Reverse Engineering Dell iDRAC6 Express: Fan Control

Part I: Accessing the Root User

November 02, 2017 by **Matt** in /Home/Firmware with **No Comments**

I have an aging Dell T710 that I bought a number of years ago. I use it to offload long running processes, handle file sharing, shared services, jails and so on. It's been running FreeBSD for a couple of years since I moved away from Linux.

Like most server-class hardware this tower is particularly loud. Dell shipped it with pulse width modulated fans (http://stuffbymatt.ca/static/Firmware/dell_idrac6_AFC0912DE-7X60.pdf) that are anything but quiet and thanks to the iDRAC6 Express software that runs on a WPCM450 integrated baseboard management controller (BMC) the fan control & throttling makes it sound like a jet is taking off. That's fine if you have somewhere to put it but we've moved to a small apartment and it has to sit in our office. As we're fans of hearing ourselves think (pun intended) I needed some way to wrestle control away from the default iDRAC firmware.

The WPCM450 runs a version of Busybox Linux on an ARM processor. As you would expect, Dell has heavily customized this software and provides access to it via a web interface, Telnet, SSH, RAC, IPMI or Serial interface to RAC/IPMI. None of these options offer the controls we're looking for and the SSH/Telnet options are locked down to the iDRAC SM-CLP command line interface. In short, Dell has turned an otherwise very useful out-of-band management tool into a glorified toaster oven.

Previous versions of the Dell BMC have been reverse engineered by others however the Dell T710 is an 11th generation server so sadly we can't use those methods.

Standard disclaimer

If you follow these instructions you may brick your iDRAC, cause damage to your hardware, open security holes, cause future upgrades to go awry and make small children cry. I take zero responsibility for anything that you do to your own systems. You will certainly void Dell's warranty for whatever that's worth.

Before we get started

Make sure you have Telnet access enabled under the Web GUI at **iDRAC Settings** > **Network/Security** > **Services**. Alternatively if you have racadm access, either through SSH to iDRAC or otherwise, you can enable Telnet by issuing the following command:

```
$ racadm config -g cfgSerial -o cfgSerialTelnetEnable 1
```

You may need to adjust the racadm command appropriately if you are accessing it indirectly.

> If you have just set up your iDRAC you should be able to gain access using the default **root** user & password **calvin**

Also, you'll need to obtain an up-to-date copy of the iDRAC firmware. You can use ESM_Firmware_9GJYW_LN32_2.90_A00.BIN or ESM_Firmware_9GJYW_WN32_2.90_A00.EXE with the latter being a bit easier to open since it's just a ZIP file. I successfully extracted the firmware from the .BIN file but it was a bit more work as I run FreeBSD and not Linux. You'll find the firmware under `payload/firmimg.d6`

Exploring firmimg.d6 with binwalk

Binwalk helps us determine how the firmware image is set up and gives us the addresses that we can use to split it up into workable sections. Be mindful that some of these addresses are spurious so we can't use binwalk's automated extraction feature.

```
Scan Time:     2017-11-01 13:02:06
Target File:   /usr/home/matt/tmp/firmimg.d6
MD5 Checksum:  24a113b5baae30dfac9a888d69a09784
Signatures:    344

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
512           0x200           uImage header, header size: 64 bytes, header CRC: 0x14BBD
103296        0x19380         gzip compressed data, maximum compression, from Unix, las
639113        0x9C089         Certificate in DER format (x509 v3), header length: 4, se
1057005       0x1020ED        Certificate in DER format (x509 v3), header length: 4, se
1090593       0x10A421        Certificate in DER format (x509 v3), header length: 4, se
1722433       0x1A4841        Certificate in DER format (x509 v3), header length: 4, se
2090901       0x1FE795        Certificate in DER format (x509 v3), header length: 4, se
2142644       0x20B1B4        SHA256 hash constants, little endian
2737477       0x29C545        Certificate in DER format (x509 v3), header length: 4, se
3060125       0x2EB19D        Certificate in DER format (x509 v3), header length: 4, se
3065589       0x2EC6F5        Certificate in DER format (x509 v3), header length: 4, se
3071865       0x2EDF79        Certificate in DER format (x509 v3), header length: 4, se
3097813       0x2F44D5        Certificate in DER format (x509 v3), header length: 4, se
3104893       0x2F607D        Certificate in DER format (x509 v3), header length: 4, se
3104937       0x2F60A9        Certificate in DER format (x509 v3), header length: 4, se
3104981       0x2F60D5        Certificate in DER format (x509 v3), header length: 4, se
3209149       0x30F7BD        Certificate in DER format (x509 v3), header length: 4, se
3395109       0x33CE25        Certificate in DER format (x509 v3), header length: 4, se
3457388       0x34C16C        Linux kernel version "2.6.23.1 (root@BDCCBFV24.blr.amer.d
3474457       0x350419        Neighborly text, "neighborighbor_position"
3474489       0x350439        Neighborly text, "neighbor_positioncheck_internal_node"
3474518       0x350456        Neighborly text, "neighbor_positione_starts"
```

```
3475888       0x3509B0        Neighborly text, "neighbor_in_cache"
3476008       0x350A28        Neighborly text, "neighborscheck_balance"
3476211       0x350AF3        LZMA compressed data, properties: 0xC0, dictionary size:
3476247       0x350B17        LZMA compressed data, properties: 0xC0, dictionary size:
3604728       0x3700F8        CRC32 polynomial table, little endian
3993948       0x3CF15C        Neighborly text, "neighbortion !(dest  NULL || src  NULL)
4000156       0x3D099C        Neighborly text, "neighboring item failed at fs/reiserfs/
4010094       0x3D306E        Neighborly text, "neighbor (%b %z) is not in the tree(n_c
4010592       0x3D3260        Neighborly text, "neighbor_father.path_length < FIRST_PAT
4011881       0x3D3769        Neighborly text, "neighbor&& PATH_OFFSET_POSITION(p_s_tb-
4012322       0x3D3922        Neighborly text, "neighbor (%d != %d - %d)gative or zero
4026862       0x3D71EE        Neighborly text, "neighbor, it must have free_space==0 (n
4109761       0x3EB5C1        Unix path: /proc/fs/cifs/SecurityFlags
4110202       0x3EB77A        Unix path: /proc/fs/cifs/PacketSigningEnabled to be on
4119170       0x3EDA82        Unix path: /proc/fs/cifs/LookupCacheEnabled to 0
4452031       0x43EEBF        LZMA compressed data, properties: 0xC0, dictionary size:
4452055       0x43EED7        LZMA compressed data, properties: 0xC0, dictionary size:
4466660       0x4427E4        LZMA compressed data, properties: 0xC8, dictionary size:
4480512       0x445E00        CramFS filesystem, little endian, size: 52203520 version
56795116      0x3629FEC       U-Boot version string, "U-Boot 1.2.0 (Jul 24 2017 - 09:59
56795728      0x362A250       CRC32 polynomial table, little endian
```

