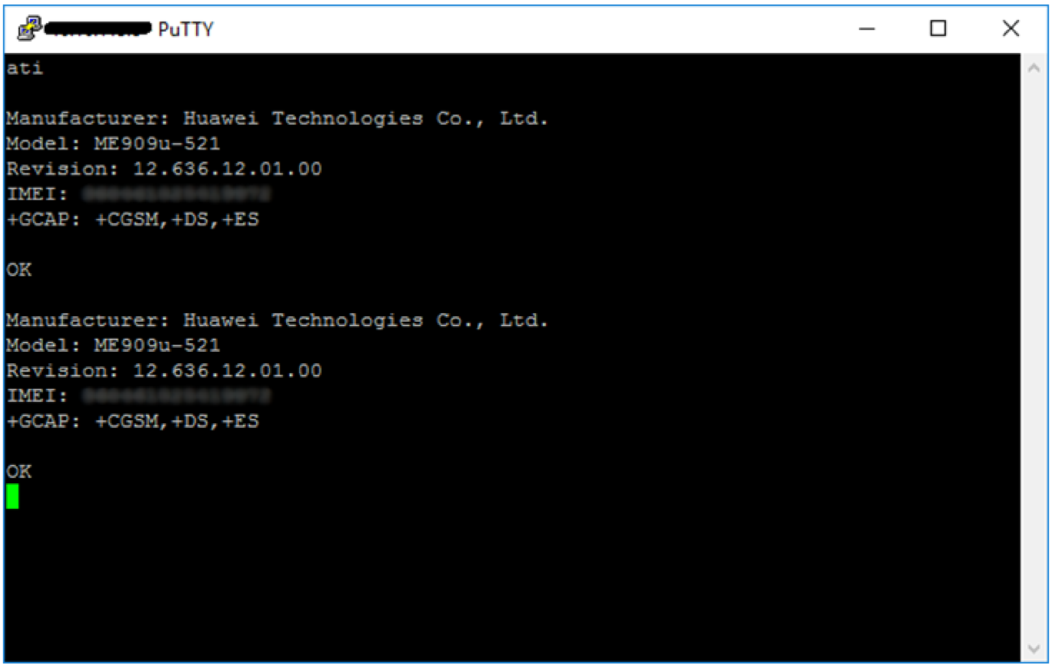


Figure 31 – Huawei modem firmware version

If it is not a misconfiguration, and all routers allow controlling their modems remotely without any authentication, remote connections to routers at TCP port 4005 should be prohibited to improve protection of the system.

10 SMS Handling

For GPRS and SMS support, Digi routers may contain different models of modems from different vendors (Telit, Huawei, Qualcomm etc.). Router under study contains Huawei ME909u-521 modem. The firmware of the router is modem-independent – it can work with many supported modem models.

In this research, we used BladeRF and YateBTS to communicate to the router using mobile network. BladeRF is a Software Defined Radio (SDR) hardware platform for RF communication. YateBTS is a software implementation of a GSM/GPRS radio access network. We ran it on Raspberry PI connected to the BladeRF board via USB 3.0.

Information about these tools can be found at the following links:

- https://www.nuand.com/ - BladeRF project page;
- https://wiki.yatebts.com/index.php/Main_Page - YateBTS wiki.

Digi WR21 router can receive system commands by SMS and send SMS. To send SMS from a Digi router, one can use “sendsms” command in system console. In addition to that, the router can send SMS to specified numbers in case of certain events.

To enable receiving SMS messages by Digi router, one can use web interface. Figure 32 shows incoming SMS settings.

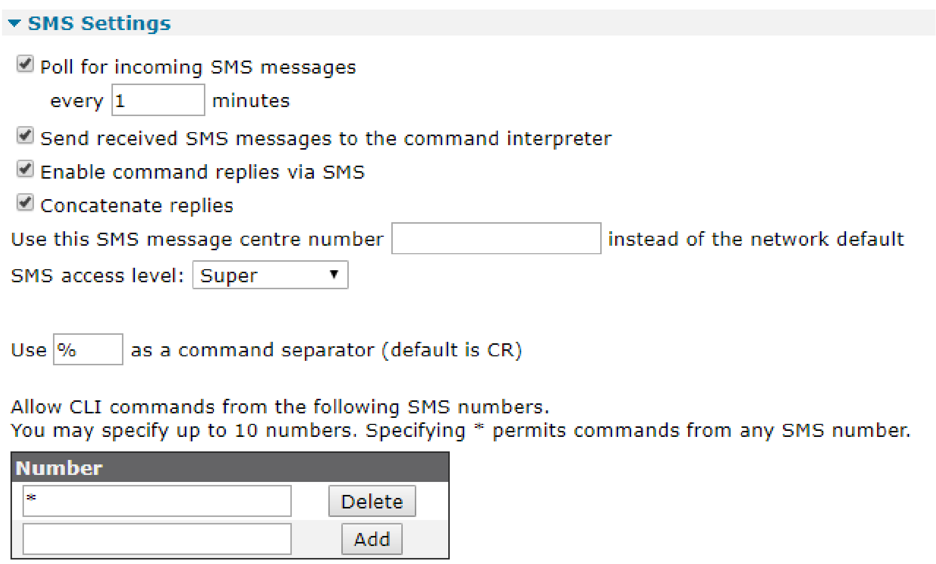


Figure 32 – Incoming SMS settings

It is possible to enable/disable executing commands from SMS messages, to configure access level for such commands and to fill whitelist of numbers that are allowed to send SMS commands to Digi router. “*” sign in the whitelist allows receiving commands from any number. If the router receives an SMS message from the number that is not in the whitelist, it rejects the message and writes the following string to debug log:

```
execute_sms: caller not in match list or filtered
```

