# Vxworks的一些研究

## Vxworks RTOS介绍

（以下的参考主要是来源于：Wikipedia）

VxWorks是美国风河系统（WindRiver）公司于1983年设计开发的一种嵌入式实时操作系统（RTOS），是嵌入式开发环境的关键组成部分。凭借良好的持续发展能力、高性能的内核以及友好的用户开发环境，其在嵌入式实时操作系统领域占据一席之地。

VxWorks支持几乎所有现代市场上的嵌入式CPU，包括x86系列、MIPS、 LoongISA、 PowerPC、Freescale ColdFire、Intel i960、SPARC、SH-4、ARM, StrongARM以及xScale CPU。

它以其良好的可靠性和卓越的实时性被广泛地应用在通信、军事、航空、航天等高精尖技术及实时性要求极高的领域中，如卫星通讯、军事演习、弹道制导、飞机导航等。

美国的F-16、F/A-18战斗机、B-2隐形轰炸机和爱国者导弹，火星探测器如1997年7月登陆的火星探测器，2008年5月登陆的凤凰号、2012年8月登陆的好奇号都使用到了VxWorks。

## Vxworks镜像

（以下的一些参考主要是来源于：[VxWorks系统的映像及其装载过程解析-拳皇14键盘优化补丁-程序博客网 (csdndocs.com)](https://www.csdndocs.com/article/1351323)）

Vxworks的映像主要是分为两个不同的大类：一类是BootRom类型，一类是Vxworks类型：

1. BootRom类型

BootRom类型映像是一个最小化、专用的VxWorks引导映像，实现最少的系统初始化，主要用于启动装载VxWorks映像，其功能类似于PC机的BIOS。BootRom运行时也建立起多任务环境，包括usrRoot任务、网络任务、TFFS任务和FTP任务等。引导映像在运行时，可能在ROM/Flash中执行(例如ROM驻留型引导映像)，也可能在RAM 中执行。在系统中其对应的编译规则文件是rules.bsp。

BootRom类型分为三种：BootRom\_res、BootRom\_uncmp和BootRom。第一种是一直运行在rom中的映象，只把data段拷贝到ram里面；第二种是非压缩方式的映象，data段和text段都要拷贝到ram里面，并在ram里面运行；第三种是压缩方式的映象，生成的时候编译器会把除掉romlnit.s和bootInit.c之外的目标文件压缩并“汇编”成一个bootrom.Z.s，最后和romInit.o，bootInit.o，version.o进行链接，生成bootrom映像。所以它也是要全部拷贝到ram中，并必须要进行解压缩的工作。而这些工作基本上都是在bootInit.c中进行的。

1. Vxworks类型

VxWorks类型映像是系统的主映像，也即是系统最终要运行的映像。该映像在运行时至少有一部分(如：数据段和BSS段)需要在RAM中运行。在系统中其对应的编译规则文件是rules.vxWorks。

VxWorks类型映像分为四种：VxWorks（RAM运行的Vxworks镜像）、VxWorks\_rom（ROM启动RAM运行的Vxworks镜像，不选哟BootRom的辅助）、VxWorks\_romResident（ROM启动ROM运行的Vxworks映像，不需要BootRom的辅助）和VxWorks\_romCompress（Vxwroks\_rom的压缩形式）

1. BootRom 类型映像和VxWorks类型映像的联系与区别

对于没有自启动功能的VxWorks类型映像(例如：VxWorks型)，在映像运行前就需要一段程序将该映像拷贝到RAM中运行，而BootRom类型映像在此时就扮演了这种“搬运工”的角色，当VxWorks系统下载完毕，BootRom的任务也就完成了。二者在系统初始化的时候，所做的功能基本相同，但是区别在于BootRom类型映像调用bootConfig.c，而VxWorks类型映像调用usrConfig.c。

