A Proposal to

The Aerospace Corporation

Implementing the Interoperable IETF/IDWG/IDXP Protocol with Proxy/Tunnel Capability

Harvey Mudd College Computer Science (2002-2003)

Team Members

Nick Hertl (Project Manager)
Will Berriel
Richard Fujiyama
Chip Bradford

Faculty Advisor

Professor Michael Erlinger

Aerospace Liaisons

Joseph Betser, Ph.D.
Rayford Sims

Abstract

The archival and correlation of network intrusion detection data is becoming more and more important for the timely recognition of intrusion attempts and prompt response to such threats. Often, however, the central repository of intrusion data is separated from a network administrator who wishes to examine this data. Currently, either transient, but insecure, holes are opened in the firewall, or permanent, and secure, channels are established in order to bypass the firewall. Clearly a transient and secure tunnel would be the best solution. Our project's goal is to implement the BEEP tuning profile for the Tunnel protocol. This promises to provide secure tunneling capabilities through firewalls for intrusion detection analysis as well as general use.

Table of Contents

1 Background Information

1.1 TCP/IP (RFC 793)

This is the main protocol stack that computers use to transport information across the Internet. It provides a reliable full duplex stream of data from one host to another, potentially using intermediate hosts (routers) to reach the eventual goal. Many application level protocols already use TCP/IP, such as HTTP (web pages) or SMTP (email). To communicate, such protocols use a standard interface or port. These ports are generally not blocked by firewalls because they are so commonly used and necessary for regular network operation. TCP/IP provides the transport for the BEEP implementations that we will use to write our profiles, but BEEP could be mapped to other transport layer protocols.

1.2 Firewall

Firewalls are computers that protect internal networks from potential hackers, as well as prevent private data from leaking out to the Internet. Every firewall has a list that specifies which ports or protocols are blocked or filtered. If you need to connect to a computer inside a firewall from outside a firewall on a port that the firewall does not allow, you are not allowed to connect. Tunnel proposes a solution to this problem for BEEP.

1.3 BEEP (RFC 3080 and RFC 3081)

BEEP is a new general protocol for application development. In the past, protocol designers have spent a large majority of their time discussing the details of how packets will look. Once all the fighting has died down, nobody has any energy to actually develop the protocol. BEEP helps with this problem by providing a very general set of capabilities for application protocols together with a comprehensive API for application protocol development. It allows the user to use tuning profiles to enable security if needed, or a number of other tuning variables. This application level protocol runs on top of TCP, integrating smoothly into the TCP/IP protocol stack.

1.4 IDXP (Internet Draft)

IDXP (Intrusion Detection eXchange Protocol) is implemented as a BEEP profile. IDXP packets transmit data encoded using Intrusion Detection Message Exchange Format (IDMEF), the focus of a previous clinic. The main purpose of IDXP is to transport intrusion detection alert data. This is currently only possible if no firewalls block the packets.

1.5 Security

It is important to ensure secure and authenticated transmission of data, especially when it involves traversing a firewall intended to increase security. BEEP provides tools and methods for easily negotiating these properties.

