

The Snort BEEP-Interface - programmer's manual

version 1.00.00 - June 30, 2007



<http://www.positif.org>

Contents

1	Introduction	3
2	Technology	3
2.1	Snort and the somec plugin	3
2.1.1	Snort	3
2.1.2	The somec plugin	3
2.2	BEEP	3
2.2.1	beepcore-c	3
2.2.2	beepng and piep profile	4
2.3	PIEPOverBEEP	4
2.4	Scratchbox	4
3	Proof of Concept	4
3.1	Implementing Proof-Of-Concept	4
3.1.1	Communication	4
3.1.2	Alert Count	6
3.2	Design features based upon encountered problems	7
3.2.1	Wait before send	7
3.2.2	Buffer Problems on the iPAQ	8
4	Running the example	8
4.1	Starting Up	9
4.2	Known Problems	9

1 Introduction

The goal of this project was to proof the concept of a communication between the packet sniffer program Snort [Snort 2006] and a PIEPoverBEEP server (an interface for the POSITIF specific protocol and messages based on BEEP see section 2.3) via BEEP [2] [Beep Community 2006], using the beepcore-c library. [1] [Beepcore-c Community 2006]. In addition Snort and BEEP had to run on an iPAQ, featuring familiar Linux. [3] [Familiar Community 2006]

The example message sent from Snort to the PIEPoverBEEP server in this proof-of-concept was produced by the spp somc.c Snort plugin by Matthias Schrfelbauer and Andreas Weitzer. [4] [5] [Weitzer et al. 2005]. It was an alert message, produced when unhallowed traffic was detected. The goal was to periodically send these alert messages from Snort to a PIEPoverBEEP server, and receive an answer message in return. As proof-of-concept, an exemplary extension of the spp somc.c snort plugin had to be implemented, that included the functionality of sending BEEP message via the beepcore-c library, and receiving its answer in return.

2 Technology

Various different technologies were addressed in this project which are described in the following.

2.1 Snort and the somc plugin

2.1.1 Snort

Snort is an open source network intrusion prevention and detection system utilizing a rule- driven language, which combines the benefits of signature, protocol and anomaly based inspection methods. With millions of downloads to date, Snort is the most widely deployed intrusion detection and prevention technology worldwide and has become the de facto standard for the industry.¹

As Snort is available as Open-Source it was easy to extend / minimize Snort, and port it onto the iPAQ. For further information see [Schrofelbauer et al. 2005].

2.1.2 The somc plugin

This Snort plugin by Matthias Schrofelbauer and Andreas Weitzer used Self Organizing Maps (SOMs) for determining if a given packet was encrypted or not. See [Schrofelbauer et al. 2005] for details.

The training plugin (spp somc.c) printed an alert message to the console, when a malicious packet was found. This plugin had to get extended to send the messages via BEEP.

2.2 BEEP

2.2.1 beepcore-c

BEEP is a P2P application protocol framework for connection-oriented, asynchronous interactions. BEEP permits simultaneous and independent exchanges within the context of a single application user-identity, supporting both textual and binary messages.² In other words, it is easy to implement reliable protocols based upon beep. Several RFCs deal with the different aspects like the BEEP-core itself (RFC 3080), or mapping BEEP on TCP (RFC 3081) or the design of Application Protocols (RFC 3117). See [Beep Community 2006] for further references.

One implementation of the BEEP-core, written in C is beepcore-c ([Beepcore-c Community 2006]). It was chosen to use beepcore-c, as Snort and its plugins were also written in C.

2.2.2 beepng and piep profile

beepcore-c ships with an example implementation and with example profiles. For this project beepng was used, as well as a provided profile piep profile, that was capable of speaking PIEPoverBEEPs (see next section) language.

beepng uses the beepcore-c functionality to register example profiles, for sending and receiving BEEP-based messages from a compatible server.

2.3 PIEPoverBEEP

For this example, an implementation of PIEPoverBEEP by Stefan Kraxberger was used, which provided an interface for the POSITIF specific protocol and messages based on BEEP RFC 3080 and RFC 3081. PIEPoverBEEP was written in Java, and used the beepcore-java3 implementation. For more information see the documentation of PIEPoverBEEP.