同时经过学习发现，其实对于Vxworks来说，其实RTOS的系统版本跟Wind的内核版本还是存在差异的。两者不能简单的划等号。

## Vxworks的镜像加载方式

（以下参考资料为：<https://blog.csdn.net/dwd112358/article/details/96445634>）

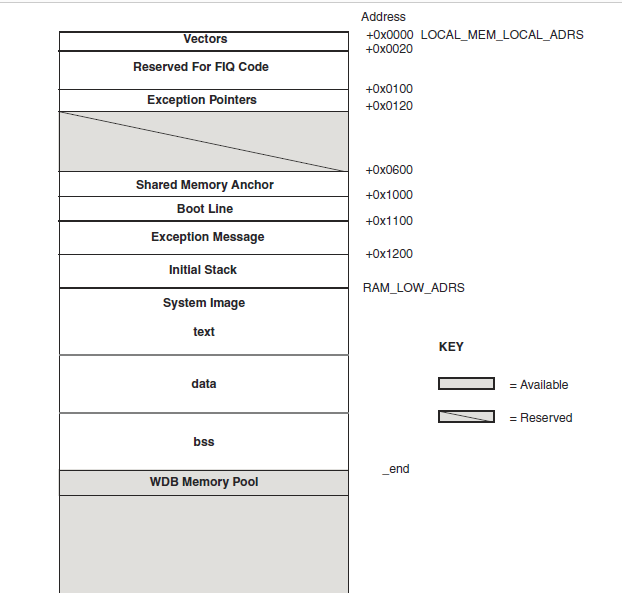
Vxworks映像的文件封装格式：.elf、.bin文件格式，两种文件格式下包含的一些内容不太一样，但是重要的组件都是存在的。

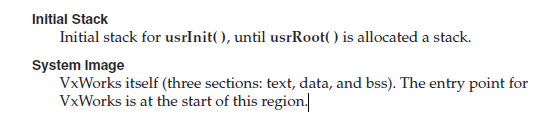
可以肯定的是，不管有无BootRom,都是在之后调用Vxworks的映像的，然后通过几个函数的作用，初始化系统，而至关重要的一个函数也是第一个被调用的函数就是usrInit函数，用于执行系统初始化程序，而该函数通常也是用于回复符号表的一个关键所在。而该函数的作用如下：Interrupt lock out,save imformation about boot type,handle all the Initialization before the kernel is actually started,then starts the kernel execution to create an initial task usrRoot(),This task completes the start up.

（下面的一些内容主要是来源于：[工控安全入门（六）——逆向角度看Vxworks - Luminous~ - 博客园 (cnblogs.com)](https://www.cnblogs.com/nongchaoer/p/11975984.html)）

