

Tesla released the P85D in 2014. At that time the vehicle came with "insane mode" acceleration with a 0-60 time of 3.2 seconds. Later in July of 2015, Tesla announced "Ludicrous mode" that cut the 0-60 time down to 2.8 seconds. This upgrade was offered both new and as a hardware and firmware change to the existing fleet of P85D vehicles. Since then, Tesla has released newer ludicrous vehicles. What makes the P85D upgrade unique was how the process required changes to the vehicle's Battery Management System(BMS). The BMS handles power requests from the drive units and other components of the car. This paper will outline the ludicrous upgrade process through reverse engineering CAN bus messages, CAN bus UDS routines and various firmware files contained on the car.

This process also involved extracting Python source code used for diagnostics to determine that the ludicrous upgrade process involved replacing the contactors and fuse with higher current versions as well as modifying the current sensing high voltage "shunt" inside the battery pack.

The process was performed on an actual donor P85D. During the upgrade attempt, the car was bricked, forcing it to be towed to another state for troubleshooting. The cause of the failure was a file contained on the security gateway that had to be modified manually in order to update the gateway with the newer configuration of the vehicle's battery pack. This paper will outline the structure of that file and how it was successfully modified.

Vehicle Overview

The Model S powertrain network consists of a standard vehicle CAN bus network running at 500 Kbit/sec. The modules on this network support standard UDS commands, this standard was critical in performing this upgrade process. This CAN bus network as well as several of the other networks all connected to the Security gateway contained within the Central Information Display. This security gateway acts as a vehicular firewall, blocking some CAN bus messages while bridging others onto CAN bus networks, such as the Chassis and Body CAN networks, as well as an ethernet network that wraps CAN bus messages inside UDP packets. This security gateway also contains the vehicle configuration as well as other critical files that this paper will outline alter.

BMS Overview

The BMS is the deciding module that allows the drive units to have only as much power as the BMS allows. It is contained on a circuit board inside the battery pack, it can only be physically accessed by removing the battery from the vehicle. Figure 1 shows a BMS purchased on the secondary market for this research:

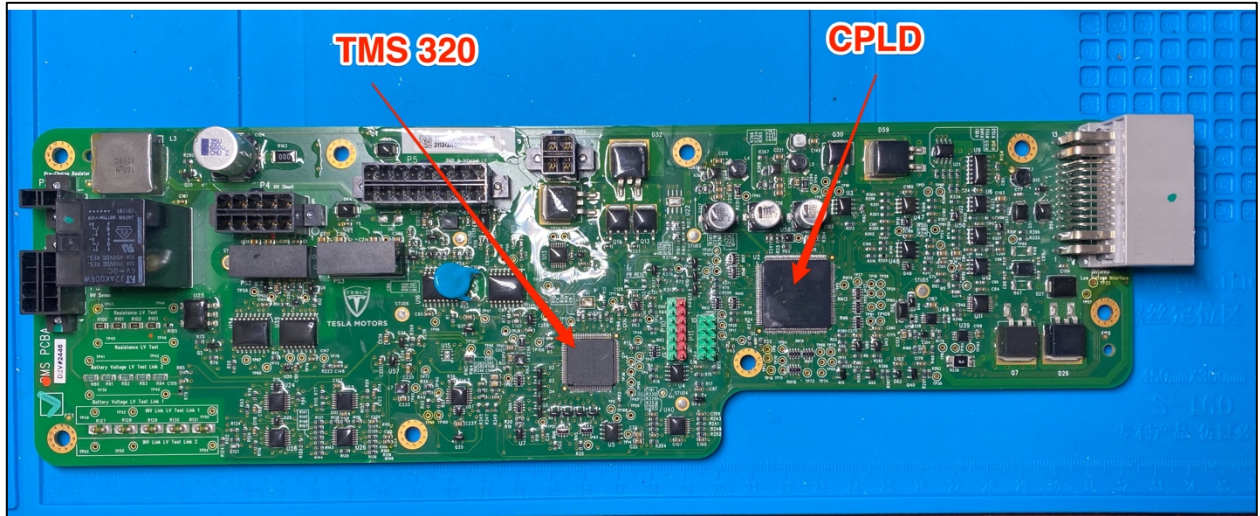


Figure 1: BMS Overview

The BMS contains the following components:

- Texas Instruments DSP TMS320C2809 -- main processor that handles power available, controls the contactors as well as requests from the charger system
- Altera CPLD – Hardware backup for the TMS 320
- High Voltage Contactors – Isolates the high voltage battery from the rest of the vehicle
- Current Shunt with STM8 – Measures current traveling through the battery pack.
- Precharge Resistor – Prevents inrush current damage by bring voltage to the rest of the car up to the level of the battery at a relatively slow rate before both contactors are open
- BMB boards on each of the 16 battery pack modules, measures temperature, voltage and contains bleed resistors to balance the 6 sets of series battery arrays inside each of the 16 modules.

Figure 2 shows all of these components as well as the high voltage sense wires that connect to each side of the contactors:

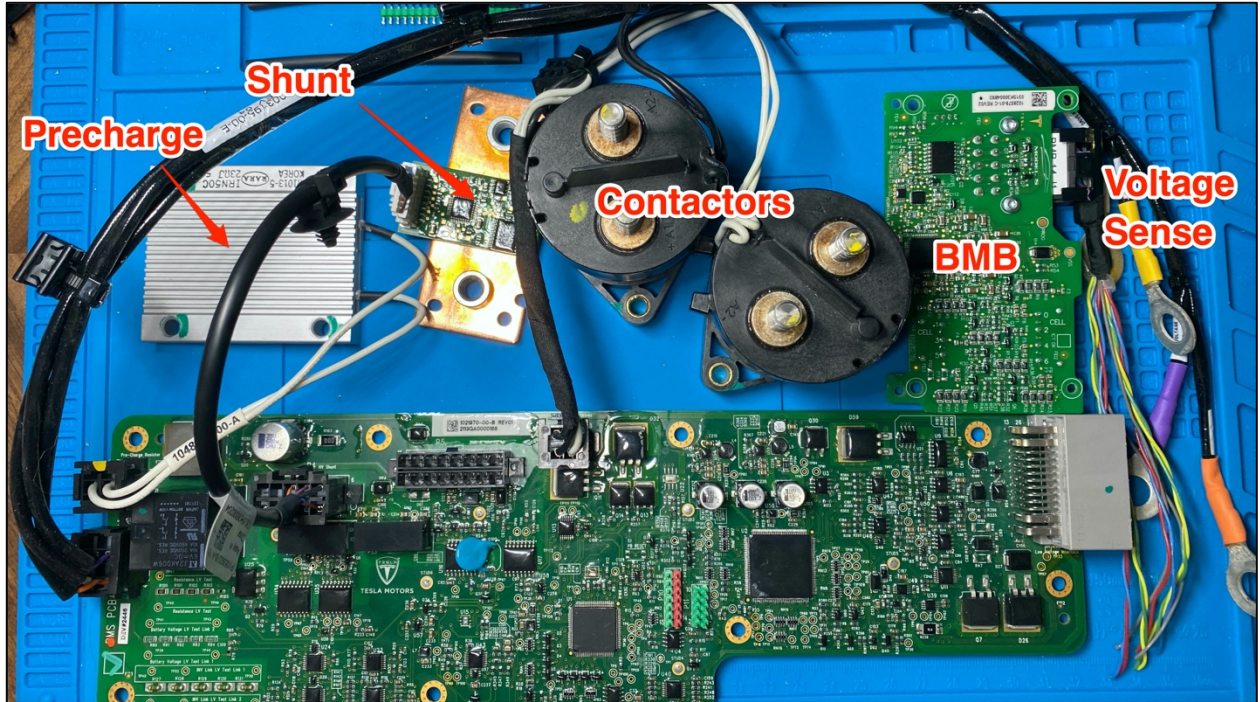


Figure 2: BMS Components

For reference the P85D contains 16 battery modules. Each module contains 444 individual cells contained in arrays with 74 in parallel and 6 in series.