PIEPoverBEEP was not modified in this project, but used as server, to which Snort had to send its messages.

2.4 Scratchbox

Because both beepng and snort have to run on an iPAQ it was necessary to compile them for that specific hardware. For that purpose Scratchbox4, a cross-compilation toolkit, was used. For more details about setting up Scratchbox, and configuring it for iPAQ, see [Weitzer et al. 2005].

3 Proof of Concept

Given at start were Snort, with the alert producing profile *somc*, as well the example beepcorec implementation beepng, featuring a piep profile. In the end, only the files

`< snort - base > /src/preprocessors/sppsomc.c`

in Snort, and

`< /beepcore - c - base > /beepcore - c/threadeddos/examples/beepng.c`

in beepcore-c had to be touched.

3.1 Implementing Proof-Of-Concept

The Proof-Of-Concept consists of two main parts: Communication between Snort and the PIEP Server using beepcore-c, and the counting of packages, so that alerts are only sent if in a given time intervall the percentage of malicious packages exceeds a specifies threshold.

3.1.1 Communication

At first it was thought to compile the BEEP functionality of beepcore-c right into the Snort plugin, to send the message directly from Snort to the PIEPoverBEEP server. But when including the necessary headers of the beepcore-c library, namespace conflicts occurred in Snort. As there was no easy or efficient way to solve this problem, another way of communication had to be established.

It was chosen to use two separate applications snort with the *somc* plugin, and an modified beepng running the piep profile, communicating between each other over a text file. *spp some.c* was extended to write the message into a text file, while *beepng.c* was extended to read the message from this file. To prevent read / write collisions on that file between the applications, also a lock file was introduced.

Listing 1 shows the `sendBEEPMessage()` function added to *spp somc.c*, that writes out the message file, while listing 2 shows parts of the modified `beepng()` function in *beepng.c*, that reads the message file into a buffer. The buffer is then copied into another buffer *b1*, together with additional information needed for the piep profile, which gets send to the PIEPoverBEEP server.

Listing 1: Parts of new sendBEEPMessage() in spp somc.c

```

1 void sendBEEPMessage ( int id ,u_char * message , float value , int intvalue)
2 {
3 /* [...] stripped of declarations */
4
5 beep_message_id_ ++;
6
7 lockFile = fopen ( lock_file_name ,"r"); /* Check for lock */
8 while ( lockFile != NULL )
9 {
10 /* [...] wait */
11 lockFile = fopen ( lock_file_name ,"r"); // idle
12 }
13 lockFile = fopen ( lock_file_name ,"w"); /* acquire lock */
14
15 /* Create and Send message */
16 outFile = fopen ( message_file_name ,"w");
17
18 fprintf ( outFile , " <? xml version =\`"1.0\`" encoding =\` UTF -8\`"? > \
19 <STM id =\`"% d%d%d%d\`">< message >%s </ message ></STM >\n" \
20 , int_4 , int_3 , int_2 , int_1 , message );
21 fclose ( outFile );
22
23 fclose ( lockFile ); /* release lock */
24 remove ( lock_file_name );
25 }

```

Listing 2: Parts of modified beepng() in beepng.c

```

1 /* [...] stripped of declarations */
2
3 lockFile = fopen ( lockfile ,"r"); /* Check for lock */
4 while ( lockFile != NULL )
5 {
6 /* [...] wait */
7 lockFile = fopen ( lockfile ,"r"); // idle
8 }
9 lockFile = fopen ( lockfile ,"w"); /* acquire lock */
10
11 inFile = fopen ( messagefile ,"r");
12 if ( inFile != NULL )
13 {
14 /* get file size */
15 fseek ( inFile , 0 , SEEK_END );
16 lSize = ftell ( inFile );
17 rewind ( inFile );
18
19 // copy the file into the buffer .
20 inBuffer = (char *) malloc ( lSize );
21 fread ( inBuffer ,1, lSize , inFile );
22
23 fclose ( inFile );
24 }
25 fclose ( lockFile ); /* release lock */
26 remove ( lockfile );
27
28 /* [...] */
29
30 // copy buffer into send buffer
31 sprintf (b1 , "%s%s", DEFAULT_CONTENT_TYPE , inBuffer );

```

3.1.2 Alert Count

Snort is only writing out a message to beepcore-c if in a given time intervall the the percentage of alerts exceeded a specified threshold. Snort is incrementing the message id at each writtenout message.

