I2NSF
Internet-Draft
Intended status: Standards Track
Expires: September 22, 2016

S. Hares
Huawei
R. Moskowitz
HTT Consulting

March 21, 2016

Secure Session Layer Services
draft-hares-i2nsf-ssls-00.txt

Abstract

Each I2NSF agent and I2NSF client needs to provide application level support for management traffic during periods of DDoS and network security attacks to deal with congestion (burst and/or continuous), high error rates and packet loss due to the attacks, and the inability to utilize a transport protocol (E.g. TCP) due to a specific protocol attack. This application level support needs to be able to select the key management system and provide "chunking" of data (in order to fit in reduced effective MTUs), compression of data (in order to fit into reduced bandwidth), small security envelope)in order to maximize room for mangement payload), and fragmentation and reassembly at the application layer for those protocols which do not support fragmentation/reassembly (E.g. UDP or SMS). The application layer needs to be able to turn off this features if the system detects these features are no longer needed.

This draft specifies a security session layer services(SSLs) which provide these features in terms of an API, and the component features (interface to key management systems, data compression, chunking of data, secure session envelope (SSE) to send data, and fragmentation and reassembly, and ability to detect existence of attack).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 22, 2016.

Copyright Notice

Table of Contents

1. Introduction

Each I2NSF agent and I2NSF client needs to provide application level support for management traffic during periods of DDoS and network security attacks to deal with congestion (burst and/or continuous), high error rates and packet loss due to the attacks, and the

inability to utilize a transport protocol (E.g. TCP) due to a specific protocol attack. Some of the services the I2NSF controller must provide during these periods of DDoS or network security attacks are:

- receiving information regarding DDoS Threats from DOTS systems,
- Changing policy on vNSF and NSF devices during these periods,
- exchanging information with user security applications using I2NSF to obtain information from the controller,
- Aid the I2NSF reporting of attacks with the the CERT (MILE) either by providing data or sendign the report
- and manages network connnectivity of devices out of compliance (SACM).

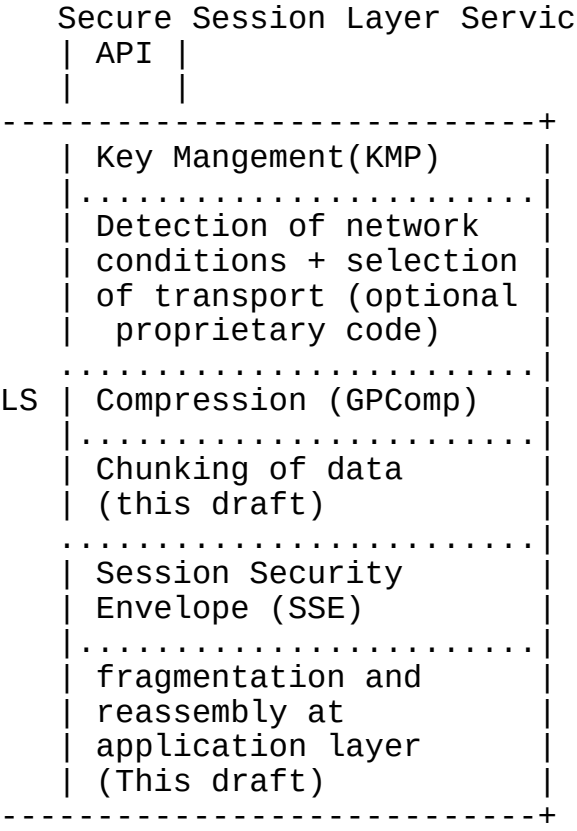
This application level support for I2NSF client-agent communication needs to be able to select the key management system and provide "chunking" of data (in order to fit in reduced effective MTUs), compression of data (in order to fit into reduced bandwidth), small security envelope)in order to maximize room for mangement payload), and fragmentation and reassembly at the application layer for those protocols which do not support fragmentation/reassembly (E.g. UDP or SMS). The application layer needs to be able to turn off this features if the system detects these features are no longer needed.

This draft specifies a security session layer (SSL) which provides these features in terms of:

- an API for the layer (section 2)
- interface to key management system (section 3),
- data compression (section 4)
- chunking of data (section 5)
- secure envelope (section 6),
- fragmentation and reassembly (section 7),
- detection of network conditions that require this service (section 8).

A diagram of the SSLS with these process is in figure 1.

