View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Electronic archive of Tomsk Polytechnic University

Available online at
http://jess.esrae.ru/

“Journal of Economics
and Social Sciences”

Analysis of Canon CAPT protocol for Linux printer support improvement

Tomsk Polytechnic University

Alexei Gordeev

Postgraduate, Systems Engineering Department, Siberian State Technological University

Abstract

The following paper discusses major deficiencies found in Canon’s own proprietary Advanced Printing Technology (CAPT) driver for Linux distributions. It points out the existence of experimental, but more clean and completely open source driver based on several previous reverse engineering attempts and poses a problem of its incompatibility with a particular printer model (LBP3000 in this case) in question. Then it proceeds to describe the effort of analysis through observation of captured conversation between Canon's own proprietary driver and the printer to point out the differences between the inner workings of original and open source drivers.
Finally, it describes the implementation of printer's support in an open source driver and concludes with the successful result of producing a driver that is able to work under modern Linux distributions and share a CAPT printer on a heterogeneous local area network.

Keywords: Canon, Advanced, Printing, Technology, Printer, CAPT, Wireshark, Protocol, Capture, Analysis;

1. Introduction

Reverse engineering is a process of extracting knowledge or design information from anything man-made and reproducing anything based the extracted information. The process often involves disassembling something (a mechanical device, electronic component, computer program, or biological, chemical, or organic matter) and analyzing its components and workings in detail. Reverse engineering may also be used to create interoperable products. In case of hardware drivers (such as in the case described in this paper), motivation for reverse engineering may be:

- Interoperability;
- Software modernization;
- Money saving.

Reverse engineering of software can be accomplished by various methods. The three main groups of software reverse engineering are:

- Analysis through observation of information exchange, most prevalent in protocol reverse engineering, which involves using bus analyzers and packet sniffers, for example, for accessing a computer bus or computer network connection and revealing the traffic data thereon. Bus or network behaviour can then be analyzed to produce a stand-alone implementation that mimics that behaviour. This is especially useful for reverse engineering device drivers. Sometimes, reverse engineering on embedded systems is greatly assisted by tools deliberately introduced by the manufacturer, such as JTAG ports or other debugging means. In Microsoft Windows, low-level debuggers such as SoftICE are popular.

- Disassembly using a disassembler, meaning the raw machine language of the program is read and understood in its own terms, only with the aid of machine-language mnemonics. This works on any computer program but can take quite some time, especially for someone not used to machine code. The Interactive Disassembler is a particularly popular tool.
- Decompilation using a decompiler, a process that tries, with varying results, to recreate the source code in some high-level language for a program only available in machine code or bytecode.

Wireshark is a Free and open source packet analyzer. It is used for network troubleshooting, analysis, software and communications protocol development, and education. Originally named Ethereal, the project was renamed Wireshark in May 2006 due to trademark issues.

Wireshark is cross-platform, using the Qt widget toolkit in current releases to implement its user interface, and using pcap to capture packets; it runs on GNU/Linux, OS X, BSD, Solaris, some other Unix-like operating systems, and Microsoft Windows. There is also a terminal-based (non-GUI) version called TShark. Wireshark, and the other programs distributed with it such as TShark, are free software, released under the terms of the GNU General Public License.

2. Canon Advanced Printing Technology support with open-source drivers on Linux

2.1. *Problems with genuine proprietary software*

i-SENSYS (also known as LaserShot) LBP3000 is a laser printer that was first released by Canon in late 2005. It features USB2.0 connectivity, and claims 14 pages per minute throughput. Despite such printers age, most of them are still in use due to reasonably cheap 3rd-party supplies and spare parts available.

An entire line of Canon printers, including LBP3000 (and not limited to laser printers), is controlled by Canon's own proprietary protocol called CAPT, which stands for Canon Advanced Printing Technology. The company claims that its use of data compression reduces their printer's memory requirement, compared to conventional laser printers, and claim that it increases the data transfer rate when printing high-resolution graphics [2].

