Semester project
Department of computer sciences
February 2005

Embedded
Network Intrusion Detection System

Authors:
Matthias Kolly
Marc Staub

Supervisors:
Nicolas Schroeter, EIF
Jean-Roland Schuler, EIF

External supervisor:
Stefan Engel, Armasuisse
Table of contents

1 Introduction ........................................................................................................... 5

2 CerfBoard 255 ....................................................................................................... 6
   2.1 Hardware.......................................................................................................... 6
   2.2 Software............................................................................................................ 6

3 Installing i-linux 5.0 ............................................................................................... 7
   3.1 Introduction...................................................................................................... 7
   3.2 Requirements................................................................................................... 7
   3.3 Setup ............................................................................................................... 7
      3.3.1 CerfBoard............................................................................................ 7
      3.3.2 Host....................................................................................................... 8
         Console 8
         Web server 8
         Configuration 8
         Dhcp server 8
   3.4 Installation ....................................................................................................... 9
   3.5 Comment......................................................................................................... 9

4 Installing the toolchain (Cross development environment) .................................... 10
   4.1 Introduction.................................................................................................... 10
   4.2 Requirements................................................................................................. 10
   4.3 Setup ............................................................................................................... 10
   4.4 Installation ...................................................................................................... 10
   4.5 Comment....................................................................................................... 10

5 Installing necessary packages ............................................................................... 12
   5.1 Introduction.................................................................................................... 12
   5.2 Requirements................................................................................................ 12
   5.3 Setup ............................................................................................................. 12
   5.4 Installation ..................................................................................................... 12
      5.4.1 libpcap 0.8.3 ..................................................................................... 12
      5.4.2 libpcre 5.0.......................................................................................... 13
      5.4.3 libnet 1.1.2.1...................................................................................... 13
   5.5 Comment....................................................................................................... 13

6 Installing Snort .................................................................................................... 14
   6.1 Introduction.................................................................................................... 14
   6.2 Requirements................................................................................................ 14
   6.3 Setup ............................................................................................................. 14
   6.4 Installation ..................................................................................................... 14
7 Setting up Snort on the CerfBoard ........................................................................... 16
   7.1 Introduction ............................................................................................................. 16
   7.2 Requirements ......................................................................................................... 16
   7.3 Networking ............................................................................................................. 16
   7.4 Snort ...................................................................................................................... 17
      7.4.1 Deployment ................................................................................................. 17
      7.4.2 Setup ............................................................................................................ 17
      7.4.3 Usage ............................................................................................................ 18
         Install the rules .................................................................................................... 18
         Run it ................................................................................................................ 18
      7.4.4 Remote access ............................................................................................... 18
   7.5 Comment .............................................................................................................. 18

8 Testing the NIDS-Sensor ............................................................................................. 19
   8.1 Introduction ........................................................................................................... 19
   8.2 Test Description ..................................................................................................... 21
      8.2.1 No rules ......................................................................................................... 22
         Description ....................................................................................................... 22
         Implementation ............................................................................................... 22
      8.2.2 One rule “false packet” ............................................................................... 22
         Description ....................................................................................................... 22
         Implementation ............................................................................................... 23
      8.2.3 Standard set of rules .................................................................................... 24
         Description ....................................................................................................... 24
         Implementation ............................................................................................... 24
      8.2.4 Standard set of rules with alerting .............................................................. 25
         Description ....................................................................................................... 25
         Implementation ............................................................................................... 25
      8.2.5 Real scenario ................................................................................................. 25
         Description ....................................................................................................... 25
         Implementation ............................................................................................... 25
   8.3 Implementation ..................................................................................................... 26
      8.3.1 Scheme ........................................................................................................... 26
      8.3.2 Tools and elements ....................................................................................... 26
         CerfBoard ........................................................................................................ 26
         PC-System ....................................................................................................... 27
         Agilent Advisor ............................................................................................... 27
         Snort-alerts ..................................................................................................... 28
      8.3.3 Other interesting tools ................................................................................... 28
         D-ITG ................................................................................................................ 28
         Etherreal .......................................................................................................... 28
         packETH .......................................................................................................... 28
         Nessus .............................................................................................................. 28
         perfmon-graph ............................................................................................... 29
         Snot ................................................................................................................. 30
         Whisker .......................................................................................................... 30
      8.3.4 Snort configuration ......................................................................................... 30
      8.3.5 Configuration of the network interface card ............................................... 30
8.4 Results

8.4.1 No rules

8.4.2 On rule “false packet”

8.4.3 Standard set of rules

8.4.4 Standard set of rules with alerting

8.4.5 Real scenario

9 Test Evaluation

10 Conclusion

11 Attachments

12 References & important links
1 INTRODUCTION

These days the number one concern of the whole computer industry is security. The Internet is strongly evolving from being a big network into the Evernet. It's almost everywhere, giving us access to desired information at any place. However, security didn't evolve as fast. Nobody can keep track of the number of worms, spyware and cracker attacks all over the world. Since we can't fix all the problems, the only solution is to at least detect the attacks.

This is done using so-called Network Intrusion Detection Systems (NIDS). Right now these systems are hosted on big servers, scanning all the traffic on a network and giving out alerts when suspicious data shows up. They don't stop any traffic; they just inform about possible bad data. Although this is working right now, it does have its problems. The biggest being the count of “false-positive” alerts, which makes it a time consuming job to process the alerts. Currently work is being done to improve the detection of attacks and to reduce the number of those “false-positive” alerts.

Another problem is the size of the machines; there's almost no flexibility due to size and required room.

Based on that, the ultimate goal is to create an independent NIDS sensor usable for distributed work. That means it has to be extremely flexible for deployment, giving fast access and still be able to do the basics: intrusion detection. This leads to embedded systems.

We were given an Intrinsyc CerfBoard 255 for our project. It's best described as a small development board with an Intel XScale PXA255 processor (a StrongARM design) and lots of I/O ports.

So our task was to install the latest available version of the Linux operating system and to convert Snort (a successful network intrusion detection software) on that platform. The goal was to do some tests in order to determine the limitations of that platform.
2 CERFBOARD 255

We are using an Intrinsyc CerfBoard 255 [1] as the target platform for Snort. Here are some specifications about that hardware and its software.

![Image of CerfBoard 255, CerfComm 250, and Cerf IO 250](source: www.intrynsic.com)

2.1 HARDWARE

Here are some details about the hardware.

