Preventing Control Character Based Attacks On Parallel Line Printer Log Devices

Marc Roessler
marc@tentacle.franken.de

August 13, 2001

Abstract

Administrators concerned about computer security sometimes use line printers for logging purposes. The basic idea is that a logfile hardcopy is impossible to modify or destroy remotely once printed. However there is a major weakness: the behavior of many printers can be influenced by control characters, enabling an attacker to render parts of the printed evidence unreadable or at least make log data analysis much more difficult. This paper shows some attacks as well as how to prevent them.

1 Introduction

In the computer security branch there has always been a need for WORM[1] and WORM-like devices. The idea is to allow writing data while preventing modification or erasal of previously written data even for fully privileged users.

This is often done by logging system information using dedicated machines (e.g. a dedicated syslog server), line printers or other means of storage not easily influenced by an attacker lacking physical access to the hardware. Line printers have the advantage of being cheap and having low power consumption when on standby. Furthermore they are relatively "dumb", so attacking a line printer is less likely to succeed than attacking a complex logging mechanism prone to have numerous software flaws. There are disadvantages as well, such as low flexibility, possible DoS[2] scenarios (printer out of paper) and high consumption of raw material. A few printers are known for low reliability in regard to paper jams or similar hardware faults. Obviously those are not the ones to be used for such an application.

Some system administrators connect a standard line printer to the parallel interface of the system and configure their syslog deamon to use the device file as one of the

[1] Write Once Read Many
[2] Denial of Service

logfiles. This is simple, convenient and usually works well. There is no problem with line printers containing only the logic needed to perform the basic print functions, but many printers (even older ones) offer numerous features. Some of these features may enable an attacker to influence logging or even modify previously written log data.

The roots of the problem are escape codes and other special characters which the printer interprets instead of printing them (or their hex value). In most cases it is possible to affect the behavior of the printer severly. For instance the implications of being able to start downloading new character fonts to the printer are quite obvious: the system administrator will not be very pleased with a font consisting of white spaces only.

Recent syslog versions will rewrite the control codes so they are printed instead of being interpreted. This will prevent attackers from causing undesired printer behavior by injecting data such as by supplying faked DNS entries. However once the attacker succeeds in writing directly to the printer device file (for example by gaining root access) he can affect the printer nevertheless, possibly rendering previously printed log entries unreadable. This is rarely mentioned when suggesting the use of line printers as logging devices [1, 2, 3].

Similar effects of control codes on standard Unix tools often used for viewing logfiles (cat, tail etc.) have already been reported [4].

2 Example attacks on a Star LC-10 printer

A Star LC-10 printer was to be used as a log printer. A quick glance into the printer manual revealed several undesired features.

Slowing down printing

Certain escape codes can cause the printing speed to decrease by magnitudes, possibly causing the printer buffer to overflow, which means loss of potentially important log data. For measuring effective printing speed one line (around 80 characters) of base64 code was used. Printing that line in draft quality (standard) took about one second to complete. In contrast, after the following characters were sent to the printer the same string took more than 45 seconds to print.

String	Effect
`0x1B 0x78 0x31`	near letter quality
`0x1B 0x2D 0x31`	underlined
`0x1B 0x5F 0x31`	overlined
`0x1B 0x68 0x02`	print characters four times as large
`0x1B 0x55 0x31`	unidirectional print

Modifying previously written data

By sending special characters it is possible to modify and overwrite previously printed data.

The Backspace character `(0x08)` moves the print head backwards by one column and sending reverse line feeds `(0x1b 0x0a)` causes the printer to feed the paper backwards. Those characters can be repeated as often as desired until either the print head hits the left margin or the beginning of the paper is reached.

By carefully crafting a string consisting of regular characters and escape codes it is possible to overwrite ('censor') previously printed characters, rendering them unreadable. The lines may still be readable using more sophisticated equipment, however this is usually beyond the capabilities of average system administrators.

