Network Working Group	S. Hartman, Ed.
Internet-Draft	Painless Security
Intended status: Standards Track	J. Howlett
Expires: September 2, 2010	JANET(UK)
	March 01, 2010

A GSS-API Mechanism for the Extensible Authentication Protocol
draft-howlett-eap-gss-00.txt

Abstract

This document defines protocols, procedures, and conventions to be employed by peers implementing the Generic Security Service Application Program Interface (GSS-API) when using the EAP mechanism.

Status of this Memo

This Internet-Draft will expire on September 2, 2010.

Copyright Notice

Table of Contents

---

1. Introduction

TOC

The Extensible Authentication Protocol (EAP) [RFC3748] (Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, “Extensible Authentication Protocol (EAP),” June 2004.) defines a framework for authenticating a network access client and server in order to gain access to a network. A variety of different EAP methods are in wide use; one of EAP's strengths is that for most types of credentials in common use, there is an EAP method that permits the credential to be used.

EAP is often used in conjunction with a backend authentication server via RADIUS [RFC3579] (Aboba, B. and P. Calhoun, “RADIUS (Remote Authentication Dial In User Service) Support For Extensible

Authentication Protocol (EAP)," September 2003.) or Diameter [RFC4072] (Eronen, P., Hiller, T., and G. Zorn, "Diameter Extensible Authentication Protocol (EAP) Application," August 2005.). In this mode, the NAS simply tunnels EAP packets over the backend authentication protocol to a home EAP/AAA server for the client. After EAP succeeds, the backend authentication protocol is used to communicate key material to the NAS. In this mode, the NAS need not be aware of or have any specific support for the EAP method used between the client and the home EAP server. The client and EAP server share a credential that depends on the EAP method; the NAS and AAA server share a credential based on the backend authentication protocol in use. The backend authentication server acts as a trusted third party enabling network access even though the client and NAS may not actually share any common authentication methods. Using AAA proxies, this mode can be extended beyond one organization to provide federated authentication for network access.

The Generic Security Services Application Programming Interface (GSS-API) [RFC2743] (Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1," January 2000.) provides a generic framework for applications to use security services including authentication and per-message data security services. Between protocols that support GSS-API directly or protocols that support SASL [RFC4422] (Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)," June 2006.), many application protocols can use GSS-API for security services. However, with the exception of Kerberos [RFC4121] (Zhu, L., Jaganathan, K., and S. Hartman, "The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2," July 2005.), few GSS-API mechanisms are in wide use on the Internet. While GSS-API permits an application to be written independent of the specific GSS-API mechanism in use, there is no facility to separate the server from the implementation of the mechanism as there is with EAP and backend authentication servers.

The goal of this specification is to combine GSS-API's support for application protocols with EAP/AAA's support for common credential types and for authenticating to a server without requiring that server to specifically support the authentication method in use. In addition, this specification supports the use of the Security Assertion Markup Language to transport assertions about attributes of client subjects to servers. Together this combination will provide federated authentication and authorisation for GSS-API applications.

This mechanism is a GSS-API mechanism that encapsulates an EAP conversation. From the perspective of RFC 3748, this specification defines a new lower-layer protocol for EAP.

Section 1.3 of [RFC5247] (Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework," August 2008.) outlines the typical conversation between EAP peers where an EAP key is derived:

- Phase 0: Discovery

- Phase 1: Authentication
- 1a: EAP authentication
- 1b: AAA Key Transport (optional)
- Phase 2: Secure Association Protocol
- 2a: Unicast Secure Association
- 2b: Multicast Secure Association (optional)

1.1. Discovery

TOC

GSS-API peers discover each other and discover support for GSS-API in an application-dependent mechanism. SASL [RFC4422] (Melnikov, A. and K. Zeilenga, “Simple Authentication and Security Layer (SASL),” June 2006.) describes how discovery of a particular SASL mechanism such as a GSS-API mechanism is conducted. The Simple and Protected Negotiation mechanism (SPNEGO) [RFC4178] (Zhu, L., Leach, P., Jaganathan, K., and W. Ingersoll, “The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism,” October 2005.) provides another approach for discovering what GSS-API mechanisms are available. The specific approach used for discovery is out of scope for this mechanism.

