INTERNET-DRAFT
Intended Status: Proposed Standard
Expires: December 2016

T. Herbert
Facebook
L. Yong
Huawei
F. Templin
Boeing

June 21, 2016

Extensions for Generic UDP Encapsulation
draft-herbert-gue-extensions-00

Abstract

This specification defines a set of the fundamental extensions for Generic UDP Encapsulation (GUE). The extensions defined in this specification are the security option, payload transform option, checksum option, fragmentation option, and the remote checksum offload option.

Status of this Memo

Copyright and License Notice

Table of Contents

1. Introduction

Generic UDP Encapsulation [I.D.nvo3-gue] is a generic and extensible encapsulation protocol. This specification defines a basic set of extensions for GUE. These extensions are the security option, payload transform option, checksum option, fragmentation option, and the remote checksum offload option.

2. GUE header format with options

The general format of GUE with the options defined in this specification is:

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+\
   |        Source port            |      Destination port         | \
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ UDP
   |           Length              |          Checksum             | /
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+/
   |Ver|C|   Hlen  |  Proto/ctype  |V|SEC|K|F|T|R|   Rsvd Flags    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                         VNID (optional)                       |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   ~                       Security (optional)                     ~
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                        Checksum (optional)                    |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   +                     Fragmentation (optional)                  +
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                   Payload transform (optional                 |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                 Remote checksum offload (optional)            |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   ~                   Private fields (optional)                   ~
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The contents of the UDP are described in [I.D.herbert-gue].

The GUE header consists of:

- o Ver: Version. Set to 0x0 to indicate GUE encapsulation header.

Note that version 0x1 does not allow options.

- o C: Control bit. May be set or unset. GUE options can be used with either control or data packets unless otherwise specified in the option definition.

- o Hlen: Length in 32-bit words of the GUE header, including optional fields but not the first four bytes of the header. Computed as (header_len - 4) / 4. The length of the encapsulated packet is determined from the UDP length and the Hlen: encapsulated_packet_length = UDP_Length - 8 - GUE_Hlen.

- o Proto/ctype: If the C bit is not set this indicates IP protocol number for the packet in the payload, else if the C bit is set this type of control message for the payload. The next header begins at the offset provided by Hlen. When the fragmentation option or payload transform option is used this field may be set to protocol number 59 for a data message, or a value of 0 for a control message, to indicate no next header for the payload.

- o V: Indicates the network virtualization option (VNID) field is present. The VNID option is described in [I.D.hy-nvo3-gue-4-nvo].

- o SEC: Indicates security option field is present. The security option is described in section 5.

- o K: Indicates checksum option field is present. The checksum option is described in section 3.

- o F: Indicates fragmentation option field is present. The fragmentation option is described in section 4.

- o T: Indicates transform option field is present. The transform option is described in section 5.

- o R: Indicates the remote checksum option field is present. The remote checksum offload option is described in section 6.

- o Private fields are described in [I.D.nvo3-gue].

3. Checksum option

The GUE checksum option provides a checksum that covers the GUE header, a GUE pseudo header, and optionally part or all of the GUE payload. The GUE pseudo header includes the corresponding IP addresses as well as the UDP ports of the encapsulating headers. This checksum should provide adequate protection against address corruption in IPv6 when the UDP checksum is zero. Additionally, the GUE checksum provides protection of the GUE header when the UDP checksum is set to zero with either IPv4 or IPv6. In particular, the GUE checksum can provide protection for some sensitive data, such as the virtual network identifier ([I.D.hy-nvo3-gue-4-nvo]), which when corrupted could lead to mis-delivery of a packet to the wrong virtual network.

3.1. Option format

The presence of the GUE checksum option is indicated by the K bit in the GUE header.

The format of the checksum option is:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|            Checksum           |        Payload coverage       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The fields of the option are:

- o Checksum: Computed checksum value. This checksum covers the GUE header (including fields and private data covered by Hlen), the GUE pseudo header, and optionally all or part of the payload (encapsulated packet).

- o Payload coverage: Number of bytes of payload to cover in the checksum. Zero indicates that the checksum only covers the GUE header and GUE pseudo header. If the value is greater than the encapsulated payload length, the packet must be dropped.

3.2. Requirements

The GUE header checksum must be set on transmit when using a zero UDP checksum with IPv6.

The GUE header checksum must be set when the UDP checksum is zero for IPv4 if the GUE header includes data that when corrupted can lead to misdelivery or other serious consequences, and there is no other mechanism that provides protection (no security field for instance). Otherwise the GUE header checksum should be used with IPv4 when the UDP checksum is zero.

