Extensible Encapsulation Protocol
Intended status: Informational
Category: Internet-Draft
A. Dubovikov
R. Haenel
L. Mangani
August 2013

The Extensible Encapsulation Protocol (“EEP”)

Status of this Memo

This Internet-Draft will expire on January 1, 2014.

Copyright Notice

Abstract

This document specifies the Extensible Encapsulation protocol (“EEP” pronounced “HEPPY”) which provides a method to duplicate an IP datagram to a collector by encapsulating the original datagram and its relative header properties (as payload, in form of concatenated chunks) within a new IP datagram transmitted over UDP/TCP/SCTP connections for remote collection. Encapsulation allows for the original content to be transmitted without altering the original IP datagram and header contents and provides flexible allocation of additional chunks containing additional arbitrary data.

1. Introduction

This document specifies a method by which an IP datagram and its headers may be Encapsulated and carried within a new IP datagram, as payload composed of concatenated chunks containing the generic header values (e.g., IP Protocol, Source IP Address) from the original datagram (3.2.1), the original packet payload and any optional vendor chunks definable by future protocol implementors(3.2.4). Once the Encapsulated datagram reaches its destination, it is decapsulated releasing the chunk types defined in this documents sections and containing the original datagram payload and header values which are ultimately delivered to a (?) collector (?) database connectivity layer

The encapsulation method described in this document is NOT designed or intended for “tunneling” of IP datagrams over network segments, and best serves as vector for passive duplication of packets intended for remote or centralized collection and long term storage and analysis.

A generic representation of the encapsulation/decapsulation flow is:

```
source --> encapsulator -----> decapsulator --> collector -> database
```

2. Motivation

The Extensible Encapsulation protocol was designed to provide an efficient, extensible and low-level framework to accurately duplicate passively obtained IP datagrams for remote collection over UDP/TCP/SCTP connections, where full retention of original datagram headers and payload MUST be provided to the collector without alterations. The definition includes both generic (internal) and vendor-specific (custom defined) chunk types providing ground for implementors to extend the spectrum of the deliverable data alongside the encapsulated IP datagram.

3. EEP Encapsulation

3.1 General Protocol Structure

EEP transmits packets over UDP/TCP/SCTP connections. Each packet starts with the HEP3 header, followed by the payload composed of concatenated chunks described and specified in the following sections.

3.1.1 EEP Header

Each packet starts with the EEP header:

```
+-------+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
| Octet |0 |1 |2 |3 |4 |5 |6 |7 |8 |9 |10|11|12|13|14|15|
| Offset|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
+-------+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
| 0-15  |EEP ID     |total|      payload               |
|       |0x45455031 |leng.|                            |
+-------+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
| 16-31 |                     ...payload... (continued) |
+-------+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
| ...   |                     ...payload... (continued) |
+-------+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

The EEP header consists of a 4-octet protocol identifier with the fixed value 0x45455031 (ASCII „EEP1") and a two-octet length value (network byte order). The length value specifies the total packet length including the EEP ID, and the length field itself and the payload. It has a possible range of values between 6 and 65535.

3.1.2 Chunk Structure & Chunk Types

After the header, the payload is structured in form of concatenated chunks. Each chunk has the following structure (octet offset relative to the start of the chunk):