The API for this SSLS allows the application to select the types of key management, and the different types of services (data compression, chunking of data, secure e)



2. API for SSLS

2.1. SSLS socket calls

The SSLS uses socket calls to set up the application session layer. The calls are shown below.

s = int socket(int domain, int type, int protocol)

where:

domain: AF_INET and AF_INET6 supported

type: SOCK_SSLS

desired protocol: Transport protocol (TCP (6), UDP (6), SCTP (132)), SMS (xx)

```
 int setsockopt(int sockfd, int level, int optname,
            const void *optval, socklen_t optlen);

int getsocketopt(int sockfd, int level, int optname
                   const void *optval, socket
 where:
  sockfd:      # socket file descriptor
  optname:     # option name (see below)
  optval;          # points to *sse_transport structure;
  optlen;          # length of option

  optnam:
  SSLS_AUTH_PRIV ]1]
  SSLS_AES_MODE[2]
  SSLS_ALGS[3]
  SSLS_SSE [4]
```

Where the opt_val structure are define in the figure below.

Figure 2

2.1.1. KMP related options

```
    Security Keying structures for:
    SSLS_AUTH_PRV, SSLS_AES_MODE, SSLS_ALGS
    options of setsockopt, getsockopt

    #struture for SSL_AUTH-PRIV optval
       struct *ssls_auth-priv_opts  {
              *ssls-x509-auth [SSLS-X509-LIMIT]
          }

          #SSL-X509-limit
          typedef struct ssls-x509-auth {
             const char name;
             void *x509-cert;  #cert struture by API
          }


          #structure for SSL_AES_MODE optval
          struct *ssls_aes_mode_opts {
        ... IKEV2 options # openikev2 API
        ... HIPv2 options # HIPv2 API
                                              #[RFC6317 + HIPv2]
         struct ssls_algs_opts;
          }

          #compression options
          struct *ssls_algs_opts {
            boolean  gpcomp-kmp; # computed with keys
       enum gmcomp-type;    #
           }
```

figure 3: setsockopt structure
for KMP related optins

2.1.2. SSE Envelope related options

```
     Security Session Envelope Related options
            #structure for SSL_SSE optval
            # SPI - is generated by KMP
            # SSE - sequence number - by SSE
            # Flags = Fragment (5 bits [0-5],

       struct *ssls_sse_opts {
            int nt_sockfd;   # new transport socket
            int *protocol;   # transport protocol for SSLS SSE
                             # can choose from (1-n )
            int *known_ports # known ports
            int chunk-size;  # chunk size
            int frag-size;   # fragment size
                             # greater than 0 means fragment]
            int SSEs-at-once # number of SSEs sent at once
            enum SSE_size;   # (compact, large, extreme)
            enum SSE-FLAG;   # compression flags
            );
```

Figure 4

2.2. OpenSSL X.509 API calls used

TBD

2.3. HIPv2 API calls used

(API calls will be added later based on HIP [RFC6317] upgraded to HIPv2.

2.3.1. HIP Structures

```
           struct addrinfo {
              int       ai_flags;          /* e.g., AI_CANONNAME */
              int       ai_family;         /* e.g., AF_HIP */
              int       ai_socktype;       /* e.g., SOCK_STREAM */
              int       ai_protocol;       /* 0 or IPPROTO_HIP */
              socklen_t ai_addrlen;        /* size of *ai_addr  */
              struct    sockaddr *ai_addr; /* sockaddr_hip */
              char     *ai_canonname;      /* canon. name of the host */
              struct    addrinfo *ai_next; /* next endpoint */
              int       ai_eflags;         /* RFC 5014 extension */
          };
```

2.3.2. HIP KMP calls

```
  #HIP uses
  #   #include <netdb.h>
 int getaddrinfo(const char *nodename,
                 const char *servname,
                 const struct addrinfo *hints,
                  struct addrinfo **res)
    void free_addrinfo(struct addrinfo *res)
```

Figure 3

3. Data Compression

The first step in making the application data easier to send through the network is to compress the data. The data compression algorithm is defined in draft-moskowitz-gpcomp-00.txt. The result of the compressed data is handed to the chunking function.

The user can disable or enable the compression function by setting SSE-DATA types to be one of the following:

- SSLS compress only - set compression, [1]
- SSLS compression and fragmentation [3],

Setting this flag to:

- no compression or fragmentation [0],
- SSLS to fragmentation only [2]

will skip the data compression step.

4. SSLS Processes

4.1. Chunking of Data

The process that "chunks" data breaks down the application stream after the compression process. If the compression process has compressed the data, the chunking process will chunk compressed data. If the user has requested no compression, this chunking process will chunk uncompressed data. The size of chunks of data the SSLS process creates to encapsulate in the secure session envelope (SSE) is specified on SSL_SSE setsockopt call.