The following program fragment was used for demonstration

```
printf("Connection attempt from 127.0.0.1 (%s)\n", str);
```

The attacker is assumed to be able to inject arbitrary data into the variable `str` somehow, e.g. by faking DNS responses. In effect the variable string may contain something like:

```
char str[]=
"\x08\x08\x08\x08\x08\x08\x08\x08\x08\x08\x08@@@@@@@@@"
"\x08\x08\x08\x08\x08\x08\x08\x08\x08#########"
"\x08\x08\x08\x08\x08\x08\x08\x08\x08XXXXXXXXX  ";
```

Fig. 1 shows what the hardcopy looks like.

Syslog guards against such attacks, but of course the string also may be written manually to the device after the log entry has been printed. There is no need for embedding the control characters inside a logged string, but it is an interesting possibility nevertheless. Anyone writing logging services should be aware of such problems.

Figure 1: Overstriking previously printed characters

Another interesting attack is deleting the last line from the printer buffer by sending the character `0x18`. When inlcuded as last character of the test string `str` shown above, only the closing bracket of the log entry is printed. It gets even worse: it enables the attacker to change the present log entry to some arbitrary string as soon as he is able to inject arbitrary characters somehow. For instance snooping a TTY and sending the data to a line printer unfiltered is not a good idea.

To give an example, if the string

```
char str[]=
"\x18 Connection attempt from 192.168.0.1 (scapegoat";
```

is injected by the attacker, the log entry generated by the example code snippet presented above will read

```
Connection attempt from 192.168.0.1 (scapegoat)
```

regardless of the fact that other data was sent to the printer before. This previously sent data is lost without a trace.

Disabling the printer

After destroying previously logged evidence the attacker is likely to try to disable the printer altogether. Otherwise a logging daemon or mechanism not discovered by the attacker may continue to generate log entries.

The attacker could try to set the printer to graphics mode by sending e.g. `0x1b 0x4c 0xc0 0x03`. As an effect the next 960 characters arriving will be interpreted as an 120dpi image, 960 dots wide. Each byte represents 8 vertical dots. Of course deciphering this is still possible although it is very painful.

The attacker probably would prefer to completely disable the printer. He could do this by sending the character `0x13`, which will set the printer “off-line”.

Of course there are still other ways to DoS the printer, such as repeatedly feeding the paper by 127 lines (`0x1b 0x66 0x31 0x7f`) or sending a large number of form feeds (`0x0c`), both of which will cause the printer to run out of paper sooner or later. Besides it is strongly advised to connect an alert to the error and paper exception lines of the printer, otherwise problems may go unnoticed.

3 How to prevent the described attacks

The best and most simple solution is to use a “dumb” line printer which does not interpret special characters. Such printers may be hard to find or “intelligent” printers may already be available.

Those “intelligent” printers may be converted to “dumb” printers by modifying the software contained within their EPROMs. However not all printers do have EPROMs which can be replaced easily. Those which do are difficult to program since the expected format of the data contained within the EPROM is usually unknown.

If an “intelligent” printer has to be used the character stream passing from the computer’s parallel interface to the printer must be filtered to contain only “sane” characters. All other characters must be changed to something harmless, such as spaces. Since the computer can not be trusted any more once it has been compromised, obviously a software based filter is not an option. External hardware safe from manipulation by a remote attacker has to be used.

For parallel line printers the suggested hardware solution centers around a TTL EPROM, such as the 27128 [3]. The data lines `d0` to `d7` from the computer are connected to the lower address pins of the EPROM `a0` to `a7`. The remaining address pins `a8` to `a13` are set to ground. The data out pins of the EPROM are connected to the printer using 74LS244 TTL buffers. All non data pins are passed through unmodified. The EPROM is permanently set to “chip enable” and “output enable”, causing it to react to changing patterns of the data lines right away.

