DIR-825L启动项分析

解压固件

binwalk解压后得知是个Squashfs文件系统

分析固件

先用firmwalk和trommel分析一下固件

```
`sudo ./firmwalker.sh
/home/iot/Desktop/_DIR825B1_FW210NAb02.bin.extracted/squashfs-root >
~/Desktop/_DIR825B1_FW210NAb02.bin.extracted/firmwalk.txt`

`sudo python3 trommel.py -p
~/Desktop/_DIR825B1_FW210NAb02.bin.extracted/squashfs-root/ -o result_trommel
-d ~/Desktop/_DIR825B1_FW210NAb02.bin.extracted/`
```

从firmwalk.txt看出来，很多关键字在/sbin/httpd和/bin/busybox两个二进制文件中高频出现

查看架构

针对busybox二进制文件查看

```
iot@research:~/Desktop/_DIR825B1_FW210NAb02.bin.extracted/squashfs-root/bin$ file busybox
busybox: ELF 32-bit MSB executable, MIPS, MIPS32 version 1 (SYSV), dynamically linked, interpreter /lib/ld-uClibc.so.0, stripped
```

得知DIR825是一个32bit的MIPS架构，且为大端序，动态链接编译的

分析启动项

/etc/rc.d/rcS

```
# 挂载文件系统
mount -a
mount -o remount +w /
mount -t tmpfs tmpfs /tmp -o size=256k
mount -t usbfs none /proc/==bus==/usb
```

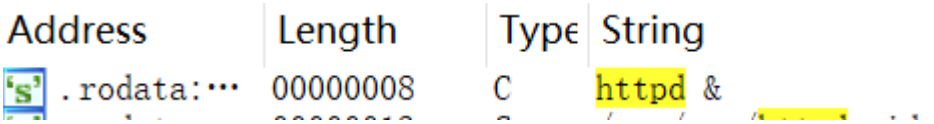
```
# 添加环境变量
export PATH=$PATH:/etc/ath

# 启动系统初始化脚本
rc init &
```

这个三个都是关于网络的配置

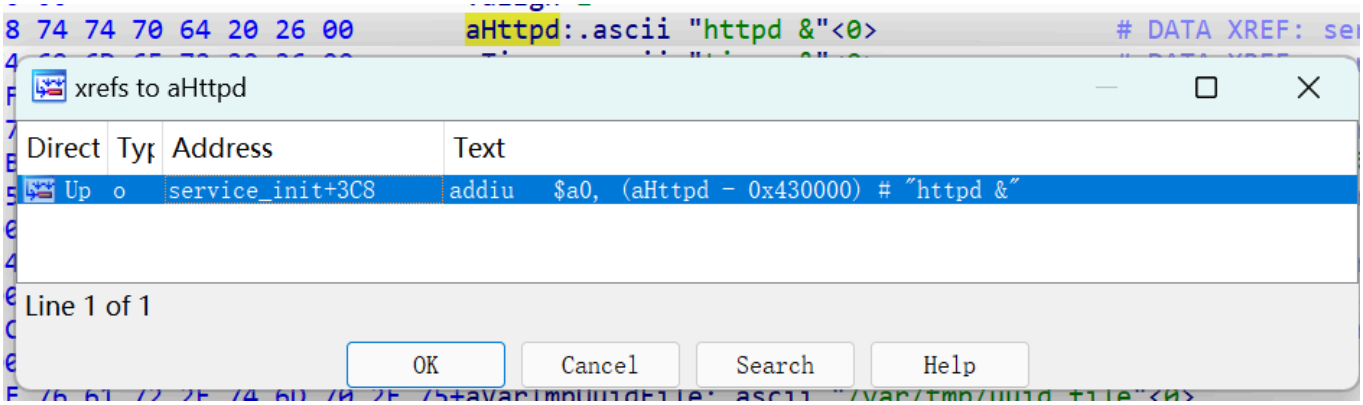
找到rc这个二进制文件`find . -name "rc"`在/sbin/rc
checksec一下发现没什么保护

```
iot@research:~/Desktop/_DIR825B1_FW210NAb02.bin.extracted/squashfs-root/sbin$ checksec --file=rc
RELRO           STACK CANARY      NX            PIE             RPATH      RUNPATH      Symbols         FORTIFY Fortified       For
tifiable        FILE
No RELRO        No canary found   NX disabled   No PIE          No RPATH   No RUNPATH   No Symbols      No      0               13r
c
```

