SS5 structures

The following structure contains network information about client when command is “connect” or “bind”:

SS5ClientInfo	
Unsigned int Socket	*Contain client socket*
Struct socksaddr_in SockAddr	*Not used*
Unsigned char SrcAddr[16]	*Contain client source address*
Unsigned int SrcPort	*Contain client source port*

The following structure contains network information about client when command is “udp associate”:

_SS5UdpClientInfo	
Unsigned int Socket	*Contain udp client socket*
Struct socksaddr_in SockAddr	*Not used*
Unsigned char SrcAddr[16]	*Contain udp client source address*
Unsigned int SrcPort	*Contain udp client source port*

The following structure contains method information if protocol is socks5:

_SS5MethodInfo	
Unsigned int Ver	*Socks versione, 4 or 5*
Unsigned int NMeth	*Number of methods supported by SS5*
Unsigned int NoAuth	
Unsigned int BasicAuth	
Unsigned int Method	*Method number*

The following structure contains information about username and password:

_SS5AuthInfo	
Unsigned char Username[16]	*Username*
Unsigned char Password[16]	*Password*

The following structure contains information about socks request when command is “connect” or “bind”:

_SS5RequestInfo	
Unsigned int Ver	*Socks versione, 4 or 5*
Unsigned int Cmd	*Socks command*
Unsigned int Rsv	
Unsigned int ATyp	*Address type (ipv4, ipv6 or fqdn)*
Unsigned char DstAddr[16]	*Remote server address (target)*
Unsigned int DstPort	*Remote server port (target)*

The following structure contains information about socks request when command is “udp associate”:

_SS5UdpRequestInfo	
Unsigned int Rsv	
Unsigned int Frag	*Fragmentation flag (not supported)*
Unsigned int ATyp	*Address type (ipv4, ipv6 or fqdn)*
Unsigned char DstAddr[16]	*Remote server address (target)*
Unsigned int DstPort	*Remote server port (target)*

The following structure contains information about upstream socks if configured:

_SS5UpstreamInfo	
Unsigned long int DstAddr	*Upstream socks address*
Unsigned int DstPort	*Upstream socks port*

The following structure contains information about SS5 facilities such as fixup, group file and bandwidth:

_SS5Facilities	
Unsigned char Fixup[16]	*Contain fixup value*
Unsigned char Group[32]	*Contain group file*
Unsigned long int Bandwidth	*Contain bandwidth*

The following structure contains information about SS5 options:

_SS5SocksOpt	
DnsOrder	*Enable dns ordering*
Verbose	*Enable verbose ouput*
Syslog	*Log to syslog instead to file*
Mute	*Disable log*
SessionTimeout	*Idle timeout in seconds*
Profiling	*Set profiling method*
LdapCriteria	*Set ldap configuration*
LdapTimeout	*Set ldap query timeout*
AuthCacheAge	*Enable and set authentication caching and aging*
StickyAge	*Set affinity aging*
Sticky	*Set affinity*
Authentication	*Set authenticaion method (passwd, pam, GEAP)*
AcceptTimeout	*Set accept timeout during BIND request*
Threaded	*Enable thread mode*
ServerBalance	*Enable Balancing (obsolete)*
PreforkProcesses	*Enable and set the number of instances to fork*