```
+------+---+---+---+---+---+---+--+--+--+--+--+--+--+--+--+
|Octet | 0 | 1 | 2 | 3 | 4 | 5 | 6| 7| 8| 9|10|11|12|13|14|15|
|Offset|   |   |   |   |   |   |  |  |  |  |  |  |  |  |  |  |
+------+---+---+---+---+---+---+--+--+--+--+--+--+--+--+--+
| 0-15 |Vendor | Type  |chunk  | chunk payload (var. length) |
|      |  ID   |  ID   |length |                             |
+------+---+---+---+---+---+---+--+--+--+--+--+--+--+--+--+
```

The chunk type is identified by a two-octet vendor ID and a two-octet type ID (both in network byte order). The vendor ID allows for grouping of chunk types for specific vendors (e.g., a vendor can receive a vendor ID and then define chunk type on its own). The chunk length field (network byte order) specifies the total length of the chunk, including the vendor ID, type ID, length and payload fields.

In combination, the Vendor and type ID fields and the length field allows a EEP implementation to skip unknown chunks and continue processing of a EEP packet.

The chunk payload depends on the type of the chunk and is defined in the following documentation section, or, for vendor specific chunks, by the vendor which maintains the chunk vendor ID. The following payload types are defined:

Payload Type	Payload Type Description
octet-string	arbitrary octet string (“byte array”)
utf8-string	UTF8 encoded character sequence
int8	8 bit unsigned integer number
uint16	16 bit unsigned integer number (network byte order)
uint32	32 bit unsigned integer number (network byte order)
inet4-addr	4 octet IPv4 address, most significant octet first
inet6-addr	16 octet Ipv6 address, most significant octet first

3.2 Chunk Types

The fields in the outer IP header are set and concatenated as chunks. The defined chunk types are described in the following section.

3.2.1 Generic Chunk Types

Chunk types with chunk vendor ID 0x0000 are called generic chunk types. The following generic chunk types are defined:

Chunk Type ID	Chunk Payload Type	Chunk Type Description
0x0001	uint8	IP protocol family
0x0002	uint8	IP protocol ID
0x0003	inet4-addr	IPv4 source address
0x0004	inet4-addr	IPv4 destination address
0x0005	inet6-addr	IPv6 source address
0x0006	inet6-addr	IPv6 destination address
0x0007	uint16	protocol source port (UDP, TCP, SCTP)

0x0008	uint16	protocol destination port (UDP, TCP, SCTP)
0x0009	uint32	timestamp seconds since 01/01/1970 epoch
0x000a	uint32	timestamp microseconds offset
0x000b	uint8	protocol type (SIP/H323/RTP/MGCP/M2UA)
0x000c	uint32	capture agent ID (202, 1201, 2033...)
0x000d	uint16	keep alive timer (sec)
0x000e	octet-string	authentication key (plaintext, TLS)
0x000f	octet-string	captured packet payload
0x0010	uint32	correlation ID
0x0011	uint16	SS7 protocol type (TCAP,INAP,ISUP)

3.2.2 Capture Protocol Types (0x000b)

Chunk Protocol ID	Chunk Protocol Type	Chunk Protocol ID	Chunk Protocol Type
0x00	reserved	0x08	MTP2 (E1/T1)
0x01	SIP	0x09	MTP3 (E1/T1)
0x02	XMPP	0x0a	M2UA (SIGTRAN)
0x03	SDP	0x0b	M2PA (SIGTRAN)
0x04	RTP	0x0c	V5UA (SIGTRAN)
0x05	RTCP	0x0d	M3UA (SIGTRAN)
0x06	MGCP	0x0e	IUA (SIGTRAN)
0x07	MEGACO (H.248)	0x0f	SUA (SIGTRAN)

3.2.3 E1/T1 Protocol Types (0x0011)

Chunk Protocol ID	Chunk Protocol Type
0x00	reserved
0x01	ISUP
0x02	SCCP
0x03	TCAP
0x04	MUP
0x05	HUP
0x06	TUP

3.2.4 Vendor Chunk Types

The vendor ID allows for grouping of chunk types. Definition of chunk types for is up to the implementor which maintains the specific vendors IDs; However the initial assignment of vendor IDs is specified in this document.

The following vendor IDs are assigned:

Chunk Vendor ID	Chunk Vendor Assignment
0x0000	Generic Vendor, Generic Chunk Types
0x0001	FreeSWITCH (www.freeswitch.org)
0x0002	Kamailio/SER (www.kamailio.org)
0x0003	OpenSIPS (www.opensips.org)
0x0004	Asterisk (www.asterisk.org)
0x0005	Homer Project (http://www.sipcapture.org)
0x0006	SipXecs (www.sipfoundry.org/)
0x0007	nTop (www.ntop.org/)

3.3 Use Cases

This document does not yet provide detailed use cases beyond the scope of the introduction to this document; these will be added in future revisions.

However current implementations of the EEP Encapsulation protocol are in the following general categories:

3.3.1 VoIP Monitoring & Signaling Troubleshooting

3.3.2 QoS Monitoring, R-Factor/MOS Reports from QoS Aware Equipment

3.3.3 Lawful Interception, RTP Session Duplication & Recording

3.3.4 SSW Session Internals (using Vendor Chunks)

4. Example EEP packet (hexadecimal octet encoding)

The following packet is decoded as an example of a valid EEP encapsulated packet. The SIP packet payload was shortened to facilitate reading of this example;

```
48 45 50 33
; EEP VERSION ID
00 71
; total length = 113 octets
00 00 00 01 00 07 02
; protocol family = 2 (IPv4)
00 00 00 02 00 07 11
; protocol ID = 17 (UDP)
00 00 00 03 00 0a d4 ca 00 01
; IPv4 source address = 212.202.0.1
00 00 00 04 00 0a 52 74 00 d3
; IPv4 destination address = 82.116.0.211
00 00 00 07 00 08 2e ea
; source port = 12010
00 00 00 08 00 08 13 c4
; destination port = 5060
00 00 00 09 00 0a 4e 49 82 cb
; seconds timestamp 1313440459 = Mon Aug 15 22:34:19 201100
00 00 0a 00 0a 00 01 d4 c0
; micro-seconds timestamp offset 120000 = 0.12 seconds
00 00 00 0b 00 07 01
; 01 – SIP
00 00 00 0c 00 0a 00 00 00 E4
; capture ID (228)
00 00 00 0f 00 14 49 4e 56 49 54 45 20 73 69 70 3a 62 6f 62
; SIP payload “INVITE sip:bob” (shortened)
```

5. Security Considerations

This document does not yet have any security considerations; these will be added in future revisions.

6. IANA Considerations

This document has no actions for IANA.

7. Acknowledgements

Parts of this document were taken or interpolated from portions of the
Homer Encapsulation Protocol (HEP) specification authored by Alexandr Dubovikov

8. References

[1] Reference 1

Author's Address

Questions about this memo can be directed to:

Alexandr Dubovikov
QSC AG
Germany

Email: alexandr.dubovikov@gmail.com

Roland Haenel
QSC AG
Germany

Email: roland@haenel.me

Lorenzo Mangani
QXIP
The Netherlands

Email: lorenzo.mangani@gmail.com