

# Release : ComLog 1.0, a WIN32 command prompt logger

**Source:** <http://www.derkeiler.com/Mailing-Lists/securityfocus/focus-ids/2002-08/0070.html>

---

**From:** Floydman ([floydian\\_99@yahoo.com](mailto:floydian_99@yahoo.com))

**Date:** 08/14/02

Date: Wed, 14 Aug 2002 12:44:49 -0400

To: [bugtraq@securityfocus.com](mailto:bugtraq@securityfocus.com), [focus-ids@securityfocus.com](mailto:focus-ids@securityfocus.com), [focus-ms@securityfocus.com](mailto:focus-ms@securityfocus.com), [forensics@securityfocus.com](mailto:forensics@securityfocus.com)

From: Floydman <[floydian\\_99@yahoo.com](mailto:floydian_99@yahoo.com)>

ComLog.pl, a WIN32 command prompt logger

by Floydman, [floydian\\_99@yahoo.com](mailto:floydian_99@yahoo.com)

August 13th 2002

This paper is available online at [www.geocities.com/floydian\\_99](http://www.geocities.com/floydian_99) and <http://securit.iquebec.com>

This paper can be freely distributed and reproduced, as long as correct credentials are maintained, and that no modifications are made to this file. For corrections, suggestions or comments, please send me an e-mail.

## Abstract

The goal of this paper is to present a new Perl tool that I made to monitor DOS sessions on Windows NT/2K (should also work on XP, but cannot try it). This tool can be used by administrator to keep a history of commands typed in the DOS command prompt and the associated output, for example on an IIS server. This can help admins to figure out what the attacker has done after compromising the machine via one of the numerous vulnerabilities this software present. Of course, like any keylogger out there, it can also be use for malicious purpose, but I think other programs out there make better spying tools.

## Preface

I started writing about my own experiences in the security field after reading Lance Spitzner's whitepapers, way back before the HoneyNet Project grew into the multi-disciplinary team that has now joined him and make the project as we know it today. I also wanted to inspire myself of what was being done on the UNIX platform (making your own tools when they don't exists) to transpose it in the Windows world, where I am more comfortable. This paper follows these traditions, as the idea for it came to me while reading the HoneyNet Project recent book, Know your ennemy. In the book, one passage relate to how the HoneyNet is designed to sniff all the traffic that passes into it since all of this traffic is suspicious by

nature. This worked great for capturing what the attackers were doing on the systems, up to the day when someone was using cryptcat to hide his session. To circumvent this problem, the team made a modification in the source code of bash to log the session and recompiled it for the next attack. However, the book says, since we can not do the same with Microsoft code, they didn't bother to make an equivalent solution for Windows. ComLog fills that gap.

Special thanks to Lance Spitzner for his comments and ideas during the improvement of the pre-release.

### Targeted audience

This document is presented to anyone who has interests in computer security, honeynets, trojan horse, keyloggers, network administration and computing in general.

1. Introduction
  2. Purpose of the program
  3. How it works
  4. Things you need to know
  5. To Install
  6. Conclusion
- Appendix A. Source code  
Appendix B. Sample session history (console)  
Appendix C. Sample session history (history.txt)

### 1. Introduction

One of the most common ways that Windows machines are hacked is through the command prompt. This is not surprising, since this is the shell environment on the Win32 platform, it permits the execution of commands without the need of a GUI, and is easily accessed by executing one single file, either `command.com` on Win 9x/Me, or `cmd.exe` on Win NT/2K. While this file in itself is not a problem, the thing is that it can be accessed by many ways by anyone who can guess where is located this file, and most of the time, it is always in the same places, because the installations were made by default. This is how most of IIS exploits work, they rely on a Unicode encoding or a special format string to fool the server into accessing files outside of the web directory and make a renamed copy of `cmd.exe` (because IIS filters commands sent directly to `cmd.exe`, thanks Microsoft for the great security), and then simply issues command to this copy via a URL in their web browser. Code Red and Nimda were doing that automatically.

Also, admins may want to keep an eye on what's going on in the DOS prompts on their users workstations, as the inside threat remains the greatest concern for computer security. I mean, what could secretaries, accountants and other office employees be doing in a DOS prompt that is related to their job? I wouldn't go as far as to remove their access to the command prompt, besides it may be needed from time to time by support people. But

right now, how can you tell if one of the employees on your network is not doing a ping scan or is netcatting his way to the restricted file server and making password dumps?

