Network Working Group
Request for Comments: 5389
Obsoletes: 3489
Category: Standards Track
J. Rosenberg
Cisco
R. Mahy
P. Matthews
Unaffiliated
D. Wing
Cisco
October 2008

Session Traversal Utilities for NAT (STUN)

Status of This Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Abstract

Session Traversal Utilities for NAT (STUN) is a protocol that serves as a tool for other protocols in dealing with Network Address Translator (NAT) traversal. It can be used by an endpoint to determine the IP address and port allocated to it by a NAT. It can also be used to check connectivity between two endpoints, and as a keep-alive protocol to maintain NAT bindings. STUN works with many existing NATs, and does not require any special behavior from them.

STUN is not a NAT traversal solution by itself. Rather, it is a tool to be used in the context of a NAT traversal solution. This is an important change from the previous version of this specification (RFC 3489), which presented STUN as a complete solution.

This document obsoletes RFC 3489.

Table of Contents

1. Introduction

The protocol defined in this specification, Session Traversal Utilities for NAT, provides a tool for dealing with NATs. It provides a means for an endpoint to determine the IP address and port allocated by a NAT that corresponds to its private IP address and port. It also provides a way for an endpoint to keep a NAT binding alive. With some extensions, the protocol can be used to do connectivity checks between two endpoints [MMUSIC-ICE], or to relay packets between two endpoints [BEHAVE-TURN].

In keeping with its tool nature, this specification defines an extensible packet format, defines operation over several transport protocols, and provides for two forms of authentication.

STUN is intended to be used in context of one or more NAT traversal solutions. These solutions are known as STUN usages. Each usage describes how STUN is utilized to achieve the NAT traversal solution. Typically, a usage indicates when STUN messages get sent, which optional attributes to include, what server is used, and what authentication mechanism is to be used. Interactive Connectivity Establishment (ICE) [MMUSIC-ICE] is one usage of STUN. SIP Outbound [SIP-OUTBOUND] is another usage of STUN. In some cases, a usage will require extensions to STUN. A STUN extension can be in the form of new methods, attributes, or error response codes. More information on STUN usages can be found in Section 14.

2. Evolution from RFC 3489

STUN was originally defined in RFC 3489 [RFC3489]. That specification, sometimes referred to as "classic STUN", represented itself as a complete solution to the NAT traversal problem. In that solution, a client would discover whether it was behind a NAT, determine its NAT type, discover its IP address and port on the public side of the outermost NAT, and then utilize that IP address and port within the body of protocols, such as the Session Initiation Protocol (SIP) [RFC3261]. However, experience since the publication of RFC 3489 has found that classic STUN simply does not work sufficiently well to be a deployable solution. The address and port learned through classic STUN are sometimes usable for communications with a peer, and sometimes not. Classic STUN provided no way to discover whether it would, in fact, work or not, and it provided no remedy in cases where it did not. Furthermore, classic STUN's algorithm for classification of NAT types was found to be faulty, as many NATs did not fit cleanly into the types defined there.

Classic STUN also had a security vulnerability -- attackers could provide the client with incorrect mapped addresses under certain topologies and constraints, and this was fundamentally not solvable through any cryptographic means. Though this problem remains with this specification, those attacks are now mitigated through the use of more complete solutions that make use of STUN.

For these reasons, this specification obsoletes RFC 3489, and instead describes STUN as a tool that is utilized as part of a complete NAT traversal solution. ICE [MMUSIC-ICE] is a complete NAT traversal solution for protocols based on the offer/answer [RFC3264] methodology, such as SIP. SIP Outbound [SIP-OUTBOUND] is a complete solution for traversal of SIP signaling, and it uses STUN in a very different way. Though it is possible that a protocol may be able to use STUN by itself (classic STUN) as a traversal solution, such usage is not described here and is strongly discouraged for the reasons described above.

The on-the-wire protocol described here is changed only slightly from classic STUN. The protocol now runs over TCP in addition to UDP. Extensibility was added to the protocol in a more structured way. A magic cookie mechanism for demultiplexing STUN with application protocols was added by stealing 32 bits from the 128-bit transaction ID defined in RFC 3489, allowing the change to be backwards compatible. Mapped addresses are encoded using a new exclusive-or format. There are other, more minor changes. See Section 19 for a more complete listing.

Due to the change in scope, STUN has also been renamed from "Simple Traversal of UDP through NAT" to "Session Traversal Utilities for NAT". The acronym remains STUN, which is all anyone ever remembers anyway.

3. Overview of Operation

This section is descriptive only.

```
                               /-----\
                             // STUN  \\
                            |   Server  |
                             \\       //
                               \-----/



                          +--------------+             Public Internet
          ................|     NAT 2    |.......................
                          +--------------+



                          +--------------+             Private NET 2
          ................|     NAT 1    |.......................
                          +--------------+




                              /-----\
                            //  STUN \\
                           |    Client |
                            \\       //               Private NET 1
                              \-----/
```

Figure 1: One Possible STUN Configuration

One possible STUN configuration is shown in Figure 1. In this configuration, there are two entities (called STUN agents) that implement the STUN protocol. The lower agent in the figure is the client, and is connected to private network 1. This network connects to private network 2 through NAT 1. Private network 2 connects to the public Internet through NAT 2. The upper agent in the figure is the server, and resides on the public Internet.

STUN is a client-server protocol. It supports two types of transactions. One is a request/response transaction in which a client sends a request to a server, and the server returns a response. The second is an indication transaction in which either agent -- client or server -- sends an indication that generates no response. Both types of transactions include a transaction ID, which is a randomly selected 96-bit number. For request/response transactions, this transaction ID allows the client to associate the response with the request that generated it; for indications, the transaction ID serves as a debugging aid.

All STUN messages start with a fixed header that includes a method, a class, and the transaction ID. The method indicates which of the various requests or indications this is; this specification defines just one method, Binding, but other methods are expected to be defined in other documents. The class indicates whether this is a request, a success response, an error response, or an indication. Following the fixed header comes zero or more attributes, which are Type-Length-Value extensions that convey additional information for the specific message.

This document defines a single method called Binding. The Binding method can be used either in request/response transactions or in indication transactions. When used in request/response transactions, the Binding method can be used to determine the particular "binding" a NAT has allocated to a STUN client. When used in either request/ response or in indication transactions, the Binding method can also be used to keep these "bindings" alive.

In the Binding request/response transaction, a Binding request is sent from a STUN client to a STUN server. When the Binding request arrives at the STUN server, it may have passed through one or more NATs between the STUN client and the STUN server (in Figure 1, there were two such NATs). As the Binding request message passes through a NAT, the NAT will modify the source transport address (that is, the source IP address and the source port) of the packet. As a result, the source transport address of the request received by the server will be the public IP address and port created by the NAT closest to the server. This is called a reflexive transport address. The STUN server copies that source transport address into an XOR-MAPPED-ADDRESS attribute in the STUN Binding response and sends the Binding response back to the STUN client. As this packet passes back through a NAT, the NAT will modify the destination transport address in the IP header, but the transport address in the XOR-MAPPED-ADDRESS attribute within the body of the STUN response will remain untouched. In this way, the client can learn its reflexive transport address allocated by the outermost NAT with respect to the STUN server.

In some usages, STUN must be multiplexed with other protocols (e.g., [MMUSIC-ICE], [SIP-OUTBOUND]). In these usages, there must be a way to inspect a packet and determine if it is a STUN packet or not. STUN provides three fields in the STUN header with fixed values that can be used for this purpose. If this is not sufficient, then STUN packets can also contain a FINGERPRINT value, which can further be used to distinguish the packets.

