从后门到漏洞——智能设备私有协议中的安全问题

魏凡 绿盟科技 格物实验室 安全研究员

目录

目录

智能设备中的私有协议

私有协议一般来说指未文档化的协议，格式未知，一般具有以下特点：

- 从功能上来讲，一般用于运维管理/服务发现等功能
- 一般监听于TCP/UDP的大端口
- 多数作为服务端，部分可以在厂商官网找到协议的客户端

私有协议的安全问题

私有协议的格式未知，所以逆向分析较难，因此公开的漏洞较少，但一出现安全问题，一般影响都较为严重。

ASUS Router infosvr UDP Broadcast root Command Execution

Several models of ASUS's routers include a service called *infosvr* that listens on UDP broadcast port 9999 on the LAN or WLAN interface. It's used by one of ASUS's tools to ease router configuration by automatically locating routers on the local subnet. This service runs with *root* privileges and contains an unauthenticated command execution vulnerability. The source code for this service, as well as the rest of the router, is available from ASUS's Support Site.

Netcore Router Udp 53413 Backdoor

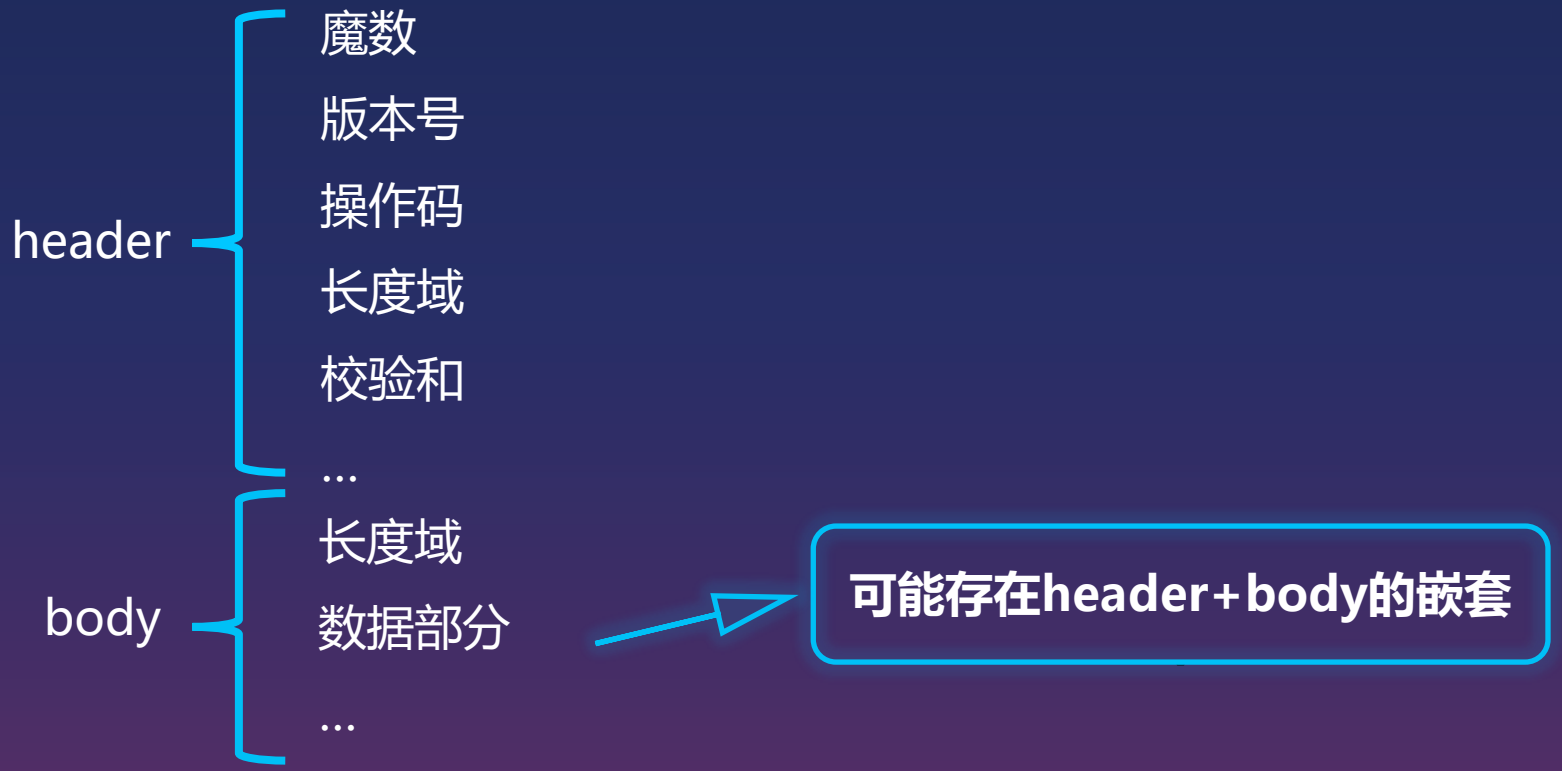
Disclosed	Created
08/25/2014	05/30/2018

Description