Split up firmware into logical sections

We'll modify two of the resulting files and use the others to put Humpty back together again.

Note that the addresses used below only apply to the 9GJYW 2.90 A00 ESM package. Any other package/firmware image is likely to have different addresses.

```
## This is the header file containing CRC32 checksums (we'll need to modify it accordin
dd if=firmimg.d6 of=01_header bs=1 count=512

## Back this up just-in-case
cp 01_header 01_header_VIRGIN

## Will be used later to reconstruct the firmware image
dd if=firmimg.d6 of=02_before_cramfs bs=1 skip=512 count=$((4480512-512))

## This is the Linux Compressed ROM image that we need to modify
dd if=firmimg.d6 of=03_cramfs bs=1 skip=4480512 count=52203520

## Will be used later to reconstruct the firmware image
dd if=firmimg.d6 of=04_after_cramfs bs=1 skip=$((4480512+52203520))
```

Unpack CramFS filesystem and make changes

Do this as root to preserve permissions. Then make modifications to image to gain shell access and finally repack for use.

```
$ cramfsck -x cramfs 03_cramfs

## Otherwise we can't su to root later on
$ chmod u+s cramfs/bin/busybox
```

Also patch `etc/sysapps_script/S_7015_telnetd_app.sh` as follows to allow backdoor access via netcat:

```
--- cramfs_VIRGIN/etc/sysapps_script/S_7015_telnetd_app.sh      1969-12-31 17:00:00.000
+++ cramfs/etc/sysapps_script/S_7015_telnetd_app.sh     2017-11-01 11:39:57.461625000 -
@@ -15,6 +15,7 @@
     # test if the variable is null is true
     if [  ] && [  == "true" ]; then
        -p  -t
+      /bin/nc -l -p 2323 -e /bin/bash &
     fi
     touch /var/run/utmp
   else
```

Finally, repack...

```
$ mkcramfs cramfs 03_cramfs_MODIFIED
```

Update header checksums

The Dell firmware upgrade process uses checksums to verify "partition" content so the header needs to be updated to reflect changes made. Your checksums will surely be different so don't reuse those below. They are for example only.

I have used this Perl script (http://stuffbymatt.ca/static/Firmware/dell_idrac6_crc32.txt) to generate the CRC32 hex digests as required by the firmware header. Save as `~/bin/crc32` and execute as seen below.