<table>
<thead>
<tr>
<th>Processor</th>
<th>Intel XScale PXA255 microprocessor 400 MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>32MB Flash</td>
</tr>
<tr>
<td></td>
<td>64MB RAM SDRAM 100MHz</td>
</tr>
<tr>
<td>I/O</td>
<td>2x 10/100BaseT Ethernet</td>
</tr>
<tr>
<td></td>
<td>1x USB 1.1 - Type B Client</td>
</tr>
<tr>
<td></td>
<td>1x USB 2.0 - Type A Master</td>
</tr>
<tr>
<td></td>
<td>1x RS232 Serial Debug</td>
</tr>
<tr>
<td></td>
<td>2x RS232 Serial</td>
</tr>
<tr>
<td></td>
<td>1x CompactFlash</td>
</tr>
<tr>
<td>CompactFlash</td>
<td>Supports type I and II cards including</td>
</tr>
<tr>
<td></td>
<td>IBM/Hitachi Microdrive, CF memory cards,</td>
</tr>
<tr>
<td></td>
<td>barcode readers, wireless modem and</td>
</tr>
<tr>
<td></td>
<td>Bluetooth cards</td>
</tr>
<tr>
<td>Power</td>
<td>5V DC</td>
</tr>
<tr>
<td></td>
<td>400mA without CompactFlash device</td>
</tr>
<tr>
<td></td>
<td>1.1A peak with CompactFlash device</td>
</tr>
</tbody>
</table>

As you can see, the specifications are pretty impressive with all these I/O ports. There are even more, the complete specification can be found here [1].

2.2 SOFTWARE

The CerfBoard comes with TurboBoot and i-linux preinstalled. There's also a cdrom that contains the x86 ARM cross compilation tool chain.
3 INSTALLING I-LINUX 5.0

3.1 INTRODUCTION

In this guide we are going to install i-linux version 5.0 (using Kernel 2.6) on the CerfBoard 255. We assume that the CerfBoard is already working correctly but on an older version of i-linux, so this is merely an upgrade. We are not going to change the boot manager and so this documentation is based on the preinstalled TurboBoot boot manager.

3.2 REQUIREMENTS

The installation process doesn't require any too special software or hardware. All we need is a host with a connection to the CerfBoard console (RS-232) using a standard cable, a web server and a dhcp server somewhere on the network the CerfBoard is going to be connected to. The necessary up to date i-linux 5.0 image files can be found on the cdrom from Intrinsyc, which can be obtained here [1].

3.3 SETUP

3.3.1 CerfBoard

First check all the DIP switches on each board, we are going to use the debug output of the CerfIO Board. Then connect the CerfBoard to the host using a standard RS-232 console cable on the J10 DEBUG port of the Cerf IO 250 Board.

Also connect the CerfBoard to your local network using the Ethernet port on the CerfBoard itself (the upper one). It needs to have access to the dhcp server. If you use the dhcp and the web server on the same host, you may connect it directly to that host using a crossed network cable.
3.3.2 Host

Console

We are using Debian Sarge GNU/Linux [2] on x86 for this but it can be done with almost any operating system on almost any platform.
We need to configure a console application like minicom on Linux or HyperTerminal on Windows in order to be able to communicate with the CerfBoard. On Debian, simply type “apt-get install minicom” as root in order to install minicom. Help on setting them up can be found using the Intrinsyc documentation [3]. Just note: Linux calls your COM-ports “ttySx”, starting from ttyS0. So don’t forget to configure minicom on the right port: Start minicom; CTRL-A; O; choose Serial port setup; A; enter /dev/ttyS0 or the port you’re using, set Bps/Par/Bits to 38400 8N1, Hardware Flow Control to no and Software Flow Control to no.

Web server

For the web server, we recommend using apache directly on the host, but you can use any web server on any system in the network. Running Debian, type “apt-get install apache” as root and you’re done. Locate the base directory of your web server. Using apache on Debian this would be ’/var/www’. Now copy the content of the ‘cdrom/i-linux-board-5.0/images’ folder to ’/var/www’. Also copy the file ‘cdrom/turbo-boot-1.6/configure.script’ to ’/var/www’. That’s it for the web server.

Configuration

We need to edit the configure.script file we just copied. Open it using your favorite text editor. Warning: If you consider using the Intrinsyc documentation for this task you’ll end up with a non-working script. So follow this guide and read the comments inside the file, since some variables changed their name.
Edit the following lines to:

```
string zImageURL = "http://<your-web-server-IP>/kernel-image-cerf-board-pxa255-2.6.7-cerfb1_arm.img";
string jffsImageURL = "http://<your-web-server-IP>/familiar-image-cerf-board-pxa25x-0.7.2-5.0.jffs2";
integer forceReload = false;
```

Just replace <your-web-server-IP> with the IP address of your web server, don’t put in localhost! We are not finished yet. Somewhere in the middle of the file you can find the following comment:

```
// Uncomment to stop checking for updates from net on every boot
```

Uncomment the line right after it, just as it says. It should read like this:
```
config.update = false;
```

If you don’t, upon each reboot your CerfBoard will retry to update instead of booting into Linux. For your convenience, you can find the script attached to this document.

Dhcp server

As noted before, the CerfBoard needs a dhcp server in order to fetch a valid IP address. Since we have one up and running we just connected the CerfBoard to our network. You may consider installing a free dhcp server on the host if you don’t have one.
3.4 INSTALLATION

Now we can start installing. Don't forget to backup any important data from your CerfBoard since the installation mechanism is going to rewrite every partition!
Start either minicom, HyperTerminal or whatever you use if you haven't already done so.
Power on the CerfBoard. You should see some boot messages on the console.
Pay attention: You'll be prompted to hit a key in order to access TurboBoot. Do so.
Now we'll change the default restore URL to your web server. Enter the following command:
```
>table.restore = "http://<your-web-server-IP>/configure.script"
```
This change is persistent across reboots. Now enter the following commands:
```
>table.clear()
>table.save()
>table.update=yes
>reboot
```
Upon rebooting, TurboBoot will automatically execute the script we configured before. You can follow the whole installation on the console output. It will fetch the necessary image files and install them on the right partition.
If you encounter any errors, it may be that you have a network problem like CerfBoard unable to find your dhcp or web server. Just control that all devices see each other using the ping command and check the console while booting CerfBoard to see if it gets an IP from your dhcp server.
After installation, CerfBoard directly boots into your new i-linux 5.0. You better restart immediately in order to check if CerfBoard reboots correctly into i-linux 5.0 and doesn't retry to install. If this happens it's almost likely due to an error in your configure.script file.
Important:
The default user on i-linux 5.0 is 'root' with the password set to 'rootme'.

3.5 COMMENT

Once everything is in place, it's rather easy to install i-linux 5.0 on the CerfBoard. If you did a backup of all important data, you can't make anything wrong. Just don't touch the boot manager...
4 INSTALLING THE TOOLCHAIN (CROSS DEVELOPMENT ENVIRONMENT)

4.1 INTRODUCTION
In this guide we are going to install the cross development environment for arm-based Hardware like the CerfBoard 255 on Debian Sarge GNU/Linux x86.

4.2 REQUIREMENTS
We need the i-linux 5.0 cdrom from Intrinsyc which can be obtained for free under [1]. We also need a working installation of Debian Sarge GNU/Linux on an x86 host. Free image files with a net installer can be found here [2]. It doesn't matter if it is Kernel 2.4 or 2.6 based, since we don't develop for that system. We use Debian Sarge since Debian Woody is getting somewhat old and we want a convenient environment for development. You may consider using an rpm-based distribution (SuSE, Mandrake, Red Hat) since there are also rpm-files on the cdrom but in this guide we only cover Debian.