Being still supported on current modern versions of Microsoft Windows operating system, CAPT printers lack proper support for open source operating systems, such as Linux. Canon actually released and periodically updates a Linux driver for their CAPT printers, but it has a number of disadvantages due, but not limited to, its semi-closed-source nature:

- Limited application range and interoperability. While Canon's own CAPT driver works fine locally on a Linux workstation, or even on a CUPS network print server, it will fail to print any printjobs sent to it from Windows workstations.
- Limited host architecture options. Linux was originally developed as a free operating system for x86-based PCs, but it eventually was ported to different processor architectures and is ubiquitously found on various types of hardware. While x86-64 architecture still remains dominant in personal computers, there is now a wide variety of reasonably cheap different ARM-based devices, such as network routers, development boards (with some of them represented and used as a desktop computer) and mini PCs. In contrast, Canon's CAPT driver contains some pre-compiled binary libraries of x86-64 architectures only and thus simply cannot be run on machine of any other architecture than x86 or x86-64.
- Unnecessary complexity. Canon's CAPT driver was designed to support a range of printers sharing the same CAPT protocol, including both printers with and without network printserver built-in, and includes an entire software stack (suite) to manage and control Canon printers. It consists of printserver application (that can be set up to use both network and local pipe for

passing printjobs), a GUI-based printjob monitor (which would be useless on a headless server-like system), and a set of printer-specific backends of its own, all of that itself running as a filter and a backend for the standard Common Unix Printer System. While such setup may be useful in configurations with many network-shared Canon-only brand printers, it leads to unnecessary complex and not user-friendly manual setup when used with a single printer, for example in small or home office. This also leads to an unnecessarily big software package size of around 25MiB (when installed), when the same exact functionality can be implemented in a single small CUPS filter program in well under 1MiB of size, at least for the specific case of locally-connected printer without network interface built-in.

2.2. *Testing available support*

There is a long-running effort of compiling a clean, open source CAPT driver for Linux by Alexey Galakhov, based on prevous reverse engineering efforts done by Nicolas Boichat and Benoit Bolsee, and co-developed by Vitaliy Tomin. At the time of writing this driver seemed to only offer full support for LBP2900 model. It also offered experimental support for LBP3000, LBP3010, LBP3018 and LBP3050.

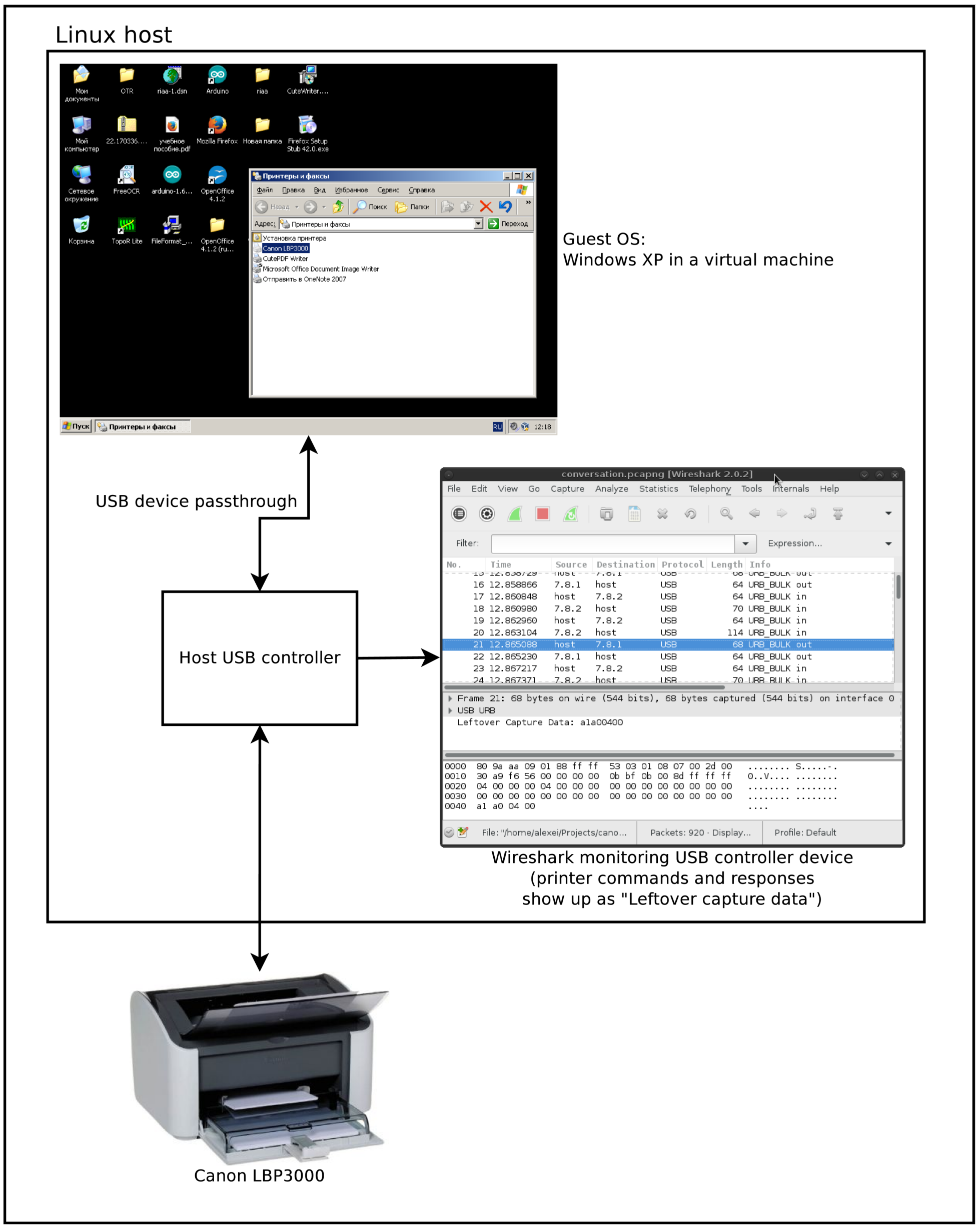
First, experimental support of LBP3000 under consideration was tested. Driver was compiled and installed, and it was observed that driver is stuck in a loop upon processing next job, after successfully printing the first job. It was also observed that driver continues to work normally when printer is power-cycled, and after printing current job, it would stuck in a loop again upon processing the next job. Appropriate excerpts from CUPS log files are presented in Table 1.