Routers manufactured by Netcore, a popular brand for networking equipment in China, have a wide-open backdoor that can be fairly easily exploited by attackers. These products are also sold under the Netis brand name outside of China. This backdoor allows cyber criminals to easily run arbitrary code on these routers, rendering it vulnerable as a security device. Some models include a non-standard echo command which doesn't honor -e, and are therefore not currently exploitable with Metasploit. See URLs or module markdown for additional options.

目录

私有协议的一般格式



协议分析——“入口点”

协议“入口点”（网络通信函数）：

recv/send，recvfrom/sendto, recvmsg/sendmsg

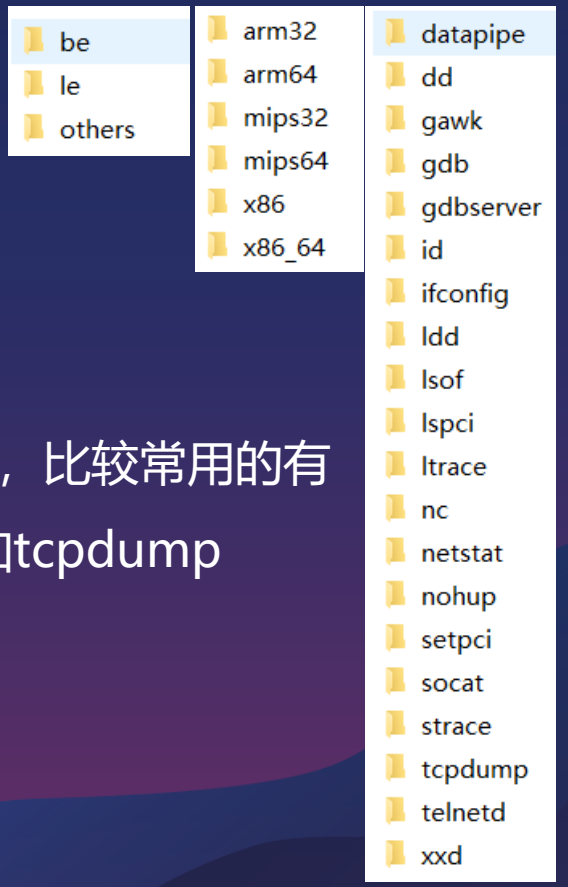
还有一些容易忽略的地方：

- 有些采用ssl连接，ssl_read/ssl_write
- recv也可用于接收udp的数据包，recvfrom也可用于接收tcp数据包
- 有时候通信函数在应用的**调用库**中

调试与调试权限获取

“拿到设备的调试权限，整个漏洞挖掘工作就成功了一半”

——来自某不知名研究员



获取调试权限的一般思路：

- 利用设备自带的TELNET/SSH功能
- 利用设备硬件调试接口
- **修改设备固件增加调试后门**
- **利用设备的已知漏洞**
- 利用设备的未知漏洞