4.3 SETUP
We are not going to explain how to install Debian Sarge on an x86 host since it is well documented and almost straightforward. Just make sure to use a pretty recent PC, since compiling takes some time. If you're a Windows user, you might consider installing Debian using Microsoft Virtual PC. That way you don't have to make risky operations on your hard drive. Just keep in mind that compiling is going to take some time...

4.4 INSTALLATION
Insert and mount the cdrom, open up a terminal window. Switch to root, issuing the following command:

```
>su
```
You need to enter the root password.
Go to the cdrom directory called 'arm-tools-1.3'. This should be:

```
>cd /media/cdrom/arm-tools-1.3/arm-tools-1.3
```
Yes, there's twice arm-tools-1.3.
We are going to install the debian package called 'cross-arm-toolchain_2.95.3-1_i386.deb'. Just enter:

```
>dpkg -i cross-arm-toolchain_2.95.3-1_i386.deb
```
That just installed our cross development environment to '/usr/local/arm'. All we need to do now is to add it to our PATH variable. Since we mainly develop as root (due to file permissions) just edit the following file with your favorite text editor: '/root/.bashrc'.

Add the following line at the end of the document:

```
PATH=$PATH:/usr/local/arm/2.95.3/bin
```
Save, quit and we're done.
You may also add that line to your default users '.bashrc' file, which can be found in his home directory.

4.5 COMMENT
We spend some time trying to build our own toolchain, but we had to cancel these efforts since you have to cope with so many details it just wouldn't be
worth investigating. However, see [4] for details if you still consider building
yourself.
Using Debian GNU/Linux on x86 hardware with the Intrinsyc packages, it is
rather easy to install the cross development environment. Since the packages
come directly from Intrinsyc we can be sure they're configured for the right
hardware.
There's also a package called 'cross-arm-toolchain_3.3.3-k2.6.4-1_i386.deb' on
the cdrom. We initially started using that one and even successfully compiled
Snort-2.3.0 using it but we where told that the glibc used by that one is too
recent for the i-linux 5.0 filesystem and so we could encounter errors. That
package is mainly intended for Kerel compilation.
5 INSTALLING NECESSARY PACKAGES

5.1 INTRODUCTION

In this guide we are going to compile all the libraries and programs needed in order to finally compile Snort-2.3.0 for our CerfBoard 255. As in all other other guides, we are working on Debian Sarge GNU/Linux.

5.2 REQUIREMENTS

We need to compile all the libraries and programs Snort depends on. Here's a list of the software needed in addition to the default i-Linux 5.0 installation:

- libpcap [5]
- libpcre [6]
- libnet [7]

We are using the most up-to-date version at time of writing:

- libpcap 0.8.3
- libpcre (pcre-5.0)
- libnet 1.1.2.1

All of them need to be compiled for our arm-linux based CerfBoard, except libpcre, which is also needed on the host computer for Snorts configure script.

5.3 SETUP

All the source tarballs have to be downloaded and extracted. Type

> tar -xzf filename.tar.gz

if it's a gzipped file or type

> tar -xjf filename.tar.bz2

if it's a bzipped file in order to extract the contents of the files.

We are going to develop as root, so we don't have to mess with file permissions. Simply open a terminal, type

> su

(switch user) and enter your root password when asked. This gives you a root-shell.

5.4 INSTALLATION

5.4.1 libpcap 0.8.3

This is the most important library for Snort since it gives access to all packets on the network.

In order to be able to cross-compile it, we need to edit the '.configure' script. So open it using your favorite text editor, go to line 4256 and edit the following:

ac_cv_linux_vers=unknown

to

ac_cv_linux_vers=2

This simply overrides the version test for the Linux kernel and doesn't hurt anything.

Now we start the configuration. Type the following on one line:

> ./configure --prefix=/usr/local/arm/build/libpcap

--includedir=/usr/local/arm/2.95.3/include --host=arm-linux --with-pcap=linux
This is going to install libpcap to '/usr/local/arm/build/libpcap' and compile it for arm-linux using the installed toolchain. The other option sets the packet capture type, which is linux in our case.

After a successful configuration, simply type

```bash
> make
```

which compiles libpcap, followed by

```bash
> make install
```

which installs it to the specified folder.

### 5.4.2 libpcre 5.0

We need a program called 'pcre-config' on the host for Snort's configure script. On Debian simply type

```bash
> apt-get install libpcre3-dev
```

in order to get it.

But you may also use the downloaded tarball. Simply type

```bash
> ./configure
> make
> make install
```

Chose your preferred method.

Now we also need the library for arm-linux. So just as for libpcap, we have to configure it for our target:

```bash
> CC_FOR_BUILD=gcc ./configure --prefix=/usr/local/arm/build/libpcap --includedir=/usr/local/arm/2.95.3/include --host=arm-linux --with-tags
```

We need to specify the local C compiler since it builds a test program for the local host. The tag option has to be set but without a value. This gives an error at the end of the configure script but it can simply be ignored. The other options have the same function as for libpcap.

Just continue with

```bash
> make
> make install
```

### 5.4.3 libnet 1.1.2.1

Libnet isn't really ready for cross-compilation but this doesn't hold us off. Start with the configuration:

```bash
> CC=gcc ./configure --prefix=/usr/local/arm/build/libnet --includedir=/usr/local/arm/2.95.3/include --host=arm-linux
```

We have to specify CC as gcc since it compiles and runs a program on the local host to test the endianess (little/big) of the machine. In fact this is absolutely wrong, since it should test the endianess of the target platform. Happily our x86 and the arm platform have the same endianess (little) and so it works.

Finish with

```bash
> make
> make install
```

### 5.5 COMMENT

After all these steps we now have everything in place in order to cross-compile Snort for the CerfBoard. Although work wasn't always proper (renaming, changing and ignoring aren't good ways) we still obtained what we need to get Snort up and running. We have found a patch for libnet which adds an option to force the endianess, but it was for an older version. There has to be done some work on these libraries and programs in order to do some proper cross-compiling. Apparently they weren't programmed with that in mind. Hopefully future version will improve that.
6  INSTALLING SNORT

6.1  INTRODUCTION
We are finally going to configure, compile and install Snort-2.3.0 on the CerfBoard 255.

6.2  REQUIREMENTS
This guide is only usable if you already compiled the necessary libraries as discussed in the other guides. You simply need to get the Snort source tarball from [8].

6.3  SETUP
Once you have the Snort tarball, simply extract it using

>tar -xzf snort-2.3.0.tar.gz

Again we are going to develop as root, so we don't have to mess with file permissions. Simply open a terminal, type

>su

(switch user) and enter your root password when asked. This gives you a root-shell.

6.4  INSTALLATION
Let's start configuring Snort. Enter the following on one line:

>./configure --prefix=/usr/local/arm/build/snort
--includedir=/usr/local/arm/2.95.3/include
--host=arm-linux
--with-libpcap-libraries=/usr/local/arm/build/libpcap/lib
--with-libpcap-includes=/usr/local/arm/2.95.3/include
--with-libpcre-libraries=/usr/local/arm/build/libpcre/lib
--with-libpcre-includes=/usr/local/arm/2.95.3/include
--with-libnet-libraries=/usr/local/arm/build/libnet/lib
--with-libnet-includes=/usr/local/arm/2.95.3/include

This has to finish without any errors.