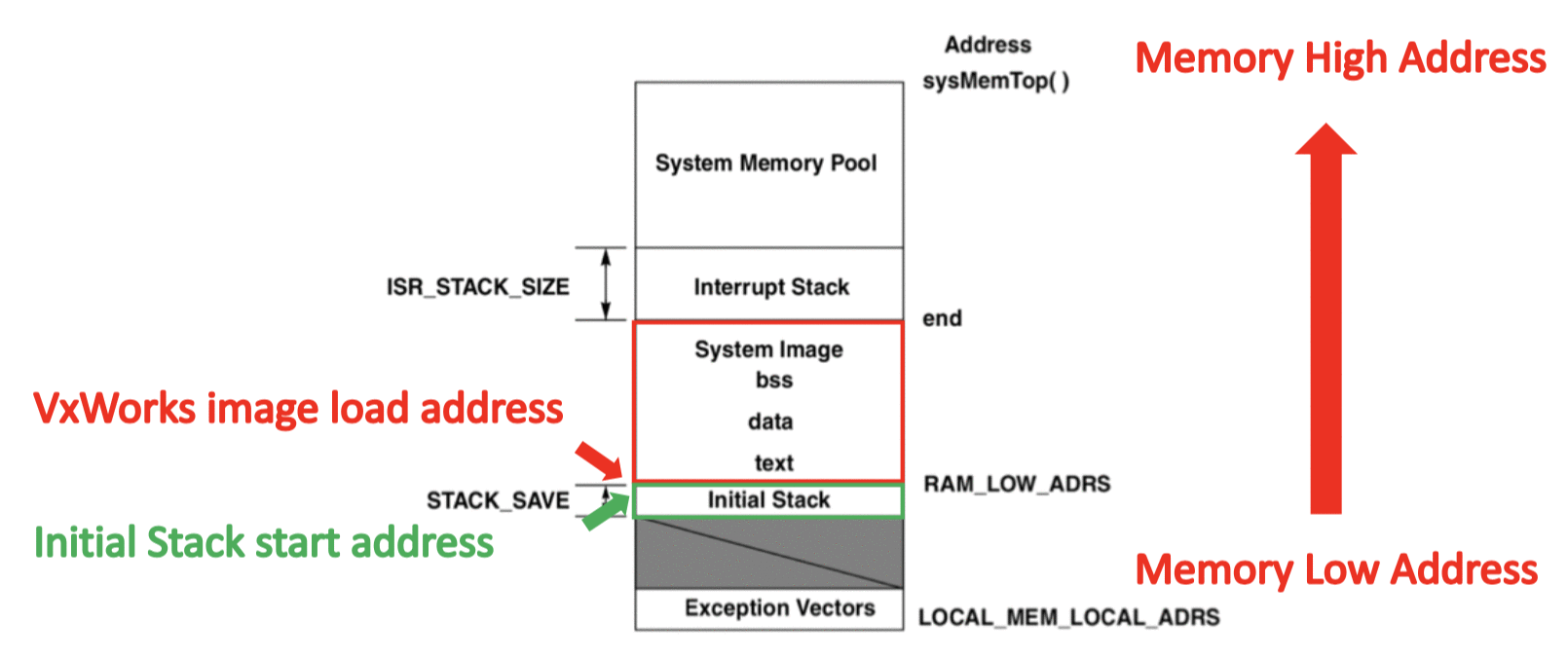
🐖此处省略，需要时补充！！！！

## Vxworks符号表的恢复





来源:已下载文件Long Nick的vxworks\_architecture\_supplement\_6.2.pdf



Vxworks镜像在加载的过程中内存地址中所呈现的关系（但是目前有问题，只是找到了在PowerPC和Mips架构中的内存布局，但是没有找到在ARM中的内内存布局，这里还需要进行进一步的处理），其中VxWorks映象由文本段（.text/.code），数据段（.data）和BSS段（.bss）组成。



通过使用Ghidra,恢复了一部分的符号表，但是没有办法回复恢复全部，同时使用IDA将Gihidra的结果输入到符号表恢复的脚本中，也还是失败的，但是好在就目前的结果来说，**还是恢复了部分的符号表，虽然我用IDA也没有看到怎么恢复的。**

1. <https://www.cnblogs.com/yangmzh3/p/11214451.html>
2. <https://cn-sec.com/archives/777131.html>
3. <https://p1kk.github.io/2021/06/12/iot/tplink%20wr886%20v6/>

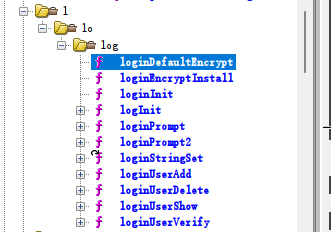
下面这个是针对Vxworks本身的一些方法研究：

* <https://blog.csdn.net/stonesharp/article/details/50530849?spm=1001.2101.3001.6650.4&utm_medium=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-4.pc_relevant_antiscanv2&depth_1-utm_source=distribute.pc_relevant.none-task-blog-2%7Edefault%7ECTRLIST%7ERate-4.pc_relevant_antiscanv2&utm_relevant_index=9>
* <https://zyen12138.github.io/2022/02/04/%E5%A6%82%E4%BD%95%E5%AF%B9ARM%E6%9E%B6%E6%9E%84%E4%B8%8B%E7%9A%84VxWorks%E7%B3%BB%E7%BB%9F%E5%9B%BA%E4%BB%B6%E8%BF%9B%E8%A1%8C%E4%BF%AE%E5%A4%8D%EF%BC%9F/>