Table 1. Excerpt from /var/log/cups/error_log

```
I [26/Mar/2016:19:02:30 +0700] Expiring subscriptions...
d [26/Mar/2016:19:02:30 +0700] select_timeout: JobHistoryUpdate=1459079615
D [26/Mar/2016:19:02:30 +0700] [Job 204] Read 4 bytes of print data...
D [26/Mar/2016:19:02:30 +0700] [Job 204] CUPS_SC_CMD_DRAIN_OUTPUT received from driver...
d [26/Mar/2016:19:02:30 +0700] select_timeout: JobHistoryUpdate=1459079615
D [26/Mar/2016:19:02:30 +0700] [Job 204] Wrote 4 bytes of print data...
d [26/Mar/2016:19:02:30 +0700] select_timeout: JobHistoryUpdate=1459079615
D [26/Mar/2016:19:02:30 +0700] [Job 204] CAPT: waiting for 6 bytes
d [26/Mar/2016:19:02:30 +0700] select_timeout: JobHistoryUpdate=1459079615
D [26/Mar/2016:19:02:30 +0700] [Job 204] Read 6 bytes of back-channel data...
d [26/Mar/2016:19:02:30 +0700] select_timeout: JobHistoryUpdate=1459079615
D [26/Mar/2016:19:02:30 +0700] [Job 204] CAPT: recv  A0 E0 06 00 88 00
d [26/Mar/2016:19:02:30 +0700] select_timeout: JobHistoryUpdate=1459079615
D [26/Mar/2016:19:02:31 +0700] [Job 204] CAPT: send  A0 E0 04 00
I [26/Mar/2016:19:02:31 +0700] Expiring subscriptions...
d [26/Mar/2016:19:02:31 +0700] select_timeout: JobHistoryUpdate=1459079615
D [26/Mar/2016:19:02:31 +0700] [Job 204] Read 4 bytes of print data…
(repeating indefinitely...)
```

2.3. *Protocol analysis with Wireshark*

In order to expand support for LBP3000, a Windows XP virtual machine was set up with original Canon CAPT driver and printer passed through directly to it on a Linux host machine with Wireshark. Then, Wireshark was set to monitor and capture data passed through USB port the printer was connected to, while virtual machine was tasked with printing a test page. This setup is shown in Figure 1.