1.2. Authentication

TOC

GSS-API authenticates a party called the GSS-API initiator to the GSS-API acceptor, optionally providing authentication of the acceptor to the initiator. Authentication starts with a mechanism-specific message called a context token sent from the initiator to the acceptor. The acceptor may respond, followed by the initiator, and so on until authentication succeeds or fails. GSS-API context tokens are reliably delivered by the application using GSS-API. The application is responsible for in-order delivery and retransmission.

EAP authentication can be started by either the peer or the authenticator. The EAP peer maps onto the GSS-API initiator and the EAP authenticator and EAP server maps onto the GSS-API acceptor. EAP messages from the peer to the authenticator are called responses; messages from the authenticator to the peer are called requests.

This specification permits a GSS-API peer to hand-off the processing of the EAP packets to a remote EAP server by using AAA protocols such as

RADIUS, RadSec or Diameter. In this case, the GSS-API peer acts as an EAP pass-through authenticator. If EAP authentication is successful, and where the chosen EAP method supports key derivation, EAP keying material may also be derived. If an AAA protocol is used, this can also be used to replicate the EAP Key from the EAP server to the EAP authenticator.
See Section 5 (Context Tokens) for details of the authentication exchange.

1.3. Secure Association Protocol

TOC

After authentication succeeds, GSS-API provides a number of per-message security services that can be used:

> GSS_Wrap() provides integrity and optional confidentiality for a message.
>
> GSS_GetMIC() provides integrity protection for data sent independently of the GSS-API
>
> GSS_Pseudo_random [RFC4401] (Williams, N., “A Pseudo-Random Function (PRF) API Extension for the Generic Security Service Application Program Interface (GSS-API),” February 2006.) provides key derivation functionality.

These services perform a function similar to security association protocols in network access. Like security association protocols, these services need to be performed near the authenticator/acceptor even when a AAA protocol is used to separate the authenticator from the EAP server. The key used for these per-message services is derived from the EAP key; the EAP peer and authenticator derive this key as a result of a successful EAP authentication. In the case that the EAP authenticator is acting as a pass-through it obtains it via the AAA protocol. See Section 6 (Acceptor Services) for details.

---

2. Requirements notation

TOC

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 [RFC2119] (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.).

---

3. EAP Channel Binding and Naming

TOC

EAP authenticates a realm. The peer knows that it has exchanged authentication with an EAP server in a given realm. Today, the peer does not typically know which NAS it is talking to securely. That is often fine for network access. However privileges to delegate to a chat server seem very different than privileges for a file server or trading site. Also, an EAP peer knows the identity of the home realm, but perhaps not even the visited realm.

In contrast, GSS-API takes a name for both the initiator and acceptor as inputs to the authentication process. When mutual authentication is used, both parties are authenticated. The granularity of these names is somewhat mechanism dependent. In the case of the Kerberos mechanism, the acceptor name typically identifies both the protocol in use (such as IMAP) and the specific instance of the service being connected to. The acceptor name almost always identifies the administrative domain providing service.

An EAP GSS-API mechanism needs to provide GSS-API naming semantics in order to work with existing GSS-API applications. EAP channel binding [I-D.ietf-emu-chbind] (Clancy, C. and K. Hoeper, “Channel Binding Support for EAP Methods,” October 2009.) is used to provide GSS-API naming semantics. Channel binding sends a set of attributes from the peer to the EAP server either as part of the EAP conversation or as part of a secure association protocol. In addition, attributes are sent in the backend authentication protocol from the authenticator to the EAP server. The EAP server confirms the consistency of these attributes. Confirming attribute consistency also involves checking consistency against a local policy database as discussed below. In particular, the peer sends the name of the acceptor it is authenticating to as part of channel binding. The acceptor sends its full name as part of the backend authentication protocol. The EAP server confirms consistency of the names.

