GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Security Essentials (Security 401)"
at http://www.giac.org/registration/gsec

Cracking the HP Secure Web Console

Michael W. Shaffer 04 November 2000

Agilent Laboratories has for some time employed a device known as the **HP Secure Web Console ®** (1) for providing network access to the console port of various UNIX hosts that we manage. Physically, the device is a small featureless box with a single serial port, a single 10BaseT Ethernet connection, and a power brick. When the serial port is connected to the console port of a host and the Ethernet connection is cabled to your existing network, the device provides network access to the host console through a java applet that may be run in any web browser. To access the applet and make a connection to the console, an administrator would point their browser at the IP address or hostname assigned to the web console device and would be greeted with a simple login screen. This initial login is based on user accounts and passwords stored in the web console device itself. The password hashes are assumed to be reasonably secure since they are stored and sent over the network as MD5 hashes. However, the hash is passed in the clear from the login applet to the device, so the possiblity does exist for sniffing and cracking of hashes by an intruder with proper access to the network.

Once the user has logged in, the web console will supply to the browser several java applets including a menu for various administrative chores and most importantly a java console client which emulates an ordinary text terminal. When first using these devices, I was somewhat disconcerted to find that the SSL security indicator on the browser is never activated, indicating that at least the HTTP traffic between browser and console was being conducted in the clear. I discussed this with several colleagues, and we were unable to reach a consensus as to whether the traffic to and from the console was really secure or not. Some insisted that the conversations were secure, while others remained unconvinced. At this point I set out to explore the issue further using some basic network surveillance tools, notably **`nmap`** (2) and **`tcpdump`** (3). The `nmap` utility is a popular and effective tool for scanning hosts and networks to determine their status, network port exposure, and to some extent what operating system or other software they may be running. The `tcpdump` tool, on the other hand, is a network traffic 'sniffer' which is commonly used for capturing, logging, and monitoring the actual data packets which a network carries.

I began by simply running `tcpdump` and watching all traffic between my workstation (shown as `wksta` in the traces shown) and a chosen web console device (shown as `webcon`) during a typical login session. After several sessions and some casual analysis of the `tcpdump` output, I began to notice that there was traffic not just to and from port 80 on the web console (`http`) but also to and from port 23 (`telnet`). The command I used to observe this traffic was:

```
tcpdump -l -x host webcon &> webcon.log &
```

This command starts `tcpdump` and directs its output into a log file for later analysis. The options used are:

- **`-l`**: Make output line buffered so that traffic can be followed from the log file as it is received by a command such as `'tail -f webcon.log'`.
- **`-x`**: Show the full data (actually up to the first 128 bytes) of each packet in the output as well as a human readable representation of the packet headers. If more data is desired from each packet, the `-s` or 'snaplen' option can be used. For instance: `'tcpdump -l -x -s 65535'` would show the maximum amount of data that `tcpdump` can process from each packet.
- **`host webcon`**: This is the 'filter expression' and appears last on the command line after any options. This expression tells `tcpdump` to focus only on traffic going to or from the host named `webcon`. The

© SANS Institute 2000 - 2005 Author retains full rights.

expression syntax allows filters to be specified on nearly any aspect or feature of a packet from the ethernet frame, IP, and protocol headers (3). If no expression is specified, then `tcpdump` will report all packets that it is able to collect.

```
Kernel filter, protocol ALL, datagram packet socket
tcpdump: listening on all devices
11:49:31.252993 eth0 > wksta.1073 > webcon.www: S ...
                        4500 003c 0000 4000 4006 4d4b 821d ecd0
                        821d fc65 0431 0050 ce21 29bb 0000 0000
                        a0c2 16d0 a5fc 0000 0204 05b4 0402 080a
                        0000 a0ab 0000 0000 0103 0300
11:49:31.258258 eth0 < webcon.www > wksta.1073: S ...
                        4500 002c 8f56 0000 1d06 2105 821d fc65
                        821d ecd0 0050 0431 ef7b 6201 ce21 29bc
                        6012 1000 4cc9 0000 0204 05b4 8888 8888
                        8888
11:49:31.258358 eth0 > wksta.1073 > webcon.www: ...
                        4500 0028 0000 4000 4006 4d5f 821d ecd0
                        821d fc65 0431 0050 ce21 29bc ef7b 6202
                        5010 16d0 5db6 0000

... much traffic from login and loading of the console applet ...

11:49:31.879889 eth0 > wksta.1075 > webcon.www: ...
                        4500 0028 0000 4000 4006 4d5f 821d ecd0
                        821d fc65 0433 0050 cdef adb8 ef7e 520a
                        5010 1920 e78e 0000
11:49:31.985539 eth0 > wksta.1076 > webcon.telnet: S ...
                        4500 003c 0000 4000 4006 4d4b 821d ecd0
                        821d fc65 0434 0017 cda3 e5d0 0000 0000
                        a0c2 16d0 ea51 0000 0204 05b4 0402 080a
                        0000 a0f4 0000 0000 0103 0300
11:49:31.990518 eth0 < webcon.telnet > wksta.1076: S ...
                        4500 002c 8f6d 0000 1d06 20ee 821d fc65
                        821d ecd0 0017 0434 ef7f 4a01 cda3 e5d1
                        6012 1000 a963 0000 0204 05b4 8888 8888
                        8888
11:49:31.990587 eth0 > wksta.1076 > webcon.telnet: ...
                        4500 0028 0000 4000 4006 4d5f 821d ecd0
                        821d fc65 0434 0017 cda3 e5d1 ef7f 4a02
                        5010 16d0 ba50 0000
```

