Security Implications from Improper De-acquisition of Medical Infusion Pumps

Research Report

Deral Heiland - Principal Security Researcher (IoT), Rapid7
Chris McGuire - Independent Security Researcher

RAPID7

INTRODUCTION

This document focuses on the physical and technical examination of various medical infusion pump devices purchased from secondary market sources. During this research project we focused on the extraction and examination of stored data within the infusion pump devices purchased on the secondary market, with the goal of identifying and locating critical data such as Protected Health Information (PHI)[1] and network configuration information such as wireless configuration passwords, that had not been properly purged from the devices prior to de-acquisition. To accomplish this testing we used various methods including Joint Test Action Group (JTAG)[2] debugging methods, along with physical communication interception methods, and destructive device methods that involve the removal and reading of flash memory storage components.

During this project we identified that most of the medical infusion pumps that were purchased from secondary market services such as eBay[3] were found to still contain wireless authentication data from the original medical organization that had deployed the devices. The authentication data identified, such as Wi-Fi Pre Shared Keys (PSK)[4], were not validated; however, the identified service set identifiers (SSIDs) were further researched using WIGLE[5], an online database of SSIDs throughout the world. The WIGLE resource revealed in every case that the device's SSIDs were still in use at the medical organizations within the United States. This brings us to the conclusion that the Wi-Fi passwords have a high probability of being valid. Through experience working with both IT networking and IT security teams within organizations, including fortune 500 companies, we have found that in most cases when Wi-Fi enabled equipment is upgraded, the previous Wi-Fi PSK are often reused, unless the underlying Wi-Fi infrastructure is upgraded and core security authentication methodologies are changed. Keeping the same Wi-Fi PSK is done to avoid the requirement of changing the Wi-Fi PSK passwords on every device throughout the network, including network infrastructure equipment and all other Wi-Fi equipment that was not being upgraded, which would be time consuming and resource expensive.

[1] https://www.hhs.gov/answers/hipaa/what-is-phi/index.html
[2] https://en.wikipedia.org/wiki/JTAG
[3] https://www.ebay.com/
[4] https://en.wikipedia.org/wiki/Pre-shared_key
[5] https://www.wigle.net/

TECHNICAL EVALUATION

In the following sections we discuss in detail various technical methods that can be used to extract data from three different brands of infusion pumps that are currently being sold on the secondary market. Although these infusion pump models are no longer being manufactured, they are believed to still be in use by many medical organizations throughout the world. The goal here is to show how simple the process can be to extract data from these devices with less than a few hundred dollars' worth of equipment that can be purchased online.

The equipment used during this exercise included the following device types, with many inexpensive versions also available online that could be used to perform the same functions. This list also includes an estimated price range for each of these devices :

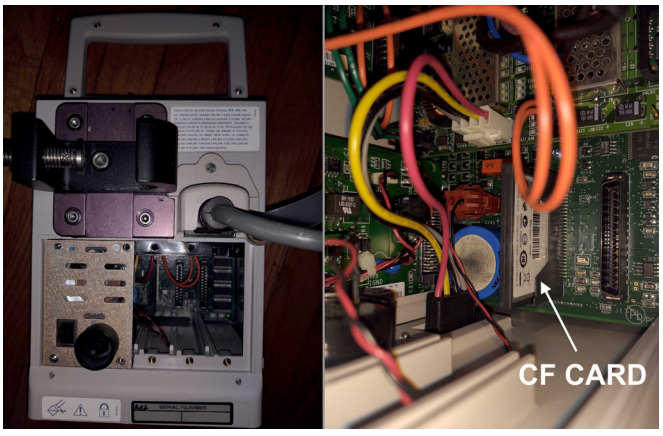
- Flash Memory chip programmer ($75-$150)
- Logic analyzer ($25-$500)
- JTAG programmer ($55-$500)
- Hot air reflow or IR reflow oven ($65-$300)
- Solder iron ($50-$125)

Alaris PC 8015

The first device examined during this study was the Alaris 8015. This device is currently in use in many medical organizations and can also be purchased on the secondary market. During the examination of this device we explored several methods that can be used to extract data from the device's memory storage. These methods included: (1) removal and examination of the internal Compact Flash card, (2) capture of serial communication while using the product's maintenance software, and (3) the physical removal and extraction of data from the flash memory chip on the main circuit board of the device. These three methods are discussed in the following sections and detail how data can be recovered from this product if proper processes are not carried out to flush this critical data prior to the product being de-acquisitioned and sold on the secondary markets.

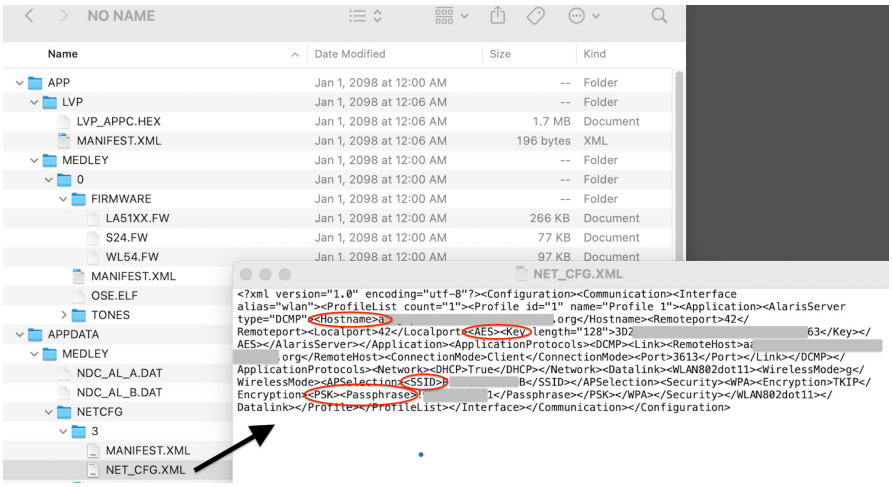
Alaris PC 8015 Non-destructive Analysis of Internal CF Card

Examination of the Alaris device showed that the infusion pump used a 64MB Compact Flash[6] (CF) card for the purpose of storing application and configuration data. The CF card is accessible through the back of the unit by removing a small cover plate and unplugging the card as shown below.



[6] https://en.wikipedia.org/wiki/CompactFlash

Once the CF card was removed from the infusion pump, we used a CF card reader to mount the FAT16 file system and examine it for critical network configuration data. After locating the network configuration file NET_CFG.XLM, we were able to identify detailed information. This information included hostname with domain information, AES keys for encryption, SSID, and the clear text PSK Passphrase. Since this device was purchased on the secondary market, data has been redacted to avoid exposing the previous medical organization's biomedical network Wi-Fi configuration data. An example of this data recovery from the NET_CFG.XLM file is shown below:

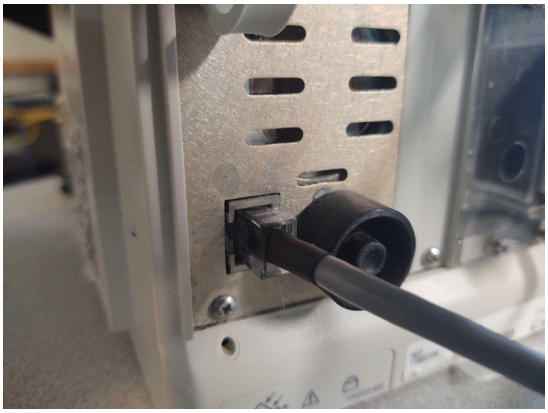


Also it is important to note that the storage of the sensitive data on the CF card was originally reported in CVE-2016-9355[7] and CVE-2016-9375[8]. This affects software versions 9.7 and 9.5 and prior, respectively. To resolve this issue, Alaris recommends updating to the latest version of the software, which removed the stored PSK data from the CF card and only stored this data on the internal flash memory.

