Packet Auditing HOWTO

C.S. Peron (maneo@bsdpro.com)
Tue Jan 8 09:36:38 CST 2002

Abstract: This document is intended to introduce a new user to the ipex packet auditing Package. C.S. Peron is the lead developer and founder of the ipex project.

Introduction:

Ipex is a free utility packet-auditing package developed primarily to act as a debugger for developmental and experimental programs for programmers and system administrators. Although it is similar to **tcpdump(1)**, it has quite a few powerful options that **tcpdump(1)** does not, it was designed around **libpcap** so they do share allot of features, and they can be used by one another.

Disclaimer:

The author of this document is not responsible for any damages incurred due to actions taken based on this document. This document is meant as an introduction on using **ipex** to debug network and protocol problems. If you do not feel comfortable taking responsibility for your own actions, you should stop reading this document now.

Copyright:

Where To Obtain ipex:

Ipex(1) is free and available at http://ipex.sourceforge.net. Ipex is released under the BSD license agreement. If you are un-familiar with this license please view the COPYING file located with the package.

Building And Installing Ipex From Source:

NOTE: Replace "X.X" with the release numbers you download

```
% zcat ipex-X.X.REL.tar.gz | tar –xvf –
% cd ipex-X.X.REL
% ./configure
% gmake
% gmake install
% gmake clean
```

Pcap Expressions:

What is a "pcap expression"? Simple really; a pcap expression determines which packets are to be captured by the device responsible for capturing packets. In **FreeBSD's** case its **/dev/bpf**. More information on **bpf** can be attained by reading the **bpf(4)** man page. In short you pass a string to ipex, and it will feed it through to **libpcap's** internal compiler, which generates the correct rule set to capture the packets you specifically want.

For more information on the allowable primitives please see the "*pcap_exprs.txt*" included in the doc/ directory of the package.

Ipex General Syntax:

Ipex has a variety of different options, reading the man page can be intimidating so in this document we will spell it out easily.

```
% ipex -help
usage: ipex [-hvnqbHxLGdO] [-f config] [-r pattern] [-o outfile]
            [-c count] [-i interface] [-u uid] [-P op=arg]
            [-t time] [-w file] [-F file] [expression]
%

SYNOPSIS: ipex [options] [pcap expression]
```

NOTE: You must specify your options before your pcap expression.

So what does this all mean? We will step through each option's required syntax and give helpful examples along the way.

Reading Your Expressions From an Infile:

Lets say you have an awfully large pcap expression or string and really don't want to have to type it all out on the command line. Ipex allows you to specify an file on the command line that ipex is to reads its configuration from. The option is **–f.**

```
% ipex –f myrules.txt
```

So what would this above example achieve? Ipex will open the file *myrules.txt* and read in any data located in the file. Skipping over comments. What are allowable comments?

```
/* this is a comment, and would be ignored by ipex */
# this is a comment and would be ignored by ipex
// this is a comment and would be ignored by ipex
; this is a comment and would also be ignored.
```

Here is a very brief example of a config file that would capture all packets going to source or destination port 25.

```
% cat myrules.txt

/*
 * ipex example -f for the HOWTO
 */
src port 25 or dst port 25
```

Locating Specific Text Patterns In Packets:

Ipex has two internal regexp handlers. Meaning, you can supply a basic and extended regular expression. This feature requires that you know something about regular expressions however if you don't we will give some very basic examples on what you might use this feature for.

```
% ipex –r 'foobar|barfoo'
```

The above example would process a packet only if it contained the string(s) "foobar" OR "barfoo". It's fairly straightforward. The '|' acts as an OR. The next example will display a little more advanced regular expression.

```
% ipex –Gr '[0-9]\{3\}-[0-9]\{4\}'
```

This particular example would process only packets, which contain a North American telephone number. I.E. 222-4455.

```
% ipex –Gr '[Aa] [Cc]at'
```

This example would process the packet provided the followings strings were located within:

- "a cat"
- "A cat"
- "A Cat"

NOTE: The –G flag enables the use of basic or traditional regexps. –r alone is not case sensitive.