Hardware Upgrades

Public releases and information on forums hinted that the parts involved were high current contactors with Inconel and a Hybrid Pyro fuse that could handle the additional current. These components were acquired on the secondary market

The pyro fuse replaces a traditional melting fuse with one that can be triggered to fire a squib charge that separates two halves of the battery very quickly. Figure 3 shows the two different types of fuses. The conventional fuse is the actual one that came out of donor vehicle.



Figure 3: Tesla Fuses

The contactors are replaced with higher current capable units. The battery pack must be removed from the vehicle in order to replace these components as well as modify the current shunt.

The battery's high voltage lines are not connected directly to ground. The positive and negative leads from the vehicle are isolated by a significant amount of resistance that is checked on a regularly interval by the BMS. When the fuse is removed half of the battery is isolated from the other half, regardless the components still contain significant voltage and can cause serious injury if precautions are not taken when the work is being completed. These components were removed using high voltage gloves and insulated tools.

The battery's contactor bay contains the BMS, current shunt, and contactors. The two contactors each connect to the negative and positive sides of the battery. The BMS also has voltage sense lines that connect to each side of the contactors terminals, therefore the BMS can detect faults in the contactors that would indicate the contactor(s) are not in the state expected by the BMS. Figure 4 shows the contactor bay with the contactors removed:



Figure 4: Contactor Bay

The donor vehicle did not encounter any issues when undergoing this process. The team also tested the battery for shorts and high potential resistance changes using electrical diagnostic equipment specifically designed to do those tests.

Firmware and Configuration Upgrades

Post hardware change, there were configuration and firmware changes made to the vehicle to change the pack configuration to support ludicrous mode power levels.

A configuration file on the gateway called "internal.dat" contains the configuration of the car and also contains a value that enables ludicrous mode called performanceaddon. There is also an entry for "packconfig" that is different on cars with ludicrous mode enabled.

Specific “packconfig” values identify whether or not the battery could support ludicrous power, had an upgrade path to ludicrous or was not capable. The donor vehicle had a pack configuration of “57”, it could be upgraded to ludicrous by changing the firmware to pack configuration “70”.

The diagnostic application Toolbox is a Windows application developed by Tesla for working on their vehicles. The application was written in python and contained plugins that were compiled and then encrypted. This application would function even when disconnected from the Internet. It stands to reason that all the information needed to decrypt the plugins would also be contained on a device running the application.

These plugins were decrypted and then decompiled using the python module “Uncompyle6”. These plugins contained source code comments and other text that would outline the firmware upgrade process as well as a process to recalibrate the current shunt that the vehicle uses to detect how much current is passing through the battery at any moment.