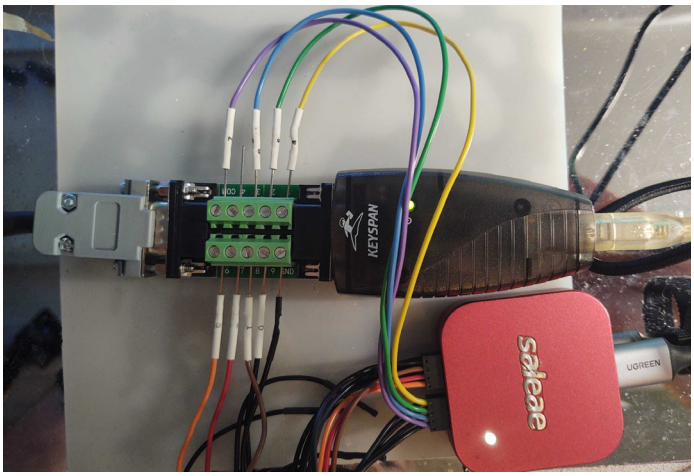
[7] https://nvd.nist.gov/vuln/detail/CVE-2016-9355
[8] https://nvd.nist.gov/vuln/detail/CVE-2016-8375

Alaris PC 8015 Non-destructive Maintenance Port

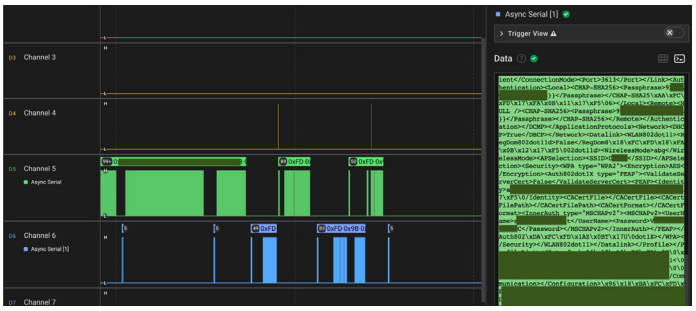
Another non-destructive method for extracting data is via the serial communication port on the back of the device. The device has a RJ45 connection port on the back that supports serial communication. By using the Alaris System Maintenance Software[9], it is possible to interact with the infusion pump to conduct maintenance, perform firmware updates, create device configuration backups, and flush the device settings and logs.



When the maintenance software is used for performing backups of network and Wi-Fi configuration, the software does prevent visual access to critical data such as passwords and keys. This software also forces some form of encryption or encoding of the configuration data before allowing this data to be saved to the local hard drive. The easiest non-destructive recovery method to gain access to the data is to monitor the serial communication while the software is requesting the configuration data from the infusion pump. An example of this is shown below, where we have placed a serial breakout board and a logic analyzer to decode the traffic between the infusion pump and the laptop running the maintenance software.



While examining and decoding serial communication between the secondary-market-purchased Alaris pump and its maintenance software, we found that it was not storing a Wi-Fi Protected Access[10] (WPA) PSK, like many other infusion pumps we had examined during this project, but that it had been configured for Protected Extensible Authentication Protocol[11] (PEAP) authentication and revealed the clear text MSCHAPv2[12] username and password used for Microsoft Active Directory authentication.

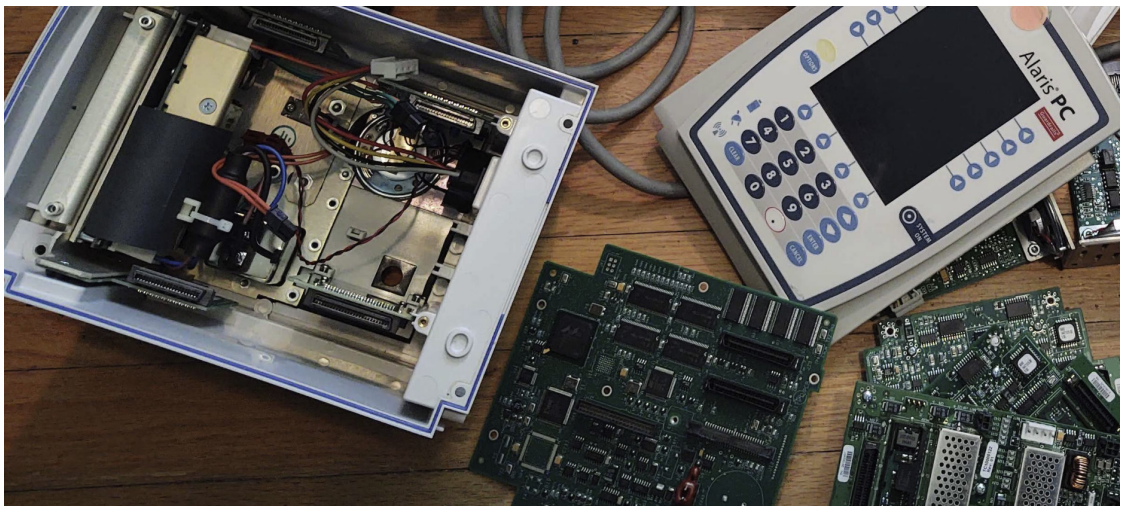


[9] https://www.medonegroup.com/pdf/manuals/techManuals/Alaris-System-Maint-Software-V9.5x-Tech-Manual.pdf
[10] https://en.wikipedia.org/wiki/Wi-Fi_Protected_Access
[11] https://en.wikipedia.org/wiki/Protected_Extensible_Authentication_Protocol
[12] https://en.wikipedia.org/wiki/MS-CHAP

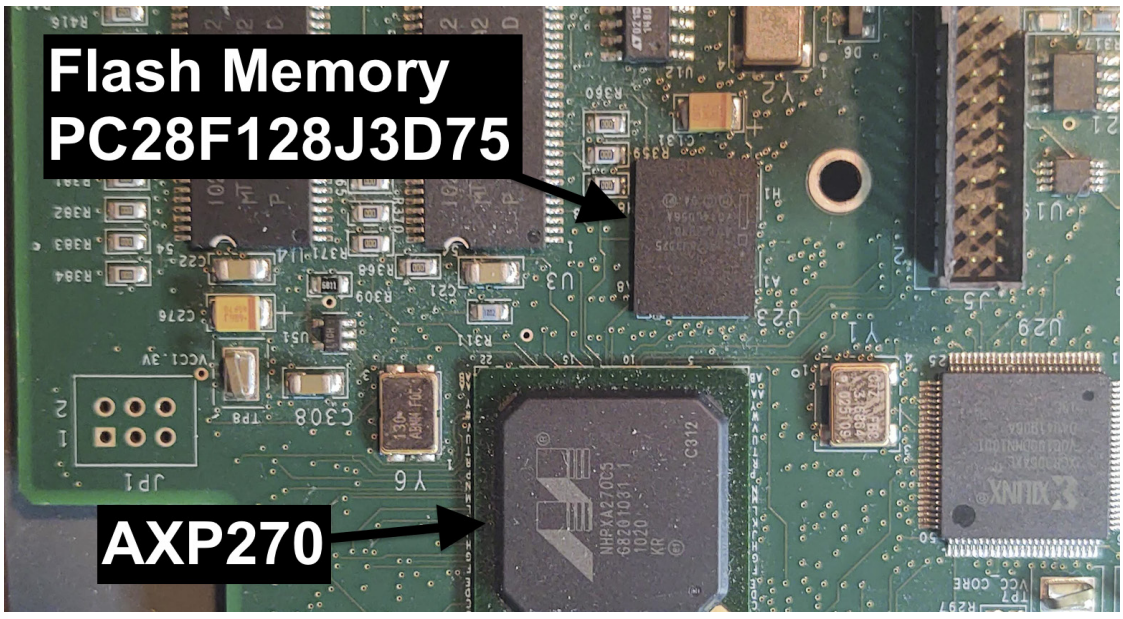
Alaris PC 8015 Unit destructive Analysis

The final method examined for extracting critical data from this infusion pump was a destructive method. The destructive method requires disassembly and removal of the flash memory chip found on the infusion pump's main circuit board. As mentioned before, the later versions of the infusion pump software stopped storing Wi-Fi credentials data on the CF card and only stored that data on the internal flash memory chip.

To gain access to the main circuit board where the flash memory chip is located, we first disassembled the infusion pump as shown below.