The secure session envelope must be bigger than the chunk.

If the SSE is using TCP or STCP, that assembles the application flow into a byte stream, then the SSE packages will contain a chunk within the secure session envelope.

If Transports that do not fragment and re-assembly are being specified, the SSL will support application layer fragmentation and reassembly. (see the fragmentation section below

4.2. Secure Session Envelope

The Secure Session Envelope (SSE) creates a secure envelope using the SPI created by the key management and running over the transport selected by the user. The SSE has three forms: compact, Large, Extreme. The SSE compact form is below in figure x. SSL defines 4 bytes of the reserved field in the FLAGS field. See [I-D.moskowitz-sse] for details on secure session envelope sizes and formats.

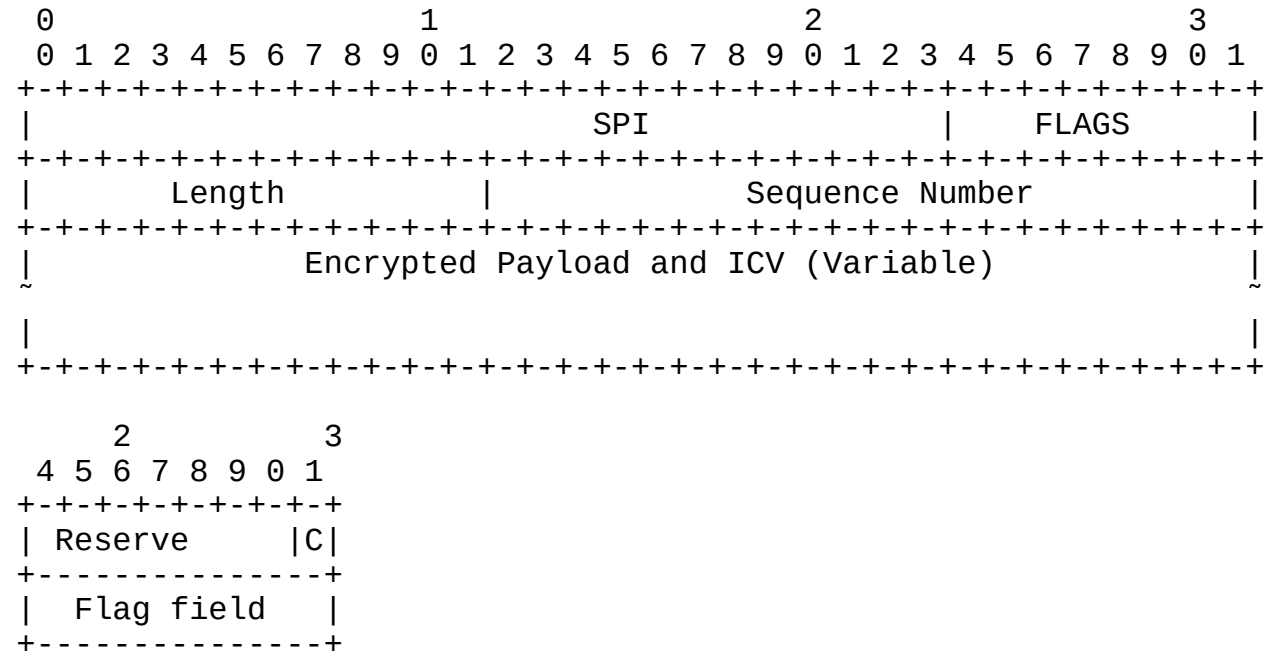
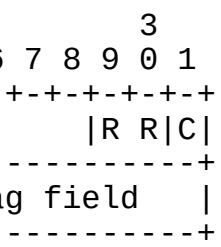


Figure 5 - Compact format of SSE

The SSLS utilizes 6 bits of the 8 bit flag in order to provide provide fragmentation and reassembly checks when the SSE gets fragmented into multiple transport packets. Each time the SSE fragments the packet to fit in the transport, it increments the fragment count in bits [24-28]. The bits for the flag word shown in figure 6.