## 2. Purpose of the program

The purpose of this program is quite simple: improve the forensics disponibility in case of a break-in in order to better determine the actions of the attacker. Forensics is a whole ball game in itself in security. It requires a good understanding of the systems involved, and experience of attacks usually made helps in focusing on the important things. Forensics is all about figuring out what has happened on a computer system (or network) after an intrusion has been done. To do this, all the information available is crucial: firewall logs, IDS logs (if any), system logs, volatile information on the systems (what processes are running in memory, ...), complete bit-by-bit disk image. All of this in order to try to understand what has happened, but on top of it, one must make sure not to alter this data by manipulating it, considering that the attacker may have gone through great efforts to hide his traces.

ComLog will not solve all the problems computer forensics face today. But it will help greatly in reducing the time necessary to understand the scope of an attack if the intruder made his attack via the command prompt. Instead of trying to reverse-engineer the attackers tool or to look for modified signature files to determine the actions of the attacker, one will only have to check these logs to determine what commands were typed by the attacker. By storing centrally the history files of the command prompts on your network, along with your antivirus log files and maybe even personal firewall log files (for more details on this, read "Securing the Microsoft internal network"), you improve your knowledge about the kind of activity that is going on on your machines. And it gives you the chance to act wisely before it is too late.

## 3. How it works

ComLog is a Perl program that sits in front of the command prompt and emulates it, capturing any data typed in and out before transmitting it to the real prompt for execution or before displaying the data on the screen. It could be considered as a wrapper. To do so, the program must be compiled with perl2exe (available from [www.indigostar.com](http://www.indigostar.com)) and renamed cmd.exe (this is why it doesn't work on Win9x/Me, these OS use command.com, which allow for smaller files, and I haven't figured how to make a .com file. Besides, it is used for boot-up. If you can help on this one, send me an e-mail). The real cmd.exe has to be renamed cm\_.exe (or whatever name you chose, but make sure to change the source code appropriately), because we still need it to execute the commands asked by the user.

So this way, when a user double-click on the command prompt icon, or run cmd.exe from the Start menu, or a cracker issues commands to the prompt by abusing the web server, web browser, database server, mail client, netcat session, etc., or to a renamed or relocated copy of it, it will be thru

ComLog that these commands will be issued. ComLog makes a timestamp in the log file (\*.clg) between each input and output, allowing the administrator to know things like at what time each event occurred, how much time each command took to perform, how much time passes between the result of the last command and the when the next command is issued, etc.

Since ComLog emulates the command prompt, there are certain things we need to keep track of in order to make the session as transparent as possible to the user. Such things to keep track of is the current directory and current drive, and to keep track of this we monitor closely 'cd' command and changes of drive. 'cls' is also a command that I cannot rely on cm\_.exe to implement, since it returns an empty screen as output, so we implement that as well. We also have to check if the drive or directory the user wishes to go to really exists, or else ComLog will contain erroneous environment data and will stop working properly. This causes a limitation in the use of the wildcard '\*' in directory names (more on that in the next chapter).

That is about all the checking and emulating that is done, all the rest that is typed by the user and/or attacker will be sent as is to the real prompt (cm\_.exe), and bad commands will simply produce the expected "The name specified is not recognized as an internal or external command, operable program or batch file." error message, as it normally would.

ComLog always start in the . (current) directory. Issuing cd commands with relative paths will create a relative path chain that will keep track of the current directory. So after browsing a couple of directories out of and back in to C:\WINNT\SYSTEM32(cd..; cd..; cd winnt; cd system32), the path should look like .\..\..\WINNT\SYSTEM32, although the user will only perceive his current directory. Issuing a cd command with a path starting with \ will replace the whole relative path that built over the session with that absolute path, in order to avoid getting too long and confusing relative path names in memory. Further relative path browsing will be appended to this absolute path, until a new absolute path is entered. Drives and current directories for each drive are kept in a self-adjusting table. Bad directories and bad drives error messages are handled by ComLog, and will replace the user's command with something neutral so not to disturb environment variables. Once these checks are done, commands are sent in the form of cm\_.exe /C currentdrive: && cd currentdir && usercommand, the && signs are letting us to issue more than one command on the same line. So this way, the new instance of cm\_.exe (which has its own environment variables) is always kept fresh on where the user wants to be. cm\_.exe terminates after executing the command. This is why we have to keep track of the environment variables ourselves. The output is captured, sanitized to hide ComLog's presence, and then sent to its log file and displayed on the screen.

