GIAC CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Security Essentials: Network, Endpoint, and Cloud (Security 401)"
at http://www.giac.org/registration/gsec

A tool for running Snort in dynamic IP address assignment environment

Shin Ishikawa
February 16, 2002

Introduction

The purpose of this paper is to detail the creation of a small tool program which aids the operation of the Snort IDS in dynamically assigned IP address environment. The configuration file of Snort (snort.conf) specifies IP numbers for the monitored network and servers. For the non-permanent IP address subscriber sites, which are the case for the most of ADSL users, these parameters need be updated every time the data link connection reset and new address is assigned. A set of small programs is written to automate Snort configuration update for the connection using PPPoE.

A program monitors the IPCP traffic and dumps PPPoE frames with IPCP negotiations. A script interprets the IPCP negotiation and sees if new IP addresses are agreed upon. If it is, snort.conf file is updated with the new IP addresses for HOME_NET and DNS_SERVERS variables and signal is sent to the running Snort process to restart and reflect the change. This paper examines the program form the secure code writing point of view and also discusses the meaning of running Snort IDS in home user environment in the age of "always connected to the Net".

Hardware and Software Environment of sample Home User

Figure 1 shows the environment for running the Snort IDS. The tool programs run in the same box the Snort IDS running.

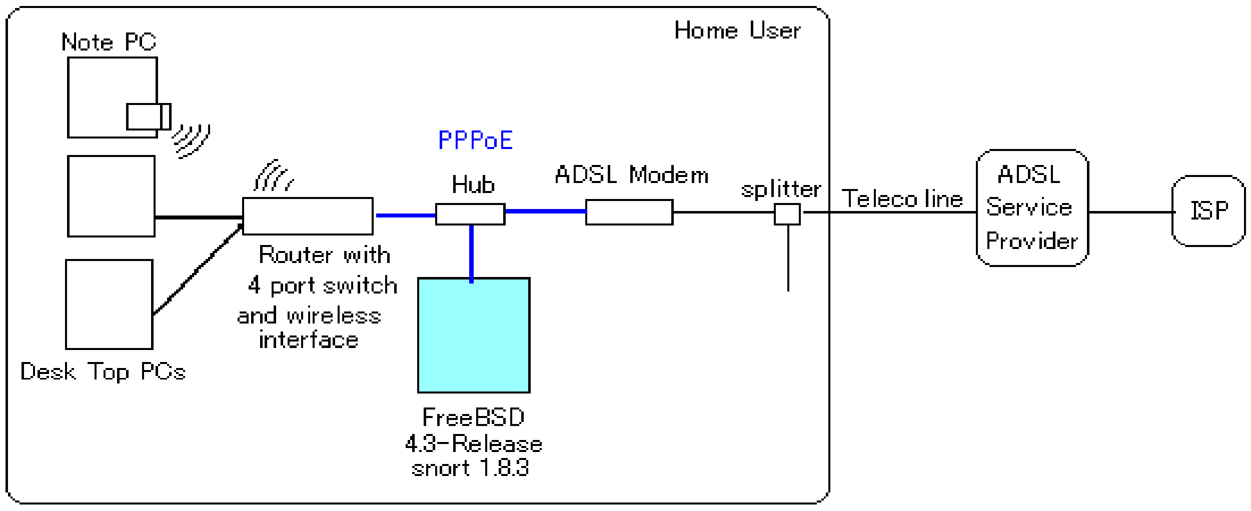

Figure 1 ADSL Home User Configuration

This home user site is connected to ISP with 1.5Mbps(down) and 512Kbps(up) ADSL. To use mobile note PC with wireless card, a router with wireless interface is installed. The built in four port switch of the router make it impossible to monitor all traffic inside LAN. Because of these, the best monitoring point for the traffic between the site and the Internet is between the router and the ADSL modem. Normally, straight cable is used to connect the two devices. In our situation, in order to attach a monitoring PC, a hub is placed. A monitoring PC is running snort-1.8.3 package port for FreeBSD 4.3-

Release. Its NIC(ed0) is not configured to any network layer protocol. It is used to "sniff" traffic in promiscuous mode.