An example plugin is shown in Figure 5:

```
__author__ = 'Otto Chiu'
__email__ = 'ochiu@teslamotors.com'
__copyright__ = 'Copyright Tesla Motors Inc. 2015'
from qxt import QtCore, QtGui
from toolbox.gui import ToolboxView
from vehicle.gui.views.vehicleroutineview import VehicleRoutineView
from vehicle.gui.widgets.signallabel import SignalLabel
from tbx_gen2.addons import Gen2Vehicle
from .flashpage import FlashPage
from .shuntcalibrationpage import ShuntCalibrationPage
DESCRIPTION = QtCore.Qt_TR_NOOP("= BMS Bootloader Update ==\nCertain HV batteries can be upgraded to make them Ludicrous capable. Once the hardware modifications are done, the BMS bootloader needs to be updated to become aware of the modification.\n\nAfter the BMS bootloader is updated, Shunt's gain amplifier needs to be calibrated to\nsupport the change.\n\nAt the end of the update the pack current and alert checks are performed to ensure it works.\n\nYou need to be connected to the battery pack over CAN on the PT bus.\n\n= Before continuing, verify: ==\n*\n*. The contactors were replaced by Ludicrous capable hardware.\n*. The HV battery fuse will be replaced by a hybrid pyro fuse.\n*. The Front DU inline fuse will be replaced by a bus bar.\n*. The vehicle is equipped with a large rear sport drive unit and a small front drive unit.\n")

class BMSBootloaderView(VehicleRoutineView):

    def __init__(self, parent=None):
        super(BMSBootloaderView, self).__init__(parent)
        self._package = None
        self.setWindowTitle(self.tr('BMS Bootloader Update'))
        self.setProperty('description', self.tr(DESCRIPTION))
        layout = self.customWidget().layout()
        bms_hwid = SignalLabel(self)
        bms_hwid.setText(self.tr('Current vehicle BMS HWID: '))
        bms_hwid.setSignalName('BMS_componentHardwareId')
        bms_hwid.setSizePolicy(QtGui.QSizePolicy.Fixed, QtGui.QSizePolicy.Minimum)
        layout.addWidget(bms_hwid, 0, 1)
        spacerItem = QtGui.QSpacerItem(40, 20, QtGui.QSizePolicy.Expanding, QtGui.QSizePolicy.Minimum)
        layout.addItem(spacerItem, 0, 2)
        return

    def initializeUi(self):
        """
        Add option to skip the shunt calibration
        """
        super(BMSBootloaderView, self).initializeUi()
        self.uiSkipShuntCal = QtGui.QCheckBox(self.tr('FOR REMAN ONLY - DO NOT USE IN A SERVICE CENTER. Skipping shunt calibration in a service center might result in a loss of drive event. Reman, check box to skip shunt calibration when a new shunt with dual gain is installed. '))
        custom_layout = self.customWidget().layout()
        custom_layout.addWidget(self.uiSkipShuntCal, 1, 1)
```

Figure 5: Section of Toolbox Python Source

Further examination of these plugins revealed many of the files contained encoded pycside binary data. A quick routine to extract this data from all of the python source code will generate binary files and then Binwalk can be used to extract data from those binary files

Below is a subset of data extracted from these binary files:

1. Firmware files for the BMS, including several application and bootloader files
2. Firmware files for the BMS that appeared to be specifically used for the “shunt calibration” routine
3. A python pickle file that, when converted to a .csv file that contained serial numbers of current shunts along with calibration values and crcs.
4. A full set of CAN Database (DBC) files for each of the three primary CAN buses on the car, these contained definitions and values for all of the signals running on the car’s CAN buses.
5. A set of Open Diagnostic data eXchange (ODX) files that contained a number of UDS routines for the various CAN bus connected modules on the car.
6. Numerous comments that help to understand the ludicrous upgrade process.

Note that the Model S and X have a developer mode that also decodes many of the CAN bus signals, these are stored in libQtCarCANData.so.1.0.0 file within the car’s firmware.

Procuring all of the necessary components was necessary for testing all of the firmware upgrade steps, as well as for calibration of the current shunt. A test bench with a CID, that included the gateway, a BMS with the High Voltage Interlock, the current shunt and a diagnostic connection to allow 3rd party connections to the Powertrain CAN were fabricated to aid in the research. This test bench allowed for testing of the ODX routines as well as validation of many of the steps required to complete this process on the donor vehicle.

The application “Vehicle Spy” by Intrepid Control Systems was used to validate the CAN DBC files and test the UDS ODX routines on the bench and donor vehicle. Previous research by other parties indicated that all Tesla modules use a static seed and key for UDS security access on CAN connected modules, this routine was added to Vehicle Spy and tested to be valid on the version of firmware running on the Bench and Donor Car.

The python extracted firmware was installed on the bench BMS using data extracted from Toolbox and while observing the CAN bus during a software upgrade.