In the first, pre-release version of ComLog, log files were stored in a file called history.txt, that was present in the . (current directory) along with cmd.exe. I thought that this was good, because ComLog would produce a log file wherever it is copied to. But this produced two

problems that are now solved with a single solution. First, if two (or more) instances of the same cmd.exe were running, they were all writing to the same log files. So issuing an intensive command like dir /s on one instance meant that one command typed in another instance had to wait until the intensive dir finishes before producing its output. The other thing was that having a potentially floating location for the log files, it was hard to keep an eye on it using a tool like LogAgent. This was solved by storing the log files in \WINNT\Help\Tutor, each instance having a random-generated filename associated to it, to prevent collision. ComLog log files have a .clg extension. Take note that if you install ComLog manually (not using the install pack), you have to create the \Tutor folder in \WINNT\Help.

#### 4. Things you need to know

ComLog does a pretty good job at pretending to be the real command prompt, but in fact it has some limitations that can affect or not its actor performance, depending on the situations. Some limitations are gone since the pre-release version, but some still remain. First of all, since I mentioned it earlier, there is a limitation in the use of the \* wildcard in directory names. In normal circumstances, under NT/2K, you can cd to a directory by supplying only the first few letters followed by a \*, which can be convenient with long path names. The limitation occurred when I implemented the checkdir() function (which I cannot remove, or else ComLog may be fooled into non-existing directories, and at that point you can kiss your cover goodbye). I tried doing it with the opendir() function, like I did with checkdrive(), but it doesn't support wildcards whatsoever (from my testing, anyway). So I use the same trick as when I send commands, and I use this to make a IF EXIST on the directory (this is usually used only in batch files, since there is little use to make an IF statement on a one-liner, but it works just as well when it is typed in DOS) to validate its existence. And the IF EXIST command cannot handle more than one layer of wildcarded directories. For example, in a ComLog session, let's pretend that I am in the root directory, and I want to move to \Program Files\Common Files. If I type cd pro\* [ENTER], cd com\* [ENTER], this will work just great. But if I type cd pro\*\com\*, the program will issue a "The system cannot find the path specified." error message. But under a real NT command prompt, that line should work.

Another problem is that it cannot handle interactive DOS programs, such as a regular FTP session or something as simple as date or time (you have to at least press enter to exit date and time). Since the command prompt output is captured by ComLog, and that ComLog waits for cm\_.exe to terminate to continue its execution, and that cm\_.exe is waiting for some sort of user input that will not come before quitting, we can see where the problem lies. Under certain circumstances, this is not so much of a problem, for example a web server that the command prompt can be abused with netcat, since netcat won't allow for interactive programs either, so the attacker behavior will already be adjusted to that limitation (or else he will face the same results). But on a local prompt, where a user might want to launch an interactive DOS program (be it a network scanner waiting

for its parameters or a mere DOS game), it will be obvious that something is not working right. Launching such a program from the GUI should not pose a problem, but if you must absolutely start a program from a DOS prompt, remember that you can always do so from `cm_.exe`.

One other thing is that ComLog will try to hide himself from the view of the user, by modifying the output when `cm_` or ComLog log files shows up. This will be the case if the user issues a `dir` or a `pslist` command to see what files or running programs are present on the system. This have the side effect of having a different file count at the end of the `dir` commands and the number of files displayed on the screen (or the user's dumpfile, as he is likely to do). While this has little chance of being detected on directories containing a lot of files, in an empty directory, this might ring a bell to an attacker. Another thing that could give it is that by hiding the bigger file size for `cmd.exe` (695 kb once compiled with `perl2exe`, compared to 204 kb on NT and 231 kb on 2K), I drop any files that is the same exact size as ComLog. So, in a directory where you have `cm_.exe` (204 kb), `cmd.exe`(695 kb), and you copy `cmd.exe` to `root.exe` and produce a `dir`, you will see `cmd.exe`(204 kb), but no trace of `cm_.exe` (that's good) or `root.exe` (not so good).