The network between the ADSL modem and the router is 10Base-T Ethernet and the PPPoE protocol is used. Although tcpdump(version 3.5/libpcap version 0.5) interprets ether frames on the LAN at PPPoE level, the Snort IDS 1.8.3 has no problem in monitoring IP payload within the PPPoE frames. The router is a low end product for home use, priced around two hundred dollars, and does not support SNMP or other useful management interfaces. Browser access from inside LAN is used to setup and monitor the configuration of the router device.

The tcpdump has been running on the monitoring PC (the blue box in Figure 1) from the day one of ADSL connection. There are no servers in the site and the router filtering is so configured that FTP is the only protocol that is allowed to have TCP active open into the home LAN. The actual traffic is examined from tcpdump header traces to see this is working as advertised and expected so far.

Placing Snort IDS was the next step, but the changing IP address posed a problem. If the router is really a Unix box, by running Snort in it, the detection of assigned IP address change would have been easier. A posting on the internet "Host attack countermeasures (Japanese)" mentions a script which periodically examines interface address. Taking the infrequency of the change of the DNS server addresses from the ISP, that is an acceptable solution. Some of the dynamic DNS servicers, such as "NO-IP.com", provide DNS entry update client programs that also use periodical change detection.

In the above home user's case, the router product is so simple and featureless that workaround that does not depend on it was necessary.

Overview of the Tool program

Figure 2 shows the overview of the tool program and its relation to the Snort process. These are running in the blue box in the Figure 1.

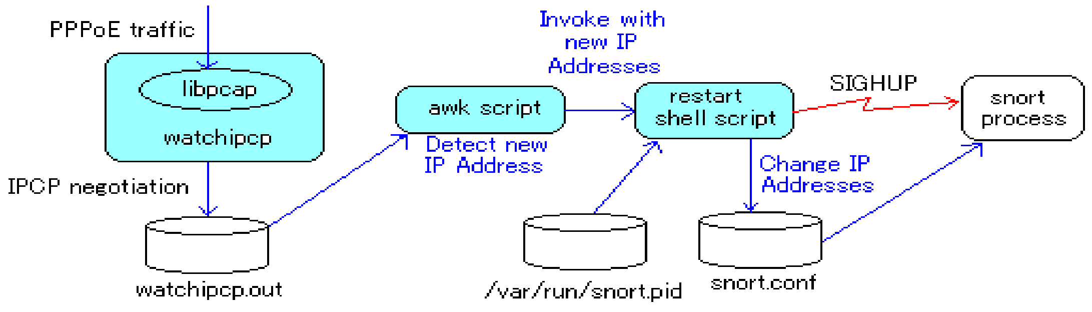

Figure 2 Tool overview

There are three programs. First one is PPPoE packet sniffer called watchipcp. This process sniffs the traffic between the ADSL modem and the router using libpcap and writes IPCP packets to a text file. Second one is an awk script, which reads the file and examines the IPCP negotiation. An ad hoc IPCP request/confirm matching is used to detect new address negotiation completion. It is far too much to implement the full

IPCP negotiation in a tool like this. Third one is a shell script that modifies snort.conf file and sends SIGHUP signal to the running Snort process to restart with the new configuration.

PPPoE traffic sniffer - watchipcp

This little program is actually a modification of the udpcksum and writepcap example programs found in the Stevens' textbook [1] and accompanying web site [2], respectively. More novice-friendly tutorial on programming with libpcap [6] is also useful.