The GUE header checksum should not be set when the UDP checksum is non-zero. In this case the UDP checksum provides adequate protection and this avoids convolutions when a packet traverses NAT that does address translation (in that case the UDP checksum is required).

3.3. GUE checksum pseudo header

The GUE pseudo header checksum is included in the GUE checksum to provide protection for the IP and UDP header elements which when corrupted could lead to misdelivery of the GUE packet. The GUE pseudo header checksum is similar to the standard IP pseudo header defined in [RFC0768] and [RFC0793] for IPv4, and in [RFC2460] for IPv6.

The GUE pseudo header for IPv4 is:

```
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                         Source Address                        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                      Destination Address                      |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |        Source port            |      Destination port         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The GUE pseudo header for IPv6 is:

```
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   +                                                               +
   |                                                               |
   +                         Source Address                        +
   |                                                               |
   +                                                               +
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   +                                                               +
   |                                                               |
   +                      Destination Address                      +
   |                                                               |
   +                                                               +
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |        Source port            |      Destination port         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Note that the GUE pseudo header does not include payload length or protocol as in the standard IP pseudo headers. The length field is deemed unnecessary because:

o If the length is corrupted this will usually be detected by a checksum validation failure on the inner packet.

- Fragmentation of packets in a tunnel should occur on the inner packet before being encapsulated or GUE fragmentation (section 4) may be performed at tunnel ingress. GUE packets are not expected to be fragmented when using IPv6. See RFC6936 for considerations of payload length and IPv6 checksum.

- A corrupted length field in itself should not lead to misdelivery of a packet.

- Without the length field, the GUE pseudo header checksum is the same for all packets of flow. This is a useful property for optimizations such as TCP Segment Offload (TSO).

3.4. Checksum computation

The GUE checksum is computed and verified following the standard process for computing the Internet checksum [RFC1071]. Checksum computation may be optimized per the mathematical properties including parallel computation and incremental updates.

3.5. Transmitter operation

The procedure for setting the GUE checksum on transmit is:

1) Create the GUE header including the checksum and payload coverage fields. The checksum field is initially set to zero.

2) Calculate the 1’s complement checksum of the GUE header from the start of the GUE header through the its length as indicated in GUE Hlen.

3) Calculate the checksum of the GUE pseudo header for IPv4 or IPv6.

4) Calculate checksum of payload portion if payload coverage is enabled (payload coverage field is non-zero). If the length of the payload coverage is odd, logically append a single zero byte for the purposes of checksum calculation.

5) Add and fold the computed checksums for the GUE header, GUE pseudo header and payload coverage. Set the bitwise not of the result in the GUE checksum field.

3.6. Receiver operation

If the GUE checksum option is present, the receiver must validate the checksum before processing any other fields or accepting the packet.

The procedure for verifying the checksum is:

1) If the payload coverage length is greater than the length of the encapsulated payload then drop the packet.

2) Calculate the checksum of the GUE header from the start of the header to the end as indicated by Hlen.

3) Calculate the checksum of the appropriate GUE pseudo header.

4) Calculate the checksum of payload if payload coverage is enabled (payload coverage is non-zero). If the length of the payload coverage is odd logically append a single zero byte for the purposes of checksum calculation.

5) Sum the computed checksums for the GUE header, GUE pseudo header, and payload coverage. If the result is all 1 bits (-0 in 1’s complement arithmetic), the checksum is valid and the packet is accepted; otherwise the checksum is considered invalid and the packet must be dropped.

3.7. Security Considerations

The checksum option is only a mechanism for corruption detection, it is not a security mechanism. To provide integrity checks or authentication of the GUE header, the GUE security option should be used.

4. Fragmentation option

The fragmentation option allows an encapsulator to perform fragmentation of packets being ingress to a tunnel. Procedures for fragmentation and reassembly are defined in this section. This specification adapts the procedures for IP fragmentation and reassembly described in [RFC0791] and [RFC2460]. Fragmentation may be performed on both data and control messages in GUE.

4.1. Motivation

This section describes the motivation for having a fragmentation option in GUE.

MTU and fragmentation issues with In-the-Network Tunneling are described in [RFC4459]. Considerations need to be made when a packet is received at a tunnel ingress point which may be too large to traverse the path between tunnel endpoints.

