Network Working Group
Internet-Draft
Expires: August 17, 2001

D. New
Invisible Worlds, Inc.
February 16, 2001

The TUNNEL Profile
draft-ietf-idwg-beep-tunnel-01

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026.

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/ietf/1id-abstracts.txt.

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

This Internet-Draft will expire on August 17, 2001.

Copyright Notice

Copyright (C) The Internet Society (2001). All Rights Reserved.

Abstract

This memo describes a BEEP profile that allows a BEEP peer to serve as an application-layer proxy. It allows authorized users to access services through a firewall.

Table of Contents

1. Rationale

The TUNNEL profile provides a mechanism for cooperating BEEP peers to form an application-layer tunnel. The peers exchange "tunnel" elements that specify a source route, with the outermost element being stripped off and used to decide the next hop. The innermost, empty "tunnel" element tells the final destination that it is, indeed, the final destination. The term "proxy" is used to refer any of the BEEP peers other than the initiator and the final destination.

In one use of this profile, a BEEP peer implementing the TUNNEL profile is co-resident with a firewall. An initiating machine inside the firewall makes a connection to the proxy, then ask that proxy to make a connection to an endpoint outside the firewall. Once this connection is established, the proxy tells the outside endpoint that it will be tunneling. If the outside machine agrees, the proxy "gets out of the way," simply passing octets transparently, and both the initiating and terminating machines perform a "tuning reset," not unlike the way starting a TLS negotiation discards cached session state and starts anew.

Another use for this profile is to limit connections to outside servers based on the user identity negotiated via SASL. For example, a manager may connect to a proxy, authenticate herself with SASL, then instruct the proxy to tunnel to an information service restricted to managers. Since each proxy knows the identity of the next proxy being requested, it can refuse to tunnel connections if inadequate levels of authorization have been established. It is also possible to use the TUNNEL profile to anonymize the true source of a BEEP connection, in much the way a NAT translates IP addresses. However, detailed discussion of such uses is beyond the scope of this document.

Once both endpoint machines are connected, the tunneling proxy machine does no further interpretation of the data. In particular, it does not look for any BEEP framing. The two endpoint machines may therefore negotiate TLS between them, passing certificates appropriate to the endpoints rather than the proxy, with the assurance that even the proxy cannot access the information exchanged.

2. Examples

While the semantics described in Section 4 may seem complex, the results are actually relatively simple. A few examples will show the operation and use of this profile. In these examples, the machine attempting to establish the connection is named "initial", while the intermediate proxies are "proxy1" or "proxy2", and the machine with

the service that "initial" wishes to access is called "final". The examples also assume that the BEEP framework[1] is implemented on top of TCP[2], or some other mapping where one transport connection carries all channels.

2.1 One-Hop Example

A simple one-hop connection through a single proxy is illustrated first.

```
  initial                 proxy1                    final
     ----- xport connect ----->
    <------- greeting -------->
     --- start TUNNEL [1] ---->
                               ----- xport connect ------>
                              <-------- greeting -------->
                               ---- start TUNNEL [2] ---->
                              <---------- ok ------------
    <------- ok -------------- [3]
    <------------- greeting [4]-------------------------->
```

Notes:
[1] The TUNNEL element looks like this:
<tunnel fqdn='final.example.com' port='10288'>
<tunnel/>
</tunnel>

[2] The TUNNEL element looks like this:
<tunnel/>

[3] At this point, immediately after sending the <ok/> element, proxy1 starts passing octets transparently. It continues to do so until either transport connection is closed, after which it closes the other.

[4] This greeting may include the TLS profile, allowing initial and final to communicate without proxy1 understanding or interfering without being caught.

2.2 Two-Hop Example

The second example shows the initiator connecting to its proxy, that proxy connecting to another, and finally that second proxy finding a service outside.

```
initial             proxy1                 proxy2                 final
  --- xport connect -->
 <---- greeting ------>
  --start TUNNEL [1]-->
                         -- xport connect --->
                        <----- greeting ----->
                         --start TUNNEL [2]-->
                                               --- xport  connect --->
                                              <------- greeting ----->
                                               ---start TUNNEL [3]--->
                                              <-------- ok ---------
                        <------- ok --------- [4]
 <------- ok --------- [5]
 <------------------------- greeting ---------------------------->

Notes:
  [1] The TUNNEL element looks like this:
      <tunnel fqdn='proxy2.example.com' port='10288'>
        <tunnel fqdn='final.example.com' port='10290'>
          <tunnel/>
        </tunnel>
      </tunnel>

  [2] The TUNNEL element looks like this:
        <tunnel fqdn='final.example.com' port='10290'>
          <tunnel/>
        </tunnel>

  [3] The TUNNEL element looks like this:
          <tunnel/>

  [4] Proxy2 starts passing octets transparently after
      sending the <ok/>.

  [5] Proxy1 starts passing octets transparently after
      sending the <ok/>.
```