Oh!, I almost forgot. If you use ComLog in its Perl format, or if you compile it and is it called `comlog.exe` or anything else except `cmd.exe`, it will work as advertised. But when you rename `cmd.exe` to `cm_.exe`, and rename `ComLog.exe` to `cmd.exe`, ComLog will go on an infinite loop when you try to execute `cmd.exe`. This is because the compiled program still need a shell environment to start into, and that shell is by default... `cmd.exe`. So the program just keep launching itself in the useless hope of ever finding a shell environment. It took me quite a while to figure this one out, tried almost everything from changing my `COMSPEC` variable, editing the registry, patching some Windows executables and DLLs (the later caused my system not to be able to boot anymore. Luckily I had a NT Recovery CD with NTFS4DOS on it, which let me put the original DLLs in place). It turns out that all you need to do is to patch 2 ActivePerl DLL with a hex editor to do the trick. This is where is stored the hardcoded call to `cmd.exe` that kept haunting me, no matter what change I was applying to my system. The files that need to be changed are `p2x560.dll` and `perl56.dll`. Make a backup of these files and open them with a hexadecimal editor. Make a string search for `cmd.exe` (there's only one instance in each file), and replace the 'd' character with '\_'. Of course, you could chose another name, but you have to realize that you are limited to a three characters filename, and that you must change the name evrywhere in the program so that it matches accordingly. Windows 200 involve one more step (not covered with the ComLog shareware install pack, to come later), thanks to Windows File Protection, a feature added in Win2K that lets the system monitor its system files and replace them with the original version if they become corrupted or deleted. This feature, which was inexistant on Windows before, will effectively keep replacing ComLog with the real `cmd.exe`. This feature can be disabled with a simple registry tweak on Win2K, a DLL have to be patched in addition of the tweak on Win2K SP2. Full details on how to disable Windows File Protection can be found at

<http://www.jsifaq.com/SUBK/tip5300/rh5392.htm> and [http://www.lanovation.com/support/docs/NT\\_2000/WFP\\_2000XP.htm](http://www.lanovation.com/support/docs/NT_2000/WFP_2000XP.htm) or by typing 'disabling Windows File Protection' on Google.

The last thing is the giveaway that is the banner displayed at the end of execution when compiled with perl2exe evaluation version. This is why you may want to download the install pack, it is compiled with a registered version of perl2exe.

## 5. To Install

For Win NT4:

1) Download the installation package (free) at <http://securit.iquebec.com/download.html>;

or 2) Installing it manually from the source code displayed below.

In the case of 1), simply download the file and execute it on the machine you want to install it on. This file will copy 2 files in your \Winnt\System32 directory (cmd.exe, cm\_.exe), and will create the directory \Tutor in \Winnt\Help\ . ComLog Log files are stored in \Winnt\Help\Tutor.

In the case of 2), first of all you need a copy of Perl installed, I suggest ActivePerl from [www.activestate.com](http://www.activestate.com) (it's free) if you don't already have one. Then you need a hex editor, I suggest XVI32 (also free) that you can download at <http://download.com.com/3000-2352-7621132.html?tag=lst-0-15>. Then, using the hex editor, you have to edit p2x560.dll and perl56.dll in your \Perl\bin directory, and change the instance of cmd.exe to cm\_.exe. Copy cmd.exe to cm\_.exe in \Winnt\System32. Copy the code presented below and save it to comlog.pl. Then, using perl2exe, compile comlog.pl, and rename the resulting comlog.exe to cmd.exe. Then copy this cmd.exe over your original cmd.exe (keep your cm\_.exe, comlog needs it). Finally, create a directory called \Tutor in \Winnt\Help, for storing the log files. To use, simply go to a Dos prompt via your preferred way.

For Win2K : The same, except that you first have to disable the Windows File Protection feature. Look at <http://www.jsifaq.com/SUBK/tip5300/rh5392.htm> and [http://www.lanovation.com/support/docs/NT\\_2000/WFP\\_2000XP.htm](http://www.lanovation.com/support/docs/NT_2000/WFP_2000XP.htm) or by typing 'disabling Windows File Protection' on Google for more details about this.

