Network Working Group
Request for Comments: 3093
Category: Informational

M. Gaynor
S. Bradner
Harvard University

1 April 2001

Firewall Enhancement Protocol (FEP)

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard of any kind. Distribution of this memo is unlimited.

Copyright Notice

Abstract

Internet Transparency via the end-to-end architecture of the Internet has allowed vast innovation of new technologies and services [1]. However, recent developments in Firewall technology have altered this model and have been shown to inhibit innovation. We propose the Firewall Enhancement Protocol (FEP) to allow innovation, without violating the security model of a Firewall. With no cooperation from a firewall operator, the FEP allows ANY application to traverse a Firewall. Our methodology is to layer any application layer Transmission Control Protocol/User Datagram Protocol (TCP/UDP) packets over the HyperText Transfer Protocol (HTTP) protocol, since HTTP packets are typically able to transit Firewalls. This scheme does not violate the actual security usefulness of a Firewall, since Firewalls are designed to thwart attacks from the outside and to ignore threats from within. The use of FEP is compatible with the current Firewall security model because it requires cooperation from a host inside the Firewall. FEP allows the best of both worlds: the security of a firewall, and transparent tunneling thought the firewall.

1.0 Terminology

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

2.0 Introduction

The Internet has done well, considering that less than 10 years ago the telco's were claiming it could not ever work for the corporate environment. There are many reasons for this; a particularly strong one is the end-to-end argument discussed by Reed, Seltzer, and Clark [2]. Innovation at the ends has proven to be a very powerful methodology creating more value than ever conceived of. But, the world is changing as Clark notes in [6]. With the connection of the corporate world to the Internet, security concerns have become paramount, even at the expense of breaking the end-to-end paradigm. One example of this is the Firewall - a device to prevent outsiders from unauthorized access into a corporation. Our new protocol, the Firewall Enhancement Protocol (FEP), is designed to restore the end-to-end model while maintaining the level of security created by Firewalls.

To see how powerful the end-to-end model is consider the following example. If Scott and Mark have a good idea and some implementation talent, they can create an artifact, use it, and send it to their friends. If it turns out to be a good idea these friends can adopt it and maybe make it better. Now enter the Firewall: if Mark happens to work at a company that installs a Firewall, he can't experiment with his friend Scott. Innovation is more difficult, maybe impossible. What business is it of an IT manager if Scott and Mark want to do some experiments to enable them to better serve their users? This is how the web was created: one guy with talent, a few good ideas, and the ability to innovate.

Firewalls are important, and we do respect the right of anybody to protecting themselves any way they want (as long as others are not inconvenienced). Firewalls work, and have a place in the Internet. However, Firewalls are built to protect from external threats, not internal ones. Our proposed protocol does not break the security model of the Firewall; it still protects against all external risks that a particular Firewall can protect against. For our protocol to work someone inside the Firewall must run an application level protocol that can access TCP port 80. Our concept allows a consistent level of security while bypassing the IT manager in charge of the Firewall. We offer freedom to innovate without additionally compromising external security, and the best part, no need to waste time involving any managers for approval.

We got this idea from the increasing number of applications that use HTTP specifically because it can bypass Firewall barriers. This piecemeal deployment of specific applications is not an efficient way to meet the challenge to innovation created by Firewalls. We decided to develop a process by which TCP/IP itself is carried over HTTP.

With this innovation anyone can use any new TCP/IP application immediately without having to go through the laborious process of dealing with Firewall access for the particular application. An unintended byproduct of this proposal is that existing TCP/IP applications can also be supported to better serve the users. With FEP, the users can decide what applications they can run.

Our protocol is simple and is partly based on the Eastlake [3] proposal for MIME encoding of IP packets. We use the ubiquitous HTTP protocol format. The IP datagram is carried in the message body of the HTTP message and the TCP packet header information is encoded into HTTP headers of the message. This ASCII encoding of the header fields has many advantages, including human readability, increasing the debuggability of new applications, and easy logging of packet information. If this becomes widely adopted, tools like tcpdump will become obsolete.