Linux host
Guest OS:
Windows XP in a virtual machine
USB device passthrough
Host USB controller
conversation.pcapng [Wireshark 2.0.2]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Filter:
Expression...
Wireshark monitoring USB controller device
(printer commands and responses
show up as "Leftover capture data")
Canon LBP3000

```
No.  Time       Source  Destination  Protocol  Length  Info
  16 12.858866  7.8.1   host         USB          64 URB_BULK out
  17 12.860848  host    7.8.2        USB          64 URB_BULK in
  18 12.860980  7.8.2   host         USB          70 URB_BULK in
  19 12.862960  host    7.8.2        USB          64 URB_BULK in
  20 12.863104  7.8.2   host         USB         114 URB_BULK in
  21 12.865088  host    7.8.1        USB          68 URB_BULK out
  22 12.865230  7.8.1   host         USB          64 URB_BULK out
  23 12.867217  host    7.8.2        USB          64 URB_BULK in
  24 12.867371  7.8.2   host         USB          70 URB_BULK in

▸ Frame 21: 68 bytes on wire (544 bits), 68 bytes captured (544 bits) on interface 0
▸ USB URB
  Leftover Capture Data: a1a00400

0000  80 9a aa 09 01 88 ff ff  53 03 01 08 07 00 2d 00   ........ S.....-.
0010  30 a9 f6 56 00 00 00 00  0b bf 0b 00 8d ff ff ff   0..V.... ........
0020  04 00 00 00 04 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0040  a1 a0 04 00                                        ....

File: "/home/alexei/Projects/cano...   Packets: 920 · Display...   Profile: Default
```

Fig. 1. Setup for protocol analysis

It was observed that conversation between printer and its driver happens in USB bulk transfer mode, utilizing USB interface's full bandwith that is not used by other USB transfers now.

According to driver's SPECS file (which contains data extracted by previous reverse engineering attempts), each printer command message generally starts with two-byte command code, succeeded by optional argument words. The byte order used by CAPT protocol is little-endian, that is the least significant byte goes first. Those commands and responses can be observed in the "Leftover capture data" field of each captured packet in Wireshark.

Job start-up command sequences (both successful and unsuccessful) were extracted from both CUPS error log file and Wireshark capture. Then command arguments were stripped away, and command codes were substituted with their mnemonic definitions found in header file capt-command.h for side-by-side comparison, shown in Table 2.

Table 2. Command sequences comparison

CUPS error log		Windows VM + proprietary driver		Notes
Code	Mnemonic	Code	Mnemonic	
Job prologue				
A1A1	CAPT_IDENT	A1A1	CAPT_IDENT	
		A0A8	CAPT_CHKXSTATUS	
E0A0	CAPT_CHKSTATUS	A0A1	CAPT_CHKJOBSTAT	
		A0A8	CAPT_CHKXSTATUS	
A3A2	CAPT_START_0	A3A2	CAPT_START_0	
		A0A1	CAPT_CHKJOBSTAT	
		A0A8	CAPT_CHKXSTATUS	
		A0A1	CAPT_CHKJOBSTAT	
A2A0	CAPT_JOB_BEGIN	A2A0	CAPT_JOB_BEGIN	
		E1A1	CAPT_JOB_SETUP	
E0A0	CAPT_CHKSTATUS	E0A0	CAPT_CHKSTATUS	Following the source code, initial implementation gets stuck at this point.
E1A1	CAPT_JOB_SETUP	A0A8	CAPT_CHKXSTATUS	
E0A0	CAPT_CHKSTATUS	A0A1	CAPT_CHKJOBSTAT	
Page prologue				
		E0A6	Unknown command	Not documented in the open-source driver at all.
E0A0	CAPT_CHKSTATUS	E0A0	CAPT_CHKSTATUS	
A0A8	CAPT_CHKXSTATUS	A0A8	CAPT_CHKXSTATUS	
E0A3	CAPT_START_1	E0A3	CAPT_START_1	
E0A2	CAPT_START_2	E0A2	CAPT_START_2	
E0A4	CAPT_START_3	E0A4	CAPT_START_3	
E0A0	CAPT_CHKSTATUS	E0A0	CAPT_CHKSTATUS	
…				

Most of extra "check status" commands on Windows driver side presumably come from the graphical printer monitor polling current job status.

The printing routines are located in prn_lbp2900.c source file. They are broken up in different printing steps, like job prologue, page prologue, page setup, etc. "lbp2900_ops_s" type structure contains a set of pointers to all such functions for a specific printer model. That way, a new printer model can be added to the driver without breaking support for already implemented printers and can reuse certain functions from other printer model implementation (ether as a 'boilerplate' during development, or just because it naturally uses the same command sequence).