Packet Hexdumps:

There are many reasons why dumping a packet in hexadecimal is useful. The biggest reason that comes off the top of my head, would be for protocol or network development. Most protocols terminate and function off invisible characters, if this is the case, doing a straight text dump will be of no real use to you. Ipex allows you spill the guts of each packet in hexadecimal.

```
% ipex -x src port 6667
ipex: open live: listening on: ed0
10:51:45.150750 207.45.69.68:6667 >> 24.66.7.3:1589 tcp AP seq=214134387
ack=1475522503 win=28740
0x0100|  3a43 6c6f 636b 4456 4121 6761 757a 6540    |:ClockDVA!gauze@
0x0110|  3231 362e 3836 2e31 3338 2e34 3020 5052    |216.86.138.40.PR
0x0120|  4956 4d53 4720 2363 203a 746f 2061 6c6c    |IVMSG.#c.:to.all
0x0130|  6f63 6174 6520 636f 6d70 7574 6572 206d    |ocate.computer.m
0x0140|  656d 6f72 7900 0a                          |emory..
```

This is an example of the output. As you can see, there was an IRC client running on this machine. Each byte of this packet is no longer a mystery.

Capturing Packets By Pid:

Packets by pid? Yes indeed. When debugging a network daemon or a developmental protocol one of the biggest problems with conventional packet sniffers is you cannot track packets via pid or process I.D. This feature has proven to be fairly handy in a few situations.

The conventional way of doing this is, doing a **netstat(1)**. Seeing the src and destination ports for the connections you want, then creating your pcap expression. This can be somewhat tedious. Ipex implements a **–P** which has a strange usage: [-P op=arg]. So what the heck is an op? What the heck is an arg?

Ipex accepts the following "ops":

"uid" - Process a packet via effective UID.
"ruid" - Process a packet via Real UID
"pid" - Process a packet via PID

So what would an "arg" be? The arg would represent the numeric value of the "op". For example lets retrieve my UID from the /etc/passwd file.

```
% cat /etc/passwd | grep csperon
csperon:*:1002:1005:csperon:/home/csperon:/bin/csh
%
```

We can see that user "csperon" has a UID of 1002. Say for example user csperon was logged into our system we could process packets from any of the processes that he owns by doing something to the effect of:

```
% ipex -P uid=1002
```

Here is another example; this time we want to monitor traffic to our telnetd. We could do something like this:

```
% sockstat -4 | grep telnetd
root     telnetd  41957 0 tcp4 24.66.7.3:23  204.112.133.186:3144
root     telnetd  41957 1 tcp4 24.66.7.3:23  204.112.133.186:3144
root     telnetd  41957 2 tcp4 24.66.7.3:23  204.112.133.186:3144
%
```

Here we see our PID of our telnetd is "41957". Now that we know our pid we can start watching packets going to and from it.

```
% ipex –P pid=41957
```

And that's all there is to it.

Tcpdump Migration:

If there is something that **tcpdump(1)** does better then ipex, you can dump the packets in a raw form to a file, then process it at a later time with either **tcpdump(1)** or **ipex(1)**.

So why would you want to dump packets in their raw form to a file? Well let's say for example you want to capture every packet going through a data-link but you want to process different packets for different purposes at a later date. This would be an application:

```
% ipex -F pdump -c 2
%
```

We capture 2 packets from the default network interface, lets say we wanted to do a hex dump of the two packets:

```
% ipex -x -w pdump
ipex: reading packets from file: pdump
11:29:49.776010 204.112.133.186:3144 >> 24.66.7.3:23 tcp A seq=942897595
ack=3018981487 win=32319
11:29:49.776323 24.66.7.3:23 >> 204.112.133.186:3144 tcp
APseq=3018981487 ack=942897595 win=11395
0x0100|  6970 6578 3a20 6f70 656e 206c 6976 653a       |ipex:.open.live:
0x0110|  206c 6973 7465 6e69 6e67 206f 6e3a 2065       |.listening.on:.e
0x0120|  6430 000a                                     |d0..
```