PPPoE and IPCP packet formats

Because we are interested in the IPCP negotiation over PPPoE only, an appropriate filter will be applied to libpcap. To see how those packets look like tcpdump trace was observed;

```
# tcpdump -n -l -ied0 -ex
tcpdump: WARNING: ed0: no IPv4 address assigned
tcpdump: listening on ed0
```

and resetting the router, among other lines, following packet trace appeared.

```
20:36:36.468102 0:3:32:a9:f0:38 0:40:26:ed:81:63 8864 60: PPPoE  [ses 0xbd50] IPCP
                        1100 bd50 000c 8021 0101 000a 0306 0a2f
                        a043 fc0d e863 2033 4480 ffff ffff ffff
                        ffff ffff ffff 0045 5ec0 5010 81d0
```

By consulting "PPPoE RFC2516" as well as base PPP RFCs, the above dump can be interpreted as follows;

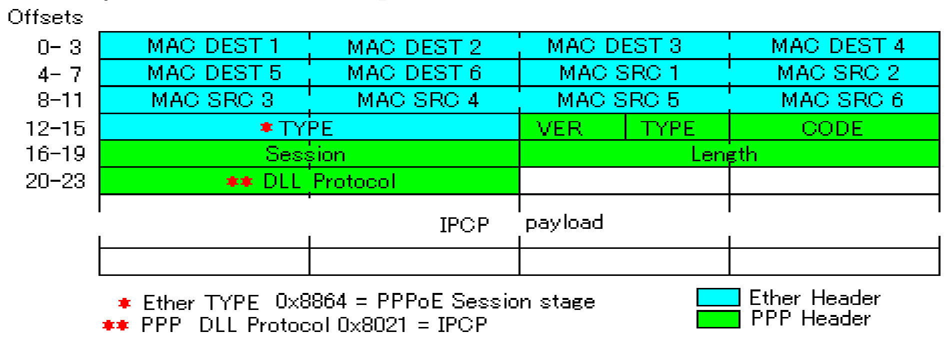

Figure 3 PPPoE packet format

The IPCP payload contains configuration request(offset 22 = 0x01) with one option;

Option code = 0x03 (IP Address)
Option length = 0x06 (including two bytes of Option code and Option length)
Option value = 0x0a2fa043 (four bytes of IP Address(note: value changed!))

From the above observation, following filter definition will be applied to the pcap library in watchipcp program. 'ether proto 0x8864 && ether[20:2] = 0x8021'

© SANS Institute 2000 - 2005 Author retains full rights.

IPCP Options

RFC 1332, 1877 and 2290 defines IPCP configuration options[7].

Option Type	Configuration Option
1	IP Address(deprecated)
2	IP Compression protocol
3	IP Address
4	Mobile IPv4
129	Primary DNS server IP Address
130	Primary NBNS server IP Address
131	Secondary DNS server IP Address
132	Secondary NBNS server IP Address

Because watchipcp program is interested in IP address and DNS address assignments, option types 3, 129 and 131 will be checked.

Intermediate file format

IPCP configuration packets' contents will be formatted into text and will be appended to an intermediate file (watchipcp.out in Figure 2). The interpretation of ongoing configuration negotiation is left to an outside script suitable for text handling. An entry for one IPCP packet in the file looks like as follows;

```
code 2 ID 03 plen=18
optCode 3 optLen 6 value 0a2fce93
optCode 129 optLen 6 value 0a2fa201
optCode 131 optLen 6 value 0a2fa209
*+*+*+
```

The last line delimits one packet data.

Although this program is small, simple and lightweight for the intended purpose, the sniffing of LAN packets is duplicate task with Snort already running in the same box. Writing this function as a Snort plugin would be more elegant.

IPCP negotiation examiner - a simple comparison awk script

PPP IPCP configuration negotiation is a complicated process. Full implementation of it is too
far beyond the reach of the current project, so, an ad hoc and trivially simple method was sed. For the same ID, when the requested option is exactly the same as its ack, an agreement is reached. This is from the observation of PPP negotiation like this:

© SANS Institute 2000 - 2005 Author retains full rights.