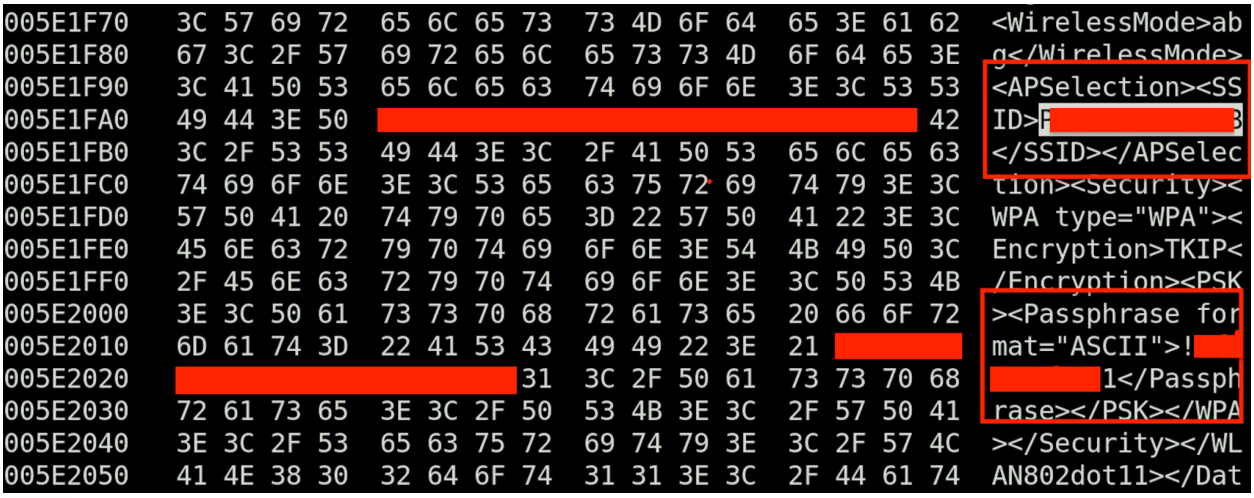


Once the unit was disassembled, the flash memory chip of interest was found located near the primary processor, an AXP270C5. The flash memory chip was identified as an Intel 128Mbit PC28F128J3D75. Since the flash memory chip is a Ball Grid Array[13] (BGA), an infrared reflow system was used to heat the circuit board and melt the solder under the flash memory chip to allow its removal for examination.



[13] https://en.wikipedia.org/wiki/Ball_grid_array

Once the chip was removed we used an RT809H flash chip programmer to extract the data from the flash memory chip. Once extracted, we examined the content of the binary file using the Linux application hexedit. Using hexedit's built-in ASCII search capability, we searched the string data for keywords including SSID and PSK. This method was successful in finding the Wi-Fi configuration settings within the binary image. An example of this is shown below.



Alaris PC 8015 Unit Data Purge Processes

During general evaluation of various documentation found online, there appeared to be no available documented data purge processes for device decommissioning; however, all of these manuals were found to be outdated. Upon meeting with the Alaris security team to discuss this topic, we were informed that they did have documented processes within their product manuals and published security service bulletins for the Alaris 8015 Infusion pump products. Access to these manuals and service bulletins[14] requires a support contract with Becton, Dickinson and Company (BD)[15]. Although we were not able to gain access to this documentation, we did identify through other forms of testing that current maintenance tools do support purging of data such as drug libraries, logs, and network configuration.

If there are any questions or concerns related to processes needed for properly purging data from the Alaris infusion pump or access to needed documentation such as operation manual, service manuals, and service bulletins for the purpose of decommissioning of the products it is recommended that your support teams reach out to BD for clarification and support.

[14] https://www.bd.com/en-us/about-bd/cybersecurity/bulletin/alaris-system-residual-data
[15] https://www.bd.com/en-us

Baxter Sigma Spectrum

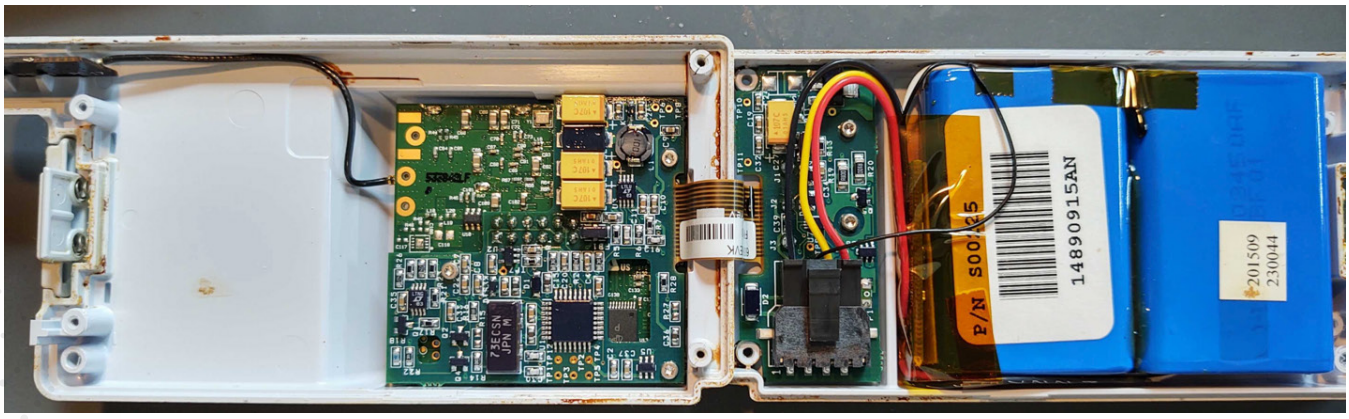
The next device examined during this study focused on the Baxter Sigma Spectrum model 35700BAX2 and associated Wireless Battery Module (WBM) running different software versions. Like the Alaris infusion pump these devices are currently in use in many medical organizations and can also be purchased on the secondary market. During the examination of this device we explored a number of methods that can be used to extract data from the devices memory storage. This infusion pump unit consisted of two independent running systems, which included the infusion pump and the WBM. This allowed for duplicate extraction points of configuration data. In the following section we examine these methods which included destructive examination, JTAG and serial communication between the Infusion pump and the WBM.

Baxter Wireless Battery Module Destructive Analysis

During this study we acquired several Sigma Spectrum units for testing along with 6 WBMs purchased separately from different sources on eBay over a period of 3 months. The WBM used for testing used firmware versions 16, 17, and 20 D29. The battery module shown in this teardown example is running software version 16.

Our first step was to open the WBM to gain access to the internal circuitry to identify the flash memory storage on the device.

Once disassembled, we determined that the 35162 battery module had a Digikey processor and was using an STMicroelectronic M29DW323DB 32Mbit TFBGA48 Flash memory chip for data storage.



To examine the content of this flash memory device we used an infrared reflow oven to heat the circuit board and remove the flash memory chip. Once removed we used an RT809H chip programmer to extract the data from the flash chip. We then examined the binary data by conducting a search using the Linux command "strings" to locate SSID information. This initial test returned no results. So next we examined the binary using the Linux application "hexedit". During this examination we noticed that ascii string data appeared garbled, under further examination it was determined that the device was storing the data using a big endian[16] method where two-byte data pairs are swapped making the ascii data difficult to read. An example of this is shown below.


```
003C0840   2C 37 32 33  36 37 2C 38  6F 56 75 6C  65 6D 49 20  ,72367,8oVulemI
003C0850   63 6E 65 72  65 6D 74 6E  2F 3C 56 45  4E 45 3E 54  cneremtn/<VENE>T
003C0860   44 3C 55 52  3E 47 2C 31  53 4E 28 20  6F 4E 6D 72  D<UR>G,1SN( oNmr
003C0870   6C 61 53 20  6C 61 6E 69  29 65 33 2C  33 37 32 39  laS lani)e3,3729
```

To resolve this issue the following command was used on a Linux platform to swap the two-byte pairs to place it in order that could be evaluated and decoded.

```
xxd -e -g2 in-file.bin | xxd -r > out-file.bin
```

After running the above command, the data byte order was corrected, and the ascii data was more easily scanned and processed. A sample of the corrected data is shown below.

```
003C0840   37 2C 33 32  37 36 38 2C  56 6F 6C 75  6D 65 20 49  7,32768,Volume I
003C0850   6E 63 72 65  6D 65 6E 74  3C 2F 45 56  45 4E 54 3E  ncrement</EVENT>
003C0860   3C 44 52 55  47 3E 31 2C  4E 53 20 28  4E 6F 72 6D  <DRUG>1,NS (Norm
003C0870   61 6C 20 53  61 6C 69 6E  65 29 2C 33  37 33 39 32  al Saline),37392
```