2) Example: Mod Filter

```
#include"SS5Main.h"
#include"SS5Mod_filter.h"

S5RetCode InitModule( struct _module *m )
{
  m->Filtering=Filtering;     <== (Pointer defined in SS5Main.h and used into SS5Core.c)

  return OK;
}

<== MAIN FUNCTION ==>
<== This function use _SS5ProxyData internal structure containing network buffer ==>
S5RetCode Filtering( unsigned char *s, struct _SS5ProxyData *pd )
{
  if( !strncmp(s,"http",strlen("http")) ) {
    if( !S5FixupHttp(pd) ) {
      return ERR_HTTP;
    }
  }
  else if( !strncmp(s,"https",strlen("https")) ) {
    if( !S5FixupHttps(pd) ) {
      return ERR_HTTPS;
    }
  }
  else if( !strncmp(s,"smtp",strlen("smtp")) ) {
    if( !S5FixupSmtp(pd) ) {
      return ERR_SMTP;
    }
  }
  else if( !strncmp(s,"pop3",strlen("pop3")) ) {
    if( !S5FixupPop3(pd) ) {
      return ERR_POP3;
    }
  }
  else if( !strncmp(s,"imap4",strlen("imap4")) ) {
    if( !S5FixupImap(pd) ) {
      return ERR_IMAP4;
    }
  }
  return OK;   <== Return value
}
```

```
<== SLAVE FUNCTIONS, not visible out of this module ==>
S5RetCode S5FixupSmtp( struct _SS5ProxyData *pd )
{
  register unsigned int index,
                        offset,
                       len;

  char s1[]="helo";
  char s2[]="ehlo";

  len=strlen(s1);

  for(offset=0;offset<DATABUF-len;offset++)
  {
    for(index=0;index<len;index++)
      if( tolower(pd->Recv[offset+index])!=s1[index] )
        break;

    if( index==len ) {
      return OK;
    }
  }

  len=strlen(s2);

  for(offset=0;offset<DATABUF-len;offset++)
  {
    for(index=0;index<len;index++)
      if( tolower(pd->Recv[offset+index])!=s2[index] )
        break;

    if( index==len ) {
      return OK;
    }
  }
  return ERR;
}

S5RetCode S5FixupPop3( struct _SS5ProxyData *pd )
{
  register unsigned int index,offset,len;

  char s[]="user";

  len=strlen(s);

  for(offset=0;offset<DATABUF-len;offset++)
  {
    for(index=0;index<len;index++)
      if( tolower(pd->Recv[offset+index]) != tolower(s[index]) )
        break;

    if( index==len ) {
      return OK;
    }
  }
  return ERR;
}
```

```
S5RetCode S5FixupImap( struct _SS5ProxyData *pd )
{
  register unsigned int index,offset,len;

  char s[]="capability";

  len=strlen(s);

  for(offset=0;offset<DATABUF-len;offset++)
  {
    for(index=0;index<len;index++)
      if( tolower(pd->Recv[offset+index]) != tolower(s[index]) )
        break;

    if( index==len ) {
      return OK;
    }
  }
  return ERR;
}

S5RetCode S5FixupHttp( struct _SS5ProxyData *pd )
{
  register unsigned int index,offset,len;

  char s[]="User-Agent:";

  len=strlen(s);

  for(offset=0;offset<DATABUF-len;offset++)
  {
    for(index=0;index<len;index++)
      if( pd->Recv[offset+index]!=s[index] )
        break;

    if( index==len ) {
      return OK;
    }
  }
  return ERR;
}
```

```
S5RetCode S5FixupHttps( struct _SS5ProxyData *pd )
{
  unsigned int length;

  /*
   * SSLv2 Record Layer: Client Hello
   *
   * Check two records:
   *
   * Length:                 must contain the len of the SSL packet
   * Handshake Message Type: must contain the "Client Hello" message
   */

  length=pd->Recv[1] + 2;

  if( length == pd->TcpRBufLen) {
    if( pd->Recv[2] == CLIENT_HELLO ) {
      return OK;
    }
  }

  /*
   * SSLv3 Record Layer: Client Hello
   * TLSv1 Record Layer: Client Hello
   *
   * Check two records:
   *
   * Length:       must contain the len of the SSL packet
   * Content Type: must contain the Handshake type
   */
  length=pd->Recv[3];
  length <<= 8;
  length +=pd->Recv[4];
  length += 5;

  if( pd->Recv[0] == HANDSHAKE ) {
    if( length == pd->TcpRBufLen ) {
      if( pd->Recv[5] == CLIENT_HELLO ) {
        return OK;
      }
    }
  }

  return ERR;
}
```