如何提取D-Link解密密钥

原创

IT老涵 于 2022-01-05 21:12:31 发布 437 收藏

分类专栏： 安全 黑客 网络 文章标签： 网络安全 安全 计算机网络

如何提取D-Link解密密钥

当我们在分析固件镜像时，遇到的最常见障碍之一便是加密。虽然已经有一些方法可用于解密固件镜像，但今天我们仍将简要介绍一下，如何提取D-Link部分路由器型号中的加密密钥，特别是D-Link DIR-X1560。该设备与 D-Link DIR-X5460是同一代路由器的一部分，最近，针对该设备，德国流行技术杂志Chip和IoT Inspector联合对其进行了Wi-Fi路由器相关的安全检查。

一、D-Link路由器固件加密

D-Link倾向于用自定义的固件更新文件格式，来加密他们的固件镜像。许多DIR系列的D-Link路由器，使用带有SHRS头的固件更新文件格式：

```
00000000  53 48 52 53 01 13 91 5D 01 13 91 60 67 C6 69 73 SHRS‘]‘`gÆis
```

这种固件格式及加密方案已经被公开记录。一位名叫0xricksanchez 的研究者，发布了一篇非常棒的writeup，记录了如何发现SHRS固件镜像（包含DIR-3060，我们已经发布了一篇相关advisory）中密钥的过程。他们从imgdecrypt二进制文件中提取出了加密密钥和初始向量IV，此二进制文件可以通过类似系列型号中的URAT shell获得。

然后，最近我们遇到了DIR-X系列的路由器，它们的固件头有一点不同。

```
00000000  65 6e 63 72 70 74 65 64  5f 69 6d 67 02 0a 00 14  |encrpted_img....|
```

头部就是一个encrpted_img字符串，然后紧跟其后的是镜像大小字段（32位大端模式）。

SHRS固件镜像的key不适用于这些encrpted_img 镜像，因此我们需要找到另一种方法，将解密密钥从这种加密格式的设备中提取出来。

二、密钥在哪里？

显然，我们无法从加密的固件镜像中提取加密密钥，因为它是加密的。所以，我们需要找到另一种方法来获取密钥key。

如果我们能够在设备上以某种方式执行代码，那么就很简单了。如有足够的本地权限，我们可以在设备运行时，访问设备上的所有内容。这是解密后的固件镜像。

如果设备制造商最近才引入固件加密，则可以追踪尚未加密的老版本固件镜像（很可能是引入加密之前的固件版本），并检查是否可以从中提取出密钥key。

直接读取设备的flash内存，是我们可以采用的另一种技术。在flash中固件不太可能被加密。拆开其中一个设备，卸掉flash内存芯片，转储内存后再读取文件系统。但是这样破坏性太太了，设备很昂贵，一旦损坏将大大浪费。

在这里，我们不会深入探讨这些问题，因为其他人已经做过了。关于这个问题Zero Day Initiative有一篇详细的文章，当你在解决固件加密时，可以考虑这个方法。

三、Shell中的操作

在我们的例子中，通过UART调试接口可以轻松得到DIR-X1560设备的shell。获取到交互式shell后，就可以轻松dump出整个文件系统。使用内置的busybox tar和nc命令可以很容易地做到这一点。首先在设备上设置监听器：

```
nc -nvlp [PORT] > filesystem.tar.gz
```

然后在设备上运行以下指令，只写明想要传出的根目录即可。

```
tar -cvz /bin/ /data/ /etc/ /etc_ro/ /lib/ /libexec/ /mnt/ /opt/ /sbin/ /usr/ /var/ /webs/ | nc [IP ADDRESS] [PO
RT]
```

最终，将得到tar.gz压缩文件，包含文件系统中任何你想要的部分，然后通过网络传输出来。

四、发现在何处解密

不同于“SHRS”固件镜像，没有明显的imgdecrypt二进制文件可供关注。因此我们可以跟踪固件上传过程，看是否可以定位到在哪里解密。

幸运的是，固件头部的字符串，可以作为独特的关键字，帮助我们在文件系统中找到相关程序。

```
$ grep -r encrpted_img
Binary file bin/fota matches
Binary file bin/httpd matches
Binary file bin/prog.cgi matches
```