Once corrected, we ran the search within the hexedit application searching for the Wi-Fi SSID that the device was attempting to connect to, information we determined early on via Wi-Fi sniffer. We located the SSID at memory offset of 0x3E01EC. Further examination of the binary data also revealed that the WPA passphrase had been converted to a 64-character hex key (PSK) and had been stored at the memory offset of 0x3E0260. Converting of passphrases to 64-character key is a common process since all operating systems require that 64-character key for authentication

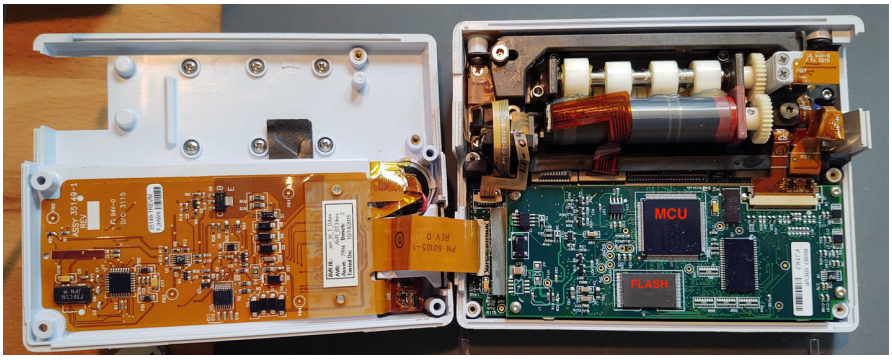
[16] https://developer.mozilla.org/en-US/docs/Glossary/Endianness

to a WPA enabled Wi-Fi access point and by storing it as a 64-character key it will help speed up the authenticating process when it is needed. An example of this is shown below.

```
003E01E0   3A 30 30 00  00 00 00 00  00 00 00 00  64 [redacted]   :00........d[redacted]
003E01F0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00   ................
003E0200   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 04   ................
003E0210   00 00 00 00  00 40 9D 59  AE C8 00 00  00 00 00 08   .....@.Y........
003E0220   00 00 00 08  00 00 00 00  00 00 00 00  00 00 00 00   ................
003E0230   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00   ................
003E0240   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00   ................
003E0250   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00   ................
003E0260   A1 [redacted]                                        .[redacted]
003E0270   [redacted]                                E7         [redacted].
```

Baxter Infusion Pump Unit Destructive Analysis

Now that we found the Wi-Fi credentials stored on the WBM the next test was to evaluate the main Baxter infusion pump unit to see if that data is also being stored there. To accomplish this we again conducted a destructive approach by disassembling the infusion pump to identify flash memory chips used to store device firmware and data.

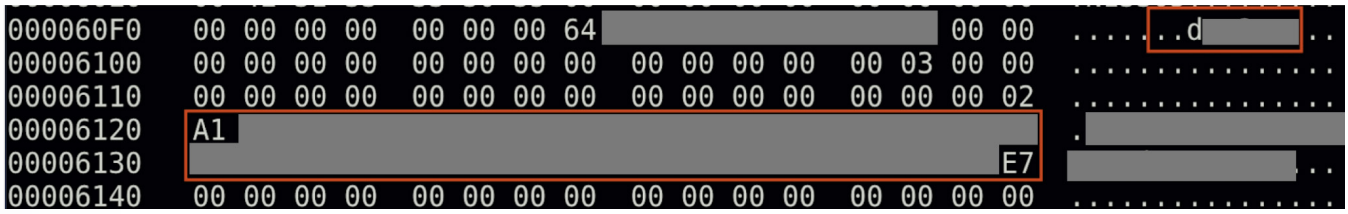


The main circuit board located above on the right side was found to contain the primary microcontroller unit (MCU), an NXP LH79520, along with a flash memory chip, a Spansion S29JL064J 64Mbit TSOP48. Prior to removing the flash memory chip, we found that all circuit boards inside the infusion pump had been coated with a conformal[17] coating to protect the circuits from moisture. So, prior to desoldering the coating had to be scrapped away from solder joints on the flash memory chip. Once this had been completed, we de-soldered the 48-pin flash memory chip and placed it in a RT809H chip programmer and extracted the stored memory for offline analysis.

Unlike the data extracted from the flash memory chip on the WBM, this data was not stored in an endian mode format that prevented it from easily being read or processed. The SSID was found located near the memory offset of

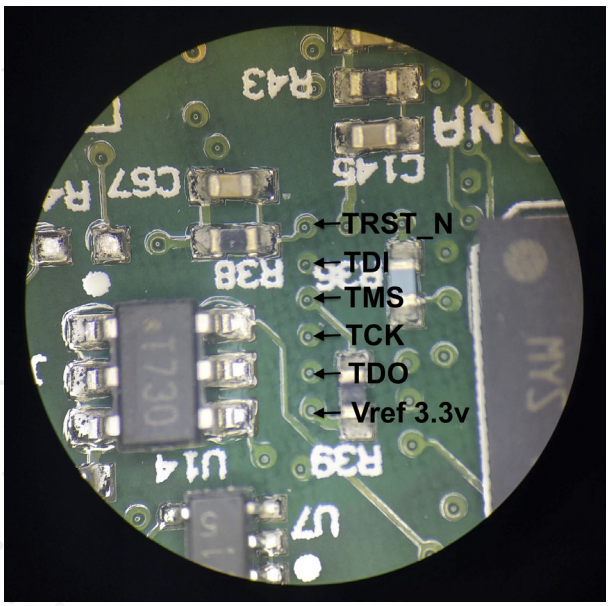
[17] https://en.wikipedia.org/wiki/Conformal_coating

0x06F0. Further examination of the binary data also revealed that the WPA passphrase, like the battery module, had been converted to a 64-character hex key (PSK) and had been stored at the memory offset of 0x6120, as shown in the following figure.

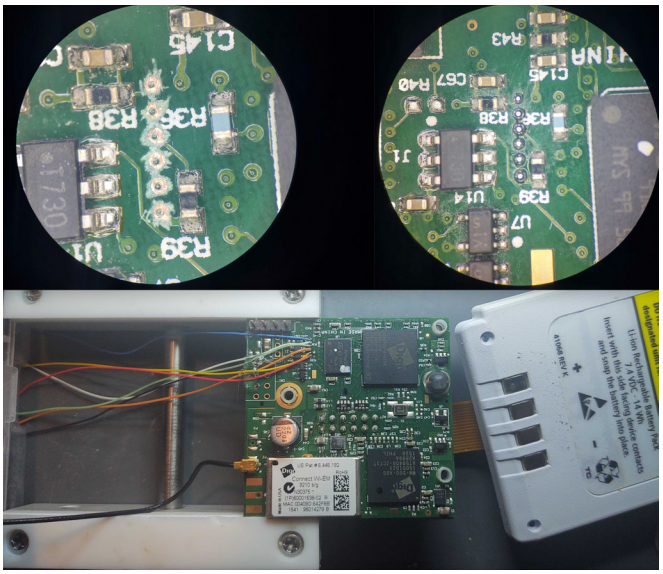


Baxter Wireless Battery Module Non-destructive Analysis Via JTAG

Next we explored gaining access to the flash memory using a non-destructive method. In this non-destructive method we used JTAG to gain access to the flash memory on the WBM, which was running software version 17. The WBM did not have a header that was easily accessible for connecting to JTAG. To locate possible JTAG connections we were required to closely examine the circuit board along with using the development manual for the Digi NS9210 MCU to identify needed pinout information. We were then able to trace circuit board paths from the MCU which allowed access to JTAG. An example of these connection points are shown below:



To attach to these connection points we cleaned off the masking and conformal coating and tin'd the connection points with solder to facilitate the connecting of 30 gauge wire wrap wire to them as shown below:



Once wired, we attached a Segger JLink JTAG debugger device to the WBM and used the JLink Commander application to extract the firmware and data stored within the flash memory of the WBM. For this device we used the JLink debugger device setting for a Digi NS9360. Also noted that a generic ARM9 setting could have also been used. After running savebin, like a previous issue, we identified that the output ASCII data was unreadable because of endian mode being used. To resolve this issue before running the savebin command again we ran the BE command to set the output as Big Endian to assure the output data would be in a readable byte order.

```
J-Link>connect
Device "NS9360" selected.


Connecting to target via JTAG
TotalIRLen = 4, IRPrint = 0x01
JTAG chain detection found 1 devices:
 #0 Id: 0x27926031, IRLen: 04, ARM926EJ-S Core
CP15.0.0: 0x41069265: ARM, Architecture 5TEJ
CP15.0.1: 0x1D0D20D2: ICache: 4kB (4*32*32), DCache: 4kB (4*32*32)
Cache type: Separate, Write-back, Format C (WT supported)
ARM9 identified.
J-Link>BE
J-Link>savebin c:\Users\percx\baxter\memdump-BE-2.bin, 0x0000, 0x800000
Opening binary file for writing... [c:\Users\percx\baxter\memdump-BE-2.bin]
Reading 8388608 bytes from addr 0x00000000 into file...O.K.
J-Link>
```

Once firmware and data was extracted, a search of the data allowed us to find the stored Wi-Fi WPA 64-character PSK . Similar to the destructive test method, this data is found by searching for the known SSID and then examining data coming after that. Also important to note that each firmware version will typically store this WPA PSK data in a different memory offset. In this case firmware version 17is found at offset 0x2D3F50 as shown below:

```
002D3EB0   FF FF C7 C0  33 2E 32 2E  30 2F 30 32  3A 30 30 3A  ....3.2.0/02:00:
002D3EC0   30 30 00 00  31 31 2E 31  2E 30 2F 30  32 3A 30 30  00..11.1.0/02:00
002D3ED0   3A 30 30 00  00 00 00 00  00 00 00 00  54 [redacted] :00.........T[redacted]
002D3EE0   [redacted]  00 00  00 00 00 00  00 00 00 00  [redacted].........
002D3EF0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 07  ................
002D3F00   00 00 00 00  00 40 9D 64  2F 8B 00 00  00 00 00 08  .....@.d/.......
002D3F10   00 00 00 08  00 00 00 00  00 00 00 00  00 00 00 00  ................
002D3F20   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
002D3F30   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
002D3F40   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
002D3F50   EA [redacted] ..[redacted]
002D3F60   [redacted] 32 [redacted] 2
002D3F70   80 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
```

Baxter Infusion Pump Unit Non-destructive JTAG Access Analysis

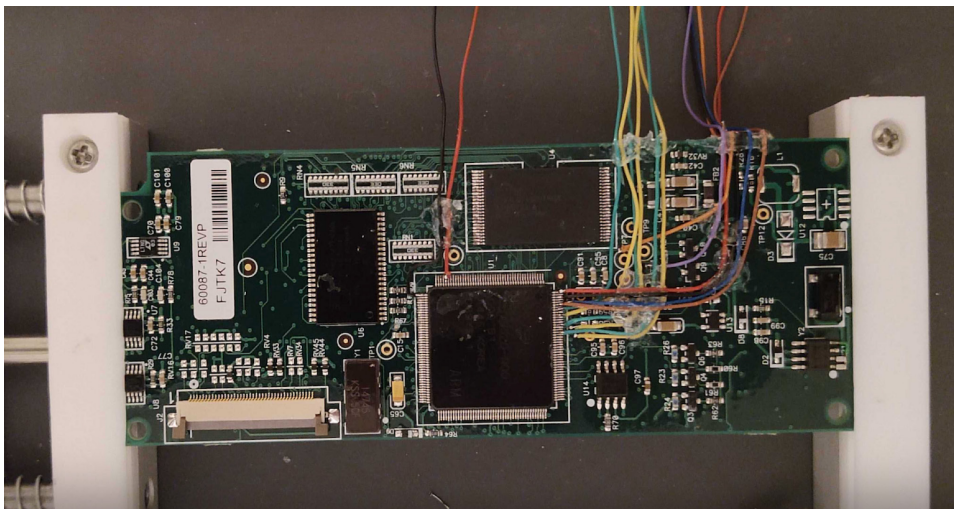
After examining the WBM, we focused on JTAG access on the main infusion pump. To accomplish this we started by disassembling the infusion pump and examined the main circuit board. On the main circuit board of the infusion pump we found it to be using an ARM7 NXP LH79520 as the primary MCU. Examination of the LH79520[18] datasheet for this MCU showed the pinout for JTAG access as:

PIN Number	NAME
170	TMS

[18] https://www.keil.com/dd/docs/datashts/philips/lh79520_ds.pdf

PIN Number	NAME
171	TDO
172	TDI
173	TCLK
174	nTRST
176	TEST2 (Pull high to enable JTAG)

Like the WBM, the infusion pump did not have any defined headers for JTAG access, so to connect up and interact with the JTAG of the MCU required direct connecting to the circuit board and the MCU. This was done using 30 gauge wire-wrap wire as shown below. Note the yellow and green wires were attached for universal asynchronous receiver transmitter (UART)[19] testing.



[19] https://en.wikipedia.org/wiki/Universal_asynchronous_receiver-transmitter

Once JTAG pins were wired we attached a Segger JLink JTAG debugger device to the infusion pump and used the JLink Commander application to extract the firmware and data stored within the flash memory of the device. The JLink command device settings for a NXP LH79520 MCU were used. Also a generic ARM7 setting could be chosen. Prior to executing a connection with JLink the device was powered off and pin 176 was pulled high prior to powering the device backup. This was done to enable JTAG on the MCU as defined by the MCU's datasheet. Below is an example of the connect, halt, and savebin commands used to extract and save the flash memory data to a binary file.

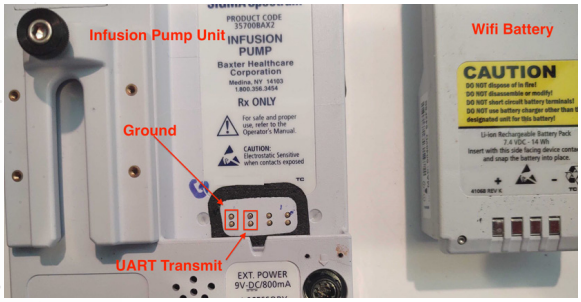
```
J-Link>connect
Device "LH79520" selected.


Connecting to target via JTAG
TotalIRLen = 4, IRPrint = 0x01
JTAG chain detection found 1 devices:
 #0 Id: 0x0F0F0F0F, IRLen: 04, ARM7TDMI Core
ARM7 identified.
J-Link>halt
PC: (R15) = 4013ED2E, CPSR = 00000033 (SVC mode, THUMB)
Current:
     R0 =0423B17E, R1 =00000400, R2 =0423B17E, R3 =4011A8DC
     R4 =000000D8, R5 =00000086, R6 =04DB2615, R7 =401EDBD0
     R8 =00000000, R9 =9FF93BDF, R10=1F572EB6, R11=00000000, R12=00000000
     R13=480272D8, R14=40144877, SPSR=20000033
USR: R8 =00000000, R9 =9FF93BDF, R10=1F572EB6, R11=00000000, R12=00000000
     R13=EBFEC7FF, R14=32FEDFF7
FIQ: R8 =FFFC402C, R9 =00000040, R10=0000007F, R11=77A9D7F7, R12=A2747E6F
     R13=48080000, R14=40120948, SPSR=60000033
IRQ: R13=48071D18, R14=4013ECF6, SPSR=20000033
SVC: R13=480272D8, R14=40144877, SPSR=20000033
ABT: R13=48071CF8, R14=7BBAF7BE, SPSR=D0000014
UND: R13=48071CD8, R14=EA7158DF, SPSR=9000001C
J-Link>savebin c:\Users\percx\baxter\Baxter_Pump_NXP-LH79520, 0x0000, 0x800000
Opening binary file for writing... [c:\Users\percx\baxter\Baxter_Pump_NXP-LH79520]
Reading 8388608 bytes from addr 0x00000000 into file...O.K.
J-Link>
```