The command sequences of a successfully printed CUPS job and a stuck CUPS job were compared. It was observed, that unsuccessful printing attempt gets stuck at the first status check after CAPT_JOB_SETUP command. Apparently, driver keeps polling the printer status for 'ready' attribute in order to proceed with the print job, and that attribute is not present at the moment. There's also no additional status check commands in-between 'job begin' and 'job setup' commands on the Windows driver side. Following the code in the source file, the offending function call was found, shown in Table 3.

Table 3. Excerpt from prn_lbp2900.c source file [1].

```
static void lbp2900_job_prologue(struct printer_state_s *state)
{
	(void) state;
	capt_sendrecv(CAPT_IDENT, NULL, 0, NULL, 0);
	sleep(1);
	capt_init_status();
	lbp2900_get_status(state->ops);

	capt_sendrecv(CAPT_START_0, NULL, 0, NULL, 0);
	capt_sendrecv(CAPT_JOB_BEGIN, magicbuf_0, ARRAY_SIZE(magicbuf_0), NULL, 0);
	lbp2900_wait_ready(state->ops);
	send_job_start();
	lbp2900_wait_ready(state->ops);
}
```

Removing this function call was sufficient to fix the problem under consideration, but it introduced another problem: print results were scrolled horizontally along the page with each consecutive page being printed, which is not acceptable.

While studying comparison of Windows and CUPS sequences further, an unknown (never documented before in capt-command.h file) command with the code E0A6 was discovered in Windows command sequence. It has a single 16-bit argument that always equals 0 and returns a single 16-bit word with the value of 0.

The command described above was placed into the send_job_start() function, which is called once by lbp2900_job_prologue() function (shown in table 4). Another test job was sent to a printer, and this time, the result finally came out absolutely normal, and the experiment goal was reached.

Table 4. Excerpt from prn_lbp2900.c source file [1].

```
static void send_job_start()
{
	uint16_t page = 1; /* nobody cares */
	uint8_t nl = 16;
	uint8_t fg = 0x01;
	uint16_t job = 1;  /* nobody cares */
	time_t rawtime = time(NULL);
	const struct tm *tm = localtime(&rawtime);
	uint8_t buf[32 + 64 + nl];
	uint8_t head[32] = {
		0x00, 0x00, 0x00, 0x00, LO(page), HI(page), 0x00, 0x00,
		0x10, 0x00, 0x0C, 0x00, nl, 0x00, 0x00, 0x00,
		fg, 0x01, LO(job), HI(job), /*-60*/ 0xC4, 0xFF, /*-120*/ 0x88, 0xFF,
		LO(tm->tm_year), HI(tm->tm_year), (uint8_t) tm->tm_mon, (uint8_t) tm->tm_mday,
		(uint8_t) tm->tm_hour, (uint8_t) tm->tm_min, (uint8_t) tm->tm_sec,
		0x01,
	};
	memcpy(buf, head, sizeof(head));
	memset(buf + 32, 0, 64 + nl);
	capt_sendrecv(CAPT_JOB_SETUP, buf, sizeof(buf), NULL, 0);

	uint8_t dummy[2] = {0,0};
```

```
        capt_sendrecv(0xE0A6, dummy, sizeof(dummy), NULL, 0);

}
```

3. Results

The support for LBP3000 printer was implemented in open-source CAPT driver, which can be compiled and used on any modern Linux (and possibly different BSD) distribution, on any machine architecture supported by those distributions. The compiled x86-64 binary CUPS filter file is extremely lightweight compared to genuine CAPT driver's 25MiB software package, only having a size of approximately 100KiB.

The resulting driver also accepts and naturally processes PostScript print jobs from different operating systems, such as Windows, which allows to share such printer on a network from a (typically) Linux-based print server or embedded appliance that has USB host interface and supports Common Unix Printing System (such as USB-enabled Linux router, or a small ARM development board, like Raspberry Pi, for example). It also enables to use IPP (Internet Printing Protocol) to share printer over the network, which allows easy setup on client machines and completely abstracts away client machines from specific printer drivers.

References

1. Galakhov, A. (2014). *Driver for Canon CAPT printers.* [Available at: https://github.com/agalakhov/captdriver] [Accessed 08/04/2016].
2. Wikimedia Foundation, Inc. (2016). *Canon Advanced Printint Technology.* [Available at: https://en.wikipedia.org/wiki/Canon_Advanced_Printing_Technology] [Accessed 08/04/2016].