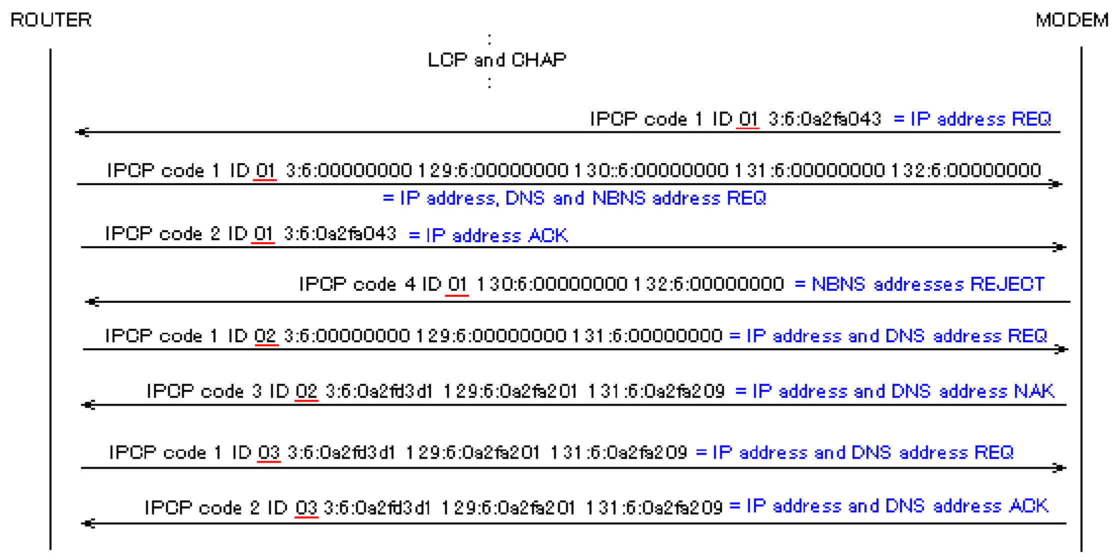
ROUTER
MODEM
LCP and CHAP
IPCP code 1 ID 01 3:6:0a2fa043 = IP address REQ
IPCP code 1 ID 01 3:6:00000000 129:6:00000000 130::6:00000000 131:6:00000000 132:6:00000000
= IP address, DNS and NBNS address REQ
IPCP code 2 ID 01 3:6:0a2fa043 = IP address ACK
IPCP code 4 ID 01 130:6:00000000 132:6:00000000 = NBNS addresses REJECT
IPCP code 1 ID 02 3:6:00000000 129:6:00000000 131:6:00000000 = IP address and DNS address REQ
IPCP code 3 ID 02 3:6:0a2fd3d1 129:6:0a2fa201 131:6:0a2fa209 = IP address and DNS address NAK
IPCP code 1 ID 03 3:6:0a2fd3d1 129:6:0a2fa201 131:6:0a2fa209 = IP address and DNS address REQ
IPCP code 2 ID 03 3:6:0a2fd3d1 129:6:0a2fa201 131:6:0a2fa209 = IP address and DNS address ACK

Figure 4 IPCP Configuration negotiation example

After some negotiations, the last two frames with ID=0x03 shows agreement of request made by Router and Ack from Modem(Server).

An awk script (watchipcp.awk) compares configuration request and corresponding ack/nak for each ID. tail command with -f option continuously reads the output of watchipcp process from the intermediate file and feeds them to the awk program.

Restarting Snort – a shell script

When new IP addresses for the Router and DNS servers determined, a shell script is invoked from inside of the awk program using system(). The script first updates snort.conf file with new HOME_NET and DNS_SERVERS variable definitions. A sed command does that using a template configuration file. Then, a SIGHUP signal is sent to the process using kill command. Snort daemon's PID is kept in /var/run/snort_ed0.pid file.

Execution

The tool consists of three processes running background and a restart shell script. Execution output example is shown in Appendix A1.3.

(1) watchipcp

This process is invoked from the shell and its standard output is redirected to an intermediate file.

```
watchipcp –ied0 –v >watchipcp.out &
```

(2) tail command and awk command

Intermediate file is read by tail command and its contents are piped to awk command.