There is a file called route.h in both the Linux kernel and glibc. Snort only needs the one from glibc but apparently the one from the kernel comes in first. So in order to be able to compile Snort correctly we have to issue the following command (on one line):

>mv /usr/local/arm/2.95.3/arm-linux/sys-include/net/route.h
/usr/local/arm/2.95.3/arm-linux/sys-include/net/route.h-kernel

Now we are ready to compile Snort:

make

After a successful compilation, we re-name the file to its old name (also on one line):

>mv /usr/local/arm/2.95.3/arm-linux/sys-include/net/route.h-kernel
/usr/local/arm/2.95.3/arm-linux/sys-include/net/route.h

And now we can finally install Snort:

make install

This installs it to '/usr/local/arm/build/snort' as configured before. If you go see the contents of this folder, you'll see a 'naked' 'snort' executable in the 'bin' folder and a man-file in the 'man' folder. There are no configuration files. In fact, these configuration files are still in the downloaded source tarball. So we need to copy them over. Assuming that you're still in the snort source directory, issue the following commands:

>mkdir /usr/local/arm/build/snort/etc
>cp etc/classification.config etc/Unicode.map etc/reference.config etc/threshold.conf /usr/local/arm/build/snort/etc

Now you have Snort-2.3.0 compiled and configured for arm-linux.

6.5 COMMENT

Once all necessary libraries and programs are in place it's rather easy to compile Snort. The only tricky part being to rename a file, which isn't that obvious. After all we now have an arm-linux version of Snort 2.3.0.
7 SETTING UP SNORT ON THE CERFBOARD

7.1 INTRODUCTION
Now that Snort and the necessary libpcre shared library are compiled for arm-linux, it's time to set things in place on the CerfBoard.

7.2 REQUIREMENTS
Login to the CerfBoard with the console (RS-232) connection. As said the login is 'root' with 'rootme' as password.

7.3 NETWORKING
We'll start setting up both network interfaces. Simply type
>ifedit
and you'll end in 'vim' with the network configuration file open.
Adjust the following:
auto eth0
   iface eth0 inet static
   address <CerfBoardIP>
   netmask 255.255.255.0
auto eth1
   iface eth1 inet static
Replace <CerfBoardIP> with any useful IP address. This fixes the IP of eth0 (the upper port) to <CerfBoardIP>/24 and won't activate eth1 (the lower port). The idea is that you connect eth0 to your management LAN so you have access to the device and eth1 to the network you're going to analyze with Snort using promiscuous mode. That way the CerfBoard won't be accessible from the suspicious network.
Type the following commands in order to activate the new network setup:
>ifdown eth0
>ifdown eth1
>ifup eth0
There's no need for 'ifup eth1' since it won't work. We need to activate it manually:
>ifconfig eth1 up
By issuing the 'ifconfig' command, we can check that everything is ok:
eth0   Link encap:Ethernet  HWaddr 00:D0:CA:F1:3F:C9
       inet addr:<CerfBoardIP>  Bcast:xxx.xxx.xxx.xxx  Mask:255.255.255.0
       UP BROADCAST MULTICAST  MTU:1500  Metric:1
       ...eth1   Link encap:Ethernet  HWaddr 00:D0:CA:F1:3E:05
       UP BROADCAST MULTICAST  MTU:1500  Metric:1
       ...lo     Link encap:Local Loopback
       inet addr:127.0.0.1  Mask:255.0.0.0
       UP LOOPBACK RUNNING  MTU:16436  Metric:1
They are all up and eth1 doesn't have an IP address.
Now simply connect eth1 to a hub or switch span-port in order to scan that network and connect eth0 to a management LAN or directly to a PC.
7.4 **Snort**

7.4.1 **Deployment**

Snort is still in the following directory on the development host: 
'/usr/local/arm/build/snort'.
Libpcre can be found here: '/usr/local/arm/build/libpcre'.
Since Snort has more than one file, we make a tar.gz archive so we don't have to copy file per file. Just type
> cd /usr/local/arm/build
> tar cf snort.tar snort
> gzip snort.tar

Now we just need to transfer the file 'snort.tar.gz' and the shared library
'libpcre.so.0' over to the CerfBoard.
We do that using 'scp':
scp /usr/local/arm/build/snort.tar.gz root@<CerfBoardIP>:
scp /usr/local/arm/build/libpcre/lib/libpcre.so.0 root@<CerfBoardIP>:
That will copy both files to '/root' on the CerfBoard.

7.4.2 **Setup**

Now logon to the CerfBoard either using 'ssh' or the debug console.
You should be in '/root'. Type the following to extract Snort from the archive:
> tar xzf snort.tar.gz
Now we're ready to put things in place:
> mkdir /etc/snort
> mkdir /etc/snort/rules
> mv snort/etc/* /etc/snort
> mv snort/bin/snort /bin
> mv snort/man/man8/snort.8 /man/man8
> rm -rf snort
> mv libpcre.so.0 /lib
> mkdir /var/log/snort

Please note that the last command has to be reissued after each reboot of the
CerfBoard because it will be deleted. Since that directory is hard-coded into
Snort (line 72 of snort.h) the only solution to bypass that is either creating a
startup script that does create that directory or to overwrite it at the command
line when starting Snort (or to modify the source code).
That gives us a working Snort installation on the CerfBoard. All we need to do
now is configuring the 'snort.conf' file using 'vim'. This really depends on your
configuration. But you at least have to specify the folder where Snort has to
look for its rules. You can find the entry at line number 109. Change it to:
var RULE_PATH /etc/rules
The configuration of the rules is done in the lower part of the same document.
By default most of them are activated. In order to activate or deactivate them
simply place or remove a '#' before the corresponding line.
After that configuration we're now ready to use Snort.
7.4.3 Usage

Install the rules

Go to [7] and download the latest rules file. For Snort-2.3.0 download the file called 'snortrules-snapshot-2_3.tar.gz'. Get the file over to your CerfBoard using scp. Logon to the CerfBoard and do the following:

```
>tar xzf snortrules-snapshot-2_3.tar.gz
>rm /etc/snort/rules/*
>cp rules/* /etc/snort/rules
```

Please note that each rules-snapshot comes with a 'snort.conf' file and that maybe new rules are only configured in that new file, not in the one you're currently using. We recommend you to use and reconfigure always the new one.

Run it

There's an annoying fact about Snort:
It searches its 'snort.conf' only in the current directory. Just as with the log directory before, it's hard-coded. So each time you would like to use Snort you should 'cd' over to '/etc/snort' and there's no option on the command line...

```
>cd /etc/snort
>snort -i eth1
```

This will run snort on interface eth1. It will load all the rules and stop immediately if there's an error with the configuration. After that it starts analyzing traffic. It doesn't monitor directly to the screen. In order to quit, just type 'control-c'. Snort now does display some statistics like all the packages seen, how many it analyzed and how many it dropped. Right next to that it shows how many alerts it found.