2.3 Failed Set-Up Example

The third example shows the initiator connecting through two proxys, the second proxy attempting to connect to the specified service and finding the destination is not a BEEP server. (Of course, specifying the telnet service can be expected to lead to this error.) The same would result if the destination did not support the TUNNEL profile.

```
initial             proxy1                 proxy2                 final
  --- xport connect -->
 <---- greeting ------>
  --start TUNNEL [1]-->
                         --- xport connect -->
                        <----- greeting ----->
                         --start TUNNEL [2]-->
                                               ---- xport connect --->
                                              <------- login: -------
                                               ----- xport close ---->
                        <---- <error> -------
                         --- xport close ---->
 <---- <error> ------
  --- xport close ---> [3]

Notes:
  [1] The TUNNEL element looks like this:
      <tunnel fqdn='proxy2.example.com' port='10288'>
        <tunnel fqdn='final.example.com' srv='_telnet._tcp'>
          <tunnel/>
        </tunnel>
      </tunnel>

  [2] The TUNNEL element looks like this:
        <tunnel fqdn='final.example.com' srv='_telnet._tcp'>
          <tunnel/>
        </tunnel>

  [3] This close is optional. "Initial" may also send another
      <tunnel> element, attempting to contact a different
      server, for example.
```


2.4 Non-BEEP Example

This example shows the initiator connecting through two proxys, the second proxy attempting to connect to the specified service and accepting that the destination is not a BEEP server. The difference at the protocol level is two-fold: The "initial" machine does not include the innermost "tunnel" element, and the final proxy ("proxy2") therefore does not expect a BEEP greeting.

```
initial             proxy1                 proxy2                 final
  --- xport connect -->
 <---- greeting ------>
  --start TUNNEL [1]-->
                         --- xport connect -->
                        <----- greeting ----->
                         --start TUNNEL [2]-->
                                               ---- xport connect --->
                                              <------- login: -------
                         <------ <ok> ------- [3]
                         <----- login: ------ [4]
 <------ <ok> --------- [3]
 <----- login: -------- [4] [5]


Notes:
  [1] The TUNNEL element looks like this:
      <tunnel fqdn='proxy2.example.com' port='10288'>
        <tunnel fqdn='final.example.com' svc='_telnet._tcp'>
        </tunnel>
      </tunnel>
      Note the lack of an innermost no-attribute <tunnel> element.

  [2] The TUNNEL element looks like this:
        <tunnel fqdn='final.example.com' srv='_telnet._tcp'>
        </tunnel>
      Note the lack of an innermost no-attribute <tunnel> element.

  [3] Each proxy starts transparently forwarding octets after this <ok>.

  [4] Each proxy forwards any data it received from the final host,
      even if that data arrived before the <ok> was sent.

  [5] After receiving the "ok" message, the "initial" peer can expect
      raw, non-BEEP data to be sent to and received from the "final"
      machine.
```


2.5 Profile Example

This example shows the initiator connecting through two proxys. The initial machine knows there is a server offering the SEP2 profile somewhere beyond proxy1, but it need not know where. Proxy1 has been locally configured to know that all SEP2 servers are beyond proxy2. Proxy2 has been locally configured to chose "final" as the server of choice for SEP2 services. Note that "final" does not necessarily need to offer the requested profile in its initial greeting.

```
initial             proxy1                 proxy2                 final
  --- xport connect -->
 <---- greeting ------>
  --start TUNNEL [1]-->
                         -- xport connect --->
                        <----- greeting ----->
                         --start TUNNEL [2]-->
                                               --- xport  connect --->
                                              <------- greeting ----->
                                               ---start TUNNEL [3]--->
                                              <-------- ok ---------
                        <------- ok --------- [4]
 <------- ok --------- [5]
 <-------------------------- greeting ---------------------------->


Notes:
  [1] The TUNNEL element looks like this:
        <tunnel profile="http://xml.resource/org/profiles/SEP2"/>
      Note the lack of an innermost no-attribute <tunnel> element.

  [2] Proxy1 maps this to
        <tunnel fqdn="proxy2.example.com" port="10288">
          <tunnel profile="http://xml.resource/org/profiles/SEP2"/>
        </tunnel>
      based on local configuration, then processes the new
      element, stripping off the outer element and routing
        <tunnel profile="http://xml.resource/org/profiles/SEP2"/>
      to proxy2.

  [3] Proxy2 receives the TUNNEL element with simply the SEP2
      URI specified. Local provisioning maps this to
        <tunnel fqdn='final.example.com' srv='_beep._tcp'>
          <tunnel/>
        </tunnel>
      Note the presence of an innermost no-attribute <tunnel> element.
      Proxy2 then strips the outermost element, looking up the
      appropriate address and port, and forwards the <tunnel/>
      element to the final machine.
```