They are invoked from the shell;

```
tail –f watchipcp.out | awk –f watchipcp.awk
```

© SANS Institute 2000 - 2005 Author retains full rights.

(3) restart sehll

Invoked from the awk script with appropriate arguments.

```
restart.sh  IP  PrimaryDNS_IP  SecondaryDNS_IP
```

Performance

To see the CPU resource consumption, the CPU Time of Snort process and watchipcp process are compared. After two day's run, Snort used CPU for 17 minute 46 seconds and watchipcp used 1 minute 21 seconds. This is about 13:1 and the added load was negligible. The monitoring PC is a 48M bytes memory Intel 486DX2. As a home ADSL user with 1.5Mbps downlink, the traffic is so light that a low spec old machine can monitor it.

Discussion

In Japan, the number of ADSL user is growing at a remarkable rate. Major ISP and Teleco provide low price services starting as low as twenty dollars a month. Magazines and media emphasize the danger of "always connected" Internet environment. How so, is not so clearly understood by most users, however. Threat of viruses in e-mail attachments and downloaded files are well recognized and anti-virus software is one of best selling retail packages.

The possibility of being abused by crackers and becoming one of offenders without knowing is new. Always powered up PCs directly connected to ADSL modem with no security precaution is becoming a horrible reality. Home users with several PCs will have to install host based IDS on each of his PC or set up network based IDS. With either IDS monitoring, it becomes possible to detect such cases.

Making the operation of Snort IDS easier for home users environment is an attempt to help such effort less demanding.

New programs can introduce new risks in the environment in which they run. Programmers need be aware of the environment in which the product runs. At least clear statement of how much efforts and consideration regarding security have been put into the program and what assumptions are made about the environment they run should accompany functional description. Just as every Internet RFC has Security consideration section, every software needs to have security assumption description in its specifications and manuals.

There are ongoing efforts to provide "checklist" for writing secure code, or more practically, avoiding dangerous pitfalls. The classic on this subject is Grafinkel and Spafford[8]. Among recent ones are [9] and Open Source community's Internet resources, such as [10]. Microsoft's recent emphasis on secure code is expected to mark the major change of tide toward quality software. The endless patching on program bugs is not a sound state of the industry.

As for the buffer overflow bug, "The best defense is often a good education on the issues"([9] p137). The book states rules for C programmers, which may be summarized as follows.

© SANS Institute 2000 - 2005 Author retains full rights.

Basic rule	Always do bound checking.
Corollary	Always validate input from user and other programs.
Individual rules	
1	Never use gets().
2	Avoid (mis)use of standard library functions(Table 7-1, p 152).
3	Beware of internal buffer size for common functions.
4	Don't assume anything about the behavior of someone else's software.

Those rules seem to be demanding too much for ordinary programmers. How hard will it be to build programs without trusting someone else's work? The book even states, "The rest is up to you". The recognition of the difficulty of writing quality code free from buffer overflow in C or C++ may be one of the reasons of Java's wide acceptance. Java also frees programmers from the burden of the allocation/disposal of memory storage.

Some compilers for C or C++ have the option of producing runtime bound checking code. Either approach have some performance penalty, though. The quality implementation of the JVM or compiler must come first. Putting efforts into the infra-structural part of the software development would be right thing..

In this spirit, watchipcp program and the tool is examined from the security programming point of view. The assumption for the tool's execution environment is, **a stand-alone Internet unreachable machine**. So, it should never be used in machines reachable from the Internet or connected to inside network. The program needs to have root privilege to access network capture device /dev/bpf. A call to setuid(getuid()) is made at the program startup after establishing access to the network device. This makes the process run in the non-privileged mode. So, watchipcp program file is SUID root.([1] p714).

The examination of watchipcp code easily reveals a bug; the bug resides in the portion written by the current author. The original code (udpcksum) from the textbook is considered flawless.