7.4.4 Remote access

In order to access the alerts file, we simply have to make a secure shell connection to the CerfBoard and go read the file 'var/log/snort/alerts'. Although we haven't tested it, you might consider writing a cron script that copies the 'alerts' file to a remote server using scp. A short test revealed that with the ~15MB free memory you could store around 80000 alerts. Of course that depends upon the length of the alert message.

7.5 Comment

Setting up Snort on the CerfBoard isn't that complicated but still has its difficulties. After all it's kind of strange Snort has to be run from the same directory the 'snort.conf' file resides in and that the log directory is hard-coded. But after all this doesn't hold us off from using it.
8  TESTING THE NIDS-SENSOR

8.1  INTRODUCTION

Once Snort is installed and well working on the embedded system, we have to check if the sensor can efficiently be used in a computer network. Therefore we have to find out the limitations of the sensor. But, how to find out the limitations? What defines the limitation?

In order to describe the limits of Snort on an embedded system, we have to think about which elements the performance of Snort depends and which elements affect the performance.

Let's have a look at how an NIDS is associated with the rest of the operating system and the attached hardware:

![Diagram showing the association between Operating System, Snort, libpcap, NIC, and Network]

Which way takes a packet passing by on the network into our operating system? First, the network interface card captures and processes it. A thread inside the operating system (libpcap in this case) polls the NIC and receives the packet. It is then stored into "userland-memory" where Snort can access. Snort does its treatment, which means, it is processing the packet by applying preprocessors, rules, logging, etc.

It is a basic need to understand each step of processing a packet from the network to be able to describe the elements affecting the performance of an NIDS.

Now, it is possible to distinct two different manners to describe the performance of an NIDS:

1) Hard- and Software limitations

   Primarily, the performance of an NIDS is dependent on the system, on which it is set up. A NIDS needs more or less CPU-time to process the packets passing by, depending on the number of bytes emitted on the cable. It could be possible that too many packets pass through in a too
small interval, so the NIDS has not got the time to analyse each packet in real-time. If the NIDS is unable to process network traffic at the rate it arrives, packets may be dropped and valuable information is lost. Significant packet loss negatively affects the overall NIDS effectiveness. Snort uses therefore so-called preprocessors to counteract against it. Preprocessors are modules who analyse packets even before they are transmitted to Snort to apply the rules.

Since the NIDS is a kind of “Networksniffer” (every packet passing by has to be processed) it is also important, that the operating system on which the NIDS is build on is able to forward every packet from the hardware-layer to the software-layer within an adequate time. Snort uses therefore on Unix like systems as well as on Windows operating systems the library called “libpcap” (“winpcap” on windows). If Snort is running on Linux, the developers of Snort propose to use a modified version of libpcap that implements a shared memory ring buffer[9].

This is based on the fact that if there are some other processes using the resources of the operating system, it can happen that the process who is polling the NIC is not able to hold down its job, because the operating system is not attributing the necessary resources. That means that packets can already be dropped at the network interface card and never been forwarded to Snort. So it would be a very bad idea to use the same machine on which the NIDS runs for other tasks (like a webserver).

Another point affecting the performance of the NIDS is due to the logging capability of Snort. If Snort has to log each occurring event (for example logging every packet), it will use a lot more CPU time to process and perhaps more important, a lot of data has to be written to the harddisc (foreclosed that disc-access is very slow).

Naturally, in an environment with a bitrate around 1Gbps, there has to be either a very powerful system, or the tasks of the NIDS have to be distributed (process the analysing task on another processor).

2) Detection logic
On the other side, the performance of a NIDS can be described by its ability to detect true attacks in the stream of network traffic it observes. The question is, what are the actual dangers on a network, on what event the supervisor has to be alerted? It is about minimising so-called “false-positives”, packets or events who misleadingly reveal alerts. Writing adapted Snort-rules or modifying existing ones will help on in that point. But that mission is very closely associated with the context of the network. That means, that the user of Snort has to know his network and what it is used for (large datatransfers, databases, Internet, etc.). That means, that a system with a fast processor can be less powerful than a system, which has got fewer resources but a better configuration (perhaps there is much more HTTP-traffic, and Snort can process faster due to the “httpinspect”-preprocessor, etc.).
In the further course of this document, we will focus on the first statement mentioned above, because it makes no sense to directly optimise a NIDS if it is even not sure, if the sensor fulfils the condition mentioned above in the first statement.

This fact raises following questions:

**Packets**
- Are there any packets that are not even received by Snort?
- Is Snort dropping some packets

**Memory**
- According to the quantity of network traffic and the number of attacks generated, is there enough memory available?

**Rules**
- Is Snort able to apply the whole set of rules if the network traffic is high (are there enough system resources, high CPU-load)?
- Is the performance of Snort reduced if more rules are applied?

**Logs**
- Are the logfiles using an important amount of memory space on the flashmemory?
- In which tempo the size of the logfiles increases or in which frequency the logfiles have to be fetched (in order that the amount of free disk space is always enough, independent on the network traffic).

It is obvious that these statements are depending on each other. For example logfiles grow faster if there are more rules to consider and so we should fetch the logfiles more frequently.

Naturally, the manner we test our IDS, serves just as a possibility to compare different systems, on which Snort is running. It is not meant as a universal benchmark test of the IDS.

### 8.2 Test Description

In order to get an idea about how our NIDS-sensor will react on high network traffic or big attacks, which is a main purpose to evaluate the real usability of the sensor, we will test it in a test environment. The tests will be made with a PC-system and our embedded system (the CerfBoard) to compare the test results and finally to get a reference-point in order to specify the performance of the embedded network intrusion detection system. The following sections describe the different test scenarios.
8.2.1 No rules

Description
A packet generator will simulate some network traffic and we will observe the behaviour of Snort related to the points mentioned above (Memory, Rules, Logs, etc.). No rules will be applied. That enables to be sure, that the measured performance variations are due to network factor changes like the bitrate or the packet-size.
With that test we have the possibility to find out, in which conditions our NIDS is able to analyse every packet or in other words, to detect the “basic-detection-threshold” of the NIDS.
“Basic-detection-threshold” means that it makes no sense to test the NIDS with a higher bitrate or bigger packet-size than the values set in this test, because at any case, the performance of the NIDS will be less than with this configuration, if there are one or more rules to consider.

Implementation
A traffic generator renders a constant flow. We will use only one UDP-packet and send it n times with a constant delay between the different packets to the sensor. The variation of the bitrate will be done by changing the interframe delay.
The packet size will be varied by adding or withdrawing bytes in the data section of the UDP-packet.

Objective of this test is to find out the necessary bitrate to get Snort to drop packets.

8.2.2 One rule “false packet”

Description
Snort is analysing each packet on the network. It permits to act on a specific content of the packet. A rule like the following raises an alert, if an ip-packet passes by with the content “false packet”:

```
ip any any -> any any (content:"false packet"; msg:"false packet detected");
```

This rule analyses every IP packet passing by (any any -> any any). That means every packet from any source and any port-number to any destination with any port-number.