1.6 Tunnel (Internet Draft)

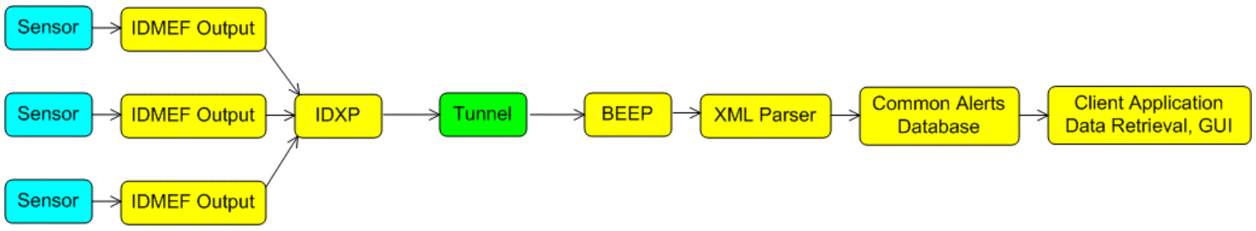


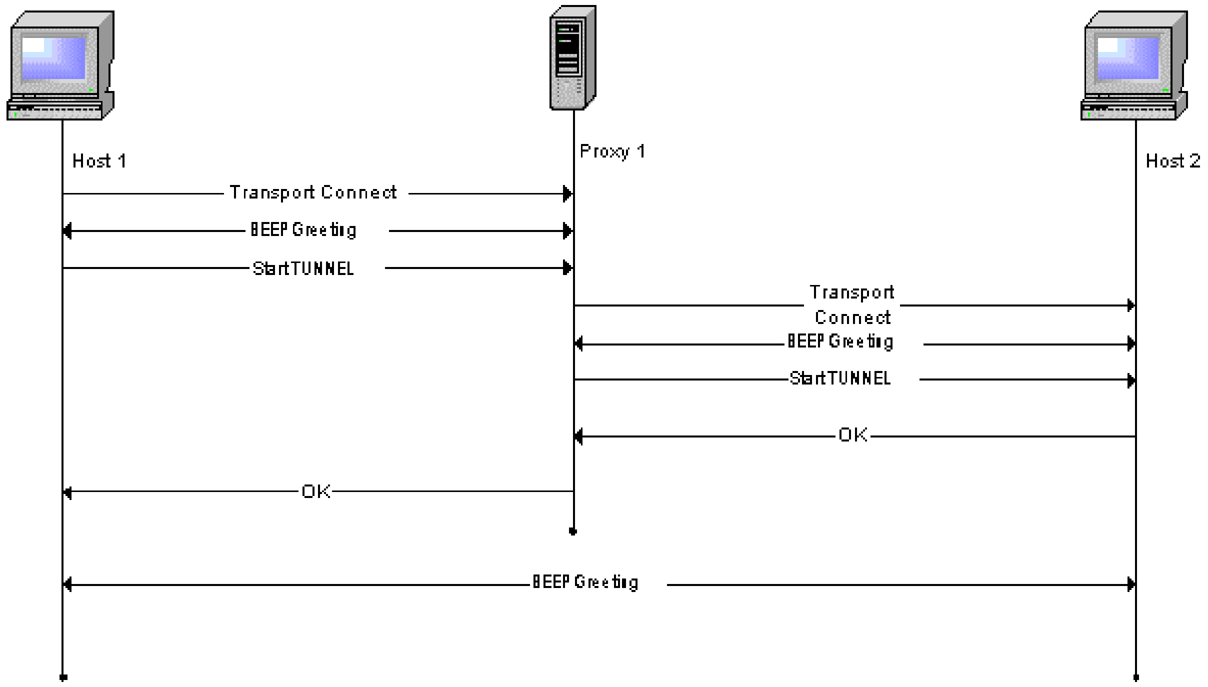
Figure 1: Conceptual Layout
Light Blue: 3rd party IDS
Yellow: Assumed to work properly
Green: The focus of our clinic

Figure 1 shows the ideal intrusion detection situation. Intrusion Sensors use their innate technology to generate alerts. All Sensors use the IDMEF syntax to specify their alerts. Alerts are then transported via the IDXP/BEEP protocol to Analyzers, where a database and UI exist to allow the manager to recognize intrusions and to take appropriate action. Tunnel exists to provide a means to transport the intrusion alerts through any number of firewalls or proxies that might exist between the Sensor and Analyzer.

Tunnel provides a way for BEEP peers to form an application-layer-tunnel. Peers exchange Tunnel packets to establish a connection between them that acts as a point-to-point connection between the two peers. It is a profile that falls between IDXP and BEEP, tuning the BEEP session to have the necessary characteristics for creating the connection between the peers. In the diagram above, each sensor will want a point-to-point connection with the XML Parser, and thus each will form its own tunnel over BEEP to allow the IDXP messages to be exchanged between the sensors and the parser.