用IDA打开这个rc二进制文件，因为之前提到过httpd重叠率较高，于是查了一下string=httpd



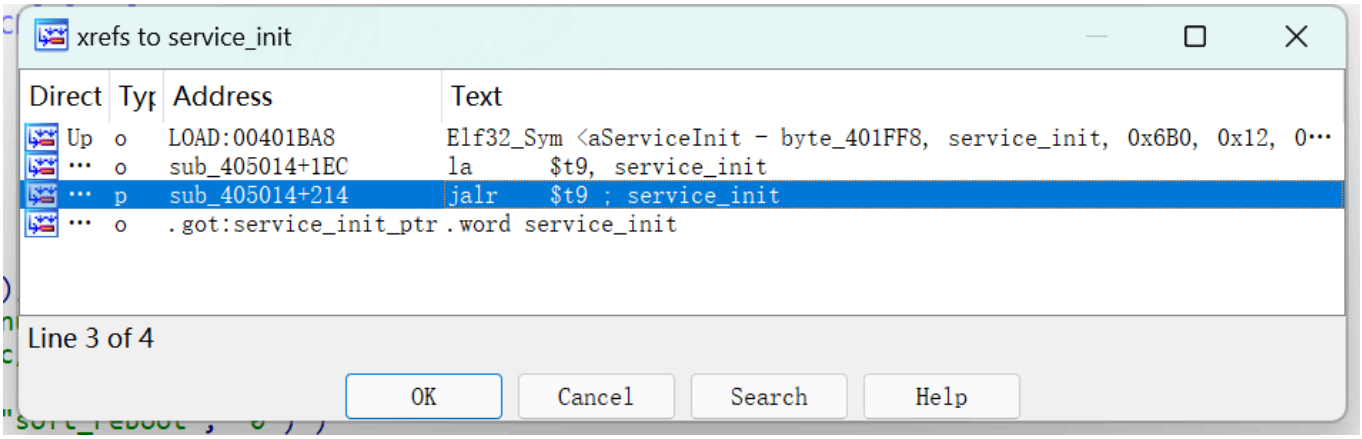
点进去看一下交叉引用



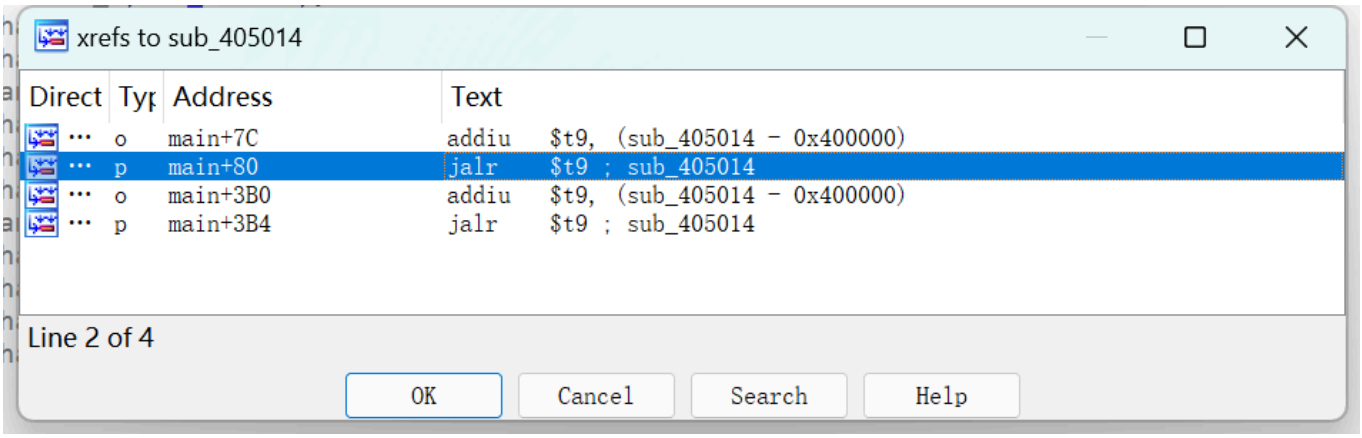
发现是在service_init()这个函数里执行了httpd，启动了http服务

```
102   system("httpd &");
```