With a traffic generator, we are creating our own packets of the following type:
ETHER Header

ETHER: Destination: 00-00-00-00-00-00
ETHER: Source: 00-60-B0-21-C8-16
ETHER: Protocol: IP
ETHER: FCS: B1E4B8CB

IP Header

IP: Version = 4
IP: Header length = 20
IP: Differentiated Services (DS) Field = 0x00
IP: 0000 00.. DS Codepoint = Default PHB (0)
IP: ..... 00 Unused
IP: Packet length = 186
IP: Id = 2
IP: Fragmentation Info = 0x0000
IP: .0. ......... Don't Fragment Bit = FALSE
IP: .... 00. ... More Fragments Bit = FALSE
IP: .... 0000 0000 0000 Fragment offset = 0
IP: Time to live = 255
IP: Protocol = UDP (17)
IP: Source address = 15.6.1.0
IP: Destination address = 15.6.1.2

UDP Header

UDP: Source port = 1024
UDP: Destination port = 1024
UDP: Length = 166
UDP: Checksum = 0
UDP: 158 bytes of data

If we send that packet over the network, Snort will raise an alert as shown below (listing of the file /var/log/alert):

```
[**] [1:0:0] false packet detected [**]
[Proprty: 0]
02/16-15:42:53.370105 15.6.1.0:1024 -> 15.6.1.2:1024
UDP TTL:255 TOS:0x0 ID:2 IpLen:20 DgmLen:536
Len: 508
```

Implementation

The proceeding will be the same as described in the previous section ("8.2.1 No rules"). The packet generated by ourselves (mentioned above) will be used as UDP-packet.

Objective of this test is to find out the necessary bitrate to get Snort to drop packets.
8.2.3 Standard set of rules

Description

A Sensor acting as NIDS is useless, if there are no rules specified. The next test we will do consists in applying the standard set of rules and generate a non alerting traffic. That allows specifying the bitrate (notabene without any alerts) that Snort is able to endure without dropping packets.

Implementation

Section “8.2.1 No rules” contains the proceeding for this test.

The standard set of rules is listed below. It is the rules including-section of the Snort configuration file (snort.conf). The version of the rules and the configuration file are at the actual state (v 1.144.2.6 2005/01/13 20:36:20).

```bash
include $RULE_PATH/local.rules
include $RULE_PATH/bad-traffic.rules
include $RULE_PATH/exploit.rules
include $RULE_PATH/scan.rules
include $RULE_PATH/finger.rules
include $RULE_PATH/ftp.rules
include $RULE_PATH/telnet.rules
include $RULE_PATH/rpc.rules
include $RULE_PATH/rservices.rules
include $RULE_PATH/dos.rules
include $RULE_PATH/ddos.rules
include $RULE_PATH/dns.rules
include $RULE_PATH/finger.rules
```

```bash
include $RULE_PATH/ftp.rules
include $RULE_PATH/telnet.rules
include $RULE_PATH/rpc.rules
include $RULE_PATH/rservices.rules
include $RULE_PATH/dos.rules
include $RULE_PATH/ddos.rules
include $RULE_PATH/dns.rules
include $RULE_PATH/ftp.rules
```

```bash
include $RULE_PATH/web-cgi.rules
include $RULE_PATH/web-coldfusion.rules
include $RULE_PATH/web-lis.rules
include $RULE_PATH/web-frontpage.rules
include $RULE_PATH/web-misc.rules
include $RULE_PATH/web-client.rules
include $RULE_PATH/web-php.rules
```
This configuration makes Snort applying 2223 rules.

8.2.4 Standard set of rules with alerting

Description
The next test simulates a DoS-attack to our NIDS. It is the “hardest” stress-test we will realise. The reached throughput in this test will probably be the smallest of all the tests that we achieve. By applying the standard set of rules together with the rule “false packet” written and explained in section 8.2.2 One rule “false packet” we will generate a dataflow to the NIDS formed by the UDP-packet we explained also in section 8.2.2 One rule “false packet”. Every packet will generate an alert.

Implementation
We will apply the same proceeding as in section 8.2.2 One rule “false packet”.

8.2.5 Real scenario

Description
As we have seen, the performance of Snort is depending on a lot of factors. The test described above includes only a dataflow of UDP-packets (which is furthermore always the same). To provide one more meaningful test-result, it would be awesome to put the sensor into a real network and to observe the behaviour. But, if we don’t know in advance exactly on what traffic the sensor will be exposed, the test would have just a statistical character and it would be very hard and imprecise to compare two sensors with each other. The solution would be to record on a real network all the traffic within a fixed period and to replay it on our test-environment. This would approve to make a “real-scenario-test” with TCP-Connections, UDP packets, ARP, DNS, etc.
Varying the delay between the packets allows to in- or decrease the bitrate and it is possible to find the detection-threshold of the NIDS. Naturally the bitrate of the recorded network stream will not be constant; but we can expect that, if packets are dropped, it will happen on throughput-peaks. But in any case, the objective of this test is, as it is for all other mentioned tests in this document, to compare two different systems running Snort who are exposed to exactly the same network stream and to observe the behaviour.

Implementation
With a protocol analyser who enables to record data streams we will record the actions on a real, more or less major network. The analyser must be connected to a span port of a switch, which is positioned in a place on the network where a lot of different traffic can be captured.
8.3 **Implementation**

This section contains details about important elements used to provide the tests.

### 8.3.1 Scheme

We were using the following network-topology as test-environment:

![Network Topology Diagram]

In each test we use a network protocol analyser to monitor the packets that are emitted to the PC or the CerfBoard running Snort. That allows checking, if every emitted packet has been analysed by Snort. If the number of monitored packets isn’t the same as Snort is reporting, then there has to be a loss of packets between the hardware layer (network interface card) and the software layer (libpcap and Snort) on the system running Snort.

### 8.3.2 Tools and elements

This section contains a description of the tools and elements that will be used to accomplish the tests.

**CerfBoard**

<table>
<thead>
<tr>
<th>Processor</th>
<th>XScale-PXA255</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock speed</td>
<td>400 MHz</td>
</tr>
<tr>
<td>BogoMIPS</td>
<td>397.31</td>
</tr>
<tr>
<td>RAM</td>
<td>64 MB SDRAM</td>
</tr>
<tr>
<td>OS</td>
<td>Intrinsyc i-linux 5.0 (familiar linux)</td>
</tr>
</tbody>
</table>

See additional specifications at the beginning of this document.
**PC-System**

In order to compare the test-results of the CerfBoard, we will use a personal computer running a similar software-configuration like on the CerfBoard.

<table>
<thead>
<tr>
<th>Processor</th>
<th>Intel Pentium 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>clock speed</td>
<td>3GHz (2994 MHz)</td>
</tr>
<tr>
<td>BogoMIPS</td>
<td>5931</td>
</tr>
<tr>
<td>RAM</td>
<td>512 MB DDR-RAM</td>
</tr>
<tr>
<td>OS</td>
<td>Debian Sarge</td>
</tr>
</tbody>
</table>

**Agilent Advisor**

The “Agilent Advisor” is a tool to analyse and generate network-packets. We will use it generally to generate network-traffic such as single packets or whole data-streams.