Flag work in SSE header

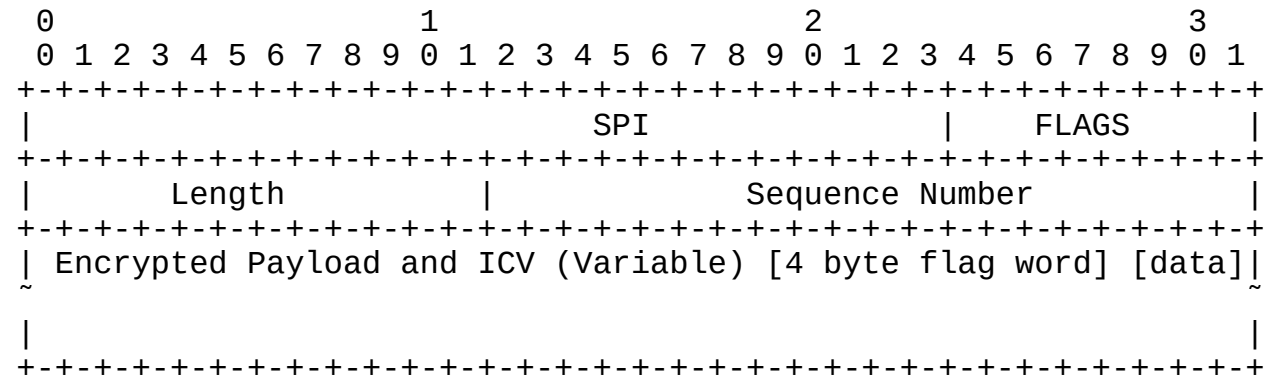
Bits [4-8] - 1-30 bit value for the fragment number
0 - no fragmentation
31 - indicates an fragmentation ACK response
Bits 5-6 - reserved
Bit 7 - compression

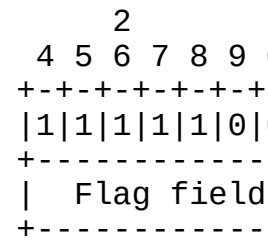
Figure 6 - SSLS redefined SSE Flag byte

4.3. Application Packet Fragmentation and Reassembly

SSE's secure envelope may be passed over UDP to avoid transport-level security attacks. Alternatively SSE's secure transport may go over the extremely limited SMS fabric so that some security management information gets through. In both cases, the user (or the "detection log") can select the transport and fragmentation.

If fragmentation is turned on, the individual SSE envelopes will track the IP messages the SSE envelope is broken into by placing the fragment number in the lowest 5 bits of the SSE Flag byte [24-28]. The SSE process receiving the traffic will send back an acknowledge SSE packet [Flag value in bits 0-4 is 0x1F or 31] within 30 bit map of sequences acked [1-30] in first 4 bits of SSE data. It is anticipate that the fragmentation process will attempt to bundle some acks.





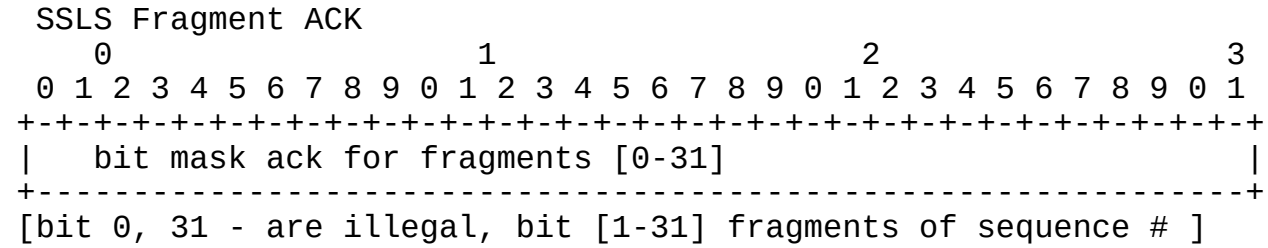


Figure 7 - SSLS ACK flag filed and first 4 bytes of payload

An example Fragmentation and ACK exchange

```
    SSLS-process-1--------IP/SMS---------SSLS Process-2
     [E.g. I2NSF Client -----------------I2NSF Agent]

    SSE-packet (SPI,(flags(fragment=1,C=1),
         length, seq 1, data )---->

     SSE-packet (SPI,(flags(fragment=2,C=1),
         length, seq 1, data )---->

     SSE-packet (SPI,(flags(fragment=3,C=1),
         length, seq 1, data )---->

     SSE-packet (SPI,(flags(fragment=1,C=1),
         length, seq 2, data )---->

      SSE-packet (SPI,(flags(fragment=2,C=1),
         length, seq 2, data )---->
                    <--SSE-packet (SPI)(flags fragment=31,C=1)
                  length, seq1,[ack-fragment 1,2])
                             <--SSE-packet (SPI)(flags fragment=32,C=1)
                               length, seq2,[ack-fragment1,2]

     SSE-packet (SPI,(flags(fragment=3,C=1),
         length, seq 1, data )---->
                    <--SSE-packet (SPI)(flags fragment=31,C=1)
                  length, seq1,[ack-fragment 3])

   Below is a set of pseudo call for the calls to socket

     pseudo
    struct sse_opts = {};
    optlen=size(sse_opts);
    optname= SSLS_SSE; #4
    s = int socket(int domain, int type, int protocol)
    errno =  int setsockopt(sockfd,level,optname,
             void struct *sse_opts,optlen);

    Errors: (Exact ERNOS added later)
     - protocol not support
     - error in known ports
     - error in chunk_size
     - error in fragment size
     - error in SSE-at-once
     - error - unsupported SSE
     - error in compression flags

     [Add read-write to socket ]
```