The firmware upgrade process works as follows, the example below upgrades a pack from pack id 57 to 70, relevant to the donor car. Other pack ids are capable of upgrade however they are not covered in this document:

1. Flash the BMS application partition with
“withSecondaryBoot_UDSBoot_BMS_GATEWAY_APP_HWID-57.hex”
2. Read the shunt calibration using UDS routine 22 (Read data by id) Id 401 (Figure 6)
3. Lookup the serial number for the shunt in the calibration pickle file
4. Write the shunt calibration using UDS routine 31 (Routine control) id 503 (Figure 7)
5. Read back the shunt calibration to confirm it changed (not sure if this was the process but I did it to be safe)
6. Flash the bootloader on the BMS and change its value from 57 to 70 using
“withSecondaryBoot_UDSBoot_BMS-R57_CSM_UPDATER_SVN-68454.hex” as updater

code and the flash the actual bootloader with “withSecondaryBoot_UDSBoot_BMS-R70-CSM_SVN-71214.hex” Flash the application partition with the firmware appropriate for the MCU software revision running on the vehicle.

7. Update the gateway “internal.dat” file with the Packconfig 70 (from 57) and add the Performanceaddon 1 line
8. Redeploy software using the CID updater executable or modify the gateway’s firmware.rc file

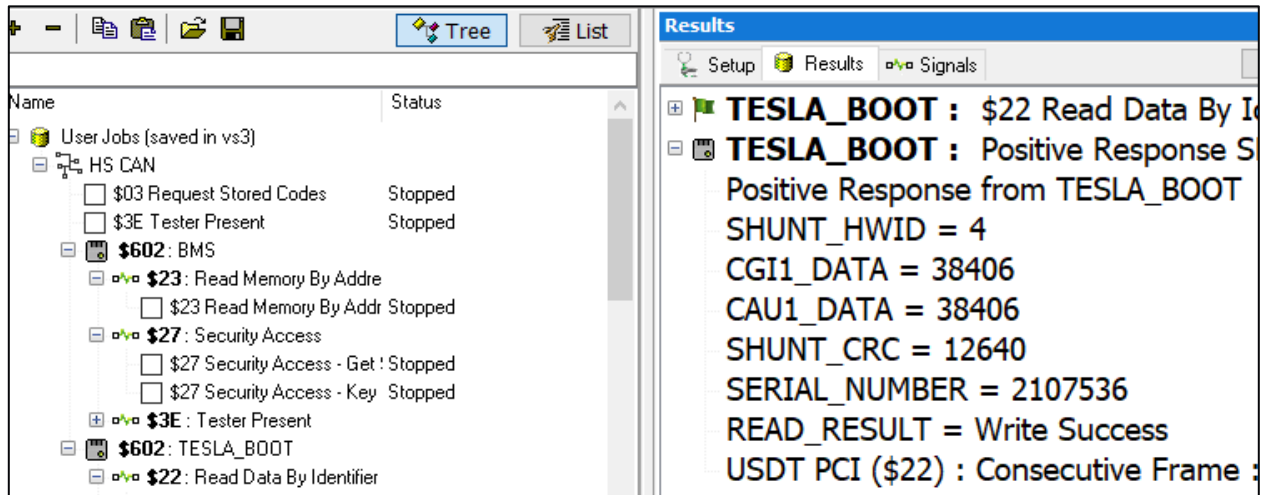


Figure 6: Shunt Read

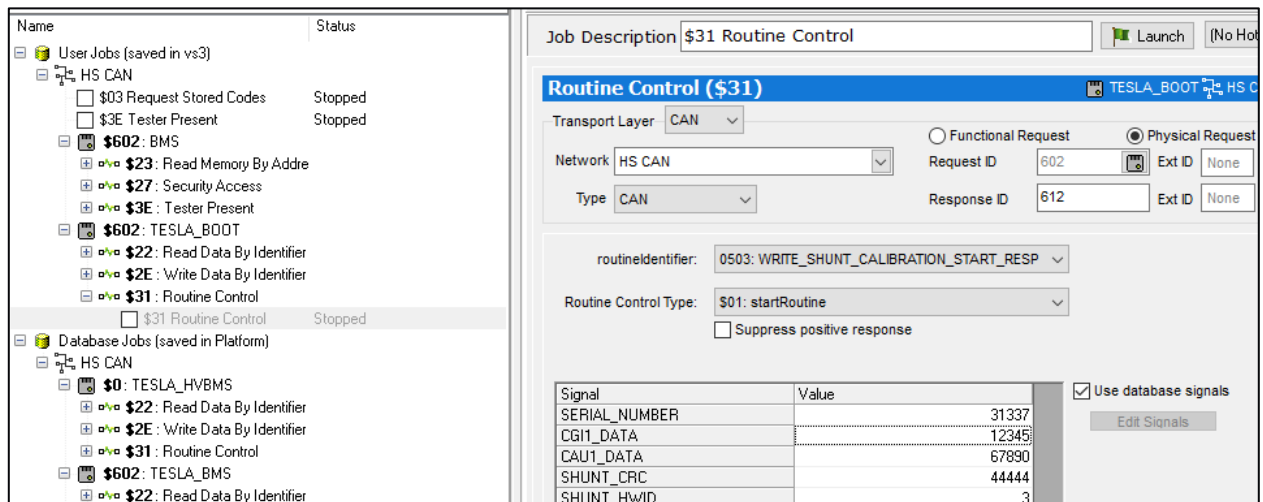


Figure 7: Write Calibration data to Shunt

The correct BMS application firmware was identified by examining the “version_map2.tsv” file within the MCU firmware. In this case the valid firmware file was “uds_BMS_ModelS_R70_CONFIG141.hex” which corresponded to an AWD car with a performance rear motor, as shown in the Figure 8


```

bms:69 bms/142/uds_BMS_ModelS_R70_CONFIG142.hex bms.hex bms 50cc7c1b 4wd=1, rearDriveUnitType=1