准备静态编译的小工具，比较常用的有busybox, gdbserver和tcpdump

目录

本议题所提到的漏洞均已送报厂商并已修复

某厂商智能设备私有协议分析

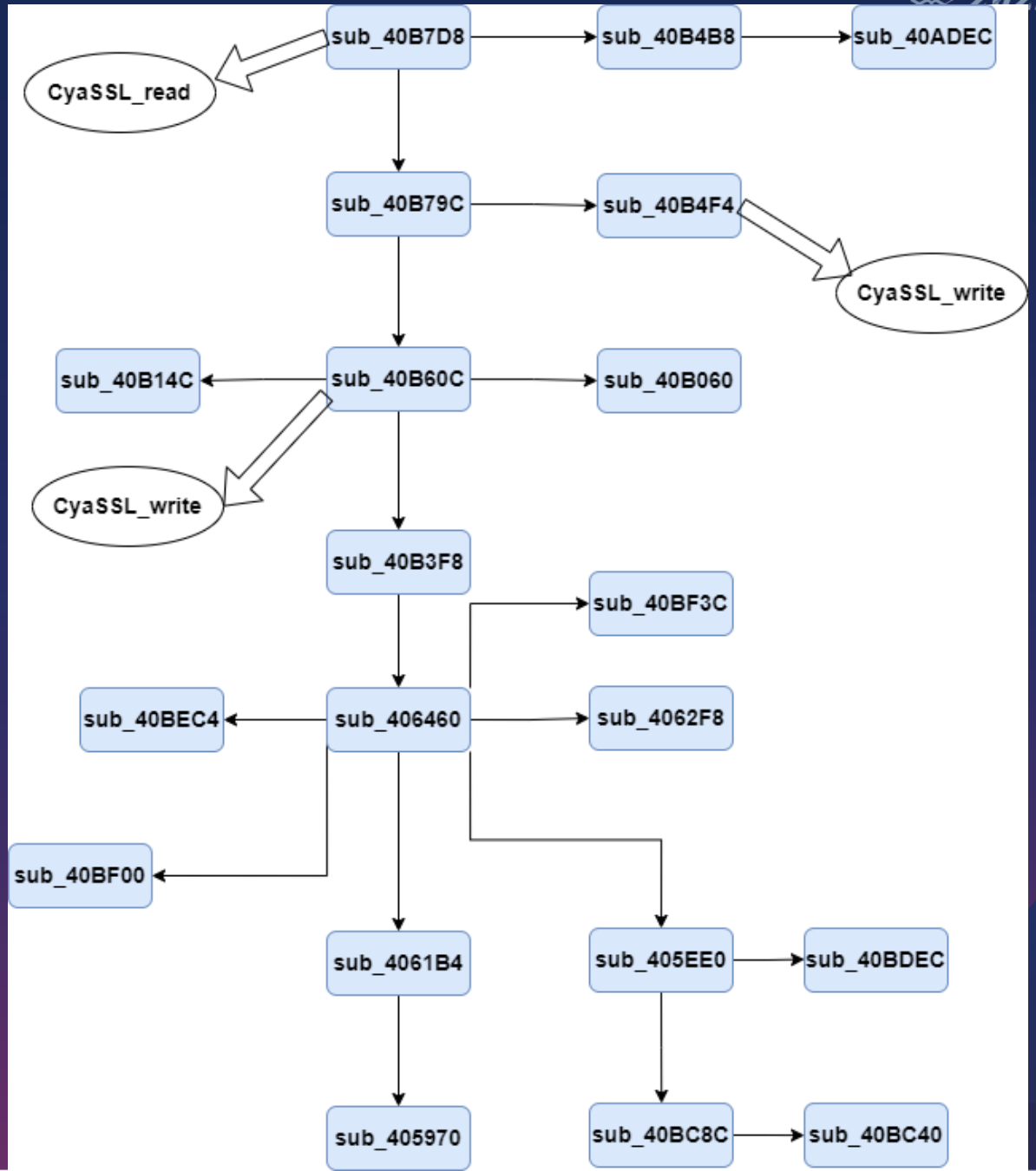
一个偶然的机会，我们拿到了一个某厂商生产的智能设备，其某个TCP端口使用了一个私有协议，该协议未文档化，官网也未提供任何管理客户端。

- CyaSSL_read
- 单向认证

```
LOAD:0040B810                lw      $a0, 0x20($s0)    # ssl
LOAD:0040B814                move    $a1, $s2          # data
LOAD:0040B818                jal     CyaSSL_read
LOAD:0040B81C                addiu   $a2, $v0, -1      # size
```

常规漏洞挖掘思路

- 找到协议“入口点”
- 跟踪数据流向
- 逆向分析



非常规的私有协议漏洞挖掘思路

主观考虑智能设备的常见协议：UPNP协议——接口的命令注入

MQTT协议——未授权访问

协议“脆弱点”——容易出现漏洞的地方

非常规思路：以发现漏洞为导向，暂时不考虑格式，先了解其功能，然后对其“脆弱点”进行重点突破。

协议提供的功能

reboot

```
D:004067DC loc_4067DC:                                  # CODE XREF: sub_406460+128↑j
D:004067DC                                              # DATA XREF: LOAD:jpt_406588↓o
D:004067DC                li      $a0, aReboot          # jumptable 00406588 case 5
D:004067E4
D:004067E4 loc_4067E4:                                  # CODE XREF: sub_406460+374↑j
D:004067E4                jal     system
D:004067E8                nop
```

restorefactory

```
D:004067B4 loc_4067B4:                                  # CODE XREF: sub_406460+128↑j
D:004067B4                                              # DATA XREF: LOAD:jpt_406588↓o
D:004067B4                lui     $a0, 0x41  # 'A'      # jumptable 00406588 case 2
D:004067B8                jal     system
D:004067BC                li      $a0, aRestorefactory  # "restorefactory"
D:004067C0                jal     sleep
D:004067C4                li      $a0, 1
D:004067C8                j       loc_4067EC
D:004067CC                nop
```