在此我们找到了两个二进制程序 prog.cgi 和fota，二者可能以某种方式来处理加密过的固件镜像。

在progl.cgi中，根据字符串encrypted_img，我们可以很容易的追踪到固件上传的地方。

```
        iVar7 = imageLen + 1;
        validLen = uploadSize + 1;
        *local_d8 = (char)iVar4;
        puVar1 = local_d8 + 1;
        if (iVar7 == 0x10) {
          iVar4 = memcmp(local_d8 + -0xf,"encrpted_img",0xc);
          if (iVar4 == 0) {
            validLen = *(uint *)(local_d8 + -3);
            img_size = img_size - 0x10;
            validLen_shift =
                 validLen << 0x18 | (validLen >> 8 & 0xff) << 0x10 | (validLen >> 0x10 & 0xff) << 8
                 | validLen >> 0x18;
            validLen = uploadSize - 0xf;
            DAT_000f9b30 = 1;
            iVar7 = imageLen + -0xf;
```

```
        log_log(7,"Dlink_uploadImage",0x28e,"found encrypted image!");
        puVar1 = local_d8 + -0xf;
      }
      else {
        if (DAT_000f9b30 == 0) {
          log_log(3,"Dlink_uploadImage",0x292,"not found ecrypted image!");
          memset(read_buf,0,0x2000);
          DAT_000fa054 = 0;
          DAT_000fa050 = 0;
          if (*param_2 != 0) {
            cmsMem_free();
```


通过跟随固件被加密部分的指针变量，发现函数FUN00033144被调用，此指针是其第一个参数。

```
    log_log(7,"Dlink_uploadImage",0x2c5,"image uploadSize=%d, imageLen=%d, validLen=%d",uploadSize
            ,imageLen,validLen);
    if (param_1 == 2) {
      uVar6 = 0;
    }
    else {
      uVar6 = param_4 & 1;
    }
    uVar2 = local_cc;
    if ((uVar6 != 0) &&
       (uVar6 = local_cc | uploadBufLen & 0xff, uploadBufLen = uVar6, uVar2 = uVar6, uVar6 != 0))
    {
      uploadBufLen = FUN_000330a8(uVar3,img_size - imageLen,0);
      uVar6 = FUN_00033144(&pimg_data,validLen,uploadBufLen,imageLen);
      if (uVar6 == 0) {
        if (uploadBufLen != 0) {
          uVar3 = uploadBufLen;
        }
        local_d8 = pimg_data;
        validLen = uVar6;
        uploadSize = uVar6;
        uploadBufLen = uVar6;
        uVar2 = local_cc;
      }
      else {
        if (uVar6 == 0x232c) {
```


在这个函数内部，出现一个极有可能是固件解密的函数： gj_decode()

```
int FUN_00033144(void *pimg_data,uint param_2,int param_3,int param_4)

{
  int iVar1;
  uint uVar2;
  int iVar3;
  uint uVar4;
  void *decrypted;
  size_t local_24;

  iVar1 = DAT_000f9b30;
```

```
if ((DAT_0001f930 != 0) && (iVar1 = ValidLen_shift, ValidLen_shift != 0)) {
  decrypted = (void *)0x0;
  local_24 = 0;
                    /* WARNING: Load size is inaccurate */
  iVar1 = gj_decode(*pimg_data,offset,&decrypted,&local_24);
                    /* WARNING: Load size is inaccurate */
  memcpy(*pimg_data,decrypted,local_24);
  if (offset < param_2) {
```


五、深入库文件

gj_decode()函数被定义在libcmd_util.so文件中，此函数代码不多，但关键的是，其内部调用了两个加密相关的函数：aes_set_key 和aes_cbc_decrypt。这两个函数也被定义在上述库文件中，但我们没必要深究这二者。因为它们的函数名已经给予了很多信息：像是CBC模式下的AES加密。