[4] Proxy2 starts transparently forwarding octets after this <ok>.

[5] Proxy1 starts transparently forwarding octets after this <ok>.

2.6 Endpoint Example

This example shows the initiator connecting through two proxys. The initial machine knows there is a server known as "operator console" somewhere beyond proxy1, but it need not know where. Proxy1 has been locally configured to know that "operator console" is beyond proxy2. Proxy2 has been locally configured to use "final" as "operator console". This example is almost identical to the previous example, except that "endpoint" is intended to route to a particular server, while "profile" is intended to route to a particular service. Otherwise, these two attributes are very similar.

```
initial             proxy1               proxy2                final
 --- xport connect -->
 <---- greeting ------>
  --start TUNNEL [1]-->
                       -- xport connect --->
                      <----- greeting ----->
                       --start TUNNEL [2]-->
                                           --- xport  connect --->
                                          <------- greeting ----->
                                           ---start TUNNEL [3]--->
                                          <-------- ok ----------
                      <------- ok --------- [4]
 <------- ok --------- [5]
 <-------------------------- greeting ---------------------------->
```

Notes:

[1] The TUNNEL element looks like this:

```
<tunnel endpoint="operator console">
</tunnel>
```

Note the lack of an innermost no-attribute <tunnel> element.

[2] Proxy1 maps this to

```
<tunnel fqdn="proxy2.example.com" port="10288">
  <tunnel endpoint="operator console">
  </tunnel>
</tunnel>
```

based on local configuration, then processes the new element, stripping off the outer element and routing

```
<tunnel endpoint="operator console">
</tunnel>
```

to proxy2.

[3] Proxy2 receives the TUNNEL element with simply the endpoint specified. Local provisioning maps this to

```
<tunnel fqdn='final.example.com' srv='_beep._tcp'>
  <tunnel/>
</tunnel>
```

Note the presence of an innermost no-attribute <tunnel> element. Proxy2 then strips the outermost element, looking up the appropriate address and port, and forwards the <tunnel/> element to the final machine.

[4] Proxy2 starts transparently forwarding octets after this <ok>.

[5] Proxy1 starts transparently forwarding octets after this <ok>.

3. Message Syntax

The only element defined in this profile is the "tunnel" element. It is described in the following DTD, with additional limitations as described afterwards.

```
<!--
  DTD for the TUNNEL Profile, as of 2001-02-03

  Refer to this DTD as:

    <!ENTITY % TUNNEL PUBLIC "-//Blocks//DTD TUNNEL//EN"
               "http://xml.resource.org/profiles/TUNNEL/tunnel.dtd">
    %TUNNEL;
  -->

<!--
  TUNNEL messages

     role          MSG                 RPY
     ======        ===                 ===
     I or L        TUNNEL              +: ok
                                       -: error
  -->

<!ELEMENT tunnel      (tunnel?)>
<!ATTLIST tunnel
          fqdn         CDATA    #IMPLIED
          ip4          CDATA    #IMPLIED
          port         CDATA    #IMPLIED
          srv          CDATA    #IMPLIED
          profile      CDATA    #IMPLIED
          endpoint     CDATA    #IMPLIED
          >
```

The format of the "fqdn" attribute is a fully qualified domain name, such as "proxy.example.com". The format of the "ip4" attribute is four sets of decimal numbers separated by periods, such as "10.23.34.45". The format of the "port" attribute is a decimal number between one and 65535, inclusive. The format of the "srv" attribute is a pair of identifiers each starting with an underline and separated by a period, such as "_sep._tcp". The format of the "profile" attribute is a URI[3]. The format of the "endpoint" attribute is any string that may appear as an attribute value.

The only allowable combinations of attributes are as follows:

- fqdn + port;
- fqdn + srv;
- ip4 + port;
- profile, but only on the innermost element;
- endpoint, but only on the innermost element; or,
- no attributes, but only on the innermost element.