The watchipcp program naively assumes every "sniffed" message is correctly formatted. That is, the length of option value in the option-length byte field is used without validity checking. This is violation of the basic rule of "every input needs be validated for boundary checking". The bad coding is located in check_ipcp() function in watchipcp.c file. See A 1.2 (5) of Appendix.

How dangerous can this be? Even if the NIC is not configured for upper layer network protocol, programs are not free from the buffer overflow DOS attacks[11]. watchipcp program examines PPPoE frames, so malformed PPP message can cause the code to crash. It is not easy to send malformed IPCP frame to the remote site, but it is not impossible; for example, an attacker may take hold of the PPP server in the ISP's location.

The awk script uses system() function, which must be used with care. The argument command string passed to it is constructed using the input data. By looking at how the formatting of the command string is done shows that only fixed pattern of output is

© SANS Institute 2000 - 2005 Author retains full rights.

allowed. The use of system() here seems safe.

Lessons learned

The libpcap plays an important role in the network packet monitoring tools. This tutorial showed its usage in checking the PPPoE frames. By the need to update IP addresses for Snort configuration, IPCP packets of PPPoE are examined. Actual configuration negotiation sequence was examined using the tcpdump command.

The presented program was found to contain a bug that can cause buffer overflow. Writing secure code, even of this small size, requires much more effort and time.

Really, a little learning is a dangerous thing.

References

[1] R. Stevens. UNIX Network Programming, Volume 1, Second Edition: Networking APIs: Sockets and XTI, Prentice Hall, 1998, pp. 708-725

[2] W. Richard Stevens' Home Page http://www.kohala.com/start/unpv12e.html

[3] Snort Home Page http://www.snort.org/

[4] NO-IP.com Home Page http://www.no-ip.com/

[5] Host attack countermeasures(Japanese) http://www.geocities.co.jp/SiliconValley-Cupertino/5128/500_compu/ids.html

[6] Packet Capture With libpcap and other Low Level Network Tricks http://www.cse.nau.edu/~mc8/Socket/Tutorials/section1.html

[7] M. Nozaka. Internet Numbers in a Nutshell(Japanese), O'Reilly Japan, 1999, pp 51-60

[8] S. Grafinkel and G. Spafford. Practical UNIX and Internet Security, 2nd ed ,O'Reilly,1996, pp 701-719

[9] J. Viega and G McGraw.Building Secure Software, Addison Wesley, 2002

[10] Secure Programming for Linux and Unix HOWTO http://www.dwheeler.com/secure-programs/

[11] L-122: FreeBSD tcpdump Remote Buffer Overflow Vulnerability http://www.ciac.org/ciac/bulletins/l-122.shtml

Appendix

A 1. watchipcp source code

The program was compiled and run on FreeBSD 4.3-Release.
As described in the text, this program is basically a little modification of textbook example. So, for those 'almost copied' codes, only the modified parts are pointed out here instead of reproducing the whole source code. watchipcp.c, which implements IPCP packet sniffing using pcap library, is presented in a whole.
A 1.1 Source code preparation
Download unpv12e.tgz from Reference URL and expand. Top of the expansion directory will be unpv12e.
(1) Make unpv12e/watchipcp directory.

© SANS Institute 2000 - 2005, Author retains full rights.

(2) Copy unpv12e/udpcksum/*.c, unpv12e/udpcksum/*.h and unpv12e/udpcksum/Makefile to unpv12e/watchipcp.
(3) Modify/create source files as described in A1.2 below.
(4) make

A 1.2 watchipcp specific codes

(1) Header file

Rename udpcksum.h to watchipcp.h .
Add following function prototypes:

```
void            dump_frame(char *, unsigned char *ptr, int len);
void            watch_ipcp(void);
void            check_ipcp(unsigned char *ptr, int len);
```

(2)main.c file

Change header file name from "udpcksum.h" to "watchipcp.h".
In main() function,

Remove argc<2 test.
Remove cases for 'l' and '0' in getopt() select loop.
Remove lines from

```
    if (optind != argc-2)
```

to

```
        Setsockopt(rawfd, IPPROTO_IP, IP_HDRINCL, &on, sizeof(on));
```

Change "test_udp()" to "watch_ipcp()" .
Change usage() description as appropriate.