追溯一下从哪里调用了service_init()这个函数




```
41  service_init();
```



读取 `/var/run/httpd.pid` 文件中的进程 ID，并向该进程发送 `SIGCHLD` 信号（信号编号 17）。

```
pid = read_pid("/var/run/httpd.pid");
if (pid != -1)
kill(pid, 17);
```

sub_405014()为调用的初始化函数

```
26  if ( strstr(v7, "init") )
27  {
28    sub_405014();
29    return 0;
30  }
```

最终追溯到main函数中，并没看到传了什么参数进去

再返回到service_init()这个函数查看它具体都做了些什么
里面的一些函数功能：

`mount_fs` 函数这个函数主要用于在文件系统中创建一组特定的目录，每个目录都有其特定的用途，主要是为了确保系统在启动或运行时所需的目录结构已经存在，并设置这些目录的权限以确

保所有用户都可以访问这些目录。

```
int mount_fs()
{
  mkdir("/tmp/var", 0x1FFu);
  mkdir("/var/log", 0x1FFu);
  mkdir("/var/run", 0x1FFu);
  mkdir("/var/tmp", 0x1FFu);
  mkdir("/var/misc", 0x1FFu);
  mkdir("/var/etc", 0x1FFu);
  mkdir("/var/etc/ppp", 0x1FFu);
  mkdir("/var/etc/ppp/peers", 0x1FFu);
  mkdir("/var/lock", 0x1FFu);
  mkdir("/var/etc/peers", 0x1FFu);
  mkdir("/var/sbin", 0x1FFu);
  mkdir("/var/firm", 0x1FFu);
  return 0;
}
```

`create_pid` 函数 的作用是创建一个文件 `/var/run/rc.pid`，并将当前进程的 ID 写入该文件。此文件通常用于记录正在运行的进程的 PID，以便于系统和其他程序跟踪和管理该进程。通过这种方式，可以确保在需要的时候，系统或其他进程能够知道该进程的 PID 并进行相应的操作（例如终止、重新启动等）。

```
int create_pid()
{
  FILE *v0; // $s0
  __pid_t v1; // $v0

  v0 = fopen("/var/run/rc.pid", "w");
  if ( v0 )
  {
    v1 = getpid();
    fprintf(v0, "%lu\n", v1);
    fclose(v0);
  }
  return 0;
}
```

`create_symlink` 的作用是在 `/var/sbin` 目录下创建一系列符号链接，这些链接都指向 `/sbin/rc`。这些链接可以使不同的命令（例如 `ip-up`，`ip-down`，`monitor` 等）都执行相同