STUN defines a set of optional procedures that a usage can decide to use, called mechanisms. These mechanisms include DNS discovery, a redirection technique to an alternate server, a fingerprint attribute for demultiplexing, and two authentication and message-integrity exchanges. The authentication mechanisms revolve around the use of a username, password, and message-integrity value. Two authentication mechanisms, the long-term credential mechanism and the short-term credential mechanism, are defined in this specification. Each usage specifies the mechanisms allowed with that usage.

In the long-term credential mechanism, the client and server share a pre-provisioned username and password and perform a digest challenge/response exchange inspired by (but differing in details) to the one defined for HTTP [RFC2617]. In the short-term credential mechanism, the client and the server exchange a username and password through some out-of-band method prior to the STUN exchange. For example, in the ICE usage [MMUSIC-ICE] the two endpoints use out-of-band signaling to exchange a username and password. These are used to integrity protect and authenticate the request and response. There is no challenge or nonce used.

4. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119] and indicate requirement levels for compliant STUN implementations.

5. Definitions

STUN Agent: A STUN agent is an entity that implements the STUN protocol. The entity can be either a STUN client or a STUN server.

STUN Client: A STUN client is an entity that sends STUN requests and receives STUN responses. A STUN client can also send indications. In this specification, the terms STUN client and client are synonymous.

STUN Server: A STUN server is an entity that receives STUN requests and sends STUN responses. A STUN server can also send indications. In this specification, the terms STUN server and server are synonymous.

Transport Address: The combination of an IP address and port number (such as a UDP or TCP port number).

Reflexive Transport Address: A transport address learned by a client that identifies that client as seen by another host on an IP network, typically a STUN server. When there is an intervening NAT between the client and the other host, the reflexive transport address represents the mapped address allocated to the client on the public side of the NAT. Reflexive transport addresses are learned from the mapped address attribute (MAPPED-ADDRESS or XOR-MAPPED-ADDRESS) in STUN responses.

Mapped Address: Same meaning as reflexive address. This term is retained only for historic reasons and due to the naming of the MAPPED-ADDRESS and XOR-MAPPED-ADDRESS attributes.

Long-Term Credential: A username and associated password that represent a shared secret between client and server. Long-term credentials are generally granted to the client when a subscriber enrolls in a service and persist until the subscriber leaves the service or explicitly changes the credential.

Long-Term Password: The password from a long-term credential.

Short-Term Credential: A temporary username and associated password that represent a shared secret between client and server. Short-term credentials are obtained through some kind of protocol mechanism between the client and server, preceding the STUN exchange. A short-term credential has an explicit temporal scope, which may be based on a specific amount of time (such as 5 minutes) or on an event (such as termination of a SIP dialog). The specific scope of a short-term credential is defined by the application usage.

Short-Term Password: The password component of a short-term credential.

STUN Indication: A STUN message that does not receive a response.

Attribute: The STUN term for a Type-Length-Value (TLV) object that can be added to a STUN message. Attributes are divided into two types: comprehension-required and comprehension-optional. STUN agents can safely ignore comprehension-optional attributes they don't understand, but cannot successfully process a message if it contains comprehension-required attributes that are not understood.

RTO: Retransmission TimeOut, which defines the initial period of time between transmission of a request and the first retransmit of that request.

6. STUN Message Structure

STUN messages are encoded in binary using network-oriented format (most significant byte or octet first, also commonly known as big-endian). The transmission order is described in detail in Appendix B of RFC 791 [RFC0791]. Unless otherwise noted, numeric constants are in decimal (base 10).

All STUN messages MUST start with a 20-byte header followed by zero or more Attributes. The STUN header contains a STUN message type, magic cookie, transaction ID, and message length.

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |0 0|     STUN Message Type     |         Message Length        |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                         Magic Cookie                          |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                                                               |
   |                     Transaction ID (96 bits)                  |
   |                                                               |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: Format of STUN Message Header

The most significant 2 bits of every STUN message MUST be zeroes. This can be used to differentiate STUN packets from other protocols when STUN is multiplexed with other protocols on the same port.

The message type defines the message class (request, success response, failure response, or indication) and the message method (the primary function) of the STUN message. Although there are four message classes, there are only two types of transactions in STUN: request/response transactions (which consist of a request message and a response message) and indication transactions (which consist of a single indication message). Response classes are split into error and success responses to aid in quickly processing the STUN message.