Listing 3: Alert count min distance() in spp somc.c

```

1 // [...]
2 i f ( SOMC_MODE_ALERT )
3 {
4 i f ( distance_min > SOMC_MIN_DISTANCE_THRESHOLD )
5 {
6 // [...]
7
8 // Calculate Percentag in Time Intervall
9 alarm_package_percentage_ = (1.0 / (double)
num_packages_since_last_message_ ) \
10 * (double) num_alerts_since_last_message_ ;
11
12 // Check on Time Intervall for percentage calculation
13 time (& end_time_ );
14 time_diff_ = difftime ( end_time_ , start_time_ );
15 i f ( time_diff_ > PERCENTAGE_ALERTS_TIME_INTERVAL )
16 {
17 time (& start_time_ ); // set new start time
18 num_alerts_since_last_message_ = 0;
19 num_packages_since_last_message_ = 0;
20 send_message_ = 1;
21 }
22 e l s e
23 send_message_ = 0;
24
25 // Check if send and if threashold is exceeded
26 i f ( ( send_message_ ) &&
27 ( alarm_package_percentage_ >
MIN_PERCENTAGE_ALERTS_BEFORE_MESSAGE ) )
28 {
29 printf ("\ nSending warning message after %g seconds and alert -
ratio of %f.\n", time_diff_ , alarm_package_percentage_ );
30 sendBEEPMessage ( somc_alert_counter_ , message , distance_min ,
somc_alert_counter_ );
31 /* time (& start_time_ ); // set new start time
32 num_alerts_since_last_message_ = 0;
33 num_packages_since_last_message_ = 0; */
34 }
35 }
36 }
37 // [...]

```

Beepcore-c is only sending a read-in message from Snort to the PIEP-Server if the message id is different from the message id last sent.

Listing 4: New-Message check in beepng() in beepng.c

```

1
2 /* [...] read new newMessageId_ from message */
3
4 // compare with old id
5 i f ( strcmp ( newMessageId_ , oldMessageId_ ) == 0)
6 {
7 printf ( " Not sending message , because this message was sent already .\n

```

```

");
8 send_message_ = 0;
9 }
10 e l s e
11 {
12 strcpy ( oldMessageId_ , newMessageId_ ); // remember this id as old id
13 printf (" Sending message , because this message was not sent yet .\n");
14 send_message_ = 1;
15 }
16
17 // [...]

```

3.2 Design features based upon encountered problems

During the implementation and testing process several problems occurred. Based upon those problems additional features had to be included into the design. They are important because further implementations must consider them as well.

3.2.1 Wait before send

To prevent beeping of flooding the network by sending to much alert messages a wait feature was included. beeping checks that at least a given amount of time has passed before sending the message. As Snort has to check every packet, and every packet may lead to another alert, sending to much messages may overwhelm systems capacity easily. Listing 5 shows a generic wait method, used both in beeping and spp some.c to prevent from message flooding.

Listing 5: Generic wait function in c

```

1 # include <time .h>
2 # define MIN_SECONDS_WAIT 4.0
3
4 time_t start_time_ ;
5 time_t end_time_ ;
6 double time_diff_ ;
7
8 /* [...] */
9
10 time (& end_time_ ); // acquire new end time
11 // calculate diff to last start time
12 time_diff_ = difftime ( end_time_ , start_time_ );
13
14 // only doFunction if at least MIN_SECONDS_WAIT seconds have passed .
15 i f ( time_diff_ > MIN_SECONDS_WAIT )
16 {
17 doFunction ();
18 time (& start_time_ ); // set new start time
19 }

```

Snort detecting BEEP packets as malicious

As the beeping-c library sends its message unencrypted by default, Snort detected those packets as unallowed, and produced an alert. Because of the alert a new BEEP message was send, and so on. To prevent this, messages from or to the PIEPOverBEEP server had to be ignored. This can be done by defining a BPF5 file for Snort. Listing 6 shows an example BPF-file for Snort.