There are four suggested alternatives in [RFC4459] to deal with this:

1) Fragmentation and Reassembly by the Tunnel Endpoints

2) Signaling the Lower MTU to the Sources

3) Encapsulate Only When There is Free MTU

4) Fragmentation of the Inner Packet

Many tunneling protocol implementations have assumed that fragmentation should be avoided, and in particular alternative #3 seems preferred for deployment. In this case, it is assumed that an operator can configure the MTUs of links in the paths of tunnels to ensure that they are large enough to accommodate any packets and required encapsulation overhead. This method, however, may not be feasible in certain deployments and may be prone to misconfiguration in others.

Similarly, the other alternatives have drawbacks that are described in [RFC4459]. Alternative #2 implies use of something like Path MTU Discovery which is not known to be sufficiently reliable. Alternative #4 is not permissible with IPv6 or when the DF bit is set for IPv4, and it also introduces other known issues with IP fragmentation.

For alternative #1, fragmentation and reassembly at the tunnel endpoints, there are two possibilities: encapsulate the large packet and then perform IP fragmentation, or segment the packet and then encapsulate each segment (a non-IP fragmentation approach).

Performing IP fragmentation on an encapsulated packet has the same issues as that of normal IP fragmentation. Most significant of these is that the Identification field is only sixteen bits in IPv4 which introduces problems with wraparound as described in [RFC4963].

The second possibility follows the suggestion expressed in [RFC2764] and the fragmentation feature described in the AERO protocol [I.D.templin-aerolink], that is for the tunneling protocol itself to incorporate a segmentation and reassembly capability that operates at the tunnel level. In this method fragmentation is part of the encapsulation and an encapsulation header contains the information for reassembly. This is different from IP fragmentation in that the IP headers of the original packet are not replicated for each fragment.

Incorporating fragmentation into the encapsulation protocol has some advantages:

- At least a 32 bit identifier can be defined to avoid issues of the 16 bit Identification in IPv4.

- Encapsulation mechanisms for security and identification, such as virtual network identifiers, can be applied to each segment.

- This allows the possibility of using alternate fragmentation and reassembly algorithms (e.g. fragmentation with Forward Error Correction).

- Fragmentation is transparent to the underlying network so it is unlikely that fragmented packet will be unconditionally dropped as might happen with IP fragmentation.

4.2. Scope

This specification describes the mechanics of fragmentation in Generic UDP Encapsulation. The operational aspects and details for higher layer implementation must be considered for deployment, but are considered out of scope for this document. The AERO protocol [I.D.templin-aerolink] defines one use case of fragmentation with encapsulation.

4.3. Option format

The presence of the GUE fragmentation option is indicated by the F bit in the GUE header.

The format of the fragmentation option is:

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |       Fragment offset   |Res|M|  Orig-proto   |               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+               +
   |                        Identification                         |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The fields of the option are:

- Fragment offset: This field indicates where in the datagram this fragment belongs. The fragment offset is measured in units of 8 octets (64 bits). The first fragment has offset zero.

- Res: Two bit reserved field. Must be set to zero for transmission. If set to non-zero in a received packet then the packet MUST be dropped.

- M: More fragments bit. Set to 1 when there are more fragments following in the datagram, set to 0 for the last fragment.

- Orig-proto: The control type (when C bit is set) or the IP protocol (when C bit is not set) of the fragmented packet.

- Identification: 40 bits. Identifies fragments of a fragmented packet.

Pertinent GUE header fields to fragmentation are:

- C: This bit is set for each fragment based on the whether the original packet being fragmented is a control or data message.

- Proto/ctype - For the first fragment (fragment offset is zero) this is set to that of the original packet being fragmented (either will be a control type or IP protocol). For other fragments, this is set to zero for a control message being fragmented, or to "No next header" (protocol number 59) for a data message being fragmented.

- F bit - Set to indicate presence of the fragmentation option field.

4.4. Fragmentation procedure

If an encapsulator determines that a packet must be fragmented (eg. the packet’s size exceeds the Path MTU of the tunnel) it should divide the packet into fragments and send each fragment as a separate GUE packet, to be reassembled at the decapsulator (tunnel egress).

For every packet that is to be fragmented, the source node generates an Identification value. The Identification must be different than that of any other fragmented packet sent within the past 60 seconds (Maximum Segment Lifetime) with the same tunnel identification-- that is the same outer source and destination addresses, same UDP ports, same orig-proto, and same virtual network identifier if present.