协议认证逻辑

取某个全局变量作为初始字符串，将该字符串进过三次处理，最后和用户传入的一个串进行比较。

```
LOAD:00406224    li      $s2, unk_41E870  # 引用全局变量作为初始字符串
LOAD:00406228    move    $s0, $v0
LOAD:0040622C    move    $a0, $s2
LOAD:00406230    move    $a1, $zero
LOAD:00406234    jal     memset
LOAD:00406238    li      $a2, 0x400
LOAD:0040623C    lbu     $v0, 0($s0)
LOAD:00406240    beqz    $v0, loc_4062D0
LOAD:00406244    move    $a0, $s0
LOAD:00406248    li      $a1, 0x10
LOAD:0040624C    jal     sub_405D98       # 第一次处理
```

```
LOAD:00406274    addiu   $a1, $sp, 0x4C+var_34
LOAD:00406278    jal     sub_405970       # 第二次处理
LOAD:0040627C    addiu   $s1, -2
LOAD:00406280    sltiu   $v0, $s1, 0x401
                 beqz    $v0, loc_4062D0
                 addiu   $s0, $s4, 2
                 move    $a0, $s2
                 move    $a1, $s0
                 jal     memcpy
                 move    $a2, $s1
                 move    $a0, $s2
                 addiu   $a1, $sp, 0x4C+var_24
                 jal     memcpy
                 li      $a2, 0x10
                 addiu   $a0, $sp, 0x4C+var_14
                 move    $a1, $s2
LOAD:004062B4    jal     sub_405970       # 第三次处理
LOAD:004062B8    move    $a2, $s1
LOAD:004062BC    addiu   $a0, $sp, 0x4C+var_14
LOAD:004062C0    move    $a1, $s0
LOAD:004062C4    jal     memcmp           # 和用户传入的串进行比较
```

第一次处理，和两个特殊字符串进行移位混淆；后续两次处理都是标准md5运算。

```
OAD:00405E10                move    $a0, $s2
OAD:00405E14                slti    $v1, $v0, 0xF
OAD:00405E18                lui     $a0, 0x41  # 'A'
OAD:00405E1C                lui     $a2, 0x41  # 'A'
OAD:00405E20                movz    $s4, $v0, $v1
OAD:00405E24                li      $a0, a
OAD:00405E28                move    $v1, $zero
                            # CODE XREF: s
                            # sub_405B6C+5
                            li      $a1, 0xFF
                            j       loc_405E90
                            li      $a2, a
```

```
OAD:00405970 sub_405970:
OAD:00405970
OAD:00405970
OAD:00405970 var_68          = -0x68
OAD:00405970 var_s0          =  0
OAD:00405970 var_s4          =  4
OAD:00405970 var_s8          =  8
OAD:00405970 var_sC          =  0xC
OAD:00405970
OAD:00405970                addiu   $sp, -0x90
OAD:00405974                sw      $s2, 0x80+var_s8($sp)
OAD:00405978                sw      $s1, 0x80+var_s4($sp)
OAD:0040597C                sw      $s0, 0x80+var_s0($sp)
OAD:00405980                move    $s2, $a0
OAD:00405984                move    $s0, $a1
OAD:00405988                move    $s1, $a2
OAD:0040598C                sw      $ra, 0x80+var_sC($sp)
OAD:00405990                jal     sub_405718      # md5_init
OAD:00405994                addiu   $a0, $sp, 0x80+var_68
OAD:00405998                move    $a2, $s1
OAD:0040599C                addiu   $a0, $sp, 0x80+var_68
OAD:004059A0                jal     sub_405754      # md5_update
OAD:004059A4                move    $a1, $s0
OAD:004059A8                move    $a0, $s2
OAD:004059AC                jal     sub_405858      # md5_final
```

通过md5算法中的4个链接变量来识别

```
OAD:00405718 sub_405718:
OAD:00405718
OAD:00405718                li      $v0, 0x67452301
OAD:00405720                sw      $v0, 8($a0)
OAD:00405724                li      $v0, 0xEFCDAB89
OAD:0040572C                sw      $v0, 0xC($a0)
OAD:00405730                li      $v0, 0x98BADCFE
OAD:00405738                sw      $v0, 0x10($a0)
OAD:0040573C                li      $v0, 0x10325476
OAD:00405744                sw      $zero, 4($a0)
OAD:00405748                sw      $zero, 0($a0)
OAD:0040574C                jr      $ra
OAD:00405750                sw      $v0, 0x14($a0)
```