Ouput of `'tcpdump -x host webcon'` during a web console login

More focused analysis revealed that the port 23 traffic began immediately after a successful login when the console java applet appeared in the user's browser. The nature of the applet and the fact that it appeared to be communicating with the well known port for the `telnet` service on the console led me to suspect that it was in fact some sort of modified `telnet` client.

At this point I decided to check my assumptions about the console applet and also see if there were any other ports open on the web console by using `nmap`. I ran a full scan of all `tcp` ports on the `webcon` device using the command:

```
nmap -sS -p 1- webcon &> nmap.log &
```

The options used for this `nmap` run are:

- **`-sS`**: Use a tcp 'SYN' scan instead of the normal full 'CONNECT' scan.
- **`-p 1-`**: Scan all tcp ports from 1 upwards (meaning all of them from 1 to 65535).
- **`webcon`**: The name of the particular host to scan. `tcpdump`, `nmap` has a huge variety of options (2), and

© SANS Institute 2000 - 2005 Author retains full rights.

it can be used to perform a variety of both tcp and udp scans on either single hosts or entire networks.

Like

```
Starting nmap V. 2.53 by fyodor@insecure.org ( www.insecure.org/nmap/ )
Interesting ports on webcon (xxx.xxx.xxx.xxx):
(The 65534 ports scanned but not shown below are in state: closed)
Port       State       Service
23/tcp     open        telnet
80/tcp     open        http

Nmap run completed -- 1 IP address (1 host up) scanned in 8 seconds
```

Ouput of `'nmap -sS -p 1- webcon'`

The scan revealed that there were indeed only two ports open, `telnet` and `http`. I now began to focus entirely on the traffic between my workstation and the web console through the `telnet` port. Continuing with my assumption that the java applet involved was using some modified 'secure' version of the standard `telnet` protocol, I decided to compare traces of similar activity through from both a web console `telnet` session and a normal `telnet` session. After making the initial connection to both sessions, I simply typed a long string of 'a' characters into each. In cryptographic terms, this could be considered a simplistic 'known plaintext' attack since I was feeding some known input to the console applet and watching the encrypted traffic that it would produce based on that input.

```
Kernel filter, protocol ALL, datagram packet socket
tcpdump: listening on all devices
19:03:51.091949 eth0 > wksta.1138 > webcon.telnet: P 3804397219:3804397220(1) ...
                         4500 0029 11df 4000 4006 e9a4 c0a8 0020
                         821d fc65 0472 0017 e2c2 7ea3 dbaa bab8
                         5018 7fb8 9e74 0000 56
19:03:51.283528 eth0 < webcon.telnet > wksta.1138: P 1:2(1) ack 1 win 4096
                         4500 0029 44f9 0000 1a06 1c8b 821d fc65
                         c0a8 0020 0017 0472 dbaa bab8 e2c2 7ea4
                         5018 1000 0e2c 0000 5673 6169 7068 730d
                         3724
19:03:51.302447 eth0 > wksta.1138 > webcon.telnet: . 1:1(0) ack 2 win 32696 ...
                         4500 0028 11e1 4000 4006 e9a3 c0a8 0020
                         821d fc65 0472 0017 e2c2 7ea4 dbaa bab9
                         5010 7fb8 f47b 0000
19:03:51.367076 eth0 > wksta.1138 > webcon.telnet: P 1:2(1) ack 2 win 32696 ...
                         4500 0029 11e2 4000 4006 e9a1 c0a8 0020
                         821d fc65 0472 0017 e2c2 7ea4 dbaa bab9
                         5018 7fb8 9e72 0000 56
19:03:51.493494 eth0 < webcon.telnet > wksta.1138: P 2:3(1) ack 2 win 4096
                         4500 0029 44fa 0000 1a06 1c8a 821d fc65
                         c0a8 0020 0017 0472 dbaa bab9 e2c2 7ea5
                         5018 1000 0e2a 0000 5656 3031 0332 47aa
                         7082
19:03:51.493559 eth0 > wksta.1138 > webcon.telnet: P 2:5(3) ack 3 win 32696 ...
                         4500 002b 11e3 4000 4006 e99e c0a8 0020
                         821d fc65 0472 0017 e2c2 7ea5 dbaa baba
                         5018 7fb8 4818 0000 5656 56
```