4. Message Semantics

When a TUNNEL channel is started, the listener expects a "tunnel" element from the initiator, either in the "start" element on channel zero or on the new channel created. As usual, if it arrives on channel zero, it is processed before the reply is returned.

In either case, the outermost "tunnel" element is examined. If it has no attributes, then this peer is hosting the BEEP service that the initiator wishes to use. In this case, the listener performs a tuning reset:

- All channels, including channel zero, are implicitly closed.
- Any previously cached information about the BEEP session is discarded.
- A new plaintext greeting is sent.

If the outermost element has a "port" attribute and an "fqdn" element, then "fqdn" is looked up as an A record via DNS, translated to an IP number. An "ip4" attribute is interpreted as the dotted-quad representation of an IPv4 address. In either case, a transport connection is established to the so-identified server. If the outermost element has a "srv" attribute, the concatenation of the "srv" element and the "fqdn" element (with a period between) is looked up in the DNS for a SRV record[4], and the appropriate server is contacted.

Alternately, if the outermost element has a "profile" attribute, then it must have no nested elements. The proxy processing this element is responsible for determining the appropriate routing to reach a peer serving the BEEP profile indicated by the URI in the attribute's value. Rather than source routing, this provides a hop-by-hop routing mechanism to a desired service.

Similarly, if the outermost element has an "endpoint" attribute, then it must have no nested elements. The proxy processing this element is responsible for determining the appropriate routing to reach a peer indicated by the value of the "endpoint" attribute. Rather than source routing, this provides a hop-by-hop routing mechanism to a desired machine. There are no restrictions on how machines are identified.

Then, if the outermost element has no nested elements, but it does have attributes other than "profile" or "endpoint", then this peer is the final BEEP hop. (This corresponds to "proxy2" in the "Non-BEEP" example above.) In this case, as soon as the final underlying transport connection is established, an "ok" element is returned over the listening session, and the tunneling of data starts. No BEEP greeting (or indeed any data) from the final hop is expected. Starting with the octet following the END(CR)(LF) trailer of the frame with the completion flag set (more=".") of the RPY carrying the "ok" element, the proxy begins copying octets directly and without any interpretation between the two underlying transport connections.

If the identified server cannot be contacted, an "error" element is returned over the listening channel and any connection established as an initiator is closed. If there is a nested "tunnel" element, and the server that has been contacted does not offer a BEEP greeting, or the BEEP greeting offered does not include the TUNNEL profile, then this too is treated as an error: the initiating transport connection is closed, and an error is returned.

If there is a nested "tunnel" element, and the identified server is contacted and offers a BEEP greeting including the TUNNEL profile, then the outermost element from the "tunnel" element received is stripped off, a new TUNNEL channel is started on the initiating session, and the stripped (inner) element is sent to start the next hop. In this case, the peer is considered a "proxy" (meaning that the next paragraph is applicable).

Once the proxy has passed the "tunnel" element on the TUNNEL channel, it awaits an "error" or an "ok" element in response. If it receives an "error" element, it closes the initiated session and it's underlying transport connection. It then passes the "error" element unchanged back on the listening session. If, on the other hand, it receives an "ok" element, it passes the "ok" element back on the listening session. Starting with the octet following the END(CR)(LF) trailer of the frame with the completion flag set (more=".") of the RPY carrying the "ok" element, the proxyy begins copying octets directly and without any interpretation between the two underlying transport connections.

5. Provisioning

While the BEEP Framework[1] is used, the attributes described are sufficient for the TCP mapping[2] of BEEP. The attributes on the "tunnel" element may need to be extended to handle other transport layers, such as IPv6 connection types.

In a mapping where multiple underlying transport connections are used, once the "ok" element is passed, all channels are closed, including channel zero. Thus, only the underlying transport connection initially established remains, and all other underlying transport connections for the session should be closed as well.

If a transport security layer (such as TLS) has been negotiated over the session, the semantics for the TUNNEL profile are ill-defined. The TUNNEL profile MUST NOT be advertised in any greetings after transport security has been negotiated.