## 6. Conclusion

So this concludes the documentation accompanying this software. As we have seen above, ComLog is a tool that can empowers network administrators into knowing what is going on with the command prompts on their machines, and can use it to determine the actions of an attacker. Since only what is going through the command prompt is logged, I think this is not too intrusive to be placed eventually on every PC on a network, since regular

office employees rarely have a professional reason to justify the use of the command prompt. ComLog being an emulator, it does show some shortcomings when compared with the real McCoy, but it should not prevent its use in most cases. However, fine knowledge of these shortcomings could help an intruder to determine that he is being watched, and more effort should be put into improving the program even more. A lot has been done recently to improve this, such as improving LogAgent to handle these logs and to make sure that ComLog has a LogAgent-compatible log file storing strategy, but more could be done to improve concealing. Finally, the source code is presented in Appendix A, and a sample session is showed as it appears on the console (Appendix B) and in the history.txt log file (Appendix C). The code should be easy to read, as I used comprehensive variable and procedure names, and I made a lot of comments along the code.

Appendix A. Source code

```
#!D:\dev\perl\bin\perl.exe
# ComLog 1.0
```

```
#####
# ComLog 1.0 #
# by Floydman floydian\_99@yahoo.com #
# Copyright 2002 SecurIT Informatique Inc. http://securit.iquebec.com #
# #
# This program captures the input/output of the Windows NT Command Promt
(cmd.exe). #
# It does so by pretending to be the real prompt and forwarding the
commands to the #
# real (and renamed) cmd.exe. I/O is stored in a random-generated log file
in #
# \WINNT\Help\Tutor.
#####

#####
# LICENSE #
# This software is Open Source. This means that its source code is open,
free and avai-#
# lable for anyone to look into, make modifications, correct bugs (let me
know, please) #
# and use for personal and commercial use. You can create your own
binaries with #
# perl2exe (www.indigostar.com), or download the install package from #
# securit.iquebec.com/download.html. #
#####

#####
# Main Program #
# This is the main structure of ComLog. #
# This programs is a loop that will be passed only once if arguments are
passed with #
# ComLog, since it means it was not launched for interactive use. Else,
the loop will #
```

SecurityFocus IDS: Release : ComLog 1.0, a WIN32 command prompt logger

```
# go on until the user types 'exit' or breaks otherwise the program
execution (CTRL-C).#
# The program then presents the prompt to the user, and gets the command
typed at the #
# keyboard. Change of drives or directories are checked for, since ComLog
have to #
# provide these parameters to the real command prompt. Everything is
logged in a #
# random generated filename to allow for multiple instances. Output is
sanitized to #
# hide ComLog's presence. #
#####

use Win32;

# Initialization of entrant parameters, current dir, command, exit and
currentdrive
$input = join (' ',@ARGV);
$index = 0;
$currentdir[$index] = '.\';
$command = "";
$exit = 1;
$currentdrive[$index] = getcurrentdrive ($currentdir[$index]);
$nothing = 1; # nop
$winnt = "Windows NT Version 4.0";
$wintwok = "Microsoft Windows 2000";
$cleanlog = 0; # Set to 0 to keep the local log files, set to 1 to delete
them after the session is done

$history = randomfilename();

open (WINVER, "cm_.exe /C ver |") or die "Can't open pipe";
@header = <WINVER>;
close (WINVER);

$header = join(" ", @header);

if ($header=~m/$winnt/) {sendoutput ("Microsoft(R) Windows NT(TM)\n(C)
Copyright 1985-1996 Microsoft Corp.\n");}
if ($header=~m/$wintwok/) {sendoutput ("Microsoft Windows 2000 [Version
5.00.2195]\n(C) Copyright 1985-1999 Microsoft Corp.\n");}

# Get the path for the fake prompt
getprompt($currentdrive[$index], $currentdir[$index]);

# If no args are passed, we cancel the $exit marker and launch the prompt
if ($input eq "") {$input = <STDIN>; $exit = 0;}

# Enters in an "interactive" session and will terminate on 'exit', the loop
will run only once if $exit is set to 1
while ($input ne "exit\n")
```

```

{
chomp $input;
$cdcommand = 0; $baddir = 0; $baddrive = 0;

SWITCH: {
    # If the command starts with 'cd', then it checks if the next character is \
    # If it is, then the string replaces the $currentdir. If it does not
start with
    # a \, then the string is appended at the end of the path in $currentdir
    # If nothing follows the cd command, then the command is put back in $input
    # since we modified the string for analysis with the shift command above
    $input=~m/^\cd/i && do {
        @input = split (//,$input);
        if ($input[0] eq "c" or $input[0] eq "C") { shift @input;}
        if ($input[0] eq "d" or $input[0] eq "D") { shift @input;}
        while ($input[0] eq " ") { shift @input;}

        if (@input)
        {
            if ($input[0] eq "\\")
            {
                $input = join ("",@input);
                $direxist = checkdir($currentdrive[$index], "\\", $input);
                if ($direxist)
                {
                    $currentdir[$index] = $input;
                    $cdcommand = 1;
                    $input = "cd ".$input;
                }
                else { $cdcommand = 1; $baddir = 1; $input="cd"; }
            }
            else
            {
                $input = join ("",@input);
                $direxist = checkdir($currentdrive[$index],$currentdir[$index],
$input);

                if ($direxist)
                {
                    $currentdir[$index] = $currentdir[$index].$input.\';
                    $cdcommand = 1;
                    $input = "cd ".$input;
                }
                else
                { $cdcommand = 1; $baddir = 1; $input="cd"; }
            }
        }
        else
        { $input="cd"; }
    }
    last SWITCH;
};
# If the command is 'cls', we nullify the command and simulate it from the