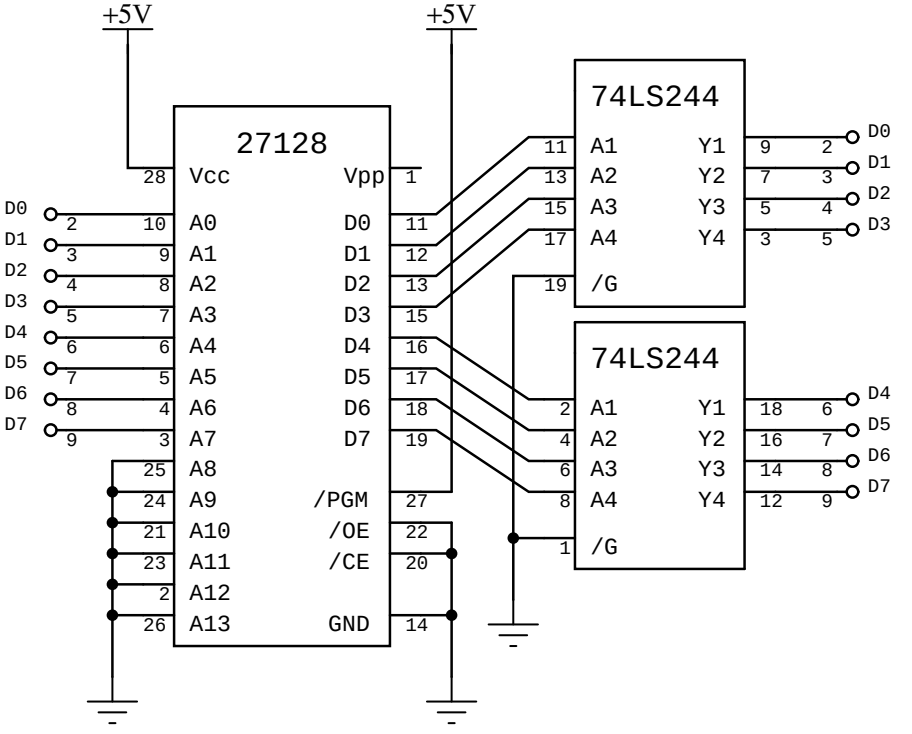

Figure 2: Schematic of the EPROM based character filter. All lines of the parallel interface not shown are passed through unmodified. Ground of this circuit must be connected to the parallel line grounds. An 78L05 power regulator should be used for providing the supply voltage of 5V.

Suppose character `A (hex 0x41)` is sent by the computer. This causes the EPROM to set its data lines according to the data located at memory address 0x41. In case data at that address is 0x41, so the character 0x41 will be sent to the printer. Therefore the

[3] Of course smaller EPROMs may be used as well since only 256 bytes are needed

character will seem to have passed through the filtering device unmodified. If some different character was stored at this memory location when programming the EPROM, that character would be sent instead.

As with any security related filter it is strongly advised to use a positive list instead of a negative one. This means that a list of acceptable characters should be built 'from scratch' rather than crossing critical characters off the list of all possible characters. In the Appendix a short Perl script can be found. It generates the raw data to be used for programming the EPROM from a passlist containing all acceptable characters. All other characters will be sent to the printer as spaces.

All lines but the data lines are passed right through to the printer. According to the specification [6] the data lines must remain stable for at least 400 ns (Star LC 10 Manual says 500 ns) before sending the strobe pulse. It may take the EPROM up to 250 ns to set all data out lines correctly, thus only about 150 ns delay between data valid and the strobe pulse can be taken for granted. This may cause this circuit not to work with some printers, however no such problems were found with the tested system.

It is advisable to build the circuit into a conductive casing. This is necessary for minimizing electromagnetic emanations which may enable local attackers without physical access to the printer to reconstruct the logged data. Another important reason for electrical shielding is the prevention of radio interference with other devices.

Weaknesses of the presented filtering circuit

A potentially serious weakness arises due to the simple design of the circuit. The propagation delay has consequences for the reliability of the character filter device. While usually only allowed characters will be passed, it can *not* be guaranteed that an attacker with full system access will not be able to sneak evil characters past the filter.

This is due to the fact that the attacker has full control over the parallel port and thus may set the lines manually, violating the protocol. For example he might choose to send the strobe pulse too early after changing the data presented at lines `d0` to `d7`. It will take the updated logic levels some time to propagate through the EPROM, so the printer might not accept the correct and stable data but do a snapshot of the still changing data lines. Of course one of those characters interpreted while data lines are still invalid may be one of the characters to be filtered.

Obviously for this attack to work precise timing is necessary, along with information on the type of printer, CPU type and CPU clock, type of EPROM, length of the printer cable and system load. Considering the fact that e.g. on a 486DX-50 one clock cycle takes about 200 ns [7] this attack seems quite unlikely to succeed.