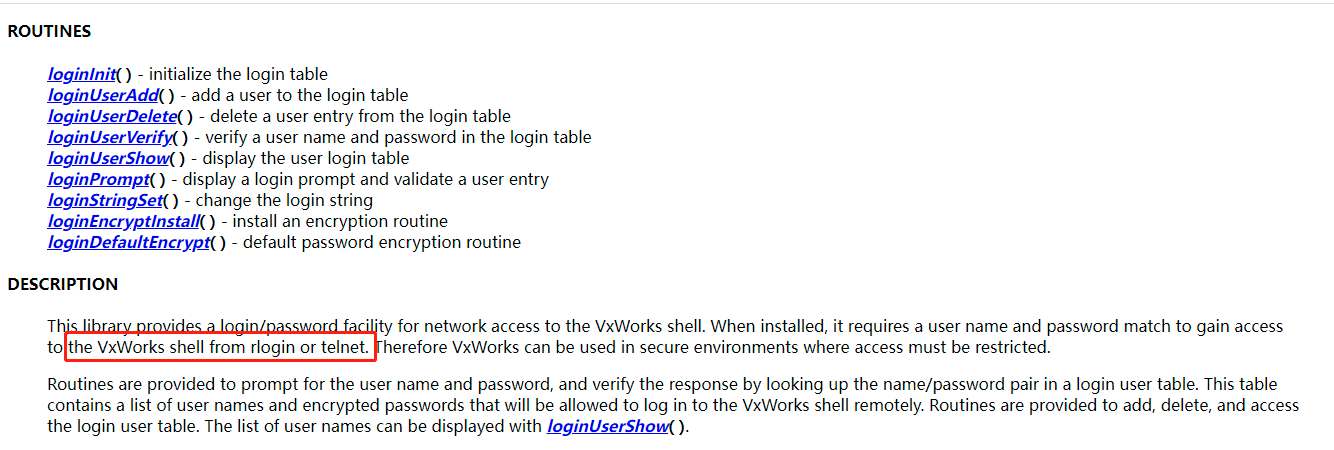
# 对于固件可能函数的分析

## 使用login关键词进行搜索

在使用Ghidra对于固件的符号表恢复之后，通过搜索一些关键词可以得到这样的一些结果：

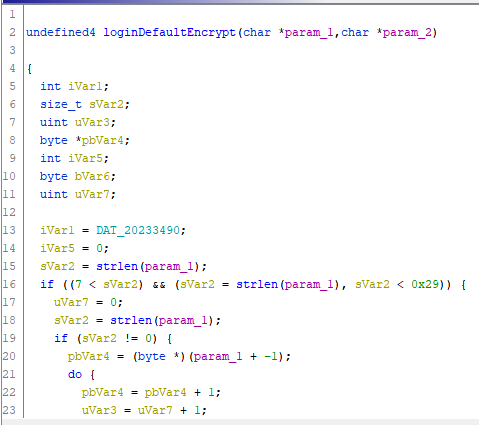
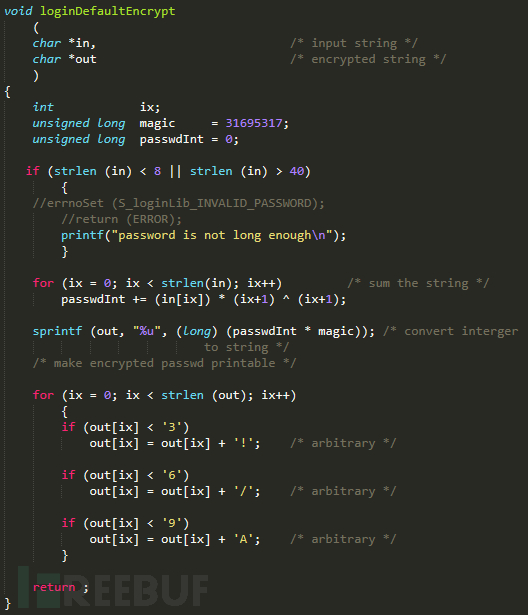


但是查询第一个函数的官方文档后（https://www.ee.ryerson.ca/~courses/ee8205/Data-Sheets/Tornado-VxWorks/vxworks/ref/loginLib.html）发现该函数（位于官方的loginLib库中）是用于远程登陆的，显然这个是没有开放端口的（但是也不排除有这种可能的存在）：