The message type field is decomposed further into the following structure:

```
                        0                 1
                        2  3  4 5 6 7 8 9 0 1 2 3 4 5

                       +--+--+-+-+-+-+-+-+-+-+-+-+-+-+
                       |M |M |M|M|M|C|M|M|M|C|M|M|M|M|
                       |11|10|9|8|7|1|6|5|4|0|3|2|1|0|
                       +--+--+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: Format of STUN Message Type Field

Here the bits in the message type field are shown as most significant (M11) through least significant (M0). M11 through M0 represent a 12-bit encoding of the method. C1 and C0 represent a 2-bit encoding of the class. A class of 0b00 is a request, a class of 0b01 is an indication, a class of 0b10 is a success response, and a class of 0b11 is an error response. This specification defines a single method, Binding. The method and class are orthogonal, so that for each method, a request, success response, error response, and indication are possible for that method. Extensions defining new methods MUST indicate which classes are permitted for that method.

For example, a Binding request has class=0b00 (request) and method=0b000000000001 (Binding) and is encoded into the first 16 bits as 0x0001. A Binding response has class=0b10 (success response) and method=0b000000000001, and is encoded into the first 16 bits as 0x0101.

> Note: This unfortunate encoding is due to assignment of values in [RFC3489] that did not consider encoding Indications, Success, and Errors using bit fields.

The magic cookie field MUST contain the fixed value 0x2112A442 in network byte order. In RFC 3489 [RFC3489], this field was part of the transaction ID; placing the magic cookie in this location allows a server to detect if the client will understand certain attributes that were added in this revised specification. In addition, it aids in distinguishing STUN packets from packets of other protocols when STUN is multiplexed with those other protocols on the same port.

The transaction ID is a 96-bit identifier, used to uniquely identify STUN transactions. For request/response transactions, the transaction ID is chosen by the STUN client for the request and echoed by the server in the response. For indications, it is chosen by the agent sending the indication. It primarily serves to correlate requests with responses, though it also plays a small role in helping to prevent certain types of attacks. The server also uses the transaction ID as a key to identify each transaction uniquely across all clients. As such, the transaction ID MUST be uniformly and randomly chosen from the interval 0 .. 2**96-1, and SHOULD be cryptographically random. Resends of the same request reuse the same transaction ID, but the client MUST choose a new transaction ID for new transactions unless the new request is bit-wise identical to the previous request and sent from the same transport address to the same IP address. Success and error responses MUST carry the same transaction ID as their corresponding request. When an agent is acting as a STUN server and STUN client on the same port, the transaction IDs in requests sent by the agent have no relationship to the transaction IDs in requests received by the agent.

The message length MUST contain the size, in bytes, of the message not including the 20-byte STUN header. Since all STUN attributes are padded to a multiple of 4 bytes, the last 2 bits of this field are always zero. This provides another way to distinguish STUN packets from packets of other protocols.

Following the STUN fixed portion of the header are zero or more attributes. Each attribute is TLV (Type-Length-Value) encoded. The details of the encoding, and of the attributes themselves are given in Section 15.

7. Base Protocol Procedures

This section defines the base procedures of the STUN protocol. It describes how messages are formed, how they are sent, and how they are processed when they are received. It also defines the detailed processing of the Binding method. Other sections in this document describe optional procedures that a usage may elect to use in certain situations. Other documents may define other extensions to STUN, by adding new methods, new attributes, or new error response codes.

7.1. Forming a Request or an Indication

When formulating a request or indication message, the agent MUST follow the rules in Section 6 when creating the header. In addition, the message class MUST be either "Request" or "Indication" (as appropriate), and the method must be either Binding or some method defined in another document.

The agent then adds any attributes specified by the method or the usage. For example, some usages may specify that the agent use an authentication method (Section 10) or the FINGERPRINT attribute (Section 8).

If the agent is sending a request, it SHOULD add a SOFTWARE attribute to the request. Agents MAY include a SOFTWARE attribute in indications, depending on the method. Extensions to STUN should discuss whether SOFTWARE is useful in new indications.

For the Binding method with no authentication, no attributes are required unless the usage specifies otherwise.

All STUN messages sent over UDP SHOULD be less than the path MTU, if known. If the path MTU is unknown, messages SHOULD be the smaller of 576 bytes and the first-hop MTU for IPv4 [RFC1122] and 1280 bytes for IPv6 [RFC2460]. This value corresponds to the overall size of the IP packet. Consequently, for IPv4, the actual STUN message would need to be less than 548 bytes (576 minus 20-byte IP header, minus 8-byte UDP header, assuming no IP options are used). STUN provides no ability to handle the case where the request is under the MTU but the response would be larger than the MTU. It is not envisioned that this limitation will be an issue for STUN. The MTU limitation is a SHOULD, and not a MUST, to account for cases where STUN itself is being used to probe for MTU characteristics [BEHAVE-NAT]. Outside of this or similar applications, the MTU constraint MUST be followed.

7.2. Sending the Request or Indication

The agent then sends the request or indication. This document specifies how to send STUN messages over UDP, TCP, or TLS-over-TCP; other transport protocols may be added in the future. The STUN usage must specify which transport protocol is used, and how the agent determines the IP address and port of the recipient. Section 9 describes a DNS-based method of determining the IP address and port of a server that a usage may elect to use. STUN may be used with anycast addresses, but only with UDP and in usages where authentication is not used.

At any time, a client MAY have multiple outstanding STUN requests with the same STUN server (that is, multiple transactions in progress, with different transaction IDs). Absent other limits to the rate of new transactions (such as those specified by ICE for connectivity checks or when STUN is run over TCP), a client SHOULD space new transactions to a server by RTO and SHOULD limit itself to ten outstanding transactions to the same server.

7.2.1. Sending over UDP

When running STUN over UDP, it is possible that the STUN message might be dropped by the network. Reliability of STUN request/ response transactions is accomplished through retransmissions of the request message by the client application itself. STUN indications are not retransmitted; thus, indication transactions over UDP are not reliable.

A client SHOULD retransmit a STUN request message starting with an interval of RTO ("Retransmission TimeOut"), doubling after each retransmission. The RTO is an estimate of the round-trip time (RTT), and is computed as described in RFC 2988 [RFC2988], with two exceptions. First, the initial value for RTO SHOULD be configurable (rather than the 3 s recommended in RFC 2988) and SHOULD be greater than 500 ms. The exception cases for this "SHOULD" are when other mechanisms are used to derive congestion thresholds (such as the ones defined in ICE for fixed rate streams), or when STUN is used in non-Internet environments with known network capacities. In fixed-line access links, a value of 500 ms is RECOMMENDED. Second, the value of RTO SHOULD NOT be rounded up to the nearest second. Rather, a 1 ms accuracy SHOULD be maintained. As with TCP, the usage of Karn's algorithm is RECOMMENDED [KARN87]. When applied to STUN, it means that RTT estimates SHOULD NOT be computed from STUN transactions that result in the retransmission of a request.

The value for RTO SHOULD be cached by a client after the completion of the transaction, and used as the starting value for RTO for the next transaction to the same server (based on equality of IP address). The value SHOULD be considered stale and discarded after 10 minutes.

Retransmissions continue until a response is received, or until a total of Rc requests have been sent. Rc SHOULD be configurable and SHOULD have a default of 7. If, after the last request, a duration equal to Rm times the RTO has passed without a response (providing ample time to get a response if only this final request actually succeeds), the client SHOULD consider the transaction to have failed. Rm SHOULD be configurable and SHOULD have a default of 16. A STUN transaction over UDP is also considered failed if there has been a hard ICMP error [RFC1122]. For example, assuming an RTO of 500 ms, requests would be sent at times 0 ms, 500 ms, 1500 ms, 3500 ms, 7500 ms, 15500 ms, and 31500 ms. If the client has not received a response after 39500 ms, the client will consider the transaction to have timed out.

7.2.2. Sending over TCP or TLS-over-TCP

For TCP and TLS-over-TCP, the client opens a TCP connection to the server.

In some usages of STUN, STUN is sent as the only protocol over the TCP connection. In this case, it can be sent without the aid of any additional framing or demultiplexing. In other usages, or with other extensions, it may be multiplexed with other data over a TCP connection. In that case, STUN MUST be run on top of some kind of framing protocol, specified by the usage or extension, which allows for the agent to extract complete STUN messages and complete application layer messages. The STUN service running on the well-known port or ports discovered through the DNS procedures in Section 9 is for STUN alone, and not for STUN multiplexed with other data. Consequently, no framing protocols are used in connections to those servers. When additional framing is utilized, the usage will specify how the client knows to apply it and what port to connect to. For example, in the case of ICE connectivity checks, this information is learned through out-of-band negotiation between client and server.