Once data was extracted, a search of the data allowed us to find the stored Wi-Fi WPA 64-character PSK. A similar method was used during the WBM testing in the previous section. By searching for the identified SSID, and then examining data coming after that it is possible to find the WPA 64-character PSK. In this case the key was found at offset 0x6120 as shown below:

```
00006090   00 00 80 3F  00 00 80 3F  00 00 00 00  0D 00 00 00  ...?...?........
000060A0   05 00 00 10  6C 04 A8 C0  01 04 A8 C0  00 FF FF FF  ....l...........
000060B0   01 04 A8 C0  00 00 00 00  00 00 00 00  00 00 00 00  ................
000060C0   00 00 00 00  00 00 00 00  00 00 00 00  30 00 00 00  ............0...
000060D0   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
000060E0   00 4E 31 33  33 30 35 00  00 00 00 00  00 00 00 00  .N13305.........
000060F0   00 00 00 00  00 00 00 54[redacted]                  .......T[redacted]
00006100   00 00 00 00  00 00 00 00  00 00 00 00  00 03 00 00  ................
00006110   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 03  ................
00006120   EA[redacted]                                        ..[redacted]
00006130   [redacted]32                                        [redacted].2
00006140   00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00  ................
```

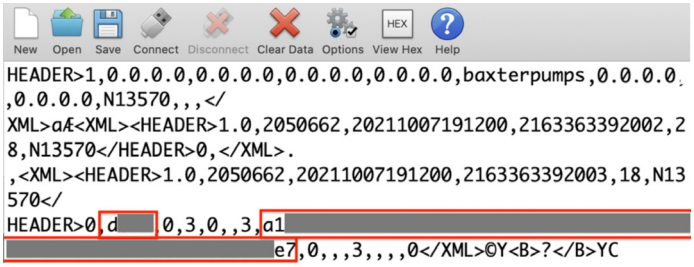
Baxter Non-destructive Inter-component Communication Analysis

During testing it was found that the infusion pump unit transfers Wi-Fi configuration data to the WBM during initial power up. This was discovered by using a logic analyzer to monitor the contact points between the units. Examples of these connections are shown in figure below. It was determined that the communication between the main infusion pump and WBM was using UART.



To capture the UART communication between the devices we created a simple shim using a flexible[20] development breakout board. This was done so the battery module could be properly installed for normal operations, but still allow us the ability to tap into and capture that communication in real time. An example of the shim and how it was placed between units is shown in the following figures.

Using standard default UART settings along with a commonly used baud rate of 57600 we were able to capture the Wi-Fi configuration data being passed to the WBM during unit power up and initialization. This captured data is shown in the following figure containing SSID and the 64 hex character PSK :



Baxter Sigma Spectrum Data Purge Process

Baxter documentation such as Spectrum Operator's Manual and Service Manual cover operational steps that can be used for resetting wireless configurations, removal of logs, and drug library from the device. A more comprehensive process for decommissioning of the Sigma Spectrum infusion pump and the WBM is currently addressed in the Product Security Bulletin[21] published on September 8, 2022. Their recommendations cover wiping data from both the infusion pump unit as well as the WBM. The process for the infusion pump unit was to remove the network settings and drug library through the standard menus on the device. The battery module was slightly more complex in that the pump needs to have the network configuration reset and networking enabled to start the process. Then the battery module must be inserted into the pump and the user must wait for the network settings to transfer to the battery module, which is designated by the network icon turning a yellow color. After this step, the WBM will have the previous network configuration wiped off of the memory. These processes have been tested and validated as a viable method to remove the network configuration and other data from the device in preparation for decommissioning. Also, Baxter will be adding this decommissioning process to their product documentation in future releases.

If there are any questions or concerns related to the processes described within Baxter's Security Bulletin for decommissioning the Baxter infusion pump products or Sigma Spectrum manuals are needed it is recommended that your support teams reach out to Baxter for clarification and support.

[20] https://www.adafruit.com/product/1518?gclid=Cj0KCQiAic6eBhCoARIsANlox84bQ9-BoQmvZFfl1zelKiOG2SjH3mSNO_Qt8Eqp-R9DIyDBHGROXBUaAp7zEALw_wcB
[21] https://www.baxter.com/sites/g/files/ebysai3896/files/2022-09/ICSMA-22-251-01_0.pdf
[22] https://www.baxter.com/contact-us

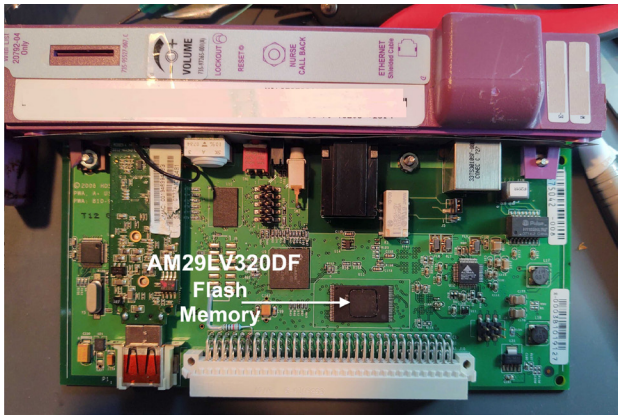
Hospira Abbott PLUM A+ - MedNet

The last device examined during this study focused on the Hospira Abbott PLUM A+ with MedNet running software version 13.40. Like the other devices this infusion pump, although no longer manufactured, is still currently in use in many medical organizations and can also be purchased on the secondary markets. The Hospira devices, being examined in this writeup, were purchased off of eBay for the purpose of data examination using various methods. The methods covered in this section include JTAG, RS232, and destructive extraction of the flash memory chips.

Hospira Infusion Pump Unit Destructive Analysis

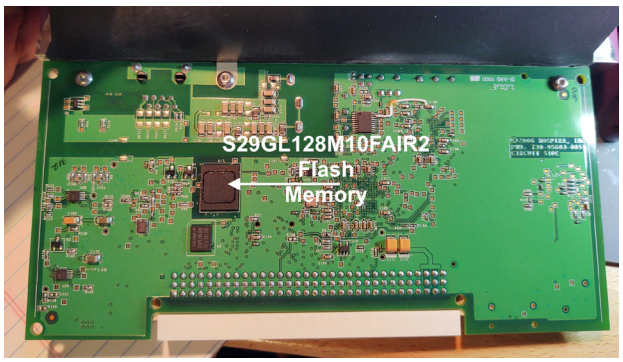
In the initial testing the first device was disassembled to identify flash memory, MCUs, and connection headers. Analysis showed that the main board for the MedNet communication board could be extracted by removing two screws and unplugging it. This board was found to contain two flash memory chips.

The first flash memory chip identified was a TSOP48 package type. After removing the sticker from its surface we identified the flash memory chip as an AMD AM29LV320DT-90EC 32 Megabit flash memory.



To examine the content of this flash memory device we used ChipQuick[23] a low temperature desoldering metal to remove the TSOP48 device. Once removed we used an RT809H chip programmer to extract the data from the flash chip. Next we used Linux command strings to examine the binary file and found this to contain what appeared to be infusion pump drug library data and user interface and pump operational control code. No configuration data was identified on the flash memory chip.

The second flash memory chip examined was a FBGA64 package device. Once the sticker was removed this chip we identified it as a Spanion S29GL128M10FAIR2 flash memory chip.