Tunnel, like most other BEEP profiles, uses XML for data transfer. Its elements are layered, so that the outermost element specifies the next hop in the connection, or through the use of an empty Tunnel element, that it has reached an endpoint. A proxy refers to any BEEP peers between the start and endpoint in the tunnel; firewalls are a common example. Once the initial connection has been setup between the begin and end points, the proxies between them transparently transmit whatever is sent to them, not checking for BEEP syntax, allowing each peer to encrypt their messages without the proxies being able to view them. In addition once a tunnel is established, peers can send non-BEEP data through it. The proxies can also use whatever security features they wish to manage their immediate connections. A proxy can limit tunnels to certain machines to only those hosts that are authorized and authenticated through Simple Authentication and Security Layer (SASL).

The IETF Tunnel Internet Draft provides examples of the expected behavior for some common expected scenarios. The simplest example would be two hosts separated by one peer, with the initiator knowing the path to the listener. As shown in the following figure:

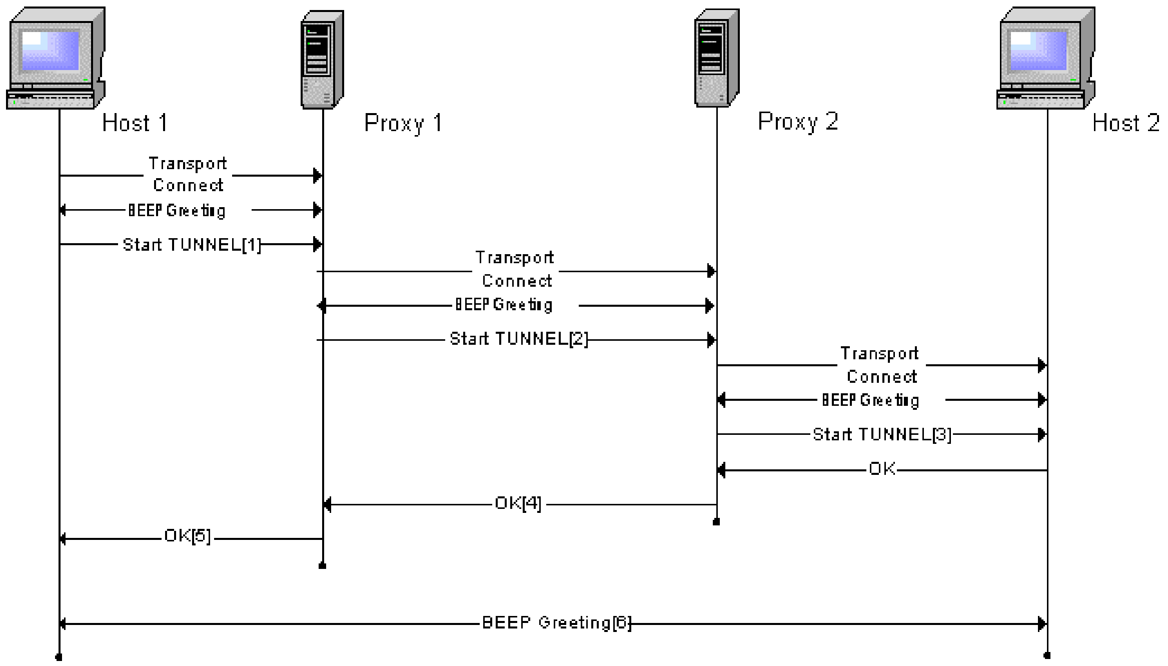


Host 1 is separated from Host 2 by Proxy 1. Host 1 connects to the proxy, starts a BEEP session, and sends the Tunnel start message with 2 layers: The outermost being the identifier for Host 2 and the innermost being an empty Tunnel element, signifying that the second hop is the endpoint (Host 2).