```

```

perl program
    $input=~m/cls/i && do {
        system("cls");
        sendinput ($input."\n");
        $input = "";
        last SWITCH;
    };

    # If the command is to change the drive letter (c:, d:, etc), we set our
    variables accordingly
    # If it's the first time the drive is accessed, an entry is added in the
    table to store it with its current directory
    $input=~m/^\w:$/ && do {
        $i = 0;
        foreach $drive (@currentdrive)
        {
            if ($currentdrive[$i]=~m/$input/i)
            { $index = $i; last SWITCH;}
            $i++;
        }
        if ($index ne $i)
        {
            $driveexist = checkdrive($input);
            if ($driveexist)
            {
                $index = $i;
                $currentdrive[$index] = $input;
                $currentdir[$index] = '.\';
            }
            else { $baddrive=1; $input = $currentdrive[$index];}
        }

        last SWITCH;
    };

    $nothing = 1;
}

# If it is a 'cd' command, then we simply execute a 'cd' command with the
$currentdir path
# If not, then we issue the 'cd' command anyway, as it is needed to
position the summoned instance
# of cm_.exe in the right directory, then we call the command typed at the
prompt
# $command contains the string of the final command to be sent to the real
DOS prompt
if ($cdcommand)
    { $command = "cm_.exe /C ".$currentdrive[$index]." && cd
    ".$currentdir[$index]; }
else
    { $command = "cm_.exe /C ".$currentdrive[$index]." && cd
    ".$currentdir[$index]." && ".$input; }

```

SecurityFocus IDS: Release : ComLog 1.0, a WIN32 command prompt logger

```
# Writes the input to the history file
sendinput ($input."\n");

# If the command is valid and it's not an empty carriage return, then we
call the command and capture the output

if ($baddir) { sendoutput("The system cannot find the path specified.\n"); }
if ($baddrive) { sendoutput("The system cannot find the drive specified.\n"); }

if ($input ne "")
{
    open (COMMANDOUTPUT, $command."|" ) or die "Can't open pipe";
    @output = <COMMANDOUTPUT>;
    close (COMMANDOUTPUT);
    sendoutput (@output);
}

# If the $exit marker is not set, get the fake prompt
if ($exit) {$input = "exit\n";} else {getprompt($currentdrive[$index],
$currentdir[$index]); $input = <STDIN>;}

} # End of While

# sends the final command (exit) to the history log file
sendinput ($input."\n");

if ($cleanlog) {unlink ($history);}

#End Of Main

#####
# procedure getcurrentdrive(currentdir) #
# This procedure gets the drive where is located the current directory. #
#####
sub getcurrentdrive
{ my ($dir) = @_;

open (PROMPT, "cd".$dir." && cd|" ) or die "Can't open pipe";
$prompt = <PROMPT>;
chomp $prompt;
close (PROMPT);

@prompt = split (//,$prompt);
$drive = join ("", ($prompt[0], $prompt[1]));
return ($drive);
}

#####
# procedure getprompt(currentdrive, currentdir) #
# This procedure receives the current drive and directory and fakes a
command prompt. #
```