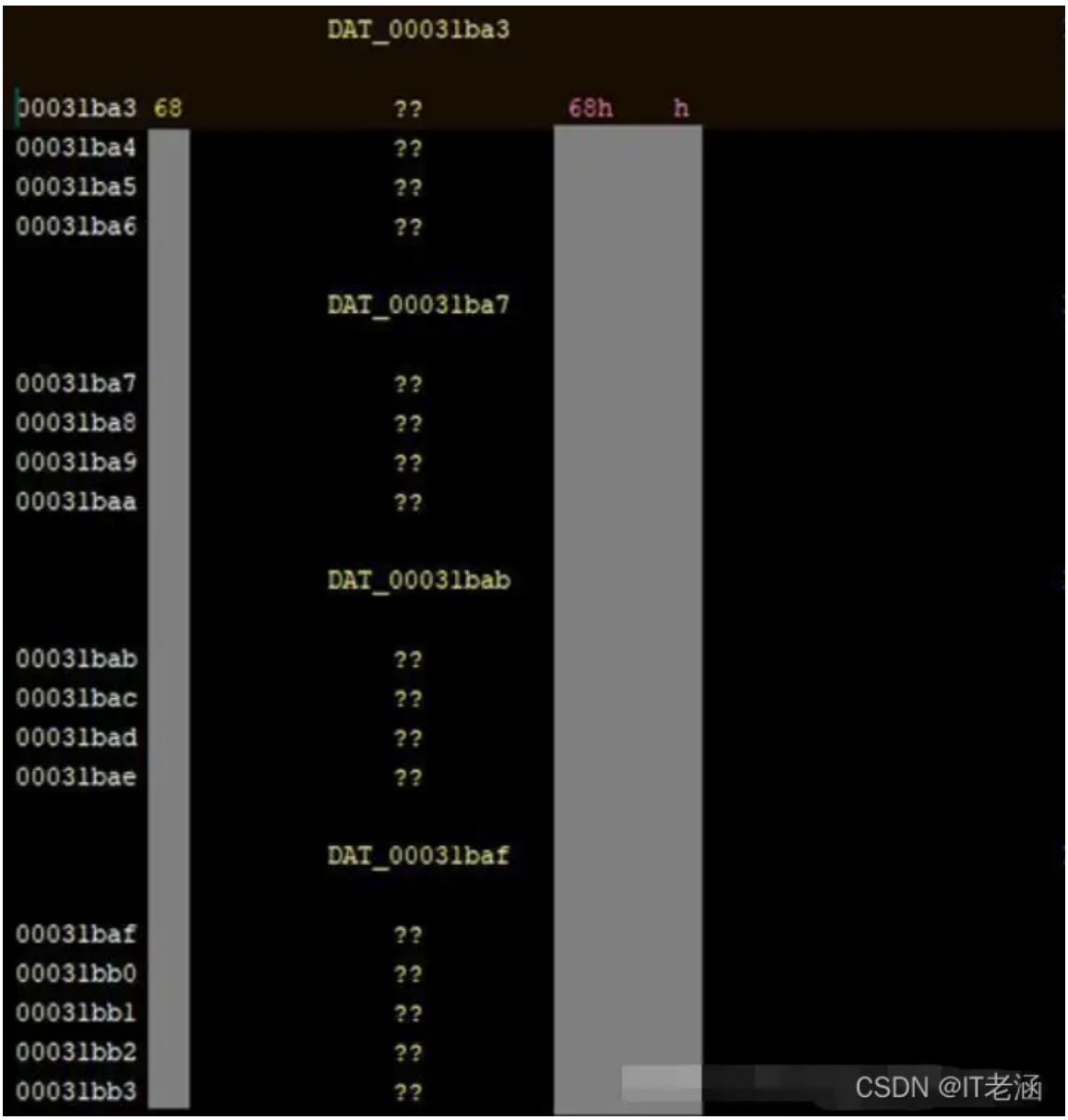
我们可以看到，aes_set_key()的第二个参数可能是AES密钥，而 aes_cbc_decrypt 的第二个参数可能是初始向量IV。

```
memcpy(pfirmware,fw_source,fw_len);
                    /* +4 is the key location. */
key_loc__ = key_loc + 4;
pfirst_dword = (undefined4 *)&DAT_00031ba3;
do {
  pnext_dword = pfirst_dword + 2;
  byte_1 = pfirst_dword[1];
  *key_loc__ = *pfirst_dword;
  key_loc__[1] = byte_1;
  key_loc__ = key_loc__ + 2;
  pfirst_dword = pnext_dword;
} while (pnext_dword != (undefined4 *)&UNK_00031bc3);
                    /* At 0 is the IV location. */
key_loc__ = key_loc;
pfirst_dword = (undefined4 *)&DAT_00031bc5;
do {
  pnext_dword = pfirst_dword + 2;
  byte_1 = pfirst_dword[1];
  *key_loc__ = *pfirst_dword;
  key_loc__[1] = byte_1;
  key_loc__ = key_loc__ + 2;
  pfirst_dword = pnext_dword;
} while (pnext_dword != (undefined4 *)&DAT_00031bd5);
aes_set_key(aes_ctx,key_loc + 4,0x100);
aes_cbc_decrypt(aes_ctx,key_loc,pfirmware,pfirmware,fw_len);
putchar(0x21);
byte_1 = 0;
}
```