## 对于logindefaultEncrypt函数的研究

下面是使用Ghrida对于该函数的一个逆向得到的结果，因为该函数从VxWorks的设计来说，通常使用的是该函数作为key的加密算法：

也是从你想和源码（参考https://www.freebuf.com/vuls/177036.html）的一个比对过程中发现，其实对于该函数的实现过程中只是存在部分的差异，就是密码长度的设置上：



但是其实从抓包过程中的结果来看，抓包结果中显示的结果应该是base64编码过的，但是具体多长还是不太清楚，从base64编码的过程来看，那个报文里面应该是存在着四个小的（以=作为编码的结束），但不知道他们具体代表了什么。

fw3jTmgCty8=

dxCEGHWjlUdEz2BucUD2EKqaREYosuVCDT4Okqgslt4=

xrIDrZ5FsD4=

BwJ5H3Jqvh+tKPORjDQz+cK8ZepNuDS9EkyZl7VigOE=

使用下面的脚本对他们进行解码：

import base64

# base64解码

def base64\_decode(base64\_data):

temp = base64.b64decode(base64\_data).hex()

return temp

data = "qgEeAGBFRQAAZEIAAFhCAAAAAAAAE0LFEAVqAAAHZhbn9rs="

tem = base64\_decode(data)

print(tem)

结果如下：

**7f0de34e6802b72f**

**7710841875a3954744cf606e7140f610aa9a444628b2e5420d3e0e92a82c96de**

**c6b203ad9e45b03e**

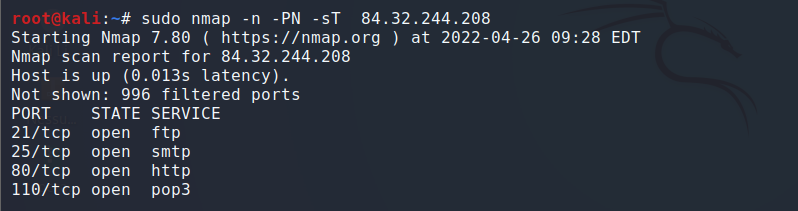
**0702791f726abe1fad28f3918c3433f9c2bc65ea4db834bd124c9997b56280e1**

**但是直接使用上述过程中的过程对该过程进行分析时，结果不一致！！！**

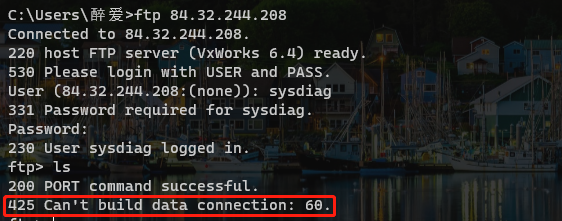
# 对于BMXP342020固件的一些研究

## 使用nmap进行扫描，查看开启的服务和端口

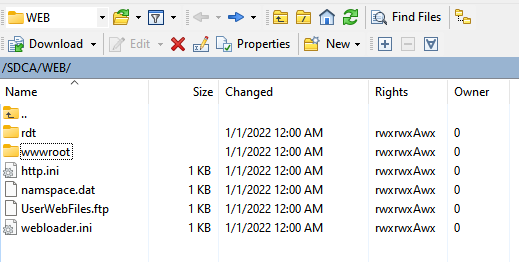
nmap -P0 -sS -sU -A -O 84.32.244.208 -p 17185(使用该命令用于查询UDP协议下开放的17185端口，该端口用于WDB RPC V2调试，同时还需要求证一件事，为啥没有找到开放的502端口？？？)



可以发现，对于目标系统而言，开启了ftp服务以及http服务，而对于ftp服务而言，正常情况下，应该是同时开放了20和21端口，其中20用于传输数据，21用于传输控制信息，而在命令行模式下只有无法建立正常的数据连接：



因此使用winscp进行处理，但是同样的问题，由于权限的限制，目前这个只能达到该目录下：



而系统的固件再上一层目录中。

## 对于80端口的一些研究

如何通过Web端拿到Shell？（参考链接：[拿shell的一些方式\_努力奋斗的小青年的博客-CSDN博客\_拿shell](https://blog.csdn.net/zdy8023/article/details/89059137)）