When STUN is run by itself over TLS-over-TCP, the TLS_RSA_WITH_AES_128_CBC_SHA ciphersuite MUST be implemented at a minimum. Implementations MAY also support any other ciphersuite. When it receives the TLS Certificate message, the client SHOULD verify the certificate and inspect the site identified by the certificate. If the certificate is invalid or revoked, or if it does not identify the appropriate party, the client MUST NOT send the STUN message or otherwise proceed with the STUN transaction. The client MUST verify the identity of the server. To do that, it follows the identification procedures defined in Section 3.1 of RFC 2818 [RFC2818]. Those procedures assume the client is dereferencing a URI. For purposes of usage with this specification, the client treats the domain name or IP address used in Section 8.1 as the host portion of the URI that has been dereferenced. Alternatively, a client MAY be configured with a set of domains or IP addresses that are trusted; if a certificate is received that identifies one of those domains or IP addresses, the client considers the identity of the server to be verified.

When STUN is run multiplexed with other protocols over a TLS-over-TCP connection, the mandatory ciphersuites and TLS handling procedures operate as defined by those protocols.

Reliability of STUN over TCP and TLS-over-TCP is handled by TCP itself, and there are no retransmissions at the STUN protocol level. However, for a request/response transaction, if the client has not received a response by Ti seconds after it sent the SYN to establish the connection, it considers the transaction to have timed out. Ti SHOULD be configurable and SHOULD have a default of 39.5s. This value has been chosen to equalize the TCP and UDP timeouts for the default initial RTO.

In addition, if the client is unable to establish the TCP connection, or the TCP connection is reset or fails before a response is received, any request/response transaction in progress is considered to have failed.

The client MAY send multiple transactions over a single TCP (or TLS-over-TCP) connection, and it MAY send another request before receiving a response to the previous. The client SHOULD keep the connection open until it:

- has no further STUN requests or indications to send over that connection, and
- has no plans to use any resources (such as a mapped address (MAPPED-ADDRESS or XOR-MAPPED-ADDRESS) or relayed address [BEHAVE-TURN]) that were learned though STUN requests sent over that connection, and
- if multiplexing other application protocols over that port, has finished using that other application, and
- if using that learned port with a remote peer, has established communications with that remote peer, as is required by some TCP NAT traversal techniques (e.g., [MMUSIC-ICE-TCP]).

At the server end, the server SHOULD keep the connection open, and let the client close it, unless the server has determined that the connection has timed out (for example, due to the client disconnecting from the network). Bindings learned by the client will remain valid in intervening NATs only while the connection remains open. Only the client knows how long it needs the binding. The server SHOULD NOT close a connection if a request was received over that connection for which a response was not sent. A server MUST NOT ever open a connection back towards the client in order to send a response. Servers SHOULD follow best practices regarding connection management in cases of overload.

7.3. Receiving a STUN Message

This section specifies the processing of a STUN message. The processing specified here is for STUN messages as defined in this specification; additional rules for backwards compatibility are defined in Section 12. Those additional procedures are optional, and usages can elect to utilize them. First, a set of processing operations is applied that is independent of the class. This is followed by class-specific processing, described in the subsections that follow.

When a STUN agent receives a STUN message, it first checks that the message obeys the rules of Section 6. It checks that the first two bits are 0, that the magic cookie field has the correct value, that the message length is sensible, and that the method value is a supported method. It checks that the message class is allowed for the particular method. If the message class is "Success Response" or "Error Response", the agent checks that the transaction ID matches a transaction that is still in progress. If the FINGERPRINT extension is being used, the agent checks that the FINGERPRINT attribute is present and contains the correct value. If any errors are detected, the message is silently discarded. In the case when STUN is being multiplexed with another protocol, an error may indicate that this is not really a STUN message; in this case, the agent should try to parse the message as a different protocol.

The STUN agent then does any checks that are required by a authentication mechanism that the usage has specified (see Section 10).

Once the authentication checks are done, the STUN agent checks for unknown attributes and known-but-unexpected attributes in the message. Unknown comprehension-optional attributes MUST be ignored by the agent. Known-but-unexpected attributes SHOULD be ignored by the agent. Unknown comprehension-required attributes cause processing that depends on the message class and is described below.

At this point, further processing depends on the message class of the request.

7.3.1. Processing a Request

If the request contains one or more unknown comprehension-required attributes, the server replies with an error response with an error code of 420 (Unknown Attribute), and includes an UNKNOWN-ATTRIBUTES attribute in the response that lists the unknown comprehension-required attributes.

The server then does any additional checking that the method or the specific usage requires. If all the checks succeed, the server formulates a success response as described below.

When run over UDP, a request received by the server could be the first request of a transaction, or a retransmission. The server MUST respond to retransmissions such that the following property is preserved: if the client receives the response to the retransmission and not the response that was sent to the original request, the overall state on the client and server is identical to the case where only the response to the original retransmission is received, or where both responses are received (in which case the client will use the first). The easiest way to meet this requirement is for the server to remember all transaction IDs received over UDP and their corresponding responses in the last 40 seconds. However, this requires the server to hold state, and will be inappropriate for any requests which are not authenticated. Another way is to reprocess the request and recompute the response. The latter technique MUST only be applied to requests that are idempotent (a request is considered idempotent when the same request can be safely repeated without impacting the overall state of the system) and result in the same success response for the same request. The Binding method is considered to be idempotent. Note that there are certain rare network events that could cause the reflexive transport address value to change, resulting in a different mapped address in different success responses. Extensions to STUN MUST discuss the implications of request retransmissions on servers that do not store transaction state.

7.3.1.1. Forming a Success or Error Response

When forming the response (success or error), the server follows the rules of Section 6. The method of the response is the same as that of the request, and the message class is either "Success Response" or "Error Response".

For an error response, the server MUST add an ERROR-CODE attribute containing the error code specified in the processing above. The reason phrase is not fixed, but SHOULD be something suitable for the error code. For certain errors, additional attributes are added to the message. These attributes are spelled out in the description where the error code is specified. For example, for an error code of 420 (Unknown Attribute), the server MUST include an UNKNOWN-ATTRIBUTES attribute. Certain authentication errors also cause attributes to be added (see Section 10). Extensions may define other errors and/or additional attributes to add in error cases.

If the server authenticated the request using an authentication mechanism, then the server SHOULD add the appropriate authentication attributes to the response (see Section 10).

The server also adds any attributes required by the specific method or usage. In addition, the server SHOULD add a SOFTWARE attribute to the message.

For the Binding method, no additional checking is required unless the usage specifies otherwise. When forming the success response, the server adds a XOR-MAPPED-ADDRESS attribute to the response, where the contents of the attribute are the source transport address of the request message. For UDP, this is the source IP address and source UDP port of the request message. For TCP and TLS-over-TCP, this is the source IP address and source TCP port of the TCP connection as seen by the server.

7.3.1.2. Sending the Success or Error Response

The response (success or error) is sent over the same transport as the request was received on. If the request was received over UDP, the destination IP address and port of the response are the source IP address and port of the received request message, and the source IP address and port of the response are equal to the destination IP address and port of the received request message. If the request was received over TCP or TLS-over-TCP, the response is sent back on the same TCP connection as the request was received on.

7.3.2. Processing an Indication

If the indication contains unknown comprehension-required attributes, the indication is discarded and processing ceases.

The agent then does any additional checking that the method or the specific usage requires. If all the checks succeed, the agent then processes the indication. No response is generated for an indication.

For the Binding method, no additional checking or processing is required, unless the usage specifies otherwise. The mere receipt of the message by the agent has refreshed the "bindings" in the intervening NATs.

Since indications are not re-transmitted over UDP (unlike requests), there is no need to handle re-transmissions of indications at the sending agent.

7.3.3. Processing a Success Response

If the success response contains unknown comprehension-required attributes, the response is discarded and the transaction is considered to have failed.

The client then does any additional checking that the method or the specific usage requires. If all the checks succeed, the client then processes the success response.