(3)pcap.c file

Change header file name from "udpcksum.h" to "watchipcp.h".
In open_pcap() function,

No need to construct filter string.
Instead, define constant CMD and pass it to the third parameter of pcap_compile();

```
#define CMD        "ether proto 0x8864 && ether[20:2] = 0x8021"
                              :
if (pcap_compile(pd, &fcode, CMD, 0, netmask) < 0)
```

Remove lines from pcap_lookupnet() to one line above pcap_compile() .
Set promiscuous flag parameter =1 in pcap_open_live()

```
        /* hardcode: promisc=1, to_ms=500 */
if ( (pd = pcap_open_live(device, snaplen, 1, 500, errbuf)) == NULL)
        :
```

© SANS Institute 2000 - 2005, Author retains full rights.

© SANS Institute 2000 - 2005 Author retains full rights.

(4) cleanup.c

Change header file name from "udpcksum.h" to "watchipcp.h".

(5) Create watchipcp.c file

Add the following lines to watchipcp.c file.

```
#include        "watchipcp.h"

void
watch_ipcp(void)
{
        int                     len;
        unsigned char           *ptr;

        for ( ; ; ) {
                ptr = next_pcap(&len);
                if( verbose ) dump_frame("111", ptr, len);
                check_ipcp(ptr, len);
        }
}

void check_ipcp(unsigned char *ptr, int len) {

        struct ether_header     *eptr;
        int     plen;
        int     pos;
        int     optLen;
        int     i;

        eptr = (struct ether_header *) ptr;
        if (ntohs(eptr->ether_type) != 0x8864 )
                 err_quit("Ethernet type %x not PPPoE", ntohs(eptr->ether_type));

                        /* skip ether header = start of ppp header */
        ptr = ptr + 14;
        len = len - 14;
        /* if( verbose ) dump_frame("222", ptr, len); */

                        /* examine ppp header fields */
        if ( ptr[6]==0x80 && ptr[7]==0x21 ) { /* this is IPCP frame */
                ptr = ptr + 8;
                printf("code %d ", ptr[0]);
                printf("ID %02x ", ptr[1]);
                plen = ptr[2]*16 + ptr[3] -4;
                printf("plen=%d\n", plen);
                ptr = ptr + 4;
                pos = 0;
                while( pos < plen )  {
                        optLen = ptr[pos+1];
                        printf("optCode %d optLen %d value ", ptr[pos], optLen );
                        for( i=pos+2; i<pos+optLen; i++ ) printf("%02x", ptr[i]);
                        printf("\n");
                        pos += optLen;
```

© SANS Institute 2000 - 2005, Author retains full rights.

© SANS Institute 2000 - 2005 Author retains full rights.

```
                }
                printf("*+*+*+\n");
                fflush(stdout);
        }
}

void
dump_frame(char *str, unsigned char *ptr, int len) {
        int     i;

        fprintf(stderr, "%s ---------------------- %d\n", str, len);
        for( i=0; i<len; i++ ) fprintf(stderr, "%02x", ptr[i]);
        fprintf(stderr, "\n----------------------\n");
}
```

(6) Makefile

Change the following lines. Note that libpcap.a is in /usr/lib in FreeBSD 4.3.

```
OBJS = main.o pcap.o watchipcp.o cleanup.o
PROGS =             watchipcp
watchipcp:      ${OBJS}
                ${CC} ${CFLAGS} -o $@ ${OBJS} /usr/lib/libpcap.a ${LIB}
```