The initial, unfragmented, and unencapsulated packet is referred to as the "original packet". This will be a layer 2 packet, layer 3 packet, or the payload of a GUE control message:

```
+-------------------------------//------------------------------+
|                        Original packet                        |
|            (e.g. an IPv4, IPv6, Ethernet packet)              |
+------------------------------//-------------------------------+
```

Fragmentation and encapsulation are performed on the original packet in sequence. First the packet is divided up in to fragments, and then each fragment is encapsulated. Each fragment, except possibly the last ("rightmost") one, is an integer multiple of 8 octets long. Fragments MUST be non-overlapping. The number of fragments should be minimized, and all but the last fragment should be approximately equal in length.

The fragments are transmitted in separate "fragment packets" as:

```
+--------------+-------------+---------------+--//--+----------+
|    first     |    second   |     third     |      |   last   |
|   fragment   |   fragment  |    fragment   | .... | fragment |
+--------------+-------------+---------------+--//--+----------+
```

Each fragment is encapsulated as the payload of a GUE packet. This is illustrated as:

```
+------------------+----------------+-----------------------+
|  IP/UDP header   |  GUE header    |        first          |
|      header      | w/ frag option |       fragment        |
+------------------+----------------+-----------------------+

+------------------+----------------+-----------------------+
|  IP/UDP header   |  GUE header    |        second         |
|      header      | w/ frag option |       fragment        |
+------------------+----------------+-----------------------+
                               o
                               o
+------------------+----------------+-----------------------+
|  IP/UDP header   |  GUE header    |         last          |
|      header      | w/ frag option |       fragment        |
+------------------+----------------+-----------------------+
```

Each fragment packet is composed of:

(1) Outer IP and UDP headers as defined for GUE encapsulation.

- The IP addresses and UDP destination port must be the same for all fragments of a fragmented packet.

- The source port selected for the inner flow identifier must be the same value for all fragments of a fragmented packet.

(2) A GUE header that contains:

- The C bit which is set to the same value for all the fragments of a fragmented packet based on whether a control message or data message was fragmented.

- o A proto/ctype. In the first fragment this is set to the value corresponding to the next header of the original packet and will be either an IP protocol or a control type. For subsequent fragments, this field is set to 0 for a fragmented control message or 59 (no next header) for a fragmented data messages.

- o The F bit is set and fragment option is present.

- o Other GUE options. Note that options apply to the individual GUE packet. For instance, the security option would be validated before reassembly.

(3) The GUE fragmentation option. The option contents include:

- o Orig-proto that identifies the first header of the original packet.

- o A Fragment Offset containing the offset of the fragment, in 8-octet units, relative to the start of the of the original packet. The Fragment Offset of the first ("leftmost") fragment is 0.

- o An M flag value of 0 if the fragment is the last ("rightmost") one, else an M flag value of 1.

- o The Identification value generated for the original packet.

(4) The fragment itself.

4.5. Reassembly procedure

At the destination, fragment packets are decapsulated and reassembled into their original, unfragmented form, as illustrated:

```
+-------------------------------//------------------------------+
|                         Original packet                       |
|            (e.g. an IPv4, IPv6, Ethernet packet)              |
+------------------------------//-------------------------------+
```

The following rules govern reassembly:

The IP/UDP/GUE headers of each packet are retained until all fragments have arrived. The reassembled packet is then composed of the decapsulated payloads in the GUE fragments, and the IP/UDP/GUE headers are discarded.

When a GUE packet is received with the fragment option, the proto/ctype in the GUE header must be validated. In the case that the packet is a first fragment (fragment offset is zero), the proto/ctype in the GUE header must equal the orig-proto value in the fragmentation option. For subsequent fragments (fragment offset is non-zero) the proto/ctype in the GUE header must be 0 for a control message or 59 (no-next-hdr) for a data message. If the proto/ctype value is invalid then for a received packet it MUST be dropped.

An original packet is reassembled only from GUE fragment packets that have the same outer Source Address, Destination Address, UDP source port, UDP destination port, GUE header C bit, virtual network identifier if present, orig-proto value in the fragmentation option, and Fragment Identification. The protocol type or control message type (depending on the C bit) for the reassembled packet is the value of the GUE header proto/ctype field in the first fragment.

The following error conditions may arise when reassembling fragmented packets with GUE encapsulation:

If insufficient fragments are received to complete reassembly of a packet within 60 seconds (or a configurable period) of the reception of the first-arriving fragment of that packet, reassembly of that packet must be abandoned and all the fragments that have been received for that packet must be discarded.

If the payload length of a fragment is not a multiple of 8 octets and the M flag of that fragment is 1, then that fragment must be discarded.