Ok, nothing really interesting there, lets see what tcpdump might have done had it handled the packets instead of ipex.

```
% tcpdump -r pdump
11:27:54.837503 userBb166.videon.wave.ca.3144 > icmp.dhs.org.telnet: .
ack 18623
33107 win 16254 (DF)
11:27:54.837618 icmp.dhs.org.telnet > userBb166.videon.wave.ca.3144: P
1:37(36)
ack 0 win 33580 (DF) [tos 0x10]
%
```

ipex can also take the output of a raw **tcpdump(1)** dump (created with –w) an process the packets with ipex's –w option. So they are fairly flexible.

Limiting Output:

There are two mechanisms in place to limit the number of packets you capture, or the amount of time you wish to capture the packets for. The **–c** and **–t** options allow you to control and limit the output.

```
% ipex –c 20
```

The above example would capture 20 packets, then exit. The **–t** is a little more involved and a little more powerful. **–t** requires a time, the time can be suffixed as: **d, h, m** and **s**.

- "**s**" – Represents Seconds
- "**m**" – Represents Minutes
- "**h**" – Represents Hours
- "**d**" – Represents Days
-

```
% ipex –t "3d 7m 2s"     # run ipex for 3 days, 7 minutes and 2 seconds
```

Running Ipex as a Daemon Process:

If you require, you have the option of running ipex as a daemon, in the background and have it write all the packets in human readable form to an output file. Lets say you have a cron job that connects to a remote machine and performs a data transfer once every week. Every so often either you or the remote machine drops packets and you can't figure out if its you that's dropping them or the remote machine.

This next example is something you could do to investigate this situation more completely.

```
% ipex –b –o packet.txt –t "14d" src port foo or dst port foo
```

The above example would launch ipex into the background. Ipex will write it's pid to /var/run/ipex.pid. the –o option indicated to ipex that you would like it to write all data to the file called "packet.txt". Ipex will run for exactly 14 days and process packets that are going to or coming from port ``foo''.

NOTE: -b requires –o. Once ipex fork()'s and turns into a daemon it closes standard in, standard out and standard error. Also, -o does not require –b, you can dump to a file using –o regardless.

Specifying A Network Interface:

For machines that have multiple network interfaces, you can specify which interface you would like to use for the operation. This interface can be specified using the **–i** option.

```
% ipex –i xl0
```

This would open */dev/xl0* and watch the packets coming and going off that interface.

Dumping Just Headers:

If you are not interested in dumping the packet contents, but would just like to see which packets are coming and going through a network interface, you can achieve this with the **–H** option.

```
% ipex –H
```

Dumping the bpf_insn Struct:

This feature is only useful for people who have located in a bug in either /dev/bpf, pcap or ipex itself. When pcap_compile is called, it takes the supplied (``pcap expression") and generates something called a (``struct bpf_program"). This struct is intern fed to bpf via an ioctl. One of this structure's elements is (``struct bpf_insn"). This determines which packets are to be captured and which are to be ignored. To view the contents of this struct use **–d**.

```
% ipex  -d src port 21 or dst port 21
```

Enabling the PCAP Code Optimizer:

During our late nights as programmers (and there are allot of those ☺) we may start getting absent minded and feed a pcap expression that isn't really efficient or even redundant at times, one way to help with this problem is something called the pcap code optimizer and you can enable it in ipex by using the **–O** flags.

```
% ipex –O src port 21 or dst port 21 and src port foo or dst port foo
```

And That concludes the ipex beginner tutorial. Now you should know enough to consult the man page for more options and examples.