6. Reply Codes

This section lists the three-digit error codes the TUNNEL profile may generate.

code	meaning
421	Service not available (E.g., the proxy does not have sufficient resources.)
450	Requested action not taken (E.g., DNS lookup failed or connection could not be established. See too 550.)
500	General syntax error (E.g., poorly-formed XML)
501	Syntax error in parameters (E.g., non-valid XML, letters in "ip4" attribute, etc.)
504	Parameter not implemented
530	Authentication required
534	Authentication mechanism insufficient (E.g., too weak, sequence exhausted, etc.)
537	Action not authorized for user
538	Encryption already enabled (E.g., TLS already negotiated, or a SASL that provides encryption already negotiated.)
550	Requested action not taken (E.g., next hop could be contacted, but malformed greeting or no TUNNEL profile advertised.)
553	Parameter invalid
554	Transaction failed (E.g., policy violation)

Note that the 450 error code is appropriate when the destination machine could not be contacted, while the 550 error code is appropriate when the destination machine could be contacted but the next phase of the protocol could not be negotiated. It is suggested that the beginning of any reply from the destination machine be included as part of the CDATA text of the error element, for debugging purposes.

7. Security Considerations

The TUNNEL profile is a profile of BEEP. In BEEP, transport security, user authentication, and data exchange are orthogonal. Refer to Section 8 of [1] for a discussion of this.

However, the intent of the TUNNEL profile is to allow bidirectional contact between two machines normally separated by a firewall. Since TUNNEL allows this connection between BEEP peers, and BEEP peers can offer a range of services with appropriate greetings, the TUNNEL profile should be configured with care. It is reasonable to strictly limit the hosts and services that a proxy is allowed to contact. It is also reasonable to limit the use of the TUNNEL profile to authorized users, as identified by a SASL profile.

Negotiation of a TLS profile in an end-to-end manner after a TUNNEL has been established will prevent intermediate proxies from observing or modifying the cleartext information exchanged, but only if TLS certificates are properly configured during the negotiation. The proxy could mount a "man in the middle" attack if public key infrastructure is not deployed.

In some environments, it is undesirable to expose the names of machines on one side of a firewall in unencrypted messages on the other side of that firewall. In this case, source routing (using the "fqdn", "ip4", "port" and "srv" attributes) can route a connection to the firewall proxy, with an innermost "profile" or "endpoint" attribute which the firewall proxy understands. Local provisioning can allow a proxy to translate a particular "profile" or "endpoint" element into a new source route to reach the desired service. This can prevents two attacks:

- o Attackers sniffing packets on one side of the firewall cannot see IP addresses or FQDNs of machines on the other side of the firewall; and,
- o Attackers cannot exhaustively attempt to connect to many FQDNs or IP addresses via source routing and use the error messages as an indication of whether the queried machine exists. For this attack to be prevented, the proxy must allow only "profile" or "endpoint" connections, always refusing to even attempt source-routed connections. This latter attack can also be thwarted by requiring a SASL identification before allowing a TUNNEL channel to be started, but this can have higher overhead.

References

[1] Rose, M.T., "The Blocks Extensible Exchange Protocol Framework", draft-ietf-beep-framework-11 (work in progress), January 2001.

[2] Rose, M.T., "Mapping the BEEP Framework onto TCP", draft-ietf-beep-tcpmapping-04 (work in progress), October 2000.

[3] Berners-Lee, T., Fielding, R.T. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[4] Gulbrandsen, A., Vixie, P. and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[5] mailto:mrose@dbc.mtview.ca.us

[6] mailto:gmatthew@orion.ac.hmc.edu

[7] mailto:bfeinste@odin.ac.hmc.edu

Author's Address

Darren New
Invisible Worlds, Inc.
1179 North McDowell Boulevard
Petaluma, CA 94954-6559
US

Phone: +1 707 789 3700
EMail: dnew@invisible.net
URI: http://invisible.net/

Appendix A. IANA Considerations

The IANA adds the following to its registry of BEEP profiles, upon approval of this document by the IESG:

Profile identification: http://xml.resource.org/profiles/TUNNEL
Message exchanged during channel creation: "tunnel"
Messages starting one-to-one exchanges: "tunnel"
Messages in positive replies: "ok"
Messages in negative replies: "error"
Messages in one-to-many exchanges: None.
Message syntax: See Section Section 3 of this document.
Message semantics: See Section Section 4 of this document.

Contact information: See the Author's Address appendix of this document.

Appendix B. Acknowledgements

The author gratefully acknowledges the contributions of Marshall Rose[5], Greg Matthews[6], and Ben Feinstein[7].

Inspiration for this profile comes from the Intrusion Detection Working Group of the IETF.

Full Copyright Statement

Copyright (C) The Internet Society (2001). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

Funding for the RFC editor function is currently provided by the Internet Society.