For the Binding method, the client checks that the XOR-MAPPED-ADDRESS attribute is present in the response. The client checks the address family specified. If it is an unsupported address family, the attribute SHOULD be ignored. If it is an unexpected but supported address family (for example, the Binding transaction was sent over IPv4, but the address family specified is IPv6), then the client MAY accept and use the value.

7.3.4. Processing an Error Response

If the error response contains unknown comprehension-required attributes, or if the error response does not contain an ERROR-CODE attribute, then the transaction is simply considered to have failed.

The client then does any processing specified by the authentication mechanism (see Section 10). This may result in a new transaction attempt.

The processing at this point depends on the error code, the method, and the usage; the following are the default rules:

- If the error code is 300 through 399, the client SHOULD consider the transaction as failed unless the ALTERNATE-SERVER extension is being used. See Section 11.

- If the error code is 400 through 499, the client declares the transaction failed; in the case of 420 (Unknown Attribute), the response should contain a UNKNOWN-ATTRIBUTES attribute that gives additional information.

- If the error code is 500 through 599, the client MAY resend the request; clients that do so MUST limit the number of times they do this.

Any other error code causes the client to consider the transaction failed.

8. FINGERPRINT Mechanism

This section describes an optional mechanism for STUN that aids in distinguishing STUN messages from packets of other protocols when the two are multiplexed on the same transport address. This mechanism is optional, and a STUN usage must describe if and when it is used. The FINGERPRINT mechanism is not backwards compatible with RFC 3489, and cannot be used in environments where such compatibility is required.

In some usages, STUN messages are multiplexed on the same transport address as other protocols, such as the Real Time Transport Protocol (RTP). In order to apply the processing described in Section 7, STUN messages must first be separated from the application packets.

Section 6 describes three fixed fields in the STUN header that can be used for this purpose. However, in some cases, these three fixed fields may not be sufficient.

When the FINGERPRINT extension is used, an agent includes the FINGERPRINT attribute in messages it sends to another agent. Section 15.5 describes the placement and value of this attribute. When the agent receives what it believes is a STUN message, then, in addition to other basic checks, the agent also checks that the message contains a FINGERPRINT attribute and that the attribute contains the correct value. Section 7.3 describes when in the overall processing of a STUN message the FINGERPRINT check is performed. This additional check helps the agent detect messages of other protocols that might otherwise seem to be STUN messages.

9. DNS Discovery of a Server

This section describes an optional procedure for STUN that allows a client to use DNS to determine the IP address and port of a server. A STUN usage must describe if and when this extension is used. To use this procedure, the client must know a server's domain name and a service name; the usage must also describe how the client obtains these. Hard-coding the domain name of the server into software is NOT RECOMMENDED in case the domain name is lost or needs to change for legal or other reasons.

When a client wishes to locate a STUN server in the public Internet that accepts Binding request/response transactions, the SRV service name is "stun". When it wishes to locate a STUN server that accepts Binding request/response transactions over a TLS session, the SRV service name is "stuns". STUN usages MAY define additional DNS SRV service names.

The domain name is resolved to a transport address using the SRV procedures specified in [RFC2782]. The DNS SRV service name is the service name provided as input to this procedure. The protocol in the SRV lookup is the transport protocol the client will run STUN over: "udp" for UDP and "tcp" for TCP. Note that only "tcp" is defined with "stuns" at this time.

The procedures of RFC 2782 are followed to determine the server to contact. RFC 2782 spells out the details of how a set of SRV records is sorted and then tried. However, RFC 2782 only states that the client should "try to connect to the (protocol, address, service)" without giving any details on what happens in the event of failure. When following these procedures, if the STUN transaction times out without receipt of a response, the client SHOULD retry the request to the next server in the ordered defined by RFC 2782. Such a retry is only possible for request/response transmissions, since indication transactions generate no response or timeout.

The default port for STUN requests is 3478, for both TCP and UDP.

Administrators of STUN servers SHOULD use this port in their SRV records for UDP and TCP. In all cases, the port in DNS MUST reflect the one on which the server is listening. The default port for STUN over TLS is 5349. Servers can run STUN over TLS on the same port as STUN over TCP if the server software supports determining whether the initial message is a TLS or STUN message.

If no SRV records were found, the client performs an A or AAAA record lookup of the domain name. The result will be a list of IP addresses, each of which can be contacted at the default port using UDP or TCP, independent of the STUN usage. For usages that require TLS, the client connects to one of the IP addresses using the default STUN over TLS port.

10. Authentication and Message-Integrity Mechanisms

This section defines two mechanisms for STUN that a client and server can use to provide authentication and message integrity; these two mechanisms are known as the short-term credential mechanism and the long-term credential mechanism. These two mechanisms are optional, and each usage must specify if and when these mechanisms are used. Consequently, both clients and servers will know which mechanism (if any) to follow based on knowledge of which usage applies. For example, a STUN server on the public Internet supporting ICE would have no authentication, whereas the STUN server functionality in an agent supporting connectivity checks would utilize short-term credentials. An overview of these two mechanisms is given in Section 3.

Each mechanism specifies the additional processing required to use that mechanism, extending the processing specified in Section 7. The additional processing occurs in three different places: when forming a message, when receiving a message immediately after the basic checks have been performed, and when doing the detailed processing of error responses.

10.1. Short-Term Credential Mechanism

The short-term credential mechanism assumes that, prior to the STUN transaction, the client and server have used some other protocol to exchange a credential in the form of a username and password. This credential is time-limited. The time limit is defined by the usage.

As an example, in the ICE usage [MMUSIC-ICE], the two endpoints use out-of-band signaling to agree on a username and password, and this username and password are applicable for the duration of the media session.

This credential is used to form a message-integrity check in each request and in many responses. There is no challenge and response as in the long-term mechanism; consequently, replay is prevented by virtue of the time-limited nature of the credential.

10.1.1. Forming a Request or Indication

For a request or indication message, the agent MUST include the USERNAME and MESSAGE-INTEGRITY attributes in the message. The HMAC for the MESSAGE-INTEGRITY attribute is computed as described in Section 15.4. Note that the password is never included in the request or indication.

10.1.2. Receiving a Request or Indication

After the agent has done the basic processing of a message, the agent performs the checks listed below in order specified:

- If the message does not contain both a MESSAGE-INTEGRITY and a USERNAME attribute:

  * If the message is a request, the server MUST reject the request with an error response. This response MUST use an error code of 400 (Bad Request).

  * If the message is an indication, the agent MUST silently discard the indication.

- If the USERNAME does not contain a username value currently valid within the server:

  * If the message is a request, the server MUST reject the request with an error response. This response MUST use an error code of 401 (Unauthorized).

  * If the message is an indication, the agent MUST silently discard the indication.

- Using the password associated with the username, compute the value for the message integrity as described in Section 15.4. If the resulting value does not match the contents of the MESSAGE-INTEGRITY attribute:

* If the message is a request, the server MUST reject the request with an error response. This response MUST use an error code of 401 (Unauthorized).

* If the message is an indication, the agent MUST silently discard the indication.

If these checks pass, the agent continues to process the request or indication. Any response generated by a server MUST include the MESSAGE-INTEGRITY attribute, computed using the password utilized to authenticate the request. The response MUST NOT contain the USERNAME attribute.

If any of the checks fail, a server MUST NOT include a MESSAGE-INTEGRITY or USERNAME attribute in the error response. This is because, in these failure cases, the server cannot determine the shared secret necessary to compute MESSAGE-INTEGRITY.

10.1.3. Receiving a Response

The client looks for the MESSAGE-INTEGRITY attribute in the response. If present, the client computes the message integrity over the response as defined in Section 15.4, using the same password it utilized for the request. If the resulting value matches the contents of the MESSAGE-INTEGRITY attribute, the response is considered authenticated. If the value does not match, or if MESSAGE-INTEGRITY was absent, the response MUST be discarded, as if it was never received. This means that retransmits, if applicable, will continue.

10.2. Long-Term Credential Mechanism

The long-term credential mechanism relies on a long-term credential, in the form of a username and password that are shared between client and server. The credential is considered long-term since it is assumed that it is provisioned for a user, and remains in effect until the user is no longer a subscriber of the system, or is changed. This is basically a traditional "log-in" username and password given to users.