3.0 FEP Protocol

Figure 1 shows a high level view of our protocol. The application (1) in host A (outside the Firewall) sends a TCP/IP datagram to host B (within the firewall). Using a tunnel interface the TCP/IP datagram is routed to our FEP software (2), which encodes the datagram within a HTTP message. Then this message is sent via a HTTP/TCP/IP tunnel (3) to host B on the normal HTTP port (4). When it arrives at host B, this packet is routed via the tunnel to the FEP software (5), which decodes the packet and creates a TCP/IP datagram to insert into host’s B protocol stack (6). This packet is routed to the application on host B (7), as if the Firewall (8) never existed.

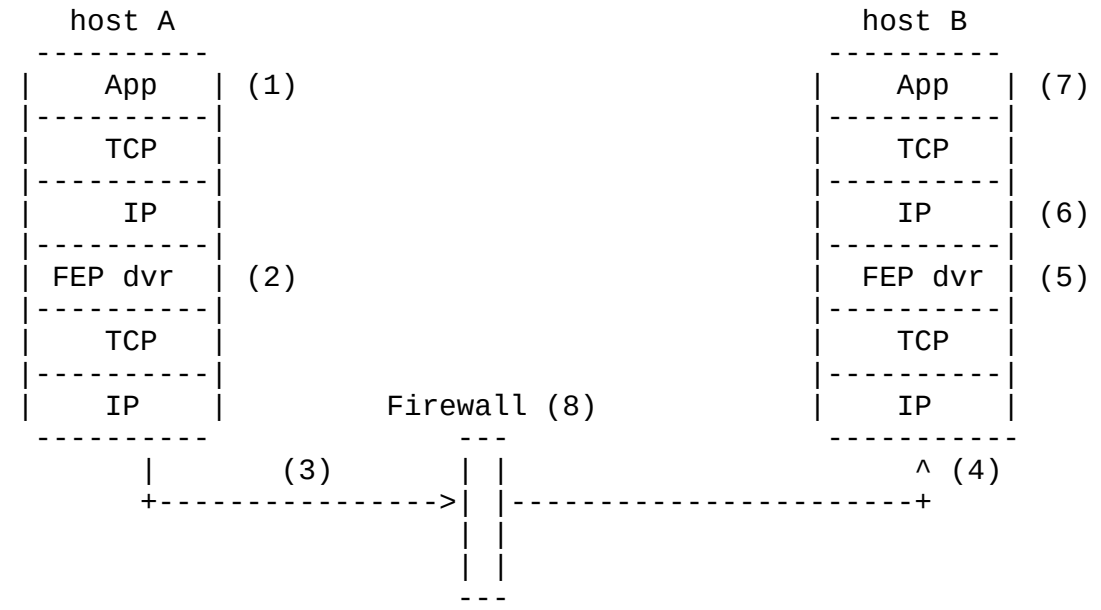

Figure 1

3.1 HTTP Method

FEP allows either side to look like a client or server. Each TCP/IP packet is sent as either a HTTP GET request or a response to a GET request. This flexibility work well with firewalls that try to verify valid HTTP commands crossing the Firewall stopping the unwanted intercepting of FEP packets.

3.2 TCP Header Encapsulation:

The TCP/IP packet is encoded into the HTTP command in two (or optionally three) steps. First, the IP packet is encoded as the message body in MIME format, as specified in [3]. Next, the TCP [4] packet header is parsed and encoded into new HTTP headers. Finally, as an option, the IP header can also be encoded into new optional HTTP headers. Encoding the TCP and optionally the IP header is strictly for human readability, since the entire IP datagram is encoded in the body part of the HTTP command.

This proposal defines the following new HTTP headers for representing TCP header information.

TCP_value_opt - This ASCII string represents the encoding type for the TCP fields where a mandatory encoding type is not specified. The legitimate values are:

TCP_binary - ASCII representation of the binary representation of the value of the field.

TCP_hexed - ASCII representation of the hex representation of the value of the field.

TCP_Sport - The 16-bit TCP Source Port number, encoded as an ASCII string representing the value of port number.

TCP_Dport - The 16-bit TCP Destination Port number, encoded as an ASCII string representing the value of the port number.