EAP channel binding is easily confused with a facility in GSS-API also called channel binding. GSS-API channel binding provides protection against man-in-the-middle attacks when GSS-API is used as authentication inside some tunnel; it is similar to a facility called cryptographic binding in EAP. See [RFC5056] (Williams, N., “On the Use of Channel Bindings to Secure Channels,” November 2007.) for a discussion of the differences between these two facilities and Section 6.1 (GSS-API Channel Binding) for how GSS-API channel binding is handled in this mechanism.

TOC

3.1. Mechanism Name Format

Before discussing how the initiator and acceptor names are validated in the AAA infrastructure, it is necessary to discuss what composes a name for an EAP GSS-API mechanism. GSS-API permits several types of generic names to be imported using GSS_Import_name(). Once a mechanism is chosen, these names are converted into a mechanism name form. This section first discusses name types that need to be imported and then discusses the structure of the mechanism name.

The GSS_C_NT_USER_NAME form represents the name of an individual user. From the standpoint of this mechanism it may take the form either of an undecorated user name or a network access identifier (NAI) [RFC4282] (Aboba, B., Beadles, M., Arkko, J., and P. Eronen, “The Network Access Identifier,” December 2005.).

The GSS_C_NT_HOSTBASED_SERVICE name form represents a service running on a host; it is textually represented as "HOST@SERVICE". This name form is required by most SASL profiles and is used by many existing applications that use the Kerberos GSS-API mechanism. While support for this name form is critical, it presents an interesting challenge in terms of channel binding. Consider a case where the server communicates with a "server proxy," or a AAA server near the server. That server proxy communicates with the EAP server. The EAP server and server proxy are in different administrative realms. The server proxy is in a position to verify that the request comes from the indicated host. However the EAP server cannot make this determination directly. So, the EAP server needs to determine whether to trust the server proxy to verify the host portion of the acceptor name. This trust decision depends both on the host name and the realm of the server proxy. In effect, the EAP server decides whether to trust that the realm of the server proxy is the right realm for the given hostname and then makes a trust decision about the server proxy itself. The same problem appears in Kerberos: there, clients decide what Kerberos realm to trust for a given hostname.

Sometimes, the client may know what AAA realm a particular host should belong to. In this case it would be desirable to use a name form that included a service, host and realm. Syntactically, this appears the same as the domain-based name discussed in [RFC5178] (Williams, N. and A. Melnikov, “Generic Security Service Application Program Interface (GSS-API) Internationalization and Domain-Based Service Names and Name Type,” May 2008.), but the semantics do not appear sufficiently similar to use the same name form.

A name form is needed to identify a SAML endpoint and a specific instance of SAML metadata associated with that endpoint. The metadata describes properties of the endpoint including public keys. One of the motivating use cases is to be able to use GSS-API to build trust in this metadata. In this case it is desirable to authenticate to an acceptor based on the endpoint and a cryptographic hash of the metadata.

The mechanism name form must be able to represent all of these names. In addition, the mechanism name form MUST make it easy for intermediate AAA proxies to extract the hostname portion when present. One possible starting point is the Kerberos name form discussed in [RFC1964] (Linn, J., “The Kerberos Version 5 GSS-API Mechanism,” June 1996.). The major down side of that approach is that there is no guaranteed way to be able to extract a hostname from a Kerberos name. Also, Kerberos naming may provide more flexibility than is needed.

3.2. Exported Mechanism Names

TOC

GSS-API provides the GSS_Export_name call. This call can be used to export the binary representation of a name. This name form can be stored on access control lists for binary comparison.
This section defines the format of the exported name token for this mechanism.

3.3. Acceptor Name RADIUS AVP

TOC

This section defines an attribute-value pair for transporting the name of the acceptor in a RADIUS or Diameter message. This AVP is included by the server to indicate the acceptor name it claims. This AVP is included in channel bindings by the client to indicate what acceptor is authenticated against.

3.4. Proxy Verification of Acceptor Name

TOC

---

4. Selection of EAP Method

TOC

The specification currently describes a single GSS-API mechanism. The peer and authenticator exchange EAP messages. The GSS-API mechanism specifies no constraints about what EAP method types are used; text in the specification says that negotiation of which EAP method to use happens at the EAP layer.
EAP does not provide a facility for an EAP server to advertise what methods are available to a peer. Instead, a server starts with its preferred method selection. If the peer does not accept that method,

the peer sends a NAK response containing the list of methods supported by the client.