协议认证逻辑分析

假设作为初始字符串的全局变量为“admin”，将利用如下算法进行处理：

（1）取“admin”字符串，和两个固定字符串一起参与移位混淆操作，得到新字符串；

（2）将第一次变换得到的新字符串进行标准md5运算，得到hash值

“\x21\x23\x2f\x29\x7a\x57\xa5\xa7\x43\x89\x4a\x0e\x4a\x80\x1f\xc3”；

（3）将第二次得到的hash值再次进行标准md5运算，得到hash值

“\x43\x44\x26\x76\xc7\x4a\xe5\x9f\x21\x9c\x2d\x87\xfd\x6b\xad\x52”

将最后得到的hash值与用户传入进行比较，若相等则认证成功。

初始字符串的赋值

在调用认证函数之前，有个函数将会给该全局变量赋值。

```
LOAD:004064B8                 move    $a2, $v0          # unk_41F3D0
LOAD:004064BC                 move    $a0, $zero
LOAD:004064C0                 jal     sub_409C1C        # 给该全局变量赋值
LOAD:004064C4                 li      $a1, 0x200
```

是有趣的是，这里将固定字符串“admin”拷贝到全局变量处。

```
LOAD:00409C84 loc_409C84:                               # CODE XREF: sub_409C1C+60↑j
LOAD:00409C84                                           # DATA XREF: LOAD:jpt_409C7C
LOAD:00409C84                 lui     $a1, 0x41  # 'A'  # jumptable 00409C7C case 0
LOAD:00409C88                 move    $a0, $s0
LOAD:00409C8C                 j       loc_409CDC
LOAD:00409C90                 li      $a1, aAdmin       # "admin"
```

运维“后门”

传入固定字符串“admin”经过三次变换后的凭证

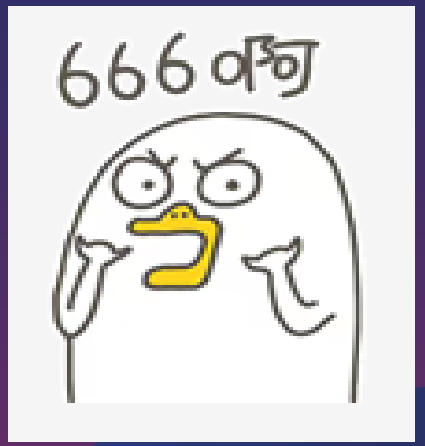
“\x43\x44\x26\x76\xc7\x4a\xe5\x9f\x21\x9c\x2d\x87\xfd\x6b\xad\x52”

即可完成认证过程，控制设备重启/恢复出厂设置。

用户：我忘了设备的管理密码了，怎么办?

厂商：IP告诉我一下，我给你远程重置一下。

用户：666啊!



协议的专利

专利所有权厂商的第一类设备

```
.text:0005258E                BLX             memset
.text:00052592                ADD             R1, SP, #0x1478+haystack
.text:00052594                LDR             R0, =aIfconfigWlan0U ; "ifconfig wlan0 up"
.text:00052596                BL              sub_51478
.text:0005259A                LDR             R0, =aTmpBaseFilesEt_1 ; "/tmp/base-files/etc/get_efuse_devid"
.text:0005259C                MOVS            R1, #0  ; oflag
```

```
.text:000F4396                LDR             R0, =aDevSlpFlashChr ; "/dev/slp_flash_chrdev"
.text:000F4398                MOVS            R3, #3
.text:000F439A                STR             R1, [SP,#0x20+var_1C]
.text:000F439C                STR             R3, [SP,#0x20+var_20]
.text:000F439E                STR             R1, [SP,#0x20+var_18]
.text:000F43A0                BLX             open
.text:000F43A4                ADDS            R3, R0, #1
.text:000F43A6                MOV             R4, R0
.text:000F43A8                BEQ             loc_F43CC
.text:000F43AA                MOVW            R1, #0xDF07 ; request
.text:000F43AE                MOV             R2, SP
.text:000F43B0                BLX             ioctl
```

```
.text:00052B66 loc_52B66                              ; CODE XREF: sub_52A08+AA↑j
.text:00052B66                LDR             R0, =aRebootF ; jumptable 00052AB2 case 6
.text:00052B68                BLX             system
```

```
.text:00051FD4                LDR             R2, =aTftpG192168112 ; "tftp -g [redacted] -r %s"
.text:00051FD6                MOV.W           R1, #0x100 ; maxlen
.text:00051FDA                MOV             R3, R5
```