Because these usernames and passwords are expected to be valid for extended periods of time, replay prevention is provided in the form of a digest challenge. In this mechanism, the client initially sends a request, without offering any credentials or any integrity checks. The server rejects this request, providing the user a realm (used to guide the user or agent in selection of a username and password) and a nonce. The nonce provides the replay protection. It is a cookie, selected by the server, and encoded in such a way as to indicate a duration of validity or client identity from which it is valid. The client retries the request, this time including its username and the realm, and echoing the nonce provided by the server. The client also includes a message-integrity, which provides an HMAC over the entire request, including the nonce. The server validates the nonce and checks the message integrity. If they match, the request is authenticated. If the nonce is no longer valid, it is considered "stale", and the server rejects the request, providing a new nonce.

In subsequent requests to the same server, the client reuses the nonce, username, realm, and password it used previously. In this way, subsequent requests are not rejected until the nonce becomes invalid by the server, in which case the rejection provides a new nonce to the client.

Note that the long-term credential mechanism cannot be used to protect indications, since indications cannot be challenged. Usages utilizing indications must either use a short-term credential or omit authentication and message integrity for them.

Since the long-term credential mechanism is susceptible to offline dictionary attacks, deployments SHOULD utilize passwords that are difficult to guess. In cases where the credentials are not entered by the user, but are rather placed on a client device during device provisioning, the password SHOULD have at least 128 bits of randomness. In cases where the credentials are entered by the user, they should follow best current practices around password structure.

10.2.1. Forming a Request

There are two cases when forming a request. In the first case, this is the first request from the client to the server (as identified by its IP address and port). In the second case, the client is submitting a subsequent request once a previous request/response transaction has completed successfully. Forming a request as a consequence of a 401 or 438 error response is covered in Section 10.2.3 and is not considered a "subsequent request" and thus does not utilize the rules described in Section 10.2.1.2.

10.2.1.1. First Request

If the client has not completed a successful request/response transaction with the server (as identified by hostname, if the DNS procedures of Section 9 are used, else IP address if not), it SHOULD omit the USERNAME, MESSAGE-INTEGRITY, REALM, and NONCE attributes. In other words, the very first request is sent as if there were no authentication or message integrity applied.

10.2.1.2. Subsequent Requests

Once a request/response transaction has completed successfully, the client will have been presented a realm and nonce by the server, and selected a username and password with which it authenticated. The client SHOULD cache the username, password, realm, and nonce for subsequent communications with the server. When the client sends a subsequent request, it SHOULD include the USERNAME, REALM, and NONCE attributes with these cached values. It SHOULD include a MESSAGE-INTEGRITY attribute, computed as described in Section 15.4 using the cached password.

10.2.2. Receiving a Request

After the server has done the basic processing of a request, it performs the checks listed below in the order specified:

- If the message does not contain a MESSAGE-INTEGRITY attribute, the server MUST generate an error response with an error code of 401 (Unauthorized). This response MUST include a REALM value. It is RECOMMENDED that the REALM value be the domain name of the provider of the STUN server. The response MUST include a NONCE, selected by the server. The response SHOULD NOT contain a USERNAME or MESSAGE-INTEGRITY attribute.

- If the message contains a MESSAGE-INTEGRITY attribute, but is missing the USERNAME, REALM, or NONCE attribute, the server MUST generate an error response with an error code of 400 (Bad Request). This response SHOULD NOT include a USERNAME, NONCE, REALM, or MESSAGE-INTEGRITY attribute.

- If the NONCE is no longer valid, the server MUST generate an error response with an error code of 438 (Stale Nonce). This response MUST include NONCE and REALM attributes and SHOULD NOT include the USERNAME or MESSAGE-INTEGRITY attribute. Servers can invalidate nonces in order to provide additional security. See Section 4.3 of [RFC2617] for guidelines.

- If the username in the USERNAME attribute is not valid, the server MUST generate an error response with an error code of 401 (Unauthorized). This response MUST include a REALM value. It is RECOMMENDED that the REALM value be the domain name of the provider of the STUN server. The response MUST include a NONCE, selected by the server. The response SHOULD NOT contain a USERNAME or MESSAGE-INTEGRITY attribute.

- Using the password associated with the username in the USERNAME attribute, compute the value for the message integrity as described in Section 15.4. If the resulting value does not match the contents of the MESSAGE-INTEGRITY attribute, the server MUST reject the request with an error response. This response MUST use an error code of 401 (Unauthorized). It MUST include REALM and NONCE attributes and SHOULD NOT include the USERNAME or MESSAGE-INTEGRITY attribute.

If these checks pass, the server continues to process the request. Any response generated by the server (excepting the cases described above) MUST include the MESSAGE-INTEGRITY attribute, computed using the username and password utilized to authenticate the request. The REALM, NONCE, and USERNAME attributes SHOULD NOT be included.

10.2.3. Receiving a Response

If the response is an error response with an error code of 401 (Unauthorized), the client SHOULD retry the request with a new transaction. This request MUST contain a USERNAME, determined by the client as the appropriate username for the REALM from the error response. The request MUST contain the REALM, copied from the error response. The request MUST contain the NONCE, copied from the error response. The request MUST contain the MESSAGE-INTEGRITY attribute, computed using the password associated with the username in the USERNAME attribute. The client MUST NOT perform this retry if it is not changing the USERNAME or REALM or its associated password, from the previous attempt.

If the response is an error response with an error code of 438 (Stale Nonce), the client MUST retry the request, using the new NONCE supplied in the 438 (Stale Nonce) response. This retry MUST also include the USERNAME, REALM, and MESSAGE-INTEGRITY.

The client looks for the MESSAGE-INTEGRITY attribute in the response (either success or failure). If present, the client computes the message integrity over the response as defined in Section 15.4, using the same password it utilized for the request. If the resulting value matches the contents of the MESSAGE-INTEGRITY attribute, the response is considered authenticated. If the value does not match, or if MESSAGE-INTEGRITY was absent, the response MUST be discarded, as if it was never received. This means that retransmits, if applicable, will continue.

11. ALTERNATE-SERVER Mechanism

This section describes a mechanism in STUN that allows a server to redirect a client to another server. This extension is optional, and a usage must define if and when this extension is used.

A server using this extension redirects a client to another server by replying to a request message with an error response message with an error code of 300 (Try Alternate). The server MUST include an ALTERNATE-SERVER attribute in the error response. The error response message MAY be authenticated; however, there are uses cases for ALTERNATE-SERVER where authentication of the response is not possible or practical.

A client using this extension handles a 300 (Try Alternate) error code as follows. The client looks for an ALTERNATE-SERVER attribute in the error response. If one is found, then the client considers the current transaction as failed, and reattempts the request with the server specified in the attribute, using the same transport protocol used for the previous request. That request, if authenticated, MUST utilize the same credentials that the client would have used in the request to the server that performed the redirection. If the client has been redirected to a server on which it has already tried this request within the last five minutes, it MUST ignore the redirection and consider the transaction to have failed. This prevents infinite ping-ponging between servers in case of redirection loops.

12. Backwards Compatibility with RFC 3489

This section defines procedures that allow a degree of backwards compatibility with the original protocol defined in RFC 3489 [RFC3489]. This mechanism is optional, meant to be utilized only in cases where a new client can connect to an old server, or vice versa. A usage must define if and when this procedure is used.

Section 19 lists all the changes between this specification and RFC 3489 [RFC3489]. However, not all of these differences are important, because "classic STUN" was only used in a few specific ways. For the purposes of this extension, the important changes are the following. In RFC 3489:

- o UDP was the only supported transport.

- o The field that is now the magic cookie field was a part of the transaction ID field, and transaction IDs were 128 bits long.

- o The XOR-MAPPED-ADDRESS attribute did not exist, and the Binding method used the MAPPED-ADDRESS attribute instead.

- o There were three comprehension-required attributes, RESPONSE-ADDRESS, CHANGE-REQUEST, and CHANGED-ADDRESS, that have been removed from this specification.

  * CHANGE-REQUEST and CHANGED-ADDRESS are now part of the NAT Behavior Discovery usage [BEHAVE-NAT], and the other is deprecated.

12.1. Changes to Client Processing

A client that wants to interoperate with an [RFC3489] server SHOULD send a request message that uses the Binding method, contains no attributes, and uses UDP as the transport protocol to the server. If successful, the success response received from the server will contain a MAPPED-ADDRESS attribute rather than an XOR-MAPPED-ADDRESS attribute. A client seeking to interoperate with an older server MUST be prepared to receive either. Furthermore, the client MUST ignore any Reserved comprehension-required attributes that might appear in the response. Of the Reserved attributes in Section 18.2, 0x0002, 0x0004, 0x0005, and 0x000B may appear in Binding responses from a server compliant to RFC 3489. Other than this change, the processing of the response is identical to the procedures described above.