After the initial connection between Host 1 and Proxy 1, Proxy 1 connects to Host 2 and starts a second BEEP session. Proxy 1 sends an initial Tunnel message, signifying that Host 2 is an endpoint, at which point Host 2 replies with an OK message (assuming it is willing to accept the tunnel). After receiving the OK, Proxy 1 sends it's own OK back to Host 1 and begins transparently forwarding all messages between Host 2 and Host 1.

One major aspect of Tunnel is that it can be directed to a host based on various criteria, the simplest would be fully qualified domain names or IPv4 addresses, but it can also work if given a service or port to connect to.

For a more complicated example, we can use 2 proxies between the 2 hosts. The following diagram shows such an exchange:



If Host 1 knows the full path to Host 2 through the 2 proxies, then it connects to Proxy 1 the same way as it would in the 1-hop example, and when it sends the initial Tunnel start message (line #1) it will send a message with 3 layers: The outermost being the identifier of Proxy 2, the middle being the identifier of Host 2, and the innermost being blank. Proxy 1 then strips off the outermost Tunnel element and initiates a connection to Proxy 2. After starting a BEEP session, it sends Proxy 2 the Tunnel message consisting of the inner 2 Tunnel elements (line #2). Proxy 2 connects to Host 2, initiates the BEEP session, and sends the innermost Tunnel element (line #3). This informs Host 2 that it is the expected endpoint. If Host 2 accepts the connection, it sends an OK to Proxy 2. Proxy 2 sends an OK to Proxy 1 and anything that Proxy 2 receives from Host 2 after the OK (line #4) is sent transparently to Proxy1. Upon receiving the OK from Proxy 2, Proxy 1 sends an OK to Host 1 and immediately after begins transparently forwarding messages from Proxy 2 to Host 1.

If Host 1 does not know the full path to Host 2, but knows that a path exists to it through Proxy 1, it can still establish a tunnel to Host 2. The connection between Host 1 and Proxy 1 begins as above, but when sending the Tunnel start message to Proxy 1, (line #1 in the diagram) Host 1 sends only the identifier for Host 2 (the service running on Host 2, the requested profile on Host 2, or a specific identifying string for Host 2) and no empty Tunnel element. Proxy 1 should be able to determine that Host 2 lies somewhere beyond Proxy 2, and thus connects to Proxy 2 as above, and sends a Tunnel start message with a single element, the identifier for Host 2 (line #2). Upon receiving this message, Proxy 2 will realize that Host 2 is connected directly to it, and thus should connect as above, and as above send a single, empty Tunnel message to Host 2, signifying that it is the endpoint (line #3). The rest of the example carries on the same way as above.

One final thing to note is that the message sent after the tunnel is built (line # 6) does not actually have to be a BEEP message, although it is in these examples. The proxies in between do not read any of the messages passing between them, so any type of data may be sent over the tunnel as needed. The hosts can work out some form of encryption preventing the proxies or anything else between them from reading the messages, keeping them secure.

The most useful features that Tunnel provides for use in Intrusion Detection come about when a manager and an analyzer are separated by a proxy. It allows the proxy to authenticate the manager, verifying that they are authorized to connect to the analyzer. It can also insulate the analyzer which is behind the proxy from outside attacks, since the analyzer's IP address does not ever need to be revealed to anyone outside the proxy.

2 Evaluation of Alternatives to Tunnel

2.1 SSL (Secure Sockets Layer)/TLS (Transport Layer Security) (RFC 2246)

TLS (Transport Layer Security) is the IETF version of SSLv3 (Secure Sockets Layer) and was mainly intended for secure transportation of HTTP traffic. In practice it is also used to secure NNTP (news), IMAP, and POP (mail) traffic as well. In the protocol stack, TLS lies between TCP and the application layers and usually provides an API similar to the BSD socket API for secured communication. Applications that wish to make use of SSL will require minimal changes to work properly. In addition to encryption, TLS provides server authentication via certificates and optionally client authentication as well. While using certificates allows a client and server to authenticate without having a pre-shared secret, spoofed certificates make it more prone to man-in-the-middle attacks. However, TLS (in the SSLv3 incarnation) is widely deployed as client configuration is simple, as it does not address any access control issues. Tunnel is able to use TLS as it is supported by BEEP.