The SSE window size for fragmentation is 30 IP fragments or 30 SMS fragments per SSE chunk. The SSE process SHOULD assign the SSE fragments in order if possible. The SSE process will send an error response to the SSE if the data chunk does not fit in 30 IP/SM fragments.

If the SSE transmitting process has not received an acknowlegement for all IP fragments for a particular SSE envelope (identified by sequence number) with a SSE-retransmit-time, it sill retransmit the unacknowledged fragments.

Several SSE envelopes may be sent with fragmentation at once. The user signals the number sent at once with multiple SSE with fragment variable on the options. If fragmentation is selected, each of these SSE envelopes may need to track up to 30 IP fragments.

4.4. Proprietary Plugins: Detect Conditions + Select Transport

The SSL process allows two properitary plugins:

1. Plugin to detect error conditions which require SSLS services which include:
    * High levels of end-to-end congestion,
    * High levels of error and loss,
    * Input from IDS/IPS that detects problems
    * Signals from other I2NSF applications
2. Proprietary actions may select transport based on input from other standardize security services (DOTS, CERT, MILE) or proprietary services.

Prototype code will provide instances to show plugin values.

5. IANA Considerations

TBD

6. Security Considerations

The SSLS shares the following security considerations with the SSE Technology:

- As SSE uses an AEAD block cipher, it is vulnerable to attack if a sequence number is reused for a given key. Thus implementations

of SSE MUST provide for rekeying prior to Sequence Number rollover. An implementation should never assume that for a given context, the sequence number space will never be exhausted. Key Management Protocols like IKEv2 [RFC7296] or HIP [RFC7401] could be used to provide for rekeying management. The KMP SHOULD not create a network layer fate-sharing limitation.

- As any security protocol can be used for a resource exhaustion attack, implementations should consider methods to mitigate flooding attacks of messages with valid SPIs but invalid content. Even with the ICV check, resources are still consumed to validate the ICV.

- SSE makes no attempt to recommend the ICV length. For constrained network implementations, other sources should guide the implementation as to ICV length selection. The ICV length selection SHOULD be the the responsibility of the KMP.

- As with any layered security protocol, SSE makes no claims of protecting lower or higher processes in the communication stack. Each layer’s risks and liabilities need be addressed at that level.

7. Acknowledgements

The authos would like to thank Frank (Liang) Xia for his comments and suggestions on this draft.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

8.2. Informative References

[I-D.hares-i2nsf-mgtflow-reqs] Hares, S., "I2NSF Data Flow Requirements", draft-hares-i2nsf-mgtflow-reqs-00 (work in progress), March 2016.

[I-D.moskowitz-sse] Moskowitz, R., Faynberg, I., Lu, H., Hares, S., and P. Giacomin, "Session Security Envelope", draft-moskowitz-sse-02 (work in progress), February 2016.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC6317] Komu, M. and T. Henderson, "Basic Socket Interface Extensions for the Host Identity Protocol (HIP)", RFC 6317, DOI 10.17487/RFC6317, July 2011, <http://www.rfc-editor.org/info/rfc6317>.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <http://www.rfc-editor.org/info/rfc7296>.

[RFC7401] Moskowitz, R., Ed., Heer, T., Jokela, P., and T. Henderson, "Host Identity Protocol Version 2 (HIPv2)", RFC 7401, DOI 10.17487/RFC7401, April 2015, <http://www.rfc-editor.org/info/rfc7401>.

Authors' Addresses

Susan Hares
Huawei
Saline
US

Email: shares@ndzh.com

Robert Moskowitz
HTT Consulting
Oak Park, MI 48237
USA

Phone: +1-248-968-9809
Email: rgm@htt-consult.com