Securing UDT using TLS/DTLS draft-tsg-tls-udt-sec-01.txt

Abstract

This document describes about providing security to UDP Based Data Transfer (UDT) protocol. UDT is application level protocol built on the top of UDP, which effectively utilizes bandwidth in the high speed network as compared with TCP. UDT relies on the above layer for security because of absence of in-build security mechanisms. This document proposes the use of Transport Layer Security (TLS)/Datagram Transport Layer Security (DTLS) for securing the UDT protocol.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

Copyright and License Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents
(http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1 Introduction .................................................. 3
1.1 Terminology ............................................... 3
2 Integration with TLS/DTLS .................................... 3
3 Mapping of UDT socket with SSL structure .................... 5
4 IANA Considerations ........................................ 6
5 Security Considerations ...................................... 6
6 Limitation ..................................................... 6
7 References .................................................... 6
  7.1 Normative References .................................. 6
  7.2 Informative References ................................. 7
Authors’ Addresses .............................................. 8
1 Introduction

TCP underutilizes the network as Bandwidth Delay Product (BDP) increases[LL07]. Several variants of TCP such as Scalable TCP, BIC TCP and logarithmic TCP have come up for efficient utilization of high speed network but have a severe problem of RTT unfairness[SY09]. UDT is designed to overcome the limitation of TCP. Though UDT [UDT] uses UDP protocol for transferring data, it is a connection oriented, unicast and also provides a reliable duplex channel[GH04]. UDT has an option for plugging user defined congestion control algorithms and has support for reliable data streaming and partial reliable messaging [GG07]. Reliable data streaming is similar to TCP whereas partial reliable messaging to UDP. Through UDT socket, application sends the data to UDT layer which in turn uses UDP channel for delivery of data. UDT socket is a logical entity which internal maps to a UDP socket.

UDT being an application level protocol is easy to deploy because of it modularized framework [GG08] and is considered as an alternative data transfer protocol which provides faster data transfer but doesn’t provide security in terms of confidentiality, authentication and integrity [BH09]. UDT layer has to depend on above layer to achieve security. UDT is implemented by each application and not by each operating system or stack. So, security must be implemented per application basis.

Absence of security in UDT leads researchers to explore already proven security mechanism, many approaches were proposed for securing UDT application, which includes IPsec, Generic Security Service Application Programming Interface (GSS-API) [BH10b], Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) [BH10a]. TLS/DTLS is most widely used security mechanism that provides secure communication over an insecure channel. This document presents integration of TLS [RFC5246] / DTLS [RFC6347] with UDT for securing the application data.

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2 Integration with TLS/DTLS

Figure 1 depicts the position of TLS/DTLS in the layered architecture of TCP/IP stack. TLS provides security to reliable data transport protocol whereas DTLS to unreliable data transport protocol. UDT application must use TLS for reliable data streaming and DTLS for
partial reliable messaging to provide security to UDT application. The application programmer has to choose whether application requires reliable data streaming or partial reliable messaging. Adding TLS/DTLS doesn’t lead to IP fragmentation as UDT layer divides data in equal packets of size 1472.

```
UDT Application

 TLS          |          DTLS

UDT          |          UDT

Reliable mode | partial reliable messaging

UDP
```

Figure 1: Position of TLS/DTLS in Layered Architecture

To provide security to UDT, TLS/DTLS has to be modified to call UDT rather TCP/UDP. This can be achieved by using Basic Input and Output (BIO) objects, which are part of OpenSSL library [OPENSSL]. OpenSSL is a generic purpose cryptographic library which supports both TLS and DTLS security mechanisms. BIO objects hide the details of underlying layer. BIO objects allow the programmer to connect to different I/O channel such as TCP socket, UDP socket, memory and terminal, etc. BIO_METHOD is a pointer in BIO object, which holds the pointer to lower layer functionality.

Similarly, a BIO_METHOD object must be created and mapped to the UDT BIO object. BIO_METHOD holds the pointer to the UDT functionality.

```
struct bio_method_st
{
    int type;
    const char *name;
    int (*bwrite)(BIO *, const char *, int);
    int (*bread)(BIO *, char *, int);
    int (*bputs)(BIO *, const char *);
    int (*bgets)(BIO *, char *, int);
    long (*ctrl)(BIO *, int, long, void *);
    int (*create)(BIO *);
    int (*destroy)(BIO *);
    long (*callback_ctrl)(BIO *, int, bio_info_cb *);
} BIO_METHOD;
```

{Tatikayala Sai Gopal}    Expires July 10 2014                  [Page 4]
bwrite, bread, bputs are wrapper functions that point to UDT::write, UDT::read and UDT::send functions respectively. These wrapper functions allow OpenSSL to call UDT functions. TLS and DTLS wrapper functions are present in bss_sock.c and bss_dgram.c. Similarly create bss_udt.c, which hold the wrapper functions for UDT. For example, sock_write() internally calls UDT::send()

```c
static int sock_write(BIO *h, const char *buf, int num) {
    // code
    ret=UDT::send(b->num,in,inl,0); // code
}
```

3 Mapping of UDT socket with SSL structure

![Diagram of Mapping of SSL BIO object with UDT functionality](image)

Figure 2: Mapping of SSL BIO object with UDT functionality

Figure 2 shows the mapping of SSL BIO object with UDT functionality. Each side a BIO object MUST be created and SHOULD be linked to the SSL structure. The following is the step to link UDT BIO object with SSL structure.
/* creating SSL structure */
ssl =SSL_new(ctx);

/* creating a bio object for UDT */
bio= BIO_new(BIO_s_udtsock() );

/* mapping BIO object with UDT socket */
BIO_set_fd( bio, udt_session_fd , BIO_NOCLOSE);

/* mapping of SSL and bio object */
SSL_set_bio(ssl,bio,bio);

BIO_s_udtsock() maps BIO object with UDT functionality. Once the mapping of UDT socket with the corresponding SSL structure is done, UDT application has to call SSL_connect() & SSL_accept() on the client and the server side respectively to establish a shared session key. Application has to call SSL_write() /SSL_read() for encryption/decryption of data using the shared session key.

4 IANA Considerations

This document uses same identifiers of TLS and DTLS. So, there is no need of new IANA registries are required.

5 Security Considerations

UDT relies on TLS/DTLS for providing authentication, confidentiality and integrity of data. Therefore, most of the security considerations are same as that of TLS and DTLS. The additional security considerations raised by UDT is a random sequence number for initial UDT handshake. Random sequence number should be unpredictable in order to avoid spoofing or session hijacking. Absence of checksum in the UDT header may result in incorrect forwarding of packet by UDP layer.

6 Limitation

Bss_udt.c can be either compiled with every application, or it can be part of the OpenSSL library. If it is part of OpenSSL library, it has to be compiled with a g++ compiler rather than with gcc Since UDT uses C++.

7 References

7.1 Normative References
7.2 Informative References


Authors’ Addresses

Tatikayala Sai Gopal,
Centre for Development of Advanced Computing(C-DAC) Hyderabad,
Hyderabad-500085
INDIA
Email: saigopalt@cdac.in

Rahul Jain,
Centre for Development of Advanced Computing(C-DAC) Hyderabad,
Hyderabad-500085
INDIA
Email: rahulj@cdac.in

Reddy Lakshmi Eswari P,
Centre for Development of Advanced Computing(C-DAC) Hyderabad,
Hyderabad-500085
INDIA
Email: prleswari@cdac.in

Jyostna G,
Centre for Development of Advanced Computing(C-DAC) Hyderabad,
Hyderabad-500085
INDIA
Email: gjyostna@cdac.in