2.2 SASL (RFC 2222)

SASL (Simple Authentication and Security Layer) is a method for adding authentication and security support to connection-based protocols. It is a framework for providing a protocol with mechanisms for authentication, integrity checking, and encryption. Some SASL mechanisms will negotiate which services to provide for the protocol, while others have a predetermined set of services. SASL allows the network administrator to configure the proper level of security for that environment; Tunnel benefits from this since BEEP makes use of SASL mechanisms.

2.3 SSH Tunneling

SSH (Secure Shell) is a pair of applications in the client/server model that are used to replace the rlogin and telnet programs. All communication between the client and server are encrypted, thus providing data confidentiality in addition to client authentication. The mass adoption of SSH implies that in most firewalls, port 22 is left open for SSH communication, and thus SSH is often used to create secure tunnels through firewalls. However, this approach is not without it's drawbacks: SSH is an application level tool and applications that wish to use SSH tunnels must manually create the tunnel through SSH, the client must name an explicit endpoint to connect to, SSH only provides client authentication and no host authentication, and all traffic is encrypted. Tunnel improves upon SSH tunneling by being more flexible in authentication and encryption details, providing address anonymity for machines behind the firewall, and for being able to create tunnels without an explicit endpoint (as in the case of services).

2.4 VPN

A VPN (Virtual Private Network) is a secure, permanent, private network built on a publicly accessible infrastructure such as the Internet or telephone network. A VPN is transparent in that the traffic it carries is unaware of any intermediate nodes between the endpoints and the intermediate nodes are unaware they are carrying traffic that is part of the VPN. In addition, a VPN provides some combination of encryption and strong authentication of remote users and hosts, and thus most VPN implementations are fairly intrusive on the client node. Tunnel is easier than a VPN to administer and deploy since most configuration is done only on the firewall. In addition, Tunnel provides more policy flexibility and is easier to configure than a VPN.

2.5 IPsec (http://www.ietf.org/html.charters/ipsec-charter.html)

IPsec (IP Security) is a protocol designed to protect IP (Internet Protocol) from attack. In doing so, it also protects all protocols that run on IP such as TCP and UDP. Thus, applications running on IP benefit from increased security without recompiling. However, by its nature as a protocol-level enhancement, IPsec requires modification of the IP stack, which usually resides in the kernel and is thus very invasive to operating systems. Combined with the fact that IPsec is a peer-to-peer protocol, deployment of IPsec is very difficult unless all machines are running the same operating system or all operating systems have interoperable IPsec implementations. Due to the strict enforcement of IP address consistency, IPsec does not operate correctly behind a network address translator (NAT). In comparison, Tunnel is easier to deploy, more configurable, operates properly with NAT, and handles proxies.

3 Technical Approach

3.1 Problem Definition

We will implement the Tunnel protocol as a BEEP profile. See above explanations of Tunnel and BEEP. If we realize, under development or analysis that the specifications for Tunnel are incomplete or flawed, we will consult with the author of the specification and clear up any issues that arise. Deliverables include functioning Tunnel profiles fully supporting everything as explained in the Tunnel Internet Draft. Client software will be written in both C and Java and work interchangeably with one another. Server and firewall software will probably be written in only C because it needs to be fast to handle multiple connections quickly.

3.2 Approach to Problem

We will begin by choosing which BEEP implementations to use in both C and Java. The next step is to implement a BEEP profile allowing a single host to communicate with itself in a contrived situation. Once the single-host scenario works, we will move on to getting two hosts communicating in a peer-to-peer setup. This will involve the use of TLS which introduces some extra complexity. The next step is to write an application for a firewall proxy, which will allow hosts on opposite sides of the firewall to communicate as if it were a normal peer-to-peer situation.