TCP_SeqNum - The 32-bit Sequence Number, encoded as an ASCII string representing the hex value of the Sequence number. This field MUST be sent as lower case because it is not urgent.

TCP_Ackl - The 32-bit Acknowledgement Number, encoded as ASCII string representing the value of the Acknowledgement number.

TCP_DODO - The 4-bit Data Offset value, encoded as an ASCII string representing the base 32 value of the actual length of TCP header in bits. (Normally this is the Data value times 32.)

TCP_6Os - The 6 reserved bits, encoded as a string of 6 ASCII characters. A "O" ("Oh") represents an "Off" bit and "O" ("Oh") represents an "On" bit. (Note these characters MUST all be sent as "off" and MUST be ignored on receipt.)

TCP_FlgBts - The TCP Flags, encoded as the set of 5 comma-separated ASCII strings: [{URG|urg}, {ACK|ack}, {PSH|psh}, {RST|rst}, {SYN|syn}, {FIN|fin}]. Capital letters imply the flag is set, lowercase means the flag is not set.

TCP_Windex - The 16-bit TCP Window Size, encoded as an ASCII string representing the value of the number of bytes in the window.

TCP_Checkit - The 16-bit TCP Checksum field, encoded as an ASCII string representing the decimal value of the ones-complement of the checksum field.

TCP_UP - The 16-bit TCP Urgent Pointer, encoded as the hex representation of the value of the field. The hex string MUST be capitalized since it is urgent.

TCP_Opp_Lst - A comma-separated list of any TCP options that may be present. Each option is encoded as an ASCII string representing the name of the option followed by option-specific information enclosed in square brackets. Representative options and their encoding follow, other IP options follow the same form:

End of Options option: ["End of Options"]

Window scale option: ["Window scale", shift_count], where shift_count is the window scaling factor represented as the ASCII string in decimal.

3.2 IPv4 Header Encapsulation:

This proposal defines the following new HTTP headers for representing IPv4 header information:

These optional headers are used to encode the IPv4 [5] header for better readability. These fields are encoded in a manner similar to the above TCP header fields.

Since the base IP packet is already present in an HTTP header, the following headers are optional. None, some or all of them may be used depending on the whim of the programmer.

IP_value_opt - This ASCII string represents the encoding type for the following fields where a mandatory encoding type is not specified. The legitimate values are the same as for TCP_value_opt.

IP_Ver - The IP Version number, encoded as an UTF-8 string. The legitimate values for the string are "four", "five", and "six." The encapsulation of the fields in the IP header are defined in this section if the value is "four", and in section 3.3 if the value is "six". Encapsulations for headers with IP_Ver value of "five" will be developed if the right orders are received. Encapsulations for headers with the IP_Ver value of "eight" are empty. Implementations MUST be able to support arbitrary native languages for these strings.

IP4_Hlen - The IP Internet Header Length field, it is encoded in the same way as TCP_DODO.

IP4_Type_of_Service (this name is case sensitive) - This is an obsolete name for a field in the IPv4 header, which has been replaced with IP_$$ and IP_CU.

IP_$$ - The 6-bit Differentiated Services field, encapsulated as an UTF-8 string representing the name of the DS codepoint in the field.

IP_CU - The 2-bit field that was the two low-order bits of the TOS field. Since this field is currently being used for experiments it has to be coded in the most general way possible, thus it is encoded as two ASCII strings of the form "bit0=X" and "bit1=X," where "X" is "on" or "off." Note that bit 0 is the MSB.

IP4_Total - The 16-bit Total Length field, encoded as an ASCII string representing the value of the field.

IP4_SSN - The IP Identification field, encoded as an ASCII string representing the value of the field.

IP4_Flags - The IP Flags, encoded as the set of 3 comma separated ASCII strings: [{"Must Be Zero"}, {"May Fragment"|"Don't Fragment"}, {"Last Fragment"|"More Fragments"}]

IP4_Frager - The 13-bit Fragment Offset field, encoded as an ASCII string representing the value of the field.