Partial output of `'tcpdump -x host webcon and port telnet'` during a secure web console session

```
Kernel filter, protocol ALL, datagram packet socket
tcpdump: listening on all devices
09:52:49.808752 eth0 > wksta.1033 > telnethost.telnet: P 319405533:319405534(1) ...
```

© SANS Institute 2000 - 2005 Author retains full rights.

```
                        4500 0035 0000 4000 4006 4d53 821d ecd0
                        821d fc64 0409 0017 1309 bddd e888 4594
                        8018 28e6 cef2 0000 0101 080a 0006 8386
                        37ba 7200 61
09:52:49.809004 eth0 < telnethost.telnet > wksta.1033: P 1:2(1) ack 1 win 32768 ...
                        4500 0035 4247 4000 3f06 0c0c 821d fc64
                        821d ecd0 0017 0409 e888 4594 1309 bdde
                        8018 8000 6dde 0000 0101 080a 37ba 7bf9
                        0006 8386 61
09:52:49.809065 eth0 > wksta.1033 > telnethost.telnet: . 1:1(0) ack 2 win 10470 ...
                        4500 0034 0000 4000 4006 4d54 821d ecd0
                        821d fc64 0409 0017 1309 bdde e888 4595
                        8010 28e6 2601 0000 0101 080a 0006 8386
                        37ba 7bf9
09:52:50.041074 eth0 > wksta.1033 > telnethost.telnet: P 1:2(1) ack 2 win 10470 ...
                        4500 0035 0000 4000 4006 4d53 821d ecd0
                        821d fc64 0409 0017 1309 bdde e888 4595
                        8018 28e6 c4df 0000 0101 080a 0006 839e
                        37ba 7bf9 61
09:52:50.041283 eth0 < telnethost.telnet > wksta.1033: P 2:3(1) ack 2 win 32768 ...
                        4500 0035 4248 4000 3f06 0c0b 821d fc64
                        821d ecd0 0017 0409 e888 4595 1309 bddf
                        8018 8000 6dad 0000 0101 080a 37ba 7c10
                        0006 839e 61
```

Partial output of `'tcpdump -x host telnethost and port telnet'` during an ordinary `telnet` session

It did not take long for me to notice that there was some highly alarming regularity in the traffic produced by the console applet in response to my string of 'a' characters. While the traffic was different from that produced by the clear text telnet session, it was also completely uniform. In the examples above, I have highlighted the byte '61' in several of the packets from the ordinary `telnet` session. This is the hexadecimal value of the ASCII 'a' character code, so it is not surprising that a string of packets carrying this value are seen when a string of 'a' characters are typed into the session. In the traffic for the secure web console session, I have highlighted the byte '56' from several of the packets which seem highly similar to those from the ordinary session. While observing the tcpdump output in realtime, I could easily see that the console applet was sending a '56' for every 'a' that I typed. This uniformity suggested initially that if an encryption algorithm was in use it must be an extremely simplistic one since the critical function of all robust encryption schemes is to introduce as much randomness as possible into their output. This behavior suggested that the encryption (more properly referred to as 'encoding' in this case) being performed must be some sort of one-to-one mapping of character values to transmitted values.

I studied this further by entering a number of strings of known, ordered characters (for instance, the lower case alphabet `'abcdefghijklmnopqrstuvwxyz'`) into the console applet and recording the transmitted byte values to build up a 'translation table'. The pattern that emerged soon suggested an elementary transformation, but for the sake of completeness I persisted until I had mapped nearly the entire printable ASCII character set in this fashion. A portion of the mapping that I recorded is shown below:

```
Encoded   Typed
  57        a
  55        b
  54        c
  53        d
  52        e
  51        f
  50        g
```

© SANS Institute 2000 - 2005 Author retains full rights.

```
5f        h
5e        i
5d        j
5c        k
5b        l
5a        m
59        n
58        o
47        p
46        q
45        r
44        s
43        t
42        u
41        v
40        w
4f        x
4e        y
4d        z
4c        {
4b        |
4a        }
49        ~
```