If the length and offset of a fragment are such that the payload length of the packet reassembled from that fragment would exceed 65,535 octets, then that fragment must be discarded.

If a fragment overlaps another fragment already saved for reassembly then the new fragment that overlaps the existing fragment MUST be discarded

If the first fragment is too small then it is possible that it does not contain the necessary headers for a stateful firewall. Sending small fragments like this has been used as an attack on IP fragmentation. To mitigate this problem, an implementation should ensure that the first fragment contains the headers of the encapsulated packet at least through the transport header.

A GUE node must be able to accept a fragmented packet that, after reassembly and decapsulation, is as large as 1500 octets. This means that the node must configure a reassembly buffer that is at least as large as 1500 octets plus the maximum-sized encapsulation headers that may be inserted during encapsulation. Implementations may find it more convenient and efficient to configure a reassembly buffer size of 2KB which is large enough to accommodate even the largest set of encapsulation headers and provides a natural memory page size boundary.

4.6. Security Considerations

Exploits that have been identified with IP fragmentation are conceptually applicable to GUE fragmentation.

Attacks on GUE fragmentation can be mitigated by:

- Hardened implementation that applies applicable techniques from implementation of IP fragmentation.

- Application of GUE security (section 5) or IPsec [RFC4301]. Security mechanisms can prevent spoofing of fragments from unauthorized sources.

- Implement fragment filter techniques for GUE encapsulation as described in [RFC1858] and [RFC3128].

- Do not accepted data in overlapping segments.

- Enforce a minimum size for the first fragment.

5. Security and payload transform options

The security option and the payload transform option are used to provide security for the GUE headers and payload. The GUE security option provides origin authentication and integrity protection of the GUE header at tunnel end points to guarantee isolation between tunnels and mitigate Denial of Service attacks. The payload transform option provides a means to perform encryption and authentication of the GUE packet that protects the payload from eavesdropping, tampering, or message forgery.

5.1. Security option format

The presence of the GUE security option is indicated in the SEC flag bits of the GUE header.

The format of the fragmentation option is:

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   ~                           Security                            ~
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The fields of the option are:

- Security (8, 16, or 32 octets). Contains the security information. The specific semantics and format of this field is expected to be negotiated between the two communicating nodes.

To provide security capability, the SEC flags MUST be set. Different sizes are allowed to provide different methods and extensibility. The use of the security field is expected to be negotiated out of band between two tunnel end points.

The possible values in the SEC flags are:

- 00 - No security field

- 01 - 64 bit security field

- 10 - 128 bit security field

- 11 - 256 bit security field

5.2. Security option usage

The GUE security field should be used to provide integrity and authentication of the GUE header. Security negotiation (interpretation of security field, key management, etc.) is expected to be negotiated out of band between two communicating hosts. Two possible uses for this field are outlined below, a more precise specification is deferred to other documents.

5.2.1. Cookies

The security field may be used as a cookie. This would be similar to cookie mechanism described in L2TP [RFC3931], and the general properties should be the same. The cookie may be used to validate the encapsulation. The cookie is a shared value between an encapsulator and decapsulator which should be chosen randomly and may be changed periodically. Different cookies may used for logical flows between the encapsulator and decapsulator, for instance packets sent with different VNIs in network virtualization [I.D.hy-nvo3-gue-4-nvo] might have different cookies.

5.2.2. Secure hash

Strong authentication of the GUE header can be provided using a secure hash. This may follow the model of the TCP authentication option [RFC5925]. In this case the security field holds a message digest for the GUE header (e.g. 16 bytes from MD5). The digest might be done over static fields in IP and UDP headers per negotiation (addresses, ports, and protocols). In order to provide enough entropy, a random salt value in each packet might be added, for instance the security field could be a 256 bit value that contains 128 bits of salt value, and a 128 bit digest value. The use of secure hashes requires shared keys which are presumably negotiated and rotated as needed out of band.

5.3. Payload Transform Option format

The presence of the GUE payload transform option is indicated by the T bit in the GUE header.