这里的反编译代码有点乱，但不要太在意细节，直接看代码逻辑。有两个do-while循环，将数据从全局变量复制到本地缓冲区，这些循环在全局地址的字节上迭代，直到到达某个特定的结束地址。

key_loc 和key_loc + 4是指向本地缓冲区的指针变量，第一个循环中，00031ba3地址处的数据拷贝到key_loc+4直到00031bc3地址，而第二个循环，00031bc5地址处的数据拷贝到key_loc直到00031bd5地址（还注意到：这两个本地指针分别是aes_cbc_decrypt()和aes_set_key()的第二个参数）

的确如此，在地址00031ba3处，可见有一个字节数组。



同样在00031bc5地址处，也可见类似的形式。

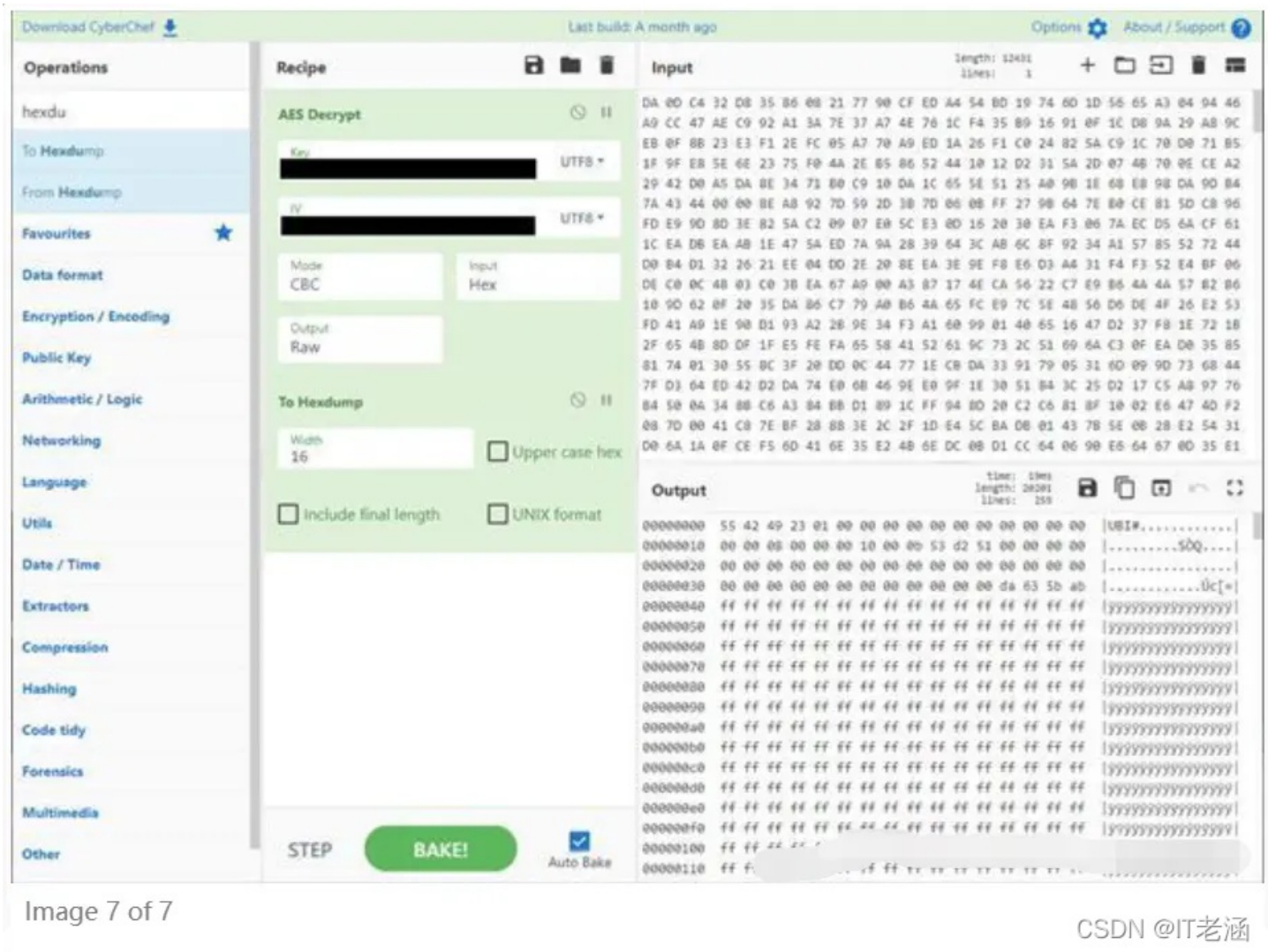
因此，我们可以假设AES密钥位于 00031ba3，而初始向量IV位于 00031bc5。

今天我们在这不会公布真实的密钥，但是那些关注且跟随者应该有能力将密钥提取出来，如他们而言，这是留给有兴趣读者的练习。

六、节省时间和脑力

通过使用CyberChef，我们可以快速、轻松地验证假设。大多数情况下，当我想要快速解决密码学相关问题时，我都会打开这个工具。它由英国GCHQ编写，如果对公务员写的东西持怀疑态度，大可不必，它只是用纯JavaScript编写，可以在你想要的任何浏览器中运行。