Providing multiple facilities to negotiate which security mechanism to use is undesirable. Section 7.3 of [RFC4462] (Hutzelman, J., Salowey, J., Galbraith, J., and V. Welch, “Generic Security Service Application Program Interface (GSS-API) Authentication and Key Exchange for the Secure Shell (SSH) Protocol,” May 2006.)describes the problem referencing the SSH key exchange negotiation and the SPNEGO GSS-API mechanism. If a client preferred an EAP method A, a non-EAP authentication mechanism B, and then an EAP method C, then the client would have to commit to using EAP before learning whether A is actually supported. Such a client might end up using C when B is available.

The standard solution to this problem is to perform all the negotiation at one layer. In this case, rather than defining a single GSS-API mechanism, a family of mechanisms should be defined. Each mechanism corresponds to an EAP method. The EAP method type should be part of the GSS-API OID. Then, a GSS-API rather than EAP facility can be used for negotiation.

Unfortunately, using a family of mechanisms has a number of problems. First, GSS-API assumes that both the initiator and acceptor know the entire set of mechanisms that are available. Some negotiation mechanisms are driven by the client; others are driven by the server. With EAP GSS-API, the acceptor does not know what methods the EAP server implements. The EAP server that is used depends on the identity of the client. The best solution so far is to accept the disadvantages of multi-layer negotiation and commit to using EAP GSS-API before a specific EAP method. This has two main disadvantages. First, authentication may fail when other methods might allow authentication to succeed. Second, a non-optimal security mechanism may be chosen.

---

5. Context Tokens

TOC

All context establishment tokens emitted by the EAP mechanism SHALL have the framing described in section 3.1 of [RFC2743], as illustrated by the following pseudo-ASN.1 structures:

```
     GSS-API DEFINITIONS ::=
              BEGIN

              MechType ::= OBJECT IDENTIFIER
              -- representing EAP mechanism
              GSSAPI-Token ::=
              -- option indication (delegation, etc.) indicated within
              -- mechanism-specific token
              [APPLICATION 0] IMPLICIT SEQUENCE {
                      thisMech MechType,
                      innerToken ANY DEFINED BY thisMech
                         -- contents mechanism-specific
                         -- ASN.1 structure not required
                      }
              END
```

The innerToken field contains an EAP packet or special token. The first EAP packet SHALL be a EAP response/identity packet from the initiator to acceptor. The acceptor SHALL respond either with an EAP request or an EAP failure packet.
The initiator and acceptor will continue exchanging response/request packets until authentication succeeds or fails.
After the EAP authentication succeeds, channel binding tokens are exchanged; see Section 6.1 (GSS-API Channel Binding) for details. Currently, the channel binding tokens are the only types of special tokens in use.

5.1. Mechanisms and Encryption Types

TOC

This mechanism family uses the security services of the Kerberos cryptographic framework [RFC3961] (Raeburn, K., “Encryption and Checksum Specifications for Kerberos 5,” February 2005.). As such, a particular encryption type needs to be chosen. A new GSS-API OID should be defined for EAP GSS-API with a given Kerberos crypto system. This document defines the eap-aes128-cts-hmac-sha1-96 GSS-API mechanism. XXX define an OID for that and use the right language to get that into the appropriate SASL registry.

5.2. Context Options

TOC

GSS-API provides a number of optional per-context services requested by flags on the call to GSS_Init_sec_context and indicated as outputs from both GSS_Init_sec_context and GSS_Accept_sec_context. This section describes how these services are handled.

Integrity, confidentiality, sequencing and replay detection are always available. Regardless of what flags are requested in GSS_Init_sec_context, implementations MUST set the flag corresponding to these services in the output of GSS_Init_sec_context and GSS_Accept_sec_context.

The PROT_READY service is never available with this mechanism. Implementations MUST NOT offer this flag or permit per-message security services to be used before context establishment.

Open issue: how is the mutual authentication request and return handled? The big question here is figuring out how this interacts with EAP and transporting state back to a pass-through authenticator.