The format of Payload Transform Field is:

```
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|    Type       | Payload Type  |           Reserved            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The fields of the option are:

Type: Payload Transform Type or Code point. Each payload transform mechanism must have one code point registered in IANA. This document specifies:

0x01: for DTLS [RFC6347]

0x80˜0xFF: for private payload transform types

A private payload transform type can be used for experimental purpose or vendor proprietary mechanisms.

Payload Type: Indicates the encrypted payload type. When payload transform option is present, proto/ctype in the base header should set to 59 ("No next header") for a data message and zero for a control message. The payload type (IP protocol or control message type) of the unencrypted payload must be encoded in this field.

The benefit of this rule is to prevent a middle box from inspecting the encrypted payload according to GUE next protocol. The assumption here is that a middle box may understand GUE base header but does not understand GUE option flag definitions.

Reserved field for DTLS type MUST set to Zero. For other transformation types, the use of reserved field may be specified.

5.3.1. Payload transform option usage

The payload transform option provides a mechanism to transform or interpret the payload of a GUE packet. The Type field describes the format of the payload before transformation and Payload Type provides the protocol of the packet after transformation. The payload transformation option is generic so that it can have both security related uses (such as DTLS) as well as non security related uses (such as compression, CRC, etc.).

The payload of a GUE packet can be secured using Datagram Transport Layer Security [RFC6347]. An encapsulator would apply DTLS to the GUE payload so that the payload packets are encrypted and the GUE header remains in plaintext. The payload transform option is set to indicate that the payload should be interpreted as a DTLS record.

5.4. Operation of security mechanisms

GUE secure transport mechanisms apply to both IPv4 and IPv6 underlay networks. The outer IP address MUST be tunnel egress IP address (dst) and tunnel ingress IP address (src). The GUE security option provides origin authentication and integrity to GUE based tunnel; GUE payload encryption provides payload privacy over an IP delivery network or Internet. The two functions are processed separately at tunnel end points. A GUE tunnel can use both functions or use one of them.

When both encryption and security are required, an encapsulator must perform payload encryption first and then encapsulate the encrypted packet with security flag and payload transform flag set in GUE header; the security option field must be filled according Section 5.2 above and the payload transform field must be filled according to Section 5.3 above. The decapsulator must decapsulate the packet first, then perform the authentication validation. If the validation is successful, it performs the payload decryption according to the encryption information in the payload encryption field in the header. Else if the validation fails, the decapsulator must discard the packet and may generate an alert to the management system. These processing rules also apply when only one function, either encryption or security, is enabled, except the other function is not performed as stated above.

If GUE fragmentation is used in concert with the GUE security option and/or payload transform option, the security and playload transformation are performed after fragmentation at the encapsulator and before reassembly at the decapsulator.

In order to get flow entropy from the payload, an encapsulator must determine the flow entropy value (e.g. a hash over the 4-tuple of a TCP connection) before performing the payload encryption. The flow entropy value can then be set in UDP src port of the GUE packet.

DTLS [RFC6347] provides packet fragmentation capability. To avoid packets fragmentation being performed multiple times by an encapsulator, an encapsulator SHOULD only perform the packet fragmentation at part of the packet encapsulation process (e.g. using the GUE fragmentation option), not in payload encryption process (i.e. DTLS layer fragmentation should be avoided).

DTLS usage [RFC6347] is limited to a single DTLS session for any specific tunnel encapsulator/decapsulator pair (identified by source and destination IP addresses). Both IP addresses MUST be unicast addresses - multicast traffic is not supported when DTLS is used. A GUE tunnel decapsulator implementation that supports DTLS can establish DTLS session(s) with one or multiple tunnel encapsulators, and likewise a GUE tunnel encapsulator implementation can establish DTLS session(s) with one or multiple decapsulators.

5.5. Considerations of Using Other Security Tunnel Mechanisms

GUE may rely on other secure tunnel mechanisms such as DTLS [RFC6347] over the whole UDP payload for securing the whole GUE packet or IPsec [RFC4301] to achieve the secure transport over an IP network or Internet.