IP4_TTL - The 8-bit Time-to-Live field, encoded as an UTF-8 string of the form "X hops to destruction." Where "X" is the decimal value -1 of the field. Implementations MUST be able to support arbitrary languages for this string.

IP4_Proto - The 8-bit Protocol field, encoded as an UTF-8 string representing the common name for the protocol whose header follows the IP header.

IP4_Checkit - The 16-bit Checksum field, encoded in the same way as TCP_Checkit.

IP4_Apparent_Source - The 32-bit Source Address field. For user friendliness this is encoded as an UTF-8 string representing the domain name of the apparent sender of the packet. An alternate form, to be used when the domain name itself might be blocked by a firewall programmed to protect the innocence of the corporate users, is an ASCII string representing the dotted quad form of the IPv4 address.

IP4_Dest_Addr - The 32-bit Destination Address field, encoded in the same way as is IP4_Apparent_Source.

IP4_Opp_Lst - A comma-separated list of all IPv4 options that are present. Each option is encoded as an ASCII string representing the name of the option followed by option-specific information enclosed in square brackets. Representative options and their encoding follow, other IP options follow the same form:

End of Options option: ["End of Options"]

Loose Source Routing option: ["Loose Source Routing", length, pointer, IP4_addr1, IP4_addr2, ...], where length and pointer are ASCII strings representing the value of those fields.

3.3 IPv6 Header Encapsulation:

This proposal defines the following new HTTP headers for representing IPv6 header information:

These optional headers encode the IPv6 [5] header for better readability. These fields are encoded in a manner similar to the above TCP header fields.

Since the base IP packet is already present in an HTTP header the following headers are optional. None, some or all of them may be used depending on the whim of the programmer. At this time only the base IPv6 header is supported. If there is sufficient interest, support will be developed for IPv6 extension headers.

IP_$$ - the 6-bit Differentiated Services field - see above

IP_CU - the 2-bit unused field - see above

IP6_Go_with_the_Flow - The 20-bit Flow Label field. Since this field is not currently in use it should be encoded as the UTF-8 string "do not care".

IP6_PayLd - The 16-bit Payload Length field, encoded as an ASCII string representing the value of the field. The use of FEP with IPv6 jumbograms is not recommended.

IP6_NxtHdr - The 8-bit Next Header field, encoded in the same way as IP4_Proto.

IP6_Hopping - The 8-bit Hop Limit field, encoded in the same way as IP4_TTL.

IP6_Apparent_Source - The 128-bit Source Address field. For user friendliness, this is encoded as an UTF-8 string representing the domain name of the apparent sender of the packet. An alternate form, to be used when the domain name itself might be blocked by a Firewall programmed to protect the innocence of the corporate users, is an ASCII string representing any one of the legitimate forms of representing an IPv6 address.

IP6_Dest_Addr - The 128-bit Destination Address field, encoded the same way as IP6_Apparent_Source.

3.4 TCP Header Compression

Compressing TCP headers in the face of a protocol such as this one that explodes the size of packets is silly, so we ignore it.

4.0 Security Considerations

Since this protocol deals with Firewalls there are no real security considerations.

5.0 Acknowledgements

We wish to thank the many Firewall vendors who have supported our work to re-enable the innovation that made the Internet great, without giving up the cellophane fig leaf of security that a Firewall provides.

6.0 Authors' Addresses

Mark Gaynor
Harvard University
Cambridge MA 02138

EMail gaynor@eecs.harvard.edu

Scott Bradner
Harvard University
Cambridge MA 02138

Phone +1 617 495 3864
EMail sob@harvard.edu

References

[1] Carpenter, B., "Internet Transparency", RFC 2775, February 2000.

[2] Saltzer, J., Reed, D., and D. Clark, "End-to-End Arguments in System Design". 2nd International Conference on Distributed Systems, Paris, France, April 1981.

[3] Eastlake, D., "IP over MIME", Work in Progress.

[4] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

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

[6] Clark, D. and M. Blumenthal, "Rethinking the Design of the Internet: The end-to-end argument vs. the brave new world". 2000.

Full Copyright Statement

Acknowledgement

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