Listing 6: BPF-file for Snort

```

1 not (dst net <ip -num > and dst port <portnum >) and
2 not (src net <ip -num > and src port <portnum >) and
3 [...]

```

Snort must be started with the parameters `-F ;bpf file;` to ignore packets from or to the given addresses. An example bpf file can be found at `/snort-2.4.2/src/bpf rules.conf`. Another solution to solve that problem would be to configure BEEP to send its messages encrypted. But one advantage of the BPF filter is, that Snort does not even look at the listed packets, so Snort does not waste resources on known trusted packets.

3.2.2 Buffer Problems on the iPAQ

Probably there is a bug in the c-standard library for the iPAQ, as inexplicable buffer-errors occurred, when determining the needed buffer size with `ftell`. This was fixed by only producing a fixed-size message, and allocating only a fixed-size buffer.

It is important to test implementation not only on a desktop system, but also on the iPAQ, after compiling in Scratchbox.

4 Running the example

After compiling Snort with the modified `spp somc.c` plugin in Scratchbox, configured for iPAQ (see [Weitzer et al. 2005]), and also compiling the modified `beepcore-c beepng` example, using the provided `piep` profile, the two binaries have to be copied onto the iPAQ. Also the `somc.rules` and the BPF file, containing the `PIEPoverBEEP` servers IP and port, must be provided.

When running the example not on the iPAQ; but on a standart Linux system, just compile Snort and Beepcore on that platform and run it from shell.

Important: Note that both `beepng` and `snort` run in the same directory. If not the paths to the message and lock files have to be adjusted in `spp somc.c` and `beepng.c` before compiling.

Also note that Snort and Beep must be started as root, to be able to check traffic and to be able to communicate with each other.

4.1 Starting Up

First check that `PIEPoverBEEP` is up and running. When testing on the iPAQ `PIEPoverBEEP` will most probably run on a different machine. Type

```
javaeu.positif.piep.BEEPServerThread - config./resources/piep.config
```

to start the `PIEPoverBEEP` server. You may have to check that `PIEPoverBEEP`'s `lib/` directory is in the `CLASSPATH`.

On the iPAQ or PC start Snort by typing

```
./snort - csomc.rules - F < bpf file > -i < device >
```

Also on the iPAQ or PC start BEEP by typing

```
./beepng -p < PIEPoverBEEP - port >< PIEPoverBEEP - ip >
```

If the given PIEPoverBEEP server is up and running, beepng reads in the messages produced by snort, and sends them to the PIEPoverBEEP server, who sends a response if the sent message was valid.

4.2 Known Problems

If beepngsticks, most probably no answer of the PIEPoverBEEP was received. This is usually the case if non valid xml was send as message. Check the PIEPoverBEEP servers output for details where the message was wrong.

Compiling problems usually occur because of missing packages. Use make clean and ./configure in Snort and beepcore-c to set up the system. Then just call make install. Usually ./configure tells you what mackage is missing. Install it with your Linux distributions package manager.

References

- [1] Beepcore c Community. Beep core c, 2007. [l.v. 26/01/2006, l.u. January 2006].
- [2] Beep Community. Blocks extensible exchange protocol. <http://www.gnu.org/software/autoconf/>, 2006. [l.v. 26/01/2006, l.u. January 2006].
- [3] Familiar Community. Linux for ipaq, 2006. [l.v. 26/01/2006, l.u. January 2006].
- [4] Inc. Sourcefire. Snort, 2007. [Online; accessed 8-June-2007].
- [5] Matthias Weitzer Andreas, Schröfelbauer. Porting snort on ipaq howto. <http://www.cis.hut.fi/research/som-research/nncr-programs.shtml>, 2005. [IAIK Seminar-Project. 2005]].