Open issue: handling of lifetime parameters.

---

6. Acceptor Services

TOC

The context establishment process may be passed through to a EAP server via a backend authentication protocol. However after the EAP authentication succeeds, security services are provided directly by the acceptor.

This mechanism uses an RFC 3961 cryptographic key called the context root key (CRK). The CRK is the result of the random-to-key operation consuming the appropriate number of bits from the EAP master session key. For example for aes128-cts-hmac-sha1-96, the random-to-key operation consumes 16 octets of key material; thus the first 16 bytes of the master session key are input to random-to-key to form the CRK.

6.1. GSS-API Channel Binding

TOC

GSS-API channel binding [RFC5554] (Williams, N., “Clarifications and Extensions to the Generic Security Service Application Program Interface (GSS-API) for the Use of Channel Bindings,” May 2009.) is a protected facility for exchanging a cryptographic name for an enclosing channel between the initiator and acceptor. The initiator sends channel binding data and the acceptor confirms that channel binding data has been checked.

The acceptor SHOULD accept any channel binding providing by the initiator if null channel bindings are passed into gss_accept_sec_context. Protocols such as HTTP Negotiate depend on this behavior of some Kerberos implementations. It is reasonable for the protocol to distinguish an acceptor ignoring channel bindings from an acceptor successfully validating them. No facility is currently

provided for an initiator implementation to expose this distinction to the initiator code.
Define a token format, token ID and key usage for this token.

6.2. Per-message security

TOC

The per-message tokens of section 4 of RFC 4121 are used. The CRK SHALL be treated as the initiator sub-session key, the acceptor sub-session key and the ticket session key.

6.3. Pseudo Random Function

TOC

The pseudo random function defined in [RFC4402] (Williams, N., “A Pseudo-Random Function (PRF) for the Kerberos V Generic Security Service Application Program Interface (GSS-API) Mechanism,” February 2006.) is used.

7. Authorization and Naming Extensions

TOC

One goal of this mechanism is to support retrieving a SAML assertion as a result of the EAP authentication. The GSS-API naming extensions will be used to access this message. This section will be expanded to discuss details.

8. Applicability Considerations

TOC

Section 1.3 of RFC 3748 provides the applicability statement for EAP. Among other constraints, EAP is scoped for use in network access. This specification anticipates using EAP beyond its current scope. The assumption is that some other document will discuss the issues surrounding the use of EAP for application authentication and expand EAP's applicability. That document will likely enumerate considerations that a specific use of EAP for application authentication needs to handle. Examples of such considerations might include the multi-layer negotiation issue, deciding when EAP or some other mechanism should be used, and so forth. This section serves as a placeholder to discuss any such issues with regard to the use of EAP and GSS-API.

9. Security Considerations

TOC

RFC 3748 discusses security issues surrounding EAP. RFC 5247 discusses the security and requirements surrounding key management that leverages the AAA infrastructure. These documents are critical to the security analysis of this mechanism.

RFC 2743 discusses generic security considerations for the GSS-API. RFC 4121 discusses security issues surrounding the specific per-message services used in this mechanism.

As discussed in Section 4 (Selection of EAP Method), this mechanism may introduce multiple layers of security negotiation into application protocols. Multiple layer negotiations are vulnerable to a bid-down attack when a mechanism negotiated at the outer layer is preferred to some but not all mechanisms negotiated at the inner layer; see section 7.3 of [RFC4462] (Hutzelman, J., Salowey, J., Galbraith, J., and V. Welch, “Generic Security Service Application Program Interface (GSS-API) Authentication and Key Exchange for the Secure Shell (SSH) Protocol,” May 2006.) for an example. One possible approach to mitigate this attack is to construct security policy such that the preference for all mechanisms negotiated in the inner layer falls between preferences for two outer layer mechanisms or falls at one end of the overall ranked preferences including both the inner and outer layer. Another approach is to only use this mechanism when it has specifically been selected for a given service. The second approach is likely to be common in practice because one common deployment will involved an EAP supplicant interacting with a user to select a given identity. Only when an identity is successfully chosen by the user will this mechanism be attempted.