Note that checksums are stored in reverse (little endian) order so they effectively "read backwards" when viewed from a hex editor.

```
$ crc32 03_cramfs_MODIFIED
f00d648a

## Use whatever hex editor strikes your fancy (hte, radare2, etc.)
## Modify 34h, 35h, 36h and 37h. This is the cram "partition" checksum.
$ hte 01_header

## Split out the actual header content from the header's own CRC32
$ dd if=01_header bs=1 skip=4 of=header_for_crc32
$ crc32 header_for_crc32
7b4bb3b9

## Use whatever hex editor strikes your fancy (hte, radare2, etc.)
## Modify 0h, 1h, 2h and 3h. This is the header's own checksum.
$ hte 01_header
```

Example from `/usr/bin/hexdump 01_header`

```
00000000  b9 b3 4b 7b 01 01 03 00  02 5a 04 00 00 1e 63 03
00000010  01 02 00 00 01 13 08 00  57 48 4f 56 00 00 00 00
00000020  00 02 00 00 e0 5b 44 00  ac 62 73 88 00 5e 44 00
00000030  00 90 1c 03 8a 64 0d f0  00 ee 60 03 38 2e 02 00
00000040  80 ea c3 89 00 00 00 00  00 00 00 00 00 00 00 00
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00
*
00000200
```

Repack firmware image, transfer to iDRAC & flash

I used TFTP to get the image file to iDRAC. Setting up a TFTP service is left as an exercise to the reader.

(I found that this is the easiest way of updating the iDRAC as I'm running FreeBSD, can't make use of Dell's bmcfwul utility, don't want to set up a boot disk or bother getting racadm running locally. YMMV.)

```
## Create final firmware image - should be EXACTLY the same size as the original but wi
$ cat 01_header 02_before_cramfs 03_cramfs_MODIFIED 04_after_cramfs > firmimg_MODIFIED.

## Copy to TFTP directory, wherever that happens to be
$ cp firmimg_MODIFIED.d6 /var/tftpd/firmimg.d6

## Connect to iDRAC and update firmware -- this can be done in several ways. I used SSH
$ ssh root@192.168.1.254

/admin1-> racadm fwupdate -g -u -a 192.168.1.1
```

The firmware transfer and flashing process will take a while. If you get anything other than a successful flash, issue `racadm gettracelog` to find out what went wrong. It will probably be a bad checksum. Once you have successfully flashed the firmware and iDRAC has reset proceed to the next step.

Connect to root backdoor via netcat

Although it won't be a proper terminal this hack will allow you to adjust some critical things.

If for some reason netcat doesn't work, use `racadm racdump` to view the iDRAC process list and determine if `nc` is running. You should see something like this:

```
6551 root       3548 S   /bin/nc -l -p 2323 -e /bin/bash
```

I've added empty lines below to make the exchange via netcat more readable but those wouldn't typically be there

```
$ netcat 192.168.1.254 2323

## Make sure we are root
whoami                            ==> send this
root                              <== expect this

## Change root's password so we can su later on
passwd                            ==>
Changing password for root        <==
New password:calvin               <===> (send "calvin")
                                  <====
Bad password: too weak            <====
Retype password:calvin            <===> (send "calvin")
                                  <====
Password for root changed by root <====
```

Before you leave netcat update a path in `/etc/passwd`. Since iDRAC SSH and Telnet access turns your user into `racuser` regardless of login name (and thereby grants access to the CLP shell) we change this to `/bin/sh` so our user can access the system, become root and so on by issuing the following commands:

```
cat /flash/data0/etc/passwd | sed -e 's#/usr/bin/clpd#/bin/sh#' > /flash/data0/etc/pass
mv /flash/data0/etc/passwd.new /flash/data0/etc/passwd
```

Now break out using Ctrl-C since Ctrl-D will close the netcat process on iDRAC

```
^C
```

Test access via iDRAC SSH or Telnet

Be aware that the root password change that we made earlier won't persist across RAC resets/firmware updates.

```
$ ssh root@192.168.1.254
root@192.168.1.254's password: calvin
[WPCM450 /flash/data0/home/root]$ whoami
racuser
[WPCM450 /flash/data0/home/root]$ su -
Password:
[WPCM450 ~]$ whoami
root
```

From here you can access the original SM-CLP command line interface by issuing the command `clpd`

We'll cover actual fan control in the next part. (http://stuffbymatt.ca/Firmware/dell_idrac6_pt2.html)

Scripting these steps

If you find yourself having to repeat these steps multiple times you can use a simple shell script to automate a number of the steps, thereby making the whole process faster and less error prone. Here is a copy of my script (http://stuffbymatt.ca/static/Firmware/dell_idrac6_deploy.txt) but you'll probably need to make changes to reflect your environment.

Related reading

Here are some other resources that might help you. They mostly pertain to older Dell PowerEdge systems and weren't completely applicable to my problem. If you aren't familiar with out-of-band systems check out OOB Management and You!

Dell PowerEdge (PE) fan woes

- Reducing Dell PE 2950/2900/2800 Fan Noise (fan mod + BMC firmware)
- How to Make a Dell PE Quieter
- Quieting the Loud Fans on a Dell PE 2950 Server
- Hacking Fan Speed on Dell PE Servers
- Quieting Dell PE 1855/1955 Blade System Fan Noise: Undocumented DRAC/MC Commands

Reverse engineering firmware

- Practical Reverse Engineering: A 5 Part Tutorial
- Reverse Engineering Linksys WAG120N
- Guessing Checksum Algorithms
- radare Reference Manual
- Netcat and Reverse Telnet

Comments

This website respects your privacy and doesn't track you with Google Analytics.
Content on this website does not represent the views of other parties.