After receiving and handling SMS message, Digi router writes information to system log (see Figure 33).

```
11:14:22, 25 Sep 2017,Modem dialing on asy 2 #:*98*1#
11:14:12, 25 Sep 2017,PPP 1 down,LL disconnect
11:14:12, 25 Sep 2017,Modem disconnected on asy 2,NO CARRIER
11:14:11, 25 Sep 2017,Modem dialing on asy 2 #:*98*1#
11:14:03, 25 Sep 2017,SMS Received: [redacted] id,Executed
11:14:02, 25 Sep 2017,DNS Query Failed on [time.devicecloud.com]
11:14:01, 25 Sep 2017,PPP 1 down,LL disconnect
11:14:01, 25 Sep 2017,Modem disconnected on asy 2,NO CARRIER
11:14:01, 25 Sep 2017,Modem dialing on asy 2 #:*98*1#
11:13:52, 25 Sep 2017,DNS Query Failed on [time.devicecloud.com]
11:13:51, 25 Sep 2017,PPP 1 down,LL disconnect
11:13:51, 25 Sep 2017,Modem disconnected on asy 2,NO CARRIER
```

Figure 33 – System log entry about SMS command execution

The router receives SMS messages from modem using AT+CMGL command. Messages are obtained as hexadecimal strings. The format of these strings is described in details in HUAWEI ME909u-521 LTE LGA Module AT Command Interface Specification (page 166).

Figure 34 shows the format of an incoming message.

1 Oct								2 Oct-12 Oct	1 Oct	1 Oct	7 Oct	1 Oct	
TP-MTI		MMS	0	0	SRI	UDHI	RP	OA	PID	DCS	SCTS	UDL	UD
Bit0	Bit1	Bit2	Bit3	Bit4	Bit5	Bit6	Bit7						

Figure 34 – Incoming SMS message format

Fields analyzed by firmware are highlighted with green color. These fields can be modified using custom binary message sender module “custom_sms.php” included in YateBTS.

Table 12 shows the format of “OA” field of an SMS message.

Table 12 – Format of “OA” field of incoming SMS message

Offset in hexadecimal symbols	Size in hexadecimal symbols	Description
0x00	2	Length of the number (oa_len)
0x02	2	Number type (type_addr)
0x04	oa_len * 2	Sender number

The size of the number should not exceed 10 octets, where one octet equals to two hexadecimal symbols.

Firmware distinguishes numbers of two types:

1. GSM-7 encoded numbers (type_addr = 5);
2. Plain text numbers (type_addr is not equal to 5).

If “DCS” field is equal to eight, then SMS data is transformed to hexadecimal string. In any other case firmware believes that SMS data is encoded using GSM-7.

“SCTS field represents message timestamp. Its length equals to seven octets.

“UDL” field contains user data length, and “UD” – user data.

User data from “UD” field of a message goes to the system command interpreter, which tries to execute commands. Digi routers ship with SMS commands support enabled by default, and with no source numbers added to the whitelist. To gain the ability of sending SMS commands to routers and receiving results of their execution, users need to add their numbers to the whitelist. We did not discover any way for malefactors to abuse command execution on a router using SMS messages. Nevertheless, we think there are a lot of opportunities for research with focus on this subsystem.

11 Conclusion

During our study of Digi wireless router we have built an inventory of hardware and software components, and performed analysis to identify implemented security mechanisms and vulnerabilities. An operating system of choice - Sarian OS - is a proprietary operating system without official documentation or third-party reviews available in public access. Main goal of this research was to gain understanding of how OS components and applications function and what resources available to them. To achieve our goal we have studied various Sarian OS subsystems such as: command line interface, task manager, file system, networking, user control subsystem, memory manager and system logging subsystem.

We also studied Sarian BIOS – the bootloader that starts at power on, loads the OS and passes control to it. BIOS provides CLI interface with functions to modify registers, flash memory, RAM, and other useful features for debugging and general research.

Network services provided on wired and wireless interfaces were analyzed to discover fingerprint, configuration options, security mechanisms and vulnerabilities.

All security findings have been shared with vendor to follow coordinated disclosure best practices. We'd like to thank Digi security team for vulnerability handling. Disclosure timelines and mitigation description are available in the document in the corresponding security findings section.

In November 2017 Digi released the firmware version 9.2.19.11 for Digi wireless routers affected by discovered vulnerabilities. This version fixes all FTP, SNMP and command line vulnerabilities mentioned in this document. Additionally, vendor released an article at Digi knowledge base, which contains information about fixed vulnerabilities and recommendations for network device users.

12 References

Digi. (2017). Digi Remote Command Interface (RCI) Reference.

Digi. (2017). Digi TransPort® WR Routers User Guide.

Freescale Semiconductor. (2010). *i.MX28 Applications Processor Reference Manual*.

Huawei. (2015). ME909u-521 LTE LGA Module AT Commnnd Interface Specification.