Sample of observed character value mapping

Many readers will already have deduced the transformation from the above sample, but to belabor the point a bit more I wrote out a complete table of the binary, hexidecimal, and ASCII character values for the uppercase alphabet. It was observation of the relationships of the binary representations of the plain and encoded values that made the relationship most apparent. The pattern I noticed suggested that a simple XOR or 'Exclusive OR' was at work. XOR is a really convenient operation since you can take two numbers, XOR them with each other, and then XOR the result with either of the first numbers to get the other number. What this means is that I only had to do a couple of XORs with the plain and 'encrypted' character values to confirm that this was indeed the operation and that the 'key' was the hexadecimal value `0x37`.

```
Encrypted                ASCII
Hex      Binary   Char   Hex       Binary    Dec
-----------------------------------------------
77    0111 0111     @     40    0100 0000    64
76    0111 0110     A     41    0100 0001    65
75    0111 0101     B     42    0100 0010    66
74    0111 0100     C     43    0100 0011    67
73    0111 0011     D     44    0100 0100    68
72    0111 0010     E     45    0100 0101    69
71    0111 0001     F     46    0100 0110    70
70    0111 0000     G     47    0100 0111    71
7f    0111 1111     H     48    0100 1000    72
7e    0111 1110     I     49    0100 1001    73
7d    0111 1101     J     4a    0100 1010    74
7c    0111 1100     K     4b    0100 1011    75
7b    0111 1011     L     4c    0100 1100    76
7a    0111 1010     M     4d    0100 1101    77
79    0111 1001     N     4e    0100 1110    78
78    0111 1000     O     4f    0100 1111    79

40    0100 0000
77    0111 0111   XOR
      ---------
37    0011 0111

41    0100 0001
76    0111 0110   XOR
```

© SANS Institute 2000 - 2005 Author retains full rights.

```
         ---------
  37     0011 0111
```

Table of encoded and plain values for the uppercase ASCII alphabet

Now that I had a guess as to what the encoding scheme for the console applet was, I decided to search the various online security forums to see if I could confirm my hypothesis and determine if the value that I had deduced was a constant or might vary from session to session or device to device. A quick visit to Google netted a brief thread (4) from the Security Portal (5) web archives of the BUGTRAQ security mailing list from around December 1999 that confirmed both my analysis and the `0x37` key value.

For completeness, I developed a simple perl script that will 'decode' the content of a web console session from the output of snooping the session traffic with a `tcpdump` command similar to those shown above. The source for the script appears below.

```
#!/usr/bin/perl

%charmap = (
  0x0a, "<>\n",
  0x0d, "<>\n"
);

$f_show = 0;

while (<>)
{
  chomp;
  $line = $_;

  if ($line =~ /^[0-9]{2}:/o)
  {
    $f_show = 0;
  }

  if ($line =~ /^\s+5018 [0-9a-f]{4} [0-9a-f]{4} 0000/o)
  {
    $f_show = 1;
  }

  if ($f_show)
  {
    if ($line =~ /^\s+5018 [0-9a-f]{4} [0-9a-f]{4} 0000 ?(.+)$/o)
    {
      $line = $1;
    }
    while ($line =~ /([0-9a-f]{2})/g)
    {
      $word = $1;
      $val = hex ($word) ^ 0x37;
      if (defined ($charmap{$val}))
      {
        $msg = $charmap{$val};
      }
      else
      {
        $msg = chr ($val);
      }
      print $msg;
    }
  }
}
```

© SANS Institute 2000 - 2005 Author retains full rights.

To use the script, simply run:

```
tcpdump -x dst host webcon and dst port telnet &> decode.log
```

and then pipe the 'decode.log' file through this perl script to see everything that was typed into the console java applet in plain text.

Conclusions

The method used for data encoding or 'scrambling' as HP refers to it in the HP Secure Web Console is not likely to provide a sufficient level of security from any but the most naive attackers who wish to snoop on the activities of system admins using these devices. Any system administrator considering deployment of new devices such as this (even those that feature the word 'Secure' in their name), should test them carefully in their own environment and question their vendors directly to be sure that the exact methods of data transmission are known and are in fact secure. Such testing does not require either expensive tools or exotic skills but simply patience and careful observation. In many cases (such as this one) a complete analysis and exploit of a security weakness is both simple and educational for the administrator who wishes to understand more about the functioning of systems under their control.

Bibliography

1. *HP Secure Web Console*.
   URL: http://www.unix.hp.com/management/confmanagement/swc/index.html (04 November 2000).
2. *NMAP -- The Network Mapper*.
   URL: http://www.insecure.org/nmap/ (04 November 2000).
3. *TCPDUMP*.
   URL: http://www.tcpdump.org/ (04 November 2000).
4. Zeverina, David. *Re: HP Secure Web Console*. 02 December 1999.
   URL: http://www.securityportal.com/list-archive/bugtraq/1999/Dec/0060.html (04 November 2000).
5. *Security Portal*.
   URL: http://www.securityportal.com/ (04 November 2000).

© SANS Institute 2000 - 2005 Author retains full rights.