功能：

- 修改配置
- 重启
- 恢复出厂设置
- 固件更新

初始字符串的获取

从配置文件中读取web管理员密码作为原始字符串。

```
t:0004F4EC          BLX             memset
t:0004F4F0          LDR             R0, =unk_15E89B ; /user_management/root
t:0004F4F2          ADD             R1, SP, #0x1A8+s
t:0004F4F4          MOV.W           R2, #0x15C
t:0004F4F8          BL              read_config
```

md5_digest(output, input, len)

运算长度len不能为0，如果为0，直接返回

```
:0004EC28 md5_digest                              ; CODE XREF: sub_
:0004EC28                                         ; sub_4EC68+57A↓p
:0004EC28
:0004EC28 s               = -0x6C
:0004EC28
:0004EC28                 PUSH            {R4-R6,LR}
:0004EC2A                 MOV             R4, R0
:0004EC2C                 SUB             SP, SP, #0x60
:0004EC2E                 MOV             R5, R2
:0004EC30                 MOV             R6, R1
:0004EC32                 CBZ             R1, loc_4EC60
:0004EC34                 CBZ             R0, loc_4EC60
:0004EC36                 CMP             R2, #0  ; 长度参数不能为0
:0004EC38                 BLE             loc_4EC60
:0004EC3A                 MOVS            R1, #0  ; c
:0004EC3C                 MOVS            R2, #0x5C ; '\' ; n
:0004EC3E                 ADD             R0, SP, #0x70+s ; s
:0004EC40                 BLX             memset
:0004EC44                 ADD             R0, SP, #0x70+s
:0004EC46                 BL              InitMd5
:0004EC4A                 ADD             R0, SP, #0x70+s ; int
:0004EC4C                 MOV             R1, R6  ; src
:0004EC4E                 MOV             R2, R5  ; n
:0004EC50                 BL              Md5Update
:0004EC54                 ADD             R0, SP, #0x70+s
:0004EC56                 MOV             R1, R4
:0004EC58                 BL              Md5Final
```

认证绕过

- 传入md5_digest函数的长度参数为用户控制，可设置为0
- 没有验证md5_digest函数的返回值，直接将结果用于memcmp进行比较
- 参与memcmp的第一个参数被初始化为空
- 控制长度参数为0+凭据为空值（16个\x00）的请求包，即可绕过认证

```
.text:0004F530        SUBS        R7, #2  ; 用户指定的长度减2
.text:0004F532        MOV         R0, R5  ; dest
.text:0004F534        MOV         R1, R4  ; src
.text:0004F536        MOV         R2, R7  ; n
.text:0004F538        BLX         memcpy
.text:0004F53C        LDM.W       R6, {R0-R3}
.text:0004F540        STM.W       R5, {R0-R3}
.text:0004F544        MOV         R1, R5
.text:0004F546        MOV         R2, R7  ; 传入的长度为用户控制
.text:0004F548        MOV         R0, R8
.text:0004F54A        BL          sub_4EC28 ; 调用md5相关函数进行运算
.text:0004F54E        MOV         R0, R8  ; s1
.text:0004F550        MOV         R1, R4  ; s2
.text:0004F552        MOVS        R2, #0x10 ; n
.text:0004F554        BLX         memcmp  ; memcmp的第一个参数为空值
```

未授权RCE

固件更新处的命令注入，tftp命令中的“文件名”参数为用户传入：

```
.text:00051FD4          LDR             R2, =aTftpG192168112 ; "tftp -g [redacted] -r %s"
.text:00051FD6          MOV.W           R1, #0x100 ; maxlen
.text:00051FDA          MOV             R3, R5
.text:00051FDC          ADD             R0, SP, #0xF80+var_D28 ; s
.text:00051FDE          BLX             snprintf
.text:00051FE2          ADD             R1, SP, #0xF80+var_E28
.text:00051FE4          ADD             R0, SP, #0xF80+var_D28
.text:00051FE6          BL              sub_51478 ; popen
```

结合前面的认证绕过，达到未授权执行任意命令。

专利所有权厂商的第二类设备

```
LOAD:00404F48    li      $s2, 0xFFFFFFFF
LOAD:00404F4C    jal     uci_alloc_context
LOAD:00404F50    nop
LOAD:00404F54    beqz    $v0, loc_404FDC
LOAD:00404F58    move    $s1, $v0
LOAD:00404F5C    move    $a1, $zero
LOAD:00404F60    li      $a2, 0x100
LOAD:00404F64    jal     memset
LOAD:00404F68    addiu   $a0, $sp, 0x144+var_104
LOAD:00404F6C    jal     strlen
LOAD:00404F70    move    $a0, $s0
LOAD:00404F74    move    $a2, $v0
LOAD:00404F78    addiu   $a0, $sp, 0x144+var_104
LOAD:00404F7C    jal     strncpy
LOAD:00404F80    move    $a1, $s0
LOAD:00404F84    move    $a0, $s1
LOAD:00404F88    addiu   $a1, $sp, 0x144+var_12C
LOAD:00404F8C    addiu   $a2, $sp, 0x144+var_104
LOAD:00404F90    jal     uci_lookup_ptr
LOAD:00404F94    li      $a3, 1
```