SecurityFocus IDS: Release : ComLog 1.0, a WIN32 command prompt logger

```
#####
sub getprompt
{ my ($drive, $dir) = @_;
```

open (PROMPT, \$drive."&& cd ".\$dir." && cd |") or die "Can't open pipe";  
\$prompt = <PROMPT>;  
chomp \$prompt;  
close (PROMPT);  
sendoutput ("\n".\$prompt.">");  
}

```
#####
# procedure sendinput(line) #
# This procedure writes the input received into the history file #
#####
sub sendinput
{ my (@line) = @_;
```

\$now = localtime;

# Opening of history file  
open (HISTORY, ">>".\$history) || die "Can't open session log file";  
lock HISTORY;  
print HISTORY "\$now\n";  
print HISTORY "@line";  
close (HISTORY);  
}

```
#####
# procedure sendoutput(output) #
# This procedure writes the output from commands received into the history
file and to #
# the console. It also sanitize the output to conceal the program presence. #
#####
sub sendoutput
{ my (@output) = @_;
```

\$now = localtime;

# Opening of history file  
open (HISTORY, ">>".\$history) || die "Can't open session log file";  
lock HISTORY;  
print HISTORY "\$now\n";  
foreach \$line (@output)  
{  
\$line=~s/cm\_.exe/cmd.exe/i;  
if (!((\$line=~m/711,342/) || (\$line=~m/\w{8}.clg/) || (\$line=~m/cm\_/)))  
{ print "\$line";  
print HISTORY "\$line";  
}  
}

```

close (HISTORY);
}

#####
# procedure checkdir($drive, $path, $dir) #
# This procedure checks for the existence of a directory before changing to
it. #
#####
sub checkdir
{ my ($drive, $dir, $unknown) = @_;

$command = "cm_.exe /C ".$drive." && cd ".$dir." && if exist ".$unknown."
echo OK";

    open (COMMANDOUTPUT, $command." |") or die "Can't open pipe";
    $output = <COMMANDOUTPUT>;
    close (COMMANDOUTPUT);
    if ($output eq "OK\n") {return 1;} else {return 0;}
}

#####
# procedure checkdrive($drive) #
# This procedure checks for the existence of a drive before changing to it. #
#####
sub checkdrive
{ my ($drive) = @_;

    opendir (TESTDRIVE, $drive."\\") or return 0;
    closedir (TESTDRIVE);
    return 1;
}

#####
# procedure randomfilename() #
# This procedure generates a random filename for the session log. #
#####
sub randomfilename
{
    for ($gen=0; $gen<8; $gen++) {$random = int (rand 26)+97; $history =
$history.chr($random);}
    $history = $history.".clg";
    $history = $ENV{SystemRoot}."/Help/tutor/".$history;
    return $history;
}
#EOF

```

#### Appendix B. Sample session history (console)

(note: this session history was made with the pre-release version of ComLog. The same commands would now produce a slightly different result.)

Microsoft(R) Windows NT(TM)  
(C) Copyright 1985–1996 Microsoft Corp.

D:\commandlog>dir  
Volume in drive D is D  
Volume Serial Number is 0480–D01C

Directory of D:\commandlog

```
08/04/02 04:35a <DIR> .  
08/04/02 04:35a <DIR> ..  
08/04/02 04:26a 655,520 cmd.exe  
08/04/02 04:20a 8,726 comlog.pl  
08/04/02 04:10a 18,433 comlog.txt  
05/13/02 02:42p 1,506 command logger.txt  
02/11/02 11:47a 971 pseudo code.txt  
08/04/02 04:26a 655,520 root.exe  
08/04/02 04:28a 583 Shortcut to cmd.exe.lnk  
05/30/02 02:04p 35 systempath.txt  
07/21/02 03:27p 878 test.txt  
13 File(s) 1,557,580 bytes  
488,346,112 bytes free
```

D:\commandlog>echo "There is a copy of cm\_.exe and history.txt in here, but  
we d  
on't see it"

D:\commandlog>echo "That last message didn't echo because it contained the  
fobid  
den words"  
"That last message didn't echo because it contained the fobidden words"

D:\commandlog>cd ..