A 1.3 Example run of watchipcp

```
# ./watchipcp -ied0 -v
device = ed0
datalink = 1
111 ---------------------- 60
004f17f39252000332a9f03888641100e251000c80210101000a03060a2fa0434f55b0f420334480ffffffff
ffffffff07be2e308b9e19c750104246
----------------------
code 1 ID 01 plen=6
optCode 3 optLen 6 value 0a2fa043
*+*+*+
                        :
                        :
                        :
111 ---------------------- 60
000332a9f038004f17f3925288641100e2510018802101030016030600a2fd34081060a2fa20183060a2f
a2090000000000000000000000000000000
----------------------
code 1 ID 03 plen=18
optCode 3 optLen 6 value 0a2fd340
optCode 129 optLen 6 value 0a2fa201
optCode 131 optLen 6 value 0a2fa209
*+*+*+
111 ---------------------- 60
004f17f39252000332a9f03888641100e2510018802102030016030600a2fd34081060a2fa20183060a2f
a2091a1319d720335080ffffffffffffff
----------------------
code 2 ID 03 plen=18
```

© SANS Institute 2000 - 2005, Author retains full rights.

```
optCode 3 optLen 6 value 0a2fd340
optCode 129 optLen 6 value 0a2fa201
optCode 131 optLen 6 value 0a2fa209
*+*+*+
^C

27 packets received by filter
0 packets dropped by kernel
#
```

A 2. Awk script

```
# watchipcp.awk

BEGIN {
        dec["0"]=0; dec["1"]=1; dec["2"]=2; dec["3"]=3; dec["4"]=4; dec["5"]=5; dec["6"]=6; dec["7"]=7;
    dec["8"]=8; dec["9"]=9; dec["a"]=10; dec["b"]=11; dec["c"]=12; dec["d"]=13; dec["e"]=14; dec["f"]=15;

        optCode[3]="IPADDR"; optCode[129]="DNSPRI"; optCode[131]="DNS2ND";

        currentID = 0
        confReq = ""
}
{
        if( $1 != "code" ) next
        if( currentID != $4 ) {
                currentID = $4
                confReq = ""
                if( $2 != 1 ) next
                # new configuration request
                confReq = saveConf()
        } else { # same ID
                if ( $2 == 1 ) {
                        confReq = saveConf()
                } else if( $2 == 2 ) {
                        ackReq = saveConf()

                        if( confReq == ackReq ) {
                                print "New session has started with configuration=" confReq
                                resetConf()
                        }
                }
        }
}

function saveConf() {
        conf = ""
    while(1) {
         getline
         if( $1 == "*+*+*+" ) break
         conf = conf " " $0
    }
        print currentID, conf
```

© SANS Institute 2000 - 2005, Author retains full rights.

```
                return conf
}

function resetConf() {
                n = split( confReq, F)
                if ( n!= 18 ) panic()
                optValue[F[ 2]] = F[ 6]
        optValue[F[ 8]] = F[12]
        optValue[F[14]] = F[18]
                for( i in optValue ) {
                                print i, optCode[i], dotdecimal( optValue[i] )
                }
        cmd = "/usr/local/share/snort/restart.sh " dotdecimal( optValue[3] ) " " dotdecimal( optValue[129] ) " "
dotdecimal( optValue[131] )
        print cmd
        system( cmd )
}
function dotdecimal( hex ) {
                val = ""
        for( j=0; j<4; j++ ) {
                                val = val "." dec[substr(hex,j*2+1,1)]*16 + dec[substr(hex,j*2+2,1)]
        }
                return substr(val,2)
}

function panic() {
                print "unexpected situation!"
                exit(1)
}
```

A 3. Shell script

```
# cat restart.sh
(cd /usr/local/share/snort; sed "s/IPADDR/$1/g" snort.conf.tmp | sed "s/DNSPRI/$2/g" | sed
"s/DNS2ND/$3/g" >snort.conf; kill -HUP `cat /var/run/snort_ed0.pid` )
```

Here, snort.conf.tmp is a template file including three lines as shown below;

```
# grep ^var snort.conf.tmp
var HOME_NET IPADDR/32
var EXTERNAL_NET any
var DNS_SERVERS [DNSPRI,DNS2ND]
#
```

© SANS Institute 2000 - 2005, Author retains full rights.