Some readers may consider this not to be secure enough. Closing the "propagation delay hole" is not trivial but possible nevertheless. The following steps need to be taken:

- Data needs to be buffered using an 8 bit D-type flip flop such as the 74HC574, which accepts data at the falling edge of the clock (strobe) signal [4]. In addition it has to be made sure that the flip flop can only be triggered while the simulated busy line (see below) is set low, i.e. after the printer acknowledged reception of data. Otherwise an attacker may (in quick succession) set the data lines differently and send a second strobe pulse, thus rendering the D flip flop transparent while the printer still reads the data. This would again enable him to exploit the mentioned weakness.

- The falling edge of the strobe signal is passed on to the printer delayed by about 1000ns (the propagation delay of EPROM, D flip flop and buffers plus some extra security margin). A 74LS123 chip may be used for this task.

- The filter circuit needs to provide a "faked" busy signal to the computer (simulated busy line) as soon as it receives the falling edge of the strobe signal. The busy line towards the computer is set low again as the busy line from the printer drops low. This "proxying" is necessary since according to the protocol the printer has to rise the busy line while strobe is still low, at most 200 ns after the falling edge. As the printer receives the strobe signal delayed by about 1000 ns he can not reply in time, causing the protocol to fail. The described function can be provided by means of a flip flop.

Obviously this improved circuit is much more complex than the one presented before. The latter was neither built nor tested, it is intended as a pointer for readers wanting improve the presented circuit.

4 Summary

While the filter device makes the task of an attacker much harder, security can not be guaranteed by the simple version of the filter.

If a "dumb" line printer is available, it should be preferred over the presented filter circuits. Each additional component raises the risks of falling victim to malicious attacks and random failures.

It should be realized, too, that the presented character filter does not prevent an attacker from DoSing the printer by consuming all its paper, e.g. by sending numerous newlines. This DoS usually can not be prevented as the attacker might as well shut down the system altogether or disable the printer by modifying the running kernel.

Generally it can be said that a single logging mechanism should be not relied upon. The proper way is to use multiple printers and remote logging hosts using regular log files, everything secured appropriately.

[4] Note that the D flip flop has to be placed *before* the EPROM! (added September 18, 2001)

A EPROM data generator

Redirect the output of this small Perl script to a file and use this file for programming the EPROM with the raw data. The passlist contains all allowed charcters as decimal numbers. Some readers may want to exclude the bell character (7) since it may be used to slow down printing.

```
#!/usr/bin/perl
# list of acceptable characters:
@passlist = (7, 9, 10, 13, 32, 33, 34, 35, 36, 37, 38,
        39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
        51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62,
        63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74,
        75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86,
        87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98,
        99, 100, 101, 102, 103, 104, 105, 106, 107, 108,
        109, 110, 111, 112, 113, 114, 115, 116, 117,
        118, 119, 120, 121, 122, 123, 124, 125, 126);
$list = ' 'x256;
foreach $passchar (@passlist) {
        substr($list,$passchar,1) = pack("C", $passchar);
}
print $list;
```

References

[1] GERHARD MOURANI. *Securing and Optimizing Linux (RedHat Edition).* June 07, 2000. Downloadable from serveral sites.

[2] SIMSON GARFINKEL. GENE SPAFFORD. *Practical Unix & Internet Security.* O'Reilly 1996.

[3] B. FRASER. *Site Security Handbook*, RFC2196. September 1997.

[4] Securax-SA-12 Security Advisory: *Remote hiding from access_log and error_log.* December 28, 2000. `http://securax.org/pers/scx-sa-12.txt`

[5] *Users Manual for Star LC-10 line printer*

[6] *IBM LaserPrinter 4029 Series Technical Reference*

[7] *Linux I/O port programming mini-HOWTO (v3.0)*, maintained by Riku Saikkonen. `http://www.linuxdoc.org/HOWTO/mini/IO-Port-Programming.html`

[8] ZHAHAI STEWART. *Interfacing the IBM PC Parallel Printer Port (Document Version 0.96)* `ftp://ftp.rmii.com/pub2/hisys/parport`