的可执行文件 `/sbin/rc`，通过不同的链接名来区分不同的功能或行为。函数最终返回创建 `/var/sbin/fwupgrade` 链接的结果。如果成功，返回 `0`；如果失败，返回 `-1`。这种设计方式在需要集中管理和执行多个不同命令的系统中非常有用，因为它允许通过一个单一的可执行文件来处理多个不同的命令调用。

```
int create_symlink()
{
  symlink("/sbin/rc", "/var/sbin/ip-up");
  symlink("/sbin/rc", "/var/sbin/ip-down");
  symlink("/sbin/rc", "/var/sbin/monitor");
  symlink("/sbin/rc", "/var/sbin/redial");
  symlink("/sbin/rc", "/var/sbin/mon6");
  symlink("/sbin/rc", "/var/sbin/red6");
  symlink("/sbin/rc", "/var/sbin/psmon");
  symlink("/sbin/rc", "/var/sbin/wantimer");
  symlink("/sbin/rc", "/var/sbin/lanmon");
  return symlink("/sbin/rc", "/var/sbin/fwupgrade");
}
```

```
/tmp/sbin # ls -la
drwxr-xr-x    2 root     root           240 Jul 21  2024 .
drwxrwxrwt   11 root     root           220 Jul 21  2024 ..
lrwxrwxrwx    1 root     root             8 Jul 21  2024 fwupgrade -> /sbin/rc
lrwxrwxrwx    1 root     root             8 Jul 21  2024 ip-down -> /sbin/rc
lrwxrwxrwx    1 root     root             8 Jul 21  2024 ip-up -> /sbin/rc
lrwxrwxrwx    1 root     root             8 Jul 21  2024 lanmon -> /sbin/rc
lrwxrwxrwx    1 root     root             8 Jul 21  2024 mon6 -> /sbin/rc
lrwxrwxrwx    1 root     root             8 Jul 21  2024 monitor -> /sbin/rc
lrwxrwxrwx    1 root     root             8 Jul 21  2024 psmon -> /sbin/rc
lrwxrwxrwx    1 root     root             8 Jul 21  2024 red6 -> /sbin/rc
lrwxrwxrwx    1 root     root             8 Jul 21  2024 redial -> /sbin/rc
lrwxrwxrwx    1 root     root             8 Jul 21  2024 wantimer -> /sbin/rc
```

初始化网络和运行脚本

```
init_network();
system("./etc/rc.d/pre_customer.sh");
system("tftpd &");
flash_get_checksum();
```

- 初始化网络。
- 运行预定义的客户脚本。
- 启动 TFTP 服务器。

- 获取闪存校验和。

初始化服务

```
LABEL_10:
  v6 = 3;
  system("klogd &");
  system("insmod %s");
  system("/sbin/gpio SYSTEM check &");
  system("httpd &");
  sleep(1u);
  system("timer &");
  init_file("/var/log/FW_log");
  syslog(6, "[Initialized, firmware version: %s ] \n", "2.10NA");
  system("dcc &");
  nvram_safe_get("lan_bridge");
  system("lld2d %s &");
  check_wlan0_domain(1);
  init_file("/var/tmp/uuid_file");
```

- 启动 `klogd`。
- 加载内核模块。
- 检查 GPIO 系统。
- 启动 HTTP 服务器。
- 启动计时器。
- 初始化日志文件。
- 记录系统日志，标明固件版本。
- 启动 `dcc`。
- 获取并配置 LAN 桥接。
- 启动 LLDP 守护进程。
- 检查 WLAN0 域。
- 初始化 UUID 文件。

危险函数

主要分析一下httpd这个重叠率较高的二进制文件！！！
(1)通过IDA搜索各种危险函数名称，通过交叉引用一个一个慢慢看静态，感觉有问题就动态调着试一试。（用插件VulFi跑一下httpd的危险函数）

function.txt

eg:Format String sprintf sub_436CEC 0x436e34 Not Checked High

具体描述的是在代码的某个子程序（`sub_436CEC`）中，调用了 `sprintf` 函数，并指出了这个问题在某个特定地址（`0x436e34`）上，但没有进行安全检查（`Not Checked`），且该问题的风险等级为高（`High`）。

- **Format String**: 格式化字符串的安全问题。这种漏洞可能导致代码执行或者信息泄露。
- **sprintf**: 这是一个 C 语言函数，用于格式化字符串。它有潜在的格式化字符串漏洞，如果使用不当，可能导致安全问题。
- **sub_436CEC**: 这是一个函数或者子程序的名称，通常是通过反汇编工具分析二进制文件时生成的标识符。
- **0x436e34**: 这是一个内存地址，表明在代码的这个地址上出现了问题。
- **Not Checked**: 表示在这个位置上的 `sprintf` 调用没有进行安全检查。
- **High**: 表示这个安全问题的风险等级是高。
  总结起来，这句话的意思是：在内存地址 `0x436e34` 处，函数 `sub_436CEC` 中调用了 `sprintf` 函数，但没有进行安全检查，这可能是一个高风险的安全漏洞。

(2)通过FACT或EMBArk分析后的最危险的危险函数去逐个分析。