复制加密固件镜像中的第一个0x1000左右字节的数据，作为16进制的ASCII粘贴到CyberChef输入中，再结合我们提取的密钥和IV，通过CyberChef的“AES Decrypt”操作，就可以得到一个理想的结果。



在这里我们可以看到UBI数据块的头部数据，说明我们正在完全的解密整个固件更新包。

一旦我们知道密钥和IV是有效的，那么很容易就可以编写一个完整的解密脚本。在这种情况下，在我们得到一个完整且正确的镜像之前，还有几个小的问题需要克服，比如数据对齐，但这些都很容易解决。

七、结论

在许多情况下，嵌入式设备中的固件被加密并不是一个很难解决的问题。一旦可以获取到物理设备，解密过程就可以大大简化。值得记住的是，嵌入式设备的固件加密主要在固件更新包中实现，而很少在存储级别实现，这一点与移动电话和笔记本电脑的最新实现相反，这二者常见的加密方法是全磁盘（至少是磁盘的重要部分）加密。

这种设置在未来可能会改变，至少是部分改变。例如，Android从4.4开始就支持 全磁盘加密 ，从7.0开始支持基于文件的加密。自Android 10以来，就需要基于文件的加密。然而应该注意的是，术语“完整磁盘”加密在Android中具有误导性，即加密的只是data/userdata分区。