- 功能上和第一类设备类似
- UCI读取WEB管理员密码作为初始字符串

认证绕过

- 传入的长度为用户控制，可以为0
- md5_digest算法中未验证传入的长度参数
- 对于标准md5算法来说，当传入md5_update函数的长度参数为0时，无论传入的原始字符串为何值，生成的hash永远为固定的串“d41d8cd98f00b204e9800998ecf8427e”

```
LOAD:00402238    addiu   $s1, -2          # 参与md5运算长度为用户传入-2
LOAD:0040223C    sltiu   $v0, $s1, 0x401
LOAD:00402240    beqz    $v0, loc_402290
LOAD:00402244    addiu   $s0, $s4, 2
LOAD:00402248    move    $a0, $s2
LOAD:0040224C    move    $a1, $s0
LOAD:00402250    jal     memcpy
LOAD:00402254    move    $a2, $s1
LOAD:00402258    move    $a0, $s2
LOAD:0040225C    addiu   $a1, $sp, 0x38+var_20
LOAD:00402260    jal     memcpy
LOAD:00402264    li      $a2, 0x10
LOAD:00402268    addiu   $a0, $sp, 0x38+var_10
LOAD:0040226C    move    $a1, $s2
LOAD:00402270    jal     md5_make_digest
LOAD:00402274    move    $a2, $s1         # 若用户传入的长度为2，则参与md5运算的长度为0
```

专利所有权厂商的第三类设备

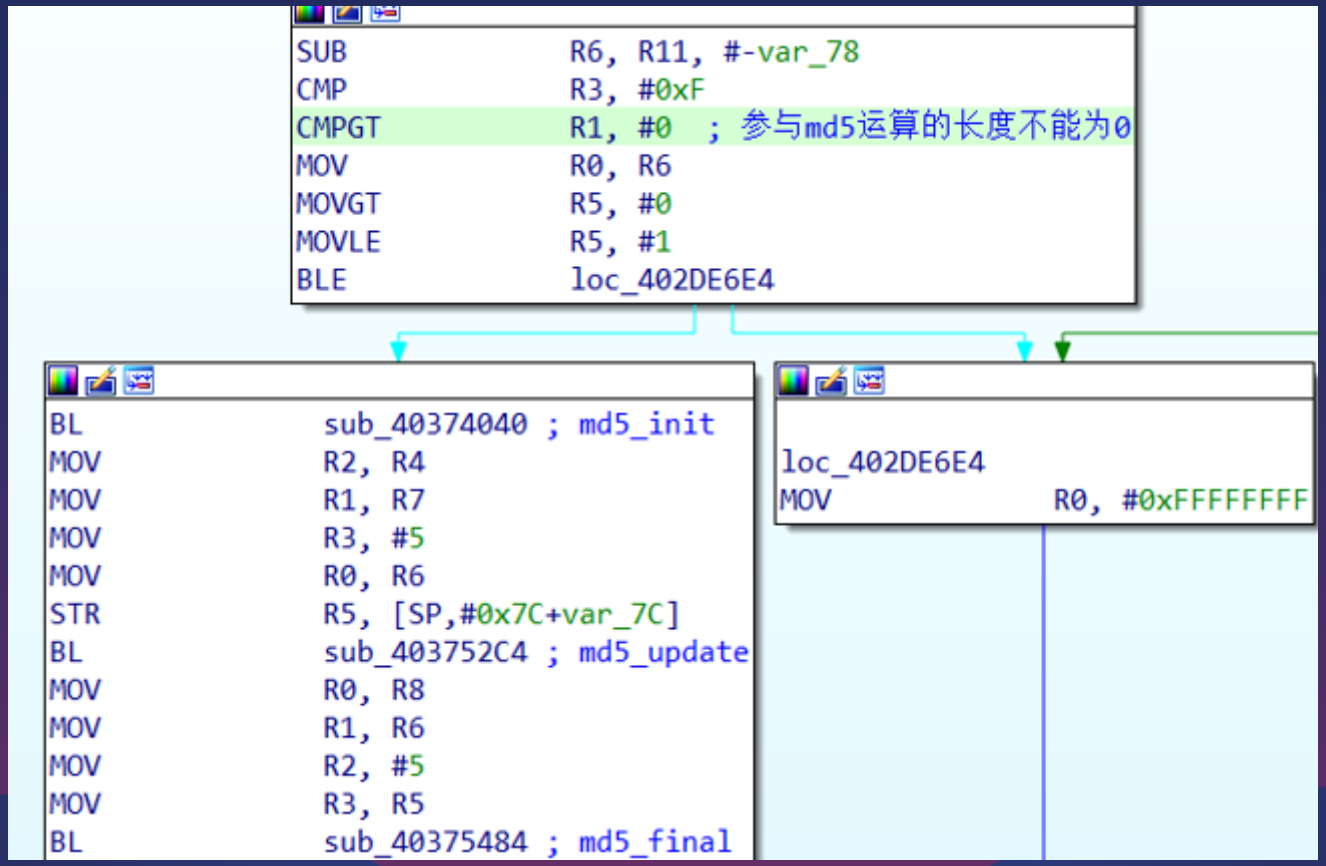
还是按照之前的认证算法进行运算，第二次参与md5运算的长度可控。

```
ROM:402DF8C8    BL      strlen
ROM:402DF8CC    MOV     R2, R8
ROM:402DF8D0    MOV     R3, #0x10
ROM:402DF8D4    MOV     R1, R0
ROM:402DF8D8    MOV     R0, R6
ROM:402DF8DC    BL      md5_digest ; 第一次参与md5运算的长度是用strlen取的
ROM:402DF8E0    CMP     R0, #0
```

```
ROM:402DF924    MOV     R3, #0x10
ROM:402DF928    BL      md5_digest ; 第二次参与md5运算的长度为用户控制
ROM:402DF92C    CMP     R0, #0
ROM:402DF930    BNE     loc_402DF94C
ROM:402DF934    MOV     R0, R10
ROM:402DF938    MOV     R1, R6
ROM:402DF93C    MOV     R2, #0x10
ROM:402DF940    BL      memcmp
```