The security of this mechanism depends on the use and verification of EAP channel binding. Today EAP channel binding is in very limited deployment. If EAP channel binding is not used, then the system may be vulnerable to phishing attacks where a user is diverted from one service to another. These attacks are possible with EAP today although not typically with common GSS-API mechanisms.

Every proxy in the AAA chain from the authenticator to the EAP server needs to be trusted to help verify channel bindings and to protect the integrity of key material. GSS-API applications may be built to assume a trust model where the acceptor is directly responsible for authentication. However, GSS-API is definitely used with trusted-third-party mechanisms such as Kerberos.

---

10. References

TOC

10.1. Normative References

TOC

[I-D.ietf-emu-chbind]	Clancy, C. and K. Hoeper, “Channel Binding Support for EAP Methods,” draft-ietf-emu-chbind-04 (work in progress), October 2009 (TXT).
[RFC2119]	Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).
[RFC2743]	Linn, J., “Generic Security Service Application Program Interface Version 2, Update 1,” RFC 2743, January 2000 (TXT).
[RFC3748]	Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, “Extensible Authentication Protocol (EAP),” RFC 3748, June 2004 (TXT).
[RFC3961]	Raeburn, K., “Encryption and Checksum Specifications for Kerberos 5,” RFC 3961, February 2005 (TXT).
[RFC4121]	Zhu, L., Jaganathan, K., and S. Hartman, “The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2,” RFC 4121, July 2005 (TXT).
[RFC4282]	Aboba, B., Beadles, M., Arkko, J., and P. Eronen, “The Network Access Identifier,” RFC 4282, December 2005 (TXT).
[RFC4401]	Williams, N., “A Pseudo-Random Function (PRF) API Extension for the Generic Security Service Application Program Interface (GSS-API),” RFC 4401, February 2006 (TXT).
[RFC4402]	Williams, N., “A Pseudo-Random Function (PRF) for the Kerberos V Generic Security Service Application Program Interface (GSS-API) Mechanism,” RFC 4402, February 2006 (TXT).
[RFC5056]	Williams, N., “On the Use of Channel Bindings to Secure Channels,” RFC 5056, November 2007 (TXT).
[RFC5554]	Williams, N., “Clarifications and Extensions to the Generic Security Service Application Program Interface (GSS-API) for the Use of Channel Bindings,” RFC 5554, May 2009 (TXT).

10.2. Informative References

TOC

[RFC1964]	Linn, J., “The Kerberos Version 5 GSS-API Mechanism,” RFC 1964, June 1996 (TXT).
[RFC3579]	Aboba, B. and P. Calhoun, “RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP),” RFC 3579, September 2003 (TXT).

[RFC4072]	Eronen, P., Hiller, T., and G. Zorn, “Diameter Extensible Authentication Protocol (EAP) Application,” RFC 4072, August 2005 (TXT).
[RFC4178]	Zhu, L., Leach, P., Jaganathan, K., and W. Ingersoll, “The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism,” RFC 4178, October 2005 (TXT).
[RFC4422]	Melnikov, A. and K. Zeilenga, “Simple Authentication and Security Layer (SASL),” RFC 4422, June 2006 (TXT).
[RFC4462]	Hutzelman, J., Salowey, J., Galbraith, J., and V. Welch, “Generic Security Service Application Program Interface (GSS-API) Authentication and Key Exchange for the Secure Shell (SSH) Protocol,” RFC 4462, May 2006 (TXT).
[RFC5178]	Williams, N. and A. Melnikov, “Generic Security Service Application Program Interface (GSS-API) Internationalization and Domain-Based Service Names and Name Type,” RFC 5178, May 2008 (TXT).
[RFC5247]	Aboba, B., Simon, D., and P. Eronen, “Extensible Authentication Protocol (EAP) Key Management Framework,” RFC 5247, August 2008 (TXT).

Authors' Addresses

TOC

	Sam Hartman (editor)
	Painless Security
Email:	hartmans-ietf@mit.edu
	Josh Howlett
	JANET(UK)
Email:	josh.howlett@ja.net