D:\>dir  
Volume in drive D is D  
Volume Serial Number is 0480–D01C

Directory of D:\

```
08/04/02 04:35a <DIR> commandlog  
06/25/02 12:57p <DIR> Dev  
12/25/00 09:34p <DIR> downloads  
08/04/02 04:20a <DIR> Log  
08/04/02 04:21a <DIR> LogAgent 2.0  
01/02/01 03:57p <DIR> movies  
07/24/01 12:59p <DIR> Musique  
08/04/02 02:42a <DIR> NONE  
06/25/02 01:59p <DIR> NTRESKIT  
08/04/02 02:59a 67,108,864 pagefile.sys  
05/20/01 06:10a <DIR> Program Files
```

```
08/04/02 04:36a <DIR> TEMP
04/30/02 02:33p <DIR> Test
10/15/00 04:06p <DIR> VIRUSES
05/24/02 06:22p <DIR> WINNT
05/24/02 06:22p 92 WINNTdun.bat
    16 File(s) 67,109,842 bytes
        488,345,088 bytes free
```

D:\>ipconfig

Windows NT IP Configuration

Ethernet adapter DE5284:

```
    IP Address. . . . . : 192.168.0.1
    Subnet Mask . . . . . : 255.255.0.0
    Default Gateway . . . . . : 192.168.0.1
```

PPP adapter NdisWan3:

```
    IP Address. . . . . : 0.0.0.0
    Subnet Mask . . . . . : 0.0.0.0
    Default Gateway . . . . . :
```

D:\>net share

Share name Resource Remark

---

```
IPC$ Remote IPC
C$ C:\ Default share
D$ D:\ Default share
G$ G:\ Default share
Log$ D:\Log
ADMIN$ D:\WINNT Remote Admin
Log D:\Log
The command completed successfully.
```

D:\>dir \log >> dirlog.txt

```
D:\>echo "The user see no display because he sent the output to a file"
"The user see no display because he sent the output to a file"
```

```
D:\>type dirlog.txt
Volume in drive D is D
Volume Serial Number is 0480-D01C
```

Directory of D:\log

```
08/04/02 04:20a <DIR> .
08/04/02 04:20a <DIR> ..
```

SecurityFocus IDS: Release : ComLog 1.0, a WIN32 command prompt logger

```
04/30/02 02:37p 3,819 adam.log
09/25/00 08:31p 373 bind.log
05/20/01 05:46a 6,917 getright.log
08/04/02 02:50a 131,546 IAMDB.RDB
09/25/00 08:31p 41 restart.log
08/04/02 04:20a 177 Scan Viruses.lnk
09/25/00 08:31p 36 shutdown.log
09/25/00 08:31p 104 startup.log
04/29/02 02:56p 218 test.bat
04/30/02 02:33p 3,721 test.txt
07/24/01 11:26a 5,553 ZALog.txt
    14 File(s) 153,044 bytes
    488,340,480 bytes free
```

```
D:\>dir
```

```
Volume in drive D is D
Volume Serial Number is 0480-D01C
```

```
Directory of D:\
```

```
08/04/02 04:35a <DIR> commandlog
06/25/02 12:57p <DIR> Dev
08/04/02 04:39a 886 dirlog.txt
12/25/00 09:34p <DIR> downloads
08/04/02 04:20a <DIR> Log
08/04/02 04:21a <DIR> LogAgent 2.0
01/02/01 03:57p <DIR> movies
07/24/01 12:59p <DIR> Musique
08/04/02 02:42a <DIR> NONE
06/25/02 01:59p <DIR> NTRESKIT
08/04/02 02:59a 67,108,864 pagefile.sys
05/20/01 06:10a <DIR> Program Files
08/04/02 04:36a <DIR> TEMP
04/30/02 02:33p <DIR> Test
10/15/00 04:06p <DIR> VIRUSES
05/24/02 06:22p <DIR> WINNT
05/24/02 06:22p 92 WINNTdun.bat
    17 File(s) 67,109,842 bytes
    488,339,456 bytes free
```

```
D:\>cd commandlog
```

```
D:\commandlog>dir
```

```
Volume in drive D is D
Volume Serial Number is 0480-D01C
```

```
Directory of D:\commandlog
```

```
08/04/02 04:35a <DIR> .
08/04/02 04:35a <DIR> ..
08/04/02 04:26a
```