md5_digest函数对长度的限制

在md5_digest函数中，对参与md5运算的长度做了判断，不能为0



认证逻辑绕过

- 考虑把参与md5_digest运算的长度设为1，则只有一个字节参与md5运算；
- 参与运算的值为0x00-0xff，生成的hash值也有256种情况；
- 协议并未限制最大尝试次数，可无限次发送认证请求。

```
fp = open("cer.txt", "w")
for i in range(0, 256):
    e = chr(i)
    m1 = hashlib.md5()
    m1.update(e.encode())
    print(m1.hexdigest())
    fp.write(m1.hexdigest())
    fp.write("\n")
```

生成“密码”字典

```
93b885adfe0da089cdf634904fd59f71
55a54008ad1ba589aa210d2629c1df41
9e688c58a5487b8eaf69c9e1005ad0bf
8666683506aacd900bbd5a74ac4edf68
ec7f7e7bb43742ce868145f71d37b53c
8bb6c17838643f9691cc6a4de6c51709
06eca1b437c7904cc3ce6546c8110110
89e74e640b8c46257a29de0616794d5d
e2ba905bf306f46faca223d3cb20e2cf
5e732a1878be2342dbfeff5fe3ca5aa3
68b329da9893e34099c7d8ad5cb9c940
13c8ffd977013703a701cf8e11deac65
58c89562f58fd276f592420068db8c09
dcb9be2f604e5df91deb9659bed4748d
4dedb2240a1e0f038dcdc8b3de92264c
d838691e5d4ad06879ca721442e883d4
6b31bdfa7f9bfece263381ffa91bd6a9
47ed733b8d10be225eceba344d533586
a8445619abd08f3ba0ebfcb31183f7f9
ffe51d3e7d8297237588704eeddc6ab2
```

认证绕过实现


构造数据包
控制长度为1
遍历256个md5 hash
发送到设备端
根据返回值判断是否认证成功
实现任意控制设备

小结

同一厂商生产的不同类型的设备，针对私有协议的实现上，往往会出现相似的“脆弱点”。

md5_digest函数：

- 不验证返回值
- 不验证传入长度
- 验证了，好像又没有验证

目录

总结与建议

总结——警惕私有协议“供应链”漏洞

后续的研究中，我们发现存在其他厂商也在使用该协议，和传统的出现在SDK上的供应链漏洞相比，这些漏洞更加“隐蔽”和难以修复。

给厂商和开发人员的建议——协议设计和实现上尽量用**白名单**的思想。

谢 谢