The use of the Agilent Advisor network protocol analyser:

- Generate packets
- Record and replay traffic
- Analyse traffic
  
  Traffic can be analysed in two different modes:
  
  - **Node**
    
    The analyser is a network terminal. That mode allows to attribute an Ethernet-address or/and an IP-address. It is possible to communicate with other elements on the network (for example to “ping” other stations).
  
  - **Monitor**
    
    In that mode the whole dataflow is sluiced through the analyser without modifying it in any way. He hasn’t got a network address and it is not possible to emit any traffic.
Snort alerts

We will use the Snort alert file (/var/log/snort/alert) to check which packets were detected by Snort and if these packets correspond with the generated “intrusions”.

8.3.3 Other interesting tools

This section contains descriptions about other tools used in the course of the project, but who were not used to accomplish the final tests.

D-ITG

“Distributed Internet Traffic Generator” is an open-source tool to simulate traffic on the network. It allows creating fully adjustable traffic with different protocols (UDP, TCP, VoIP, Telnet, DNS). It allows also logging precisely the properties of the generation (bitrate, packetrate, etc.).

Available under [10]

Etherreal

Opensource network protocol analyzer

You may read the Ethereal-manual in order to make it more powerful by disabling certain options like “Updating list of packets in real time” etc.

Available under [11]

packETH

Opensource packet generator - allows generating fully adjustable packets.

Available under [12]

Nessus

A very popular open-source vulnerability scanner.

In correlation with benchmarking Snort, Nessus can be used to provide portscans, CGI-attacks, etc.

Available under [13]
perfmon-graph

perfmon-graph is a Perl script that generates an HTML page with graphs from Snort's perfmonitor preprocessor output by using RRDTool. This tool is just a small Script, it is not an official release and is distributed on no license or conditions. It has been tested on output from the perfmonitor included in Snort 2.0.6, 2.1.0 and 2.2.0.

Here is a small example of the generated HTML-page:

![Graphs from perfmon-graph](image)

It is also possible to visualize the following information:

- Kpackets per second
- Average bytes per packet
- SYN + SYN/ACK packets per second
- Session events per second
- Stream events per second
- frag2 events per second
- CPU stats %

This is a very useful tool to monitor the behaviour of Snort. It can be used for long term testing of the sensor.

We do not use this tool because our tests are just “short-lived”, that means maximally 15 minutes. If in such a short time we would like to have accurate outputs from the perfmonitor preprocessor, the preprocessor itself would eat too much CPU-time and our test results would be inaccurate.

Available under [14]
Snot

Tool that can be used to produce a denial of service (DoS) of Snort. Since the “stream4” preprocessor and due to Snort’s highly optimised nature this tool has lost its reason for existence. The stream4 module provides TCP stream reassembly and stateful analysis capabilities to Snort. Robust stream reassembly capabilities allow Snort to ignore “stateless” attacks such as Snot produce. But Snot can nevertheless be used to test Snort by switching off the stream4 module. Snot gets as input from a text file which contains all rules of Snort (the user has to join the rules himself). As output it creates a dataflow who lets the rules of Snort responding and to register an alert. Older versions of Snort react to such an attack with a denial of service. We did not use this tool because we don’t know exactly what traffic it will produce. If we have to compare two systems it is to favour that both systems are exposed to the same traffic.

Available under [15]

Whisker

A CGI-Scanner, that was long time one of the best tools to check web server systems on their potential security vulnerabilities. It is written in Perl.

Available under [16]

8.3.4 Snort configuration

When Snort is lunched he reads a configuration file called snort.conf. This file contains the configuration of the system (which rules to include, configuration of preprocessors, etc.). We took the standard configuration of Snort as it is delivered under www.snort.org. We did not effect any changes, because it is not the objective of this document to provide the best performing configuration of Snort (as it is already mentioned in the section “8.1 Introduction”). The listing of snort.conf can be found in the attachment section.

8.3.5 Configuration of the network interface card

The network interface card (NIC) of the CerfBoard and the PC were configured as following:

- 100Mb Fast Ethernet Base TX
- half-duplex
- no ip-address attributed
8.4 **Results**

This section contains the measured data together with conclusion from the tests described above (Chapter 8.2 Test Description).

The objective of the tests is to find out the necessary bitrate that the sensor is able to process correctly. By exiting Snort, it displays a summary of detected and dropped packets, risen alerts, etc. So it is possible to simply visualize the statistics made in the course of the test.

Naturally, the measurements are not representing exact values, but it shows the dimension in which the bitrate is situated.
8.4.1 No rules

In this test we could assert for the CerfBoard, that with a packet size of 64 Bytes and a bitrate of ca. 26.06 Mbps, packets are already dropped at a lower layer, so that Snort doesn’t even see the packets.

Here is a table containing the necessary bitrate to drop packets before Snort:

<table>
<thead>
<tr>
<th>Packet size</th>
<th>CerfBoard</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 B</td>
<td>3.66 Mbps</td>
<td>51.2 Mbps</td>
</tr>
<tr>
<td>204 B</td>
<td>11.74 Mbps</td>
<td>90.47 Mbps</td>
</tr>
<tr>
<td>504 B</td>
<td>27.7 Mbps</td>
<td>95.52 Mbps</td>
</tr>
<tr>
<td>756 B</td>
<td>33.25 Mbps</td>
<td>96.68 Mbps</td>
</tr>
<tr>
<td>1004 B</td>
<td>41.26 Mbps</td>
<td>97.48 Mbps</td>
</tr>
<tr>
<td>1304 B</td>
<td>51 Mbps</td>
<td>98.12 Mbps</td>
</tr>
</tbody>
</table>

In this test we could assert for the CerfBoard, that with a packet size of 64 Bytes and a bitrate of ca. 26.06 Mbps, packets are already dropped at a lower layer, so that Snort doesn’t even see the packets.

Here is a table containing the necessary bitrate to drop packets before Snort:

<table>
<thead>
<tr>
<th>Packet size</th>
<th>CerfBoard</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 B</td>
<td>26.06 Mbps</td>
<td></td>
</tr>
<tr>
<td>204 B</td>
<td>63.45 Mbps</td>
<td></td>
</tr>
<tr>
<td>504 B</td>
<td>not dropping any more</td>
<td></td>
</tr>
</tbody>
</table>

PC:

<table>
<thead>
<tr>
<th>Packet size</th>
<th>Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 B</td>
<td>63 Mbps</td>
</tr>
<tr>
<td>204 B</td>
<td>not dropping any more</td>
</tr>
</tbody>
</table>
8.4.2 On rule “false packet”

<table>
<thead>
<tr>
<th>Packet size</th>
<th>CerfBoard</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 B</td>
<td>0.30 Mbps</td>
<td>2.48 Mbps</td>
</tr>
<tr>
<td>204 B</td>
<td>0.94 Mbps</td>
<td>21.04 Mbps</td>
</tr>
<tr>
<td>554 B</td>
<td>2.42 Mbps</td>
<td>52.9 Mbps</td>
</tr>
<tr>
<td>756 B</td>
<td>3.27 Mbps</td>
<td>76.72 Mbps</td>
</tr>
<tr>
<td>1004 B</td>
<td>3.75 Mbps</td>
<td>79.84 Mbps</td>
</tr>
<tr>
<td>1304 B</td>
<td>5.41 Mbps</td>
<td>98.05 Mbps (max)</td>
</tr>
</tbody>
</table>

As in the test before, packets with a size of 64 Bytes at a bitrate of 26.07 Mbps aren’t even forwarded to Snort (for the CerfBoard).