## Smtp和pop3服务

还开放了smtp和pop3服务，但是目前尚且不太明白smtp和pop3的在该设备中的功能是啥？（这是一个后门吗？开启该功能后会不会向服务器发送一些敏感信息？）

# 附件

## 恢复符号表的IDA脚本

（需要更改一些内容，同时有些部分还是不太确定，如ARM架构下的符号表是不是就是16个字节一个表项，这个还是非常重要的问题，有一些能够恢复，有一些是恢复不了的）

#coding=utf-8

from idaapi import \*

from idc import \*

symbol\_interval = 20 #符号表间隔

load\_address = 0x1f204 #固件内存加载基址

symbol\_table\_start = 0x3b8be8 + load\_address #符号表起始地址

symbol\_table\_end = 0x40b424 + load\_address #符号表结束地址

ea = symbol\_table\_start

eaEnd = symbol\_table\_end

while ea < eaEnd:

offset = 0 #4个字节为一组数据

#将函数名指针位置的数据转换为字符串

create\_strlit(get\_wide\_dword(ea-offset), BADADDR)

#将函数名赋值给变量sName

sName = get\_strlit\_contents(get\_wide\_dword(ea))

print(sName)

if sName:

#将bytes转为str

sName = str(sName,encoding="utf-8")

#开始修复函数名

eaFunc = get\_wide\_dword(ea - offset +4)

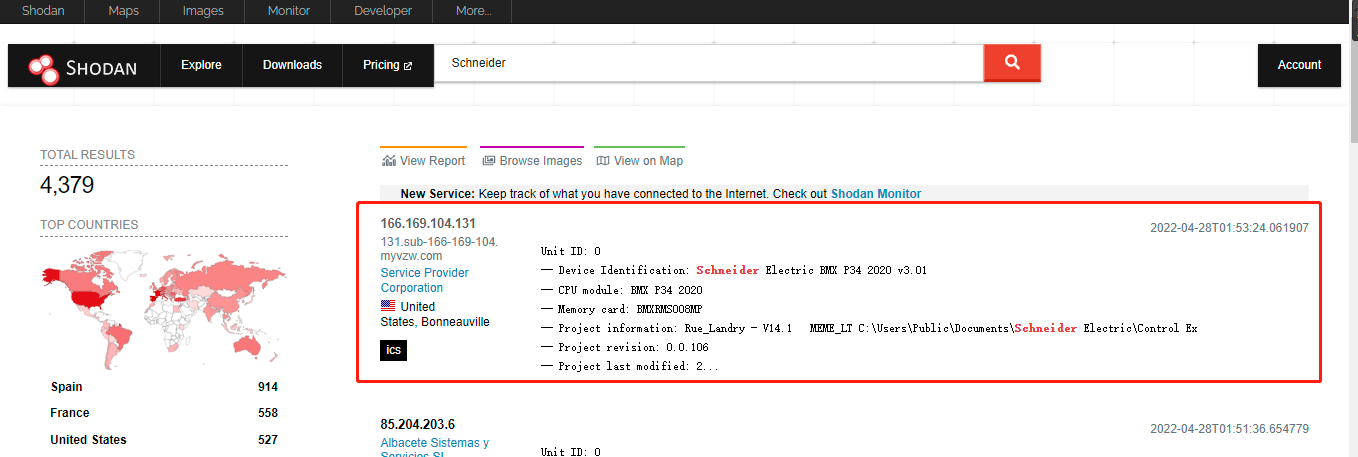
set\_name(eaFunc, sName)

create\_insn(eaFunc)

ida\_funcs.add\_func(eaFunc, BADADDR)

ea += symbol\_interval

## 使用shodan



这一款如果是真的，那其实跟我这一款的是一样的。而且将搜索的关键字设置为上图中显示的内容，可以找到更多同款设备

## 在对于WDB RPC V2协议而言，不少的报告说明该协议开放有较高的危险性，但是在测试过程中并没有发现该设备有开放该协议端口。