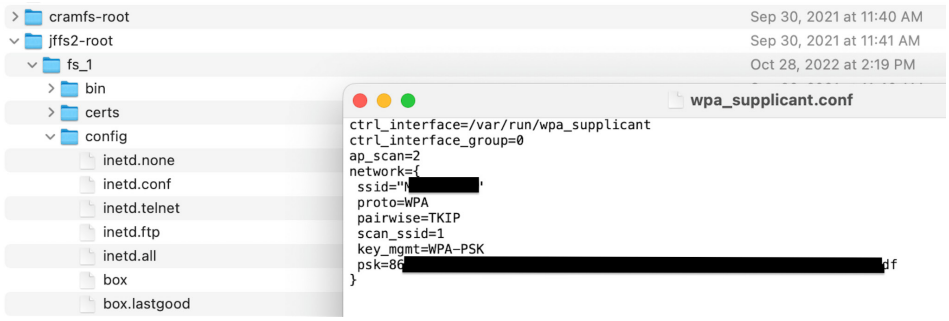
Examination of the content of this flash memory device we used an infrared reflow oven to heat the circuit board and remove the FBGA64 chip. Once removed we again used an RT809H chip programmer to extract the binary data from the flash chip. After extracting we used the

[23] https://www.chipquik.com/store/index.php?cPath=200

application binwalk[24] to initially examine the binary and discovered it contained CramFS and JFFS2 file system structure for LxNETES, an embedded Linux OS.

```
DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
4608          0x1200          CramFS filesystem, little endian, size: 4
046848 version 2 sorted_dirs CRC 0x92FC745D, edition 0, 1366 blocks, 21
0 files
4194304       0x400000        JFFS2 filesystem, little endian
4194452       0x400094        Zlib compressed data, compressed
4194664       0x400168        JFFS2 filesystem, little endian
```

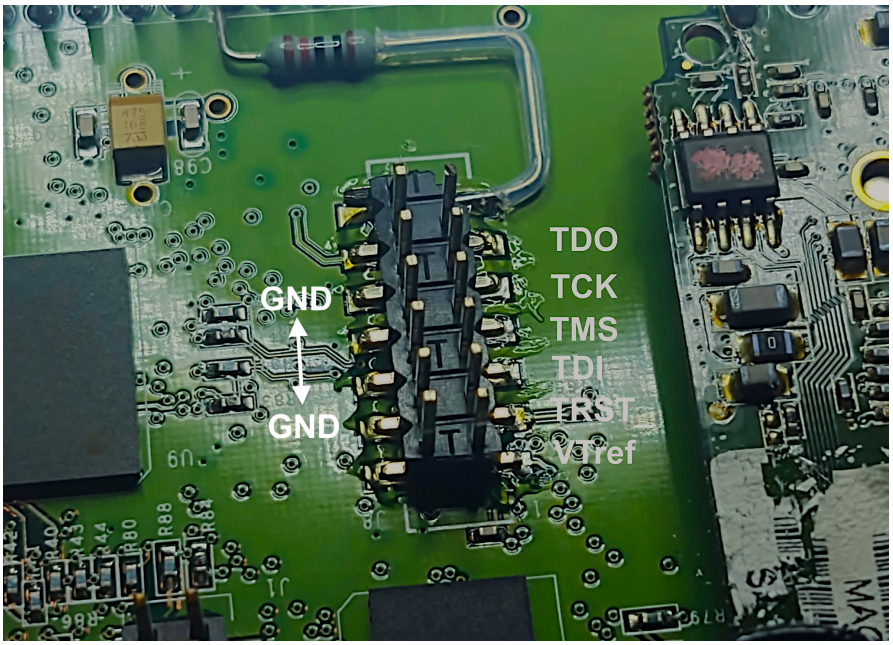
Using binwalk we extracted most of the structured file systems and were able to examine the configuration file wpa_supplicant.conf and extract the stored SSID and WPA PSK as shown below. Again like other examples we have blanked out the data to avoid exposing the original medical organizations credentials that were found on this device.



Hospira Infusion Pump Unit JTAG Non-destructive Analysis

To gain access to the flash memory using a non-destructive method for JTAG access, on a second Hospira infusion pump, we opened the device and identified the MCU as a Digi NS7520 which has a ARM7TDMI core and also 14 pin header pads on the circuit board believed to be for JTAG. To validate the JTAG connections we examined a non-functional damaged device by removing the primary MCU to gain access to the BGA. Using this along with the Digi NS7520 hardware reference manual[25] and a multimeter, we traced the circuit board paths from the MCU to the 14 pin header to validate it as a JTAG header and identify the pinout connections. Once these header pads were confirmed we soldered a 2.54mm gull wing header to the circuit boards as shown below.

[24] https://github.com/ReFirmLabs/binwalk
[25] https://www.digi.com/resources/documentation/digidocs/PDFs/90000353.pdf



The identification of the JTAG header pinout could have also been accomplished by using a tool such as a jtagulator[26] or similar device used for enumerating JTAG pins.

Once the JTAG pinout was properly identified we then attached a Segger JLink JTAG debugger device to the infusion pump and used the JLink Commander application to connect to and interact with the MCU. Using the NS7520 datasheet as a resource we identified the offset to the flash memory chip to be at hex address 0x10000000. The size of the flash memory chip, S29GL128M10FAIR2, was 16777216 bytes. Using the JLink commander application we attempted to read the memory with the savebin command. As shown below, the JLink failed to read memory because of memory access timeout during the read process.

```
J-Link>savebin dumpflash.bin, 0x10000000, 0x1000000
Opening binary file for writing... [dumpflash.bin]
Reading 16777216 bytes from addr 0x10000000 into file...Memory access: CPU temp. halted: I
ccesses#Legacy_stop_mode

****** Error: Read memory error @ address 0x10038000, word access: Memory access timeout.
Could not read memory.
J-Link>
```

Upon further testing and evaluation it was discovered that following process steps were effective in extracting all of the flash memory. The first step is to wait until the infusion pump is fully booted before initiating the connection command from the JLink Commander. Next we set the JLink connection device type to ARM7, target interface to JTAG, device position to auto-detect, and speed to 5000. The higher speed allows reading over 256k of memory before the memory access timeout would occur.

[26] http://www.grandideastudio.com/jtagulator/

```
J-Link>connect
Please specify device / core. <Default>: ARM7
Type '?' for selection dialog
Device>
Please specify target interface:
  J) JTAG (Default)
TIF>
Device position in JTAG chain (IRPre,DRPre) <Default>: -1,-1 => Auto-detect
JTAGConf>
Specify target interface speed [kHz]. <Default>: 4000 kHz
Speed>5000
Device "ARM7" selected.


Connecting to target via JTAG
TotalIRLen = 4, IRPrint = 0x01
JTAG chain detection found 1 devices:
 #0 Id: 0x3F0F0F0F, IRLen: 04, ARM7TDMI Core
Memory zones:
  Zone: Default Description: Default access mode
ARM7 identified.
```

Then by reading the data out of the flash in 256k blocks we can avoid the memory access timeout issue and the process reset and reboot that was triggered when the flash memory read timeout occurred. By reading the data in 256k blocks we were required to run 64 separate read operations to pull all of the flash chips memory. The process can be done by using the following savebin command

```
savebin d1.bin, 0x10000000, 0x40000
```

After each read operation the memory offset was incremented by 0x40000 along with changing the file name that the data was written out to. The following image shows an example of this process:

```
J-Link>savebin d59.bin, 0x10e80000, 0x40000
Opening binary file for writing... [d59.bin]
Reading 262144 bytes from addr 0x10E80000 into file...O.K.
J-Link>savebin d60.bin, 0x10ec0000, 0x40000
Opening binary file for writing... [d60.bin]
Reading 262144 bytes from addr 0x10EC0000 into file...O.K.
J-Link>savebin d61.bin, 0x10f00000, 0x40000
Opening binary file for writing... [d61.bin]
Reading 262144 bytes from addr 0x10F00000 into file...O.K.
J-Link>savebin d62.bin, 0x10f40000, 0x40000
Opening binary file for writing... [d62.bin]
Reading 262144 bytes from addr 0x10F40000 into file...O.K.
J-Link>savebin d63.bin, 0x10f80000, 0x40000
Opening binary file for writing... [d63.bin]
Reading 262144 bytes from addr 0x10F80000 into file...O.K.
J-Link>savebin d64.bin, 0x10fc0000, 0x40000
Opening binary file for writing... [d64.bin]
Reading 262144 bytes from addr 0x10FC0000 into file...O.K.
```

After all 64 of 256k blocks had been read, we then concatenated them together in proper order into a single file. Once that was completed the binary file was processed using binwalk as described above in the Hospira Infusion Pump Unit Destructive Analysis.

Once binwalk completed extracting most of the file system structure was recovered and we were able to examine the configuration file wpa_supplicant.conf and extract the stored SSID and WPA PSK as shown below.