CerfBoard:
- 64 B: 26.07 Mbps
- 204 B: 62.25 Mbps
- 504 B: not dropping any more

PC:
- 64 B: 63 Mbps
- 204 B: not dropping any more
8.4.3 Standard set of rules

We have done the same test again to find out the bitrate threshold necessarily that packets aren’t forwarded to Snort. Snort applies 2223 rules, and the bitrate dimension is located in the same field.

CerfBoard:

<table>
<thead>
<tr>
<th>Packet size</th>
<th>CerfBoard</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 B</td>
<td>2.25 Mbps</td>
<td>42.81 Mbps</td>
</tr>
<tr>
<td>204 B</td>
<td>5.79 Mbps</td>
<td>90.469 Mbps (max)</td>
</tr>
<tr>
<td>554 B</td>
<td>10.61 Mbps</td>
<td>96.267 Mbps (max)</td>
</tr>
<tr>
<td>756 B</td>
<td>12.5 Mbps</td>
<td>97.238 Mbps (max)</td>
</tr>
<tr>
<td>1004 B</td>
<td>13.13 Mbps</td>
<td>97.907 Mbps (max)</td>
</tr>
<tr>
<td>1304 B</td>
<td>14.25 Mbps</td>
<td>98.381 Mbps (max)</td>
</tr>
</tbody>
</table>

PC:

| 64 B      | 26.47 Mbps |
| 204 B     | 61.9 Mbps  |

not dropping any more
8.4.4 Standard set of rules with alerting

<table>
<thead>
<tr>
<th>Packet size (Bytes)</th>
<th>CerfBoard [Mbps]</th>
<th>PC [Mbps]</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 B</td>
<td>0.29</td>
<td>2.48</td>
</tr>
<tr>
<td>204 B</td>
<td>0.83</td>
<td>17.934</td>
</tr>
<tr>
<td>554 B</td>
<td>2.07</td>
<td>48.113</td>
</tr>
<tr>
<td>756 B</td>
<td>2.71</td>
<td>61.691</td>
</tr>
<tr>
<td>1004 B</td>
<td>3.40</td>
<td>74.622</td>
</tr>
<tr>
<td>1304 B</td>
<td>4.18</td>
<td>90.716</td>
</tr>
</tbody>
</table>

necessary bitrate to drop packets

![Graph showing throughput vs. packet size for CerfBoard and PC](image-url)
8.4.5 Real scenario

The recorded network traffic in this test contained a lot of very small packets (<80 Bytes). We could observe that, if we increase the bitrate, at a fixed rate (about 2 Mbps average while 10 seconds), some packets aren’t even forwarded to Snort (they are already dropped at libpcap). It is the same phenomenon as we asserted in the tests below. This statement indicates, that the CerfBoard have to be used very carefully. The summary of Snort indicates that about 10 % of the packets were dropped, but about 40 % of the total quantity of packets was even not seen by Snort.

In this test it is hard to say at which bitrate Snort drops packets because on real networks the bitrate is changing continuously and the exact measurement of the bitrate is very expensive, since a bitrate value is finally an average value. For example in a dataflow which usually is never climbing higher than 2 Mbps there could be within 0.6 milliseconds 100 packets of 70 Bytes. That means that within this 0.6 milliseconds there was an average bitrate of 93 Mbps. In this case, neither the CerfBoard nor the PC would have been able to detect even the packets.

At this test it is not possible to draw a conclusion because the interpretation of the test results is to approximate. Test results were reporting that at 2 Mbps the CerfBoard is already dropping packets at libpcap-level. But the value “2 Mbps” is an average of 10 seconds (calculated by the “agilent advisor”, see section “8.3.2 Tools and elements” for further details).
9 TEST EVALUATION

As we can see, there are some big differences between Snort on PC and Snort on a CerfBoard. Where are these differences coming from? The major difference between the two systems is the processor. He is the most important element for the processing speed. We chose consciously a fast CPU for the PC, because actually, if we would build up a NIDS formed of 2223 rules, we would take a powerful machine. But differences are obviously. Remarkable is particularly the loss of packets at libpcap-level. This problem represents a big disadvantage of the CerfBoard.

The difference between dropping packets at libpcap-level and at Snort-level is the following: If Snort reports that some packets were dropped, we can try to optimise our system by removing useless rules. But if nobody is reporting something (that means we have lost packets at libpcap-level), we even don’t know that there is something wrong.

If the CerfBoard would be used as NIDS, it would be possible for someone to hide a “hack” into small packets, and the NIDS would not even see a trace of it. That’s why we rather discourage to use a CerfBoard as a simple NIDS.
10 CONCLUSION

When we started our work, we didn't even think about getting as far as we are by now. We had a lot of small problems that prevented us from getting further. But after all we still succeeded, got the platform up and running and could even do some tests. If you follow our documentation you should have no problems at all getting Snort running on the embedded system.

On the performance side, we are disappointed to say the least. After some initial tests we saw a working device logging everything correctly. But then we started serious testing and it showed up that Snort doesn't even see all the packets that are on the cable after a certain bitrate. Apparently the system is too slow to cope with most Ethernet traffic. This doesn't even depend upon how many rules are loaded.

So we now have mixed feelings. We have a device that - if you only look at Snort's stats (Snort-level) - seems working right, but as soon as you monitor the actual traffic you'll see that Snort receives only a bunch of the packets on the network.

So it seems the problem lies in libpcap. Apparently it simply isn't able to do its work fast enough.

There seemed to be a solution: Someone made an improved libpcap for Linux, which should be faster than the original one. Due to lack of time we didn't try it out. But we don't believe it would speed up things fast enough when looking at the test results (see test evaluation).

So finally our conclusion is that when you are really considering putting in place such an embedded NIDS, the CerfBoard 255 isn't the right choice at the moment. But since development never stands still, we expect its successors to be usable soon...

In the meantime we have to take a look at alternatives. During our project, a very interesting device just became available: The Apple Mac mini [17]. For around the same price as the CerfBoard it delivers a huge performance jump being only around double as big. It shouldn't be any problem to install Linux on it and we already heard of Snort running on Mac OS X (BSD-Unix based operation system).
11 ATTACHMENTS

- configure.script
- snort.conf
- Cahier des charges
- Planning

12 REFERENCES & IMPORTANT LINKS

1. http://www.intrinsyc.com
3. On Intrinsyc CDROM: i-linux-5.0/documentation/index.html -> I-Linux Basics -> Connecting the CerfBoard
7. http://www.packetfactory.net/libnet
11. http://www.ethereal.com