Work on multiple hops and interoperability will really get going in the second semester. Interoperability refers to using C and Java implementations interchangeably in a single exchange between two hosts. Multiple hops refers to the possibility that not only one firewall sits between a host and its desired destination host. For a more detailed look at the proposed schedule, see Appendix A.

4 Capabilities

4.1 Dependencies

Implementation of Tunnel as a BEEP profile depends upon a working version of BEEP. Some versions exist, but are still works in progress and cannot be guaranteed to function properly. We will start with the most trusted one available in both Java and C and go from there. If BEEP cannot operate in the way that it should, this may become a problem, but we hope that only the obscure functionality will have problems if any.

4.2 Facilities

4.2.1 Hardware

We currently have a single consumer grade personal computer running Redhat 7.3, which will begin as our single host for testing BEEP profiles. It has an Intel Pentium 4 processor and 256MB of RAM, which should be enough for the relatively lightweight open-source software we anticipate running. If this is not powerful enough to run the Tunnel Protocol as we write it, we cannot expect any adoption. Additional machines when needed for firewall or peer-to-peer testing are also available, just not yet acquired.

4.2.2 Software

We plan to use Java and C to implement the BEEP Profiles for Tunnel. The two languages are needed to fulfill not only the desires of Aerospace, but also the requirements for acceptance as an Internet Standard. We plan to use a free web-server daemon (apache) for communicating our work to others via the Internet. Additionally, we have a variety of Microsoft operating system licenses available to us, which may become useful for cross-platform compatibility testing.

4.3 Literature

Tunnel, which is not even a standard yet, and BEEP, which is also relatively new, have understandably few books published about them. One that looks particularly useful, written by the author of the BEEP RFC is listed below along with a reference we have on SSL and TLS:

- Rose, Marshall T. BEEP: The Definitive Guide. O'Reilly 2002.
- Rescorla, Eric. SSL and TLS: Designing and Building Secure Systems. Addison Wesley.

We will also be using the Internet Drafts generated by the IDWG for further information with regard to the exact specifications of the standard. Additionally we may use RFC 3080 which describes the BEEP protocol.

5 Personnel

- Nick Hertl (Project Manager)

  Nick is a senior Computer Science major at Harvey Mudd College. He has worked in System Administration for three years, mostly for the HMC Computer Science Department. He spent the past two summers working at Microsoft, the most recent of which involved some high-level network programming. His main experience with low level network protocols comes from a Computer Networks class taken in the Spring of 2002 with Professor Mike Erlinger. He skis on both snow and water in his rare free time.

- Will Berriel

  Will is senior Computer Science major at Harvey Mudd College. He has taken courses in Operating Systems and Comp uter Networks. He has worked with BEEP before in developing an IDXP profile for PermaBEEP and client and server applications to transmit IDMEF messages between an analyzer and a viewer. He currently competes for the Claremont Colleges in Cross Country and Track.

- Richard Fujiama

  Richard is a senior Computer Science major at Harvey Mudd College. He has several years of experience as a Windows and Unix system administrator as well as experience gained from research with Mike Erlinger on the IDXP protocol. He is comfortable with network programming in the C and Java languages. His interests include operating systems, martial arts, and entrepreneurship.

- Chip Bradford

  Chip is a senior Computer Science major at Harvey Mudd College. He has extensive experience programming C/C++, especially in low-level environments such as Operating Systems and the network stack. His summer was spent setting up a framework for automated testing of a piece of billing software developed by LPASystems for Xerox. He is currently tutoring the Networks class at Harvey Mudd. His other interests include computer games and parties.

Faculty Advisor
Professor Michael Erlinger

Company Liaisons
Joseph Betser, Ph.D.
Rayford Sims