IPsec [RFC4301] was designed as a network security mechanism, and therefore it resides at the network layer. As such, if the tunnel is secured with IPsec, the UDP header would not be visible to intermediate routers anymore in either IPsec tunnel or transport mode. The big drawback here prohibits intermediate routers to perform load balancing based on the flow entropy in UDP header. In addition, this method prohibits any middle box function on the path.

By comparison, DTLS [RFC6347] was designed with application security and can better preserve network and transport layer protocol information than IPsec [RFC4301]. Using DTLS to secure the GUE tunnel, both GUE header and payload will be encrypted. In order to differentiate plaintext GUE header from encrypted GUE header, the destination port of the UDP header between two must be different, which essentially requires another standard UDP port for GUE with

DTLS. The drawback on this method is to prevent a middle box operation to GUE tunnel on the path.

Use of two independent tunnel mechanisms such as GUE and DTLS to carry a network protocol over an IP network adds some overlap and process complex. For example, fragmentation will be done twice.

As the result, a GUE tunnel SHOULD use the security mechanisms specified in this document to provide secure transport over an IP network or Internet when it is needed. GUE tunnels with the GUE security options can be used as a secure transport mechanism over an IP network and Internet.

6. Remote checksum offload option

Remote checksum offload is mechanism that provides checksum offload of encapsulated packets using rudimentary offload capabilities found in most Network Interface Card (NIC) devices. Many NIC implementations can only offload the outer UDP checksum in UDP encapsulation. Remote checksum offload is described in [UDPENCAP].

In remote checksum offload the outer header checksum, that is in the outer UDP header, is enabled in packets and, with some additional meta information, a receiver is able to deduce the checksum to be set for an inner encapsulated packet. Effectively this offloads the computation of the inner checksum. Enabling the outer checksum in encapsulation has the additional advantage that it covers more of the packet than the inner checksum including the encapsulation headers.

The remote offload checksum option should not be used when GUE fragmentation is also being performed. In this case the offload of the outer UDP checksum does not cover the whole transport segment so remote checksum offload would not properly set the inner transport layer checksum.

6.1. Option format

The presence of the GUE remote checksum offload option is indicated by the R bit in the GUE header.

The format of remote checksum offload field is:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Checksum start         |       Checksum offset         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

The fields of the option are:

- o Checksum start: starting offset for checksum computation relative to the start of the encapsulated payload. This is typically the offset of a transport header (e.g. UDP or TCP).

- o Checksum offset: Offset relative to the start of the encapsulated packet where the derived checksum value is to be written. This typically is the offset of the checksum field in the transport header (e.g. UDP or TCP).

6.2. Transmitter operation

The typical actions to set remote checksum offload on transmit are:

1) Transport layer creates a packet and indicates in internal packet meta data that checksum is to be offloaded to the NIC (normal transport layer processing for checksum offload). The checksum field is populated with the bitwise not of the checksum of the pseudo header or zero as appropriate.

2) Encapsulation layer adds its headers to the packet including the offload meta data. The start offset and checksum offset are set accordingly.

3) Encapsulation layer arranges for checksum offload of the outer header checksum (e.g. UDP).

4) Packet is sent to the NIC. The NIC will perform transmit checksum offload and set the checksum field in the outer header. The inner header and rest of the packet are transmitted without modification.

6.3. Receiver operation

The typical actions a host receiver does to support remote checksum offload are:

1) Receive packet and validate outer checksum following normal processing (e.g. validate non-zero UDP checksum).

2) Validate the checksum option. If checksum start is greater than the length of the packet, then the packet must be dropped. If checksum offset is greater then the length of the packet minus two, then the packet must be dropped.

3) Deduce full checksum for the IP packet. If a NIC is capable of

receive checksum offload it will return either the full checksum of the received packet or an indication that the UDP checksum is correct. Either of these methods can be used to deduce the checksum over the IP packet [UDPENCAP].

4) From the packet checksum, subtract the checksum computed from the start of the packet (outer IP header) to the offset in the packet indicted by checksum start in the meta data. The result is the deduced checksum to set in the checksum field of the encapsulated transport packet.

   In pseudo code:

```
  csum: initialized to checksum computed from start (outer IP
        header) to the end of the packet
  start_of_packet: address of start of packet
  encap_payload_offset: relative to start_of_packet
  csum_start: value from meta data
  checksum(start, len): function to compute checksum from start
        address for len bytes

  csum -= checksum(start_of_packet, encap_payload_offset +
                   csum_start)
```

4) Write the resultant checksum value into the packet at the offset provided by checksum offset in the meta data.

   In pseudo code:

```
  csum_offset: offset of checksum field

  *(start_of_packet + encap_payload_offset +
    csum_offset) = csum
```

5) Checksum is verified at the transport layer using normal processing. This should not require any checksum computation over the packet since the complete checksum has already been provided.

6.4. Security Considerations

Remote checksum offload allows a means to change the GUE payload before being received at a decapsulator. In order to prevent misuse of this mechanism, a decapsulator should apply security checks on the GUE payload only after checksum remote offload has been processed.

7. Processing order of options

Options must be processed in a specific order for both sending and receive. Note that some options, such as the checksum option, depend on other fields in the GUE header to be set.

The order of processing options to send a GUE packet are:

1) Set VNID option.

2) Fragment if necessary and set fragmentation option. VNID is copied into each fragment. Note that if payload transformation will increase the size of the payload that must be accounted for when deciding how to fragment

3) Set Remote checksum offload.

4) Perform payload transform (potentially on each fragment) and set payload transform option.

5) Set security option.

6) Calculate GUE checksum and set checksum option.

On reception the order of actions is reversed.

1) Verify GUE checksum.

2) Verify security option.

3) Perform payload transformation (i.e. decrypt payload)

4) Adjust packet for remote checksum offload.

5) Perform reassembly.

6) Receive on virtual network indicated by VNID.

Note that the relative processing order of private fields is unspecified.

8. Security Considerations

Encapsulation of network protocol in GUE should not increase security risk, nor provide additional security in itself. GUE requires that the source port for UDP packets should be randomly seeded to mitigate some possible denial service attacks.

If the integrity and privacy of data packets being transported through GUE is a concern, GUE security and payload encryption SHOULD be used to remove the concern. If the integrity is the only concern, the tunnel may consider use of GUE security only for optimization. Likewise, if the privacy is the only concern, the tunnel may use GUE encryption function only.

If GUE payload already provides secure mechanism, e.g., the payload is IPsec packets, it is still valuable to consider use of GUE security.

9. IANA Consideration

IANA is requested to assign flags for the extensions defined in this specification. Specifically, an assignment is requested for the V, SEC, K, F, T and R flags in the "GUE flag-fields" registry (proposed in [I.D.nvo3-gue]).

10. References

10.1. Normative References

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[I.D.nvo3-gue] T. Herbert, L. Yong, and O. Zia, "Generic UDP Encapsulation" draft-ietf-nvo3-gue-03

10.2. Informative References

[RFC1071] Braden, R., Borman, D., and C. Partridge, "Computing the Internet checksum", RFC1071, September 1988.

[RFC1624] Rijsinghani, A., Ed., "Computation of the Internet Checksum via Incremental Update", RFC1624, May 1994.

[RFC1936] Touch, J. and B. Parham, "Implementing the Internet Checksum in Hardware", RFC1936, April 1996.

[RFC4459] MTU and Fragmentation Issues with In-the-Network Tunneling.

P. Savola. April 2006.

[RFC4963] Heffner, J., Mathis, M., and B. Chandler, "IPv4 Reassembly Errors at High Data Rates", RFC 4963, DOI 10.17487/RFC4963, July 2007, <http://www.rfc-editor.org/info/rfc4963>.

[RFC2764] B. Gleeson, A. Lin, J. Heinanen, G. Armitage, A. Malis, "A Framework for IP Based Virtual Private Networks", RFC2764, February 2000.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[RFC1858] Ziemba, G., Reed, D., and P. Traina, "Security Considerations for IP Fragment Filtering", RFC 1858, October 1995.

[RFC3128] Miller, I., "Protection Against a Variant of the Tiny Fragment Attack (RFC 1858)", RFC 3128, June 2001.

[RFC3931] Lau, J., Townsley, W., et al, "Layer Two Tunneling Protocol - Version 3 (L2TPv3)", RFC3931, 1999

[RFC5925] Touch, J., et al, "The TCP Authentication Option", RFC5925, June 2010.

[RFC6347] Rescoria, E., Modadugu, N., "Datagram Transport Layer Security Version 1.2", RFC6347, 2012.

[I.D.hy-nvo3-gue-4-nvo] Yong, L., Herbert, T., "Generic UDP Encapsulation (GUE) for Network Virtualization Overlay" draft-hy-nvo3-gue-4-nvo-03

[I.D.draft-mathis-frag-harmful] M. Mathis, J. Heffner, and B. Chandler, "Fragmentation Considered Very Harmful"

[I.D.templin-aerolink] F. Templin, "Transmission of IP Packets over AERO Links" draft-templin-aerolink-62.txt

[UDPENCAP] T. Herbert, "UDP Encapsulation in Linux", http://people.netfilter.org/pablo/netdev0.1/papers/UDP-Encapsulation-in-Linux.pdf

Authors’ Addresses

Tom Herbert
Facebook
1 Hacker Way

Menlo Park, CA
USA

EMail: tom@herbertland.com

Lucy Yong
Huawei USA
5340 Legacy Dr.
Plano, TX 75024
USA

Email: lucy.yong@huawei.com

Fred L. Templin
Boeing Research & Technology
P.O. Box 3707
Seattle, WA 98124
USA

Email: fltemplin@acm.org