bms:70 bms/140/uds_BMS_ModelS_R70_CONFIG140.hex bms.hex bms 5af9e748 4wd=0, rearDriveUnitType=0
bms:70 bms/141/uds_BMS_ModelS_R70_CONFIG141.hex bms.hex bms fa07311c 4wd=1, rearDriveUnitType=0
bms:70 bms/142/uds_BMS_ModelS_R70_CONFIG142.hex bms.hex bms 0103dc7b 4wd=1, rearDriveUnitType=1

```

Figure 8: version_map2.tsv file

During bench testing, an error was identified after the BMS was upgraded from 57 to 70 with the shunt connected. The BMS generated the error BMS_w140_SW_OverCurrent_Sense after the BMS was configured to version 70, but not before when the BMS was version 57.

The logic between the shunt and BMS was analyzed by constructing a breakout board for the original current shunt and it was determined the post upgrade error cleared when a single shunt wire was disconnected from the original shunt that came with the BMS.

When this wire was disconnected before the upgrade process was the BMS generated a separate error. This indicated the Hardware and Software upgrades had to happen at the same time.

A conversation with a former Tesla tech confirmed a wire needed to be cut on the shunt during the upgrade process. Further analysis indicated the shunt wire that was disconnected in fact connected to the CPLD, as this device was a hardware backup to the TMS320 it made sense that the CPLD would need a way to detect an overcurrent condition.

The rear seat was removed from the vehicle and the fuse leading to the front drive unit was replaced with a bus bar, per the notes in the Python code.

After the upgrade the car indicated a higher current was available, as shown in Figure 9:

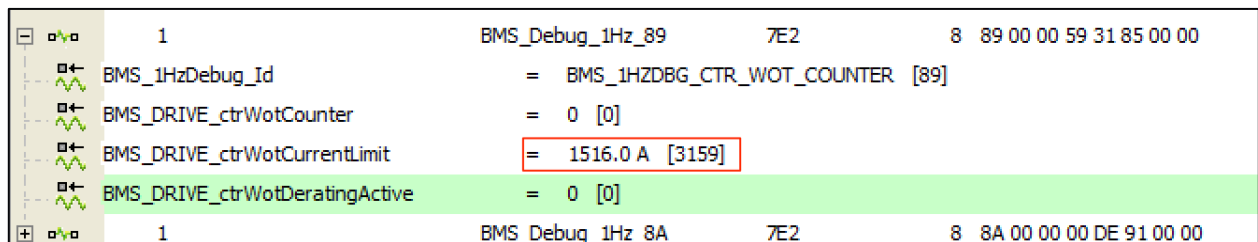


Figure 9: New Current Limits

For reference the current limits prior to the upgrade are shown in Figure 10:

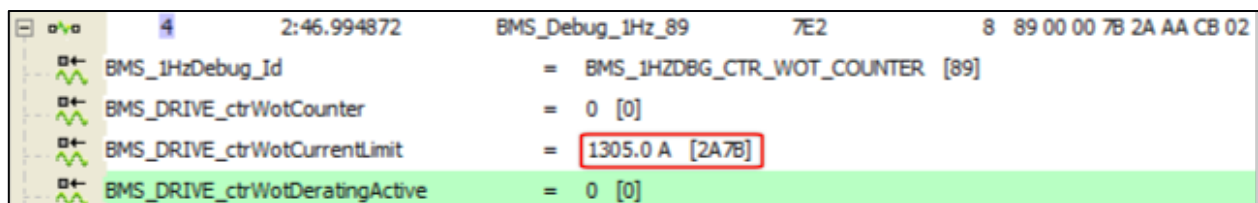
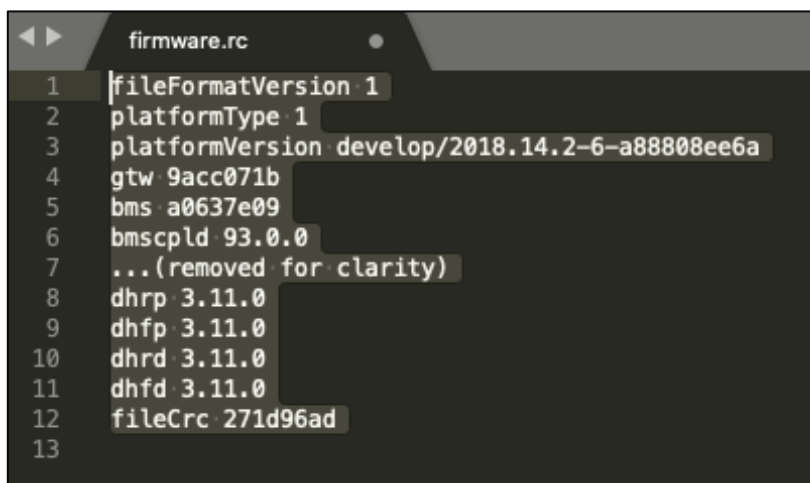


Figure 10: Pre-upgrade Current Limits

The Donor Car would not engage into drive and the gateway indicated a BMS mismatch, this error was expected as it was generated on the bench during testing. The error was supposed to clear with either a firmware redeploy or firmware upgrade, however both failed.

After analysis of the logs, an error regarding “firmware.rc” was seen. The only reference I could find to this file was in a previous security talk where the file was used by the security gateway. Similar to how the “internal.dat” configuration file was requested. This file was then downloaded from the security gateway and examined. A snippet is shown in Figure 11:



```
firmware.rc
1 fileFormatVersion 1
2 platformType 1
3 platformVersion develop/2018.14.2-6-a88808ee6a
4 gtw 9acc071b
5 bms a0637e09
6 bmscpld 93.0.0
7 ...(removed for clarity)
8 dhrp 3.11.0
9 dhfp 3.11.0
10 dhrd 3.11.0
11 dhfd 3.11.0
12 fileCrc 271d96ad
13
```

Figure 11: Section of firmware.rc

Note the 4-byte values for bms and filecrc. The value for the bms corresponded to a packid 57 firmware from the “version_map2.tsv” file and the file crc value was a crc32/jamcrc value of the entire file, minus the crc line. The firmware.rc file bms line was modified, the crc value recalculated and this file was uploaded to the gateway. Once the gateway was rebooted the errors cleared and the car could be put into drive and driven with the additional power from ludicrous.