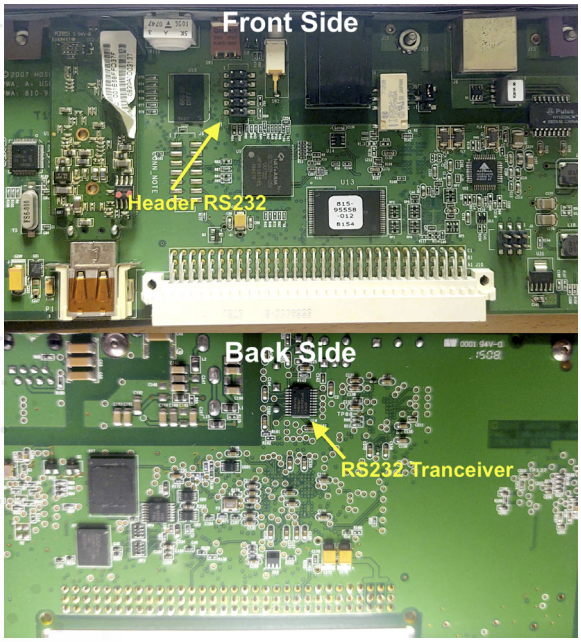
wpa_supplicant.conf

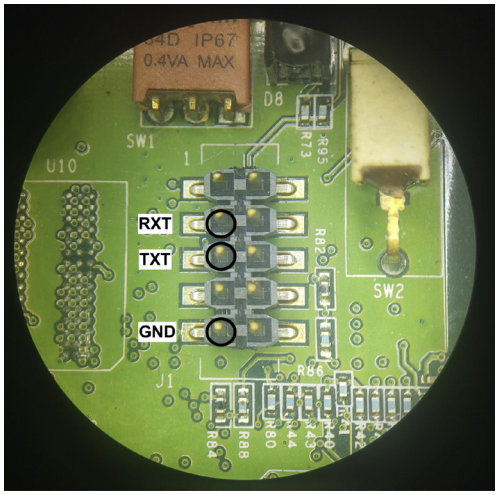
```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
ap_scan=2
network={
 ssid="[redacted]"
 proto=WPA
 pairwise=TKIP
 scan_ssid=1
 key_mgmt=WPA-PSK
 psk=86[redacted]df
}
```

Hospira Infusion Pump Unit RS232 Non-destructive Analysis

Further examination of the Hospira identified an RS232 transceiver chip Sipex SP3222E, shown below on the Back Side image of the MedNet circuit board. This chip was found to be connected to the 2.54mm 10 pin header shown on the Front Side image.

Using a logic analyzer on the 10 pin header we were able to determine which pins were active on the RS232 header and also identified that it exposed a console connection to the primary NS7520 MCU. The image below shows the correct pins for the active RS232 RXT, TXT and GND for console interaction.





Using a Waveshare[27] USB to RS232/485/TTL device and serial communication application with the following settings:

- Baud rate: 38400
- Data Bits: 8
- Parity: None
- Stop Bit: 1

We were able to connect and interact via the RS232 serial Once connected it was determined that we had root level

Untitled_0

New Open Save Connect Disconnect Clear Data Options View Hex Help

```
LxNETES Bootloader $Revision: 1.8 $

ABCDEFGHJKLHNO@00001618

Bootstrap:


Linux version 2.4.22-uc0-fs2 (svc-build@reunion2) (gcc
version 2.95.3 20010315 (release)(ColdFire patches -
20010318 from http://fiddes.net/coldfire/)(uClinux XIP and
shared lib patches from http://www.snapgear.com/)) #614 Wed
Feb 21 14:28:16 PST 2007 CE Type - Sedona

Processor: ARM/VLSI ARM 7 TDMI revision 0

NetSilicon Chip Revision: 0x29 (NS7520)

Architecture: Custom
```

[27] https://www.waveshare.com/usb-to-rs232-485-ttl.htm

access. This allowed us to identify and open the file /ram/mnt/jffs2/config/wpa_supplicant.conf and extract the stored SSID and WPA PSK as shown in the image below.

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
ap_scan=2
network={
 ssid="p[redacted]n"
 proto=WPA
 pairwise=TKIP
 scan_ssid=1
 key_mgmt=WPA-PSK
 psk=e2b[redacted]e8a
}
```

Hospira Infusion Pump Data Purge Process

A little history on the Hospira infusion pump product line. This product has changed hands several times over the years. First it was acquired by Pfizer[28] in 2015 and then was sold to ICU Medical Inc.[29] in 2017.

During general evaluation of various documentation found online for the Hospira Plum A+ MedNet unit no single procedure could be located that detailed the needed steps for removing all critical data such as PHI, and Wi-Fi configuration data in preparation of decommissioning and transfer of an infusion pump. Although, support documentation did reference a factory reset which would reset the ethernet IP address back to 192.168.0.100. This process was found located in section 6.4.2 in the manual Hospira MedNet Technical Service Manual[30]. Also, the document "Hospira MedNet Wireless connectivity Engine Configuration Guide"[31] which we located on the fccid.io website detailed information for configuration of the Wireless MedNet unit, it is assumed this access could also provide a means to reset and or remove the Wi-Fi configuration data.

If there are any questions or concerns related to the proper methods used to perform a factory reset and purging all critical data from the Hospira infusion pump products it is highly recommended that your support teams reach out to ICU Medical Inc. for clarification.

During general evaluation of various documentation found online for the Hospira Plum A+ MedNet unit no single procedure could be located that detailed the needed steps for removing all critical data such as PHI, and Wi-Fi configuration data in preparation of decommissioning and transfer of an infusion pump.

[28] https://www.pfizer.com/news/press-release/press-release-detail/pfizer-completes-acquisition-hospira
[29] https://www.icumed.com/about-us/news-events/news/2017/icu-medical-completes-the-acquisition-of-hospira-infusion-systems-from-pfizer
[30] https://www.medonegroup.com/pdf/manuals/techManuals/Plum-A-Plus-3-with-Hospira-MedNet.pdf
[31] https://fccid.io/STJ80411396001/User-Manual/User-Manual-769980.pdf

CONCLUSION

Through various testing methods described within this document a total of 13 infusion pump devices were examined. During this examination, eight of the devices were found to contain Wi-Fi PSK access credentials and one of the devices was found to contain PEAP authentication data for Windows Active Directory.

The discovery of this data on de-acquisitioned medical devices being sold on the secondary market points out a serious systemic issue. To effectively resolve this issue, organizations that leverage medical technologies should build out policies and processes to properly handle the acquisition and de-acquisition of medical technology. The policies should define ownership and governance of the processes within the organization and what is expected to maintain solid security and protection of the data that is stored on these devices, which may include critical infrastructure configuration data and PHI related data.

To support security requirements, processes need to be defined — especially processes around de-acquisition of medical devices. These processes should detail steps on how the data is purged from the device and how that is confirmed prior to sending the devices out for resale or redistribution to other organizations outside the control of the initial owner. For medical devices that are leased and are not owned by the medical organization, some form of contractual arrangement needs to be made during the acquisition phase that includes processes and expectations for purging the data from the devices prior to resale or redistribution. By developing and following defined policies and processes that support the purging of critical data from the devices prior to de-acquisition, issues like what we documented in this research paper can be completely avoided.

For further guidance on properly securing infusion pumps, the National Institute of Standards and Technology (NIST) has released a series of special publications NIST.SP.1800-8[32] that can be used to build out the needed security controls for the purpose of managing and maintaining a secure infusion pump environment.

The information provided in this report is intended for informational purposes only and Rapid7 makes no warranties, express or implied, regarding the suitability of the content for any specific purpose. The content within this report is based on data and findings available up to the date of its publication, which is mentioned within the document.

The information contained herein is provided "as is," and readers are advised to use their own discretion when applying the information to their specific situations. Furthermore, any third-party sources, tools, or software mentioned in this report are included for informational purposes only. Rapid7 does not take responsibility for the accuracy, functionality, or security of these external resources.

Rapid7 is not liable for any damages, losses, or consequences that may arise from the use of the information provided within. This includes but is not limited to direct, indirect, incidental, or consequential damages related to actions taken based on the content of this report.

Any reproduction, distribution, or unauthorized use of this report's contents without explicit permission from the authors and publishers is strictly prohibited.