12.2. Changes to Server Processing

A STUN server can detect when a given Binding request message was sent from an RFC 3489 [RFC3489] client by the absence of the correct value in the magic cookie field. When the server detects an RFC 3489 client, it SHOULD copy the value seen in the magic cookie field in the Binding request to the magic cookie field in the Binding response message, and insert a MAPPED-ADDRESS attribute instead of an XOR-MAPPED-ADDRESS attribute.

The client might, in rare situations, include either the RESPONSE-ADDRESS or CHANGE-REQUEST attributes. In these situations, the server will view these as unknown comprehension-required attributes and reply with an error response. Since the mechanisms utilizing those attributes are no longer supported, this behavior is acceptable.

The RFC 3489 version of STUN lacks both the magic cookie and the FINGERPRINT attribute that allows for a very high probability of correctly identifying STUN messages when multiplexed with other protocols. Therefore, STUN implementations that are backwards compatible with RFC 3489 SHOULD NOT be used in cases where STUN will be multiplexed with another protocol. However, that should not be an issue as such multiplexing was not available in RFC 3489.

13. Basic Server Behavior

This section defines the behavior of a basic, stand-alone STUN server. A basic STUN server provides clients with server reflexive transport addresses by receiving and replying to STUN Binding requests.

The STUN server MUST support the Binding method. It SHOULD NOT utilize the short-term or long-term credential mechanism. This is because the work involved in authenticating the request is more than the work in simply processing it. It SHOULD NOT utilize the ALTERNATE-SERVER mechanism for the same reason. It MUST support UDP and TCP. It MAY support STUN over TCP/TLS; however, TLS provides minimal security benefits in this basic mode of operation. It MAY utilize the FINGERPRINT mechanism but MUST NOT require it. Since the stand-alone server only runs STUN, FINGERPRINT provides no benefit. Requiring it would break compatibility with RFC 3489, and such compatibility is desirable in a stand-alone server. Stand-alone STUN servers SHOULD support backwards compatibility with [RFC3489] clients, as described in Section 12.

It is RECOMMENDED that administrators of STUN servers provide DNS entries for those servers as described in Section 9.

A basic STUN server is not a solution for NAT traversal by itself. However, it can be utilized as part of a solution through STUN usages. This is discussed further in Section 14.

14. STUN Usages

STUN by itself is not a solution to the NAT traversal problem. Rather, STUN defines a tool that can be used inside a larger solution. The term "STUN usage" is used for any solution that uses STUN as a component.

At the time of writing, three STUN usages are defined: Interactive Connectivity Establishment (ICE) [MMUSIC-ICE], Client-initiated connections for SIP [SIP-OUTBOUND], and NAT Behavior Discovery [BEHAVE-NAT]. Other STUN usages may be defined in the future.

A STUN usage defines how STUN is actually utilized -- when to send requests, what to do with the responses, and which optional procedures defined here (or in an extension to STUN) are to be used. A usage would also define:

- o Which STUN methods are used.

- o What authentication and message-integrity mechanisms are used.

- o The considerations around manual vs. automatic key derivation for the integrity mechanism, as discussed in [RFC4107].

- o What mechanisms are used to distinguish STUN messages from other messages. When STUN is run over TCP, a framing mechanism may be required.

- o How a STUN client determines the IP address and port of the STUN server.

- o Whether backwards compatibility to RFC 3489 is required.

- o What optional attributes defined here (such as FINGERPRINT and ALTERNATE-SERVER) or in other extensions are required.

In addition, any STUN usage must consider the security implications of using STUN in that usage. A number of attacks against STUN are known (see the Security Considerations section in this document), and any usage must consider how these attacks can be thwarted or mitigated.

Finally, a usage must consider whether its usage of STUN is an example of the Unilateral Self-Address Fixing approach to NAT traversal, and if so, address the questions raised in RFC 3424 [RFC3424].

15. STUN Attributes

After the STUN header are zero or more attributes. Each attribute MUST be TLV encoded, with a 16-bit type, 16-bit length, and value. Each STUN attribute MUST end on a 32-bit boundary. As mentioned above, all fields in an attribute are transmitted most significant bit first.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Type                  |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Value (variable)                ....
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: Format of STUN Attributes

The value in the length field MUST contain the length of the Value part of the attribute, prior to padding, measured in bytes. Since STUN aligns attributes on 32-bit boundaries, attributes whose content is not a multiple of 4 bytes are padded with 1, 2, or 3 bytes of padding so that its value contains a multiple of 4 bytes. The padding bits are ignored, and may be any value.

Any attribute type MAY appear more than once in a STUN message. Unless specified otherwise, the order of appearance is significant: only the first occurrence needs to be processed by a receiver, and any duplicates MAY be ignored by a receiver.

To allow future revisions of this specification to add new attributes if needed, the attribute space is divided into two ranges. Attributes with type values between 0x0000 and 0x7FFF are comprehension-required attributes, which means that the STUN agent cannot successfully process the message unless it understands the attribute. Attributes with type values between 0x8000 and 0xFFFF are comprehension-optional attributes, which means that those attributes can be ignored by the STUN agent if it does not understand them.

The set of STUN attribute types is maintained by IANA. The initial set defined by this specification is found in Section 18.2.

The rest of this section describes the format of the various attributes defined in this specification.

15.1. MAPPED-ADDRESS

The MAPPED-ADDRESS attribute indicates a reflexive transport address of the client. It consists of an 8-bit address family and a 16-bit port, followed by a fixed-length value representing the IP address. If the address family is IPv4, the address MUST be 32 bits. If the address family is IPv6, the address MUST be 128 bits. All fields must be in network byte order.

The format of the MAPPED-ADDRESS attribute is:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|0 0 0 0 0 0 0 0|    Family     |           Port                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                 Address (32 bits or 128 bits)                 |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5: Format of MAPPED-ADDRESS Attribute

The address family can take on the following values:

0x01:IPv4
0x02:IPv6

The first 8 bits of the MAPPED-ADDRESS MUST be set to 0 and MUST be ignored by receivers. These bits are present for aligning parameters on natural 32-bit boundaries.

This attribute is used only by servers for achieving backwards compatibility with RFC 3489 [RFC3489] clients.

15.2. XOR-MAPPED-ADDRESS

The XOR-MAPPED-ADDRESS attribute is identical to the MAPPED-ADDRESS attribute, except that the reflexive transport address is obfuscated through the XOR function.

The format of the XOR-MAPPED-ADDRESS is:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|x x x x x x x x|    Family     |         X-Port                |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                X-Address (Variable)
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6: Format of XOR-MAPPED-ADDRESS Attribute

The Family represents the IP address family, and is encoded identically to the Family in MAPPED-ADDRESS.

X-Port is computed by taking the mapped port in host byte order, XOR'ing it with the most significant 16 bits of the magic cookie, and then the converting the result to network byte order. If the IP address family is IPv4, X-Address is computed by taking the mapped IP address in host byte order, XOR'ing it with the magic cookie, and converting the result to network byte order. If the IP address family is IPv6, X-Address is computed by taking the mapped IP address in host byte order, XOR'ing it with the concatenation of the magic cookie and the 96-bit transaction ID, and converting the result to network byte order.

The rules for encoding and processing the first 8 bits of the attribute's value, the rules for handling multiple occurrences of the attribute, and the rules for processing address families are the same as for MAPPED-ADDRESS.

Note: XOR-MAPPED-ADDRESS and MAPPED-ADDRESS differ only in their encoding of the transport address. The former encodes the transport address by exclusive-or'ing it with the magic cookie. The latter encodes it directly in binary. RFC 3489 originally specified only MAPPED-ADDRESS. However, deployment experience found that some NATs rewrite the 32-bit binary payloads containing the NAT's public IP address, such as STUN's MAPPED-ADDRESS attribute, in the well-meaning but misguided attempt at providing a generic ALG function. Such behavior interferes with the operation of STUN and also causes failure of STUN's message-integrity checking.

15.3. USERNAME

The USERNAME attribute is used for message integrity. It identifies the username and password combination used in the message-integrity check.

The value of USERNAME is a variable-length value. It MUST contain a UTF-8 [RFC3629] encoded sequence of less than 513 bytes, and MUST have been processed using SASLprep [RFC4013].

15.4. MESSAGE-INTEGRITY

The MESSAGE-INTEGRITY attribute contains an HMAC-SHA1 [RFC2104] of the STUN message. The MESSAGE-INTEGRITY attribute can be present in any STUN message type. Since it uses the SHA1 hash, the HMAC will be 20 bytes. The text used as input to HMAC is the STUN message, including the header, up to and including the attribute preceding the MESSAGE-INTEGRITY attribute. With the exception of the FINGERPRINT attribute, which appears after MESSAGE-INTEGRITY, agents MUST ignore all other attributes that follow MESSAGE-INTEGRITY.

The key for the HMAC depends on whether long-term or short-term credentials are in use. For long-term credentials, the key is 16 bytes:

```
key = MD5(username ":" realm ":" SASLprep(password))
```

That is, the 16-byte key is formed by taking the MD5 hash of the result of concatenating the following five fields: (1) the username, with any quotes and trailing nulls removed, as taken from the USERNAME attribute (in which case SASLprep has already been applied); (2) a single colon; (3) the realm, with any quotes and trailing nulls removed; (4) a single colon; and (5) the password, with any trailing nulls removed and after processing using SASLprep. For example, if the username was 'user', the realm was 'realm', and the password was 'pass', then the 16-byte HMAC key would be the result of performing an MD5 hash on the string 'user:realm:pass', the resulting hash being 0x8493fbc53ba582fb4c044c456bdc40eb.

For short-term credentials:

```
key = SASLprep(password)
```

where MD5 is defined in RFC 1321 [RFC1321] and SASLprep() is defined in RFC 4013 [RFC4013].

The structure of the key when used with long-term credentials facilitates deployment in systems that also utilize SIP. Typically, SIP systems utilizing SIP's digest authentication mechanism do not actually store the password in the database. Rather, they store a value called H(A1), which is equal to the key defined above.

Based on the rules above, the hash used to construct MESSAGE-INTEGRITY includes the length field from the STUN message header. Prior to performing the hash, the MESSAGE-INTEGRITY attribute MUST be inserted into the message (with dummy content). The length MUST then be set to point to the length of the message up to, and including, the MESSAGE-INTEGRITY attribute itself, but excluding any attributes after it. Once the computation is performed, the value of the MESSAGE-INTEGRITY attribute can be filled in, and the value of the length in the STUN header can be set to its correct value -- the length of the entire message. Similarly, when validating the MESSAGE-INTEGRITY, the length field should be adjusted to point to the end of the MESSAGE-INTEGRITY attribute prior to calculating the HMAC. Such adjustment is necessary when attributes, such as FINGERPRINT, appear after MESSAGE-INTEGRITY.

15.5. FINGERPRINT

The FINGERPRINT attribute MAY be present in all STUN messages. The value of the attribute is computed as the CRC-32 of the STUN message up to (but excluding) the FINGERPRINT attribute itself, XOR'ed with the 32-bit value 0x5354554e (the XOR helps in cases where an application packet is also using CRC-32 in it). The 32-bit CRC is the one defined in ITU V.42 [ITU.V42.2002], which has a generator polynomial of x32+x26+x23+x22+x16+x12+x11+x10+x8+x7+x5+x4+x2+x+1. When present, the FINGERPRINT attribute MUST be the last attribute in the message, and thus will appear after MESSAGE-INTEGRITY.

The FINGERPRINT attribute can aid in distinguishing STUN packets from packets of other protocols. See Section 8.

As with MESSAGE-INTEGRITY, the CRC used in the FINGERPRINT attribute covers the length field from the STUN message header. Therefore, this value must be correct and include the CRC attribute as part of the message length, prior to computation of the CRC. When using the FINGERPRINT attribute in a message, the attribute is first placed into the message with a dummy value, then the CRC is computed, and then the value of the attribute is updated. If the MESSAGE-INTEGRITY attribute is also present, then it must be present with the correct message-integrity value before the CRC is computed, since the CRC is done over the value of the MESSAGE-INTEGRITY attribute as well.

15.6. ERROR-CODE

The ERROR-CODE attribute is used in error response messages. It contains a numeric error code value in the range of 300 to 699 plus a textual reason phrase encoded in UTF-8 [RFC3629], and is consistent in its code assignments and semantics with SIP [RFC3261] and HTTP [RFC2616]. The reason phrase is meant for user consumption, and can be anything appropriate for the error code. Recommended reason phrases for the defined error codes are included in the IANA registry for error codes. The reason phrase MUST be a UTF-8 [RFC3629] encoded sequence of less than 128 characters (which can be as long as 763 bytes).

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Reserved, should be 0         |Class|     Number    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Reason Phrase (variable)                                ..
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7: ERROR-CODE Attribute

To facilitate processing, the class of the error code (the hundreds digit) is encoded separately from the rest of the code, as shown in Figure 7.

The Reserved bits SHOULD be 0, and are for alignment on 32-bit boundaries. Receivers MUST ignore these bits. The Class represents the hundreds digit of the error code. The value MUST be between 3 and 6. The Number represents the error code modulo 100, and its value MUST be between 0 and 99.

The following error codes, along with their recommended reason phrases, are defined:

300 Try Alternate: The client should contact an alternate server for this request. This error response MUST only be sent if the request included a USERNAME attribute and a valid MESSAGE-INTEGRITY attribute; otherwise, it MUST NOT be sent and error code 400 (Bad Request) is suggested. This error response MUST be protected with the MESSAGE-INTEGRITY attribute, and receivers MUST validate the MESSAGE-INTEGRITY of this response before redirecting themselves to an alternate server.

> Note: Failure to generate and validate message integrity for a 300 response allows an on-path attacker to falsify a 300 response thus causing subsequent STUN messages to be sent to a victim.

400 Bad Request: The request was malformed. The client SHOULD NOT retry the request without modification from the previous attempt. The server may not be able to generate a valid MESSAGE-INTEGRITY for this error, so the client MUST NOT expect a valid MESSAGE-INTEGRITY attribute on this response.

401 Unauthorized: The request did not contain the correct credentials to proceed. The client should retry the request with proper credentials.

420 Unknown Attribute: The server received a STUN packet containing a comprehension-required attribute that it did not understand. The server MUST put this unknown attribute in the UNKNOWN-ATTRIBUTE attribute of its error response.

438 Stale Nonce: The NONCE used by the client was no longer valid. The client should retry, using the NONCE provided in the response.

500 Server Error: The server has suffered a temporary error. The client should try again.

15.7. REALM

The REALM attribute may be present in requests and responses. It contains text that meets the grammar for "realm-value" as described in RFC 3261 [RFC3261] but without the double quotes and their surrounding whitespace. That is, it is an unquoted realm-value (and is therefore a sequence of qdtext or quoted-pair). It MUST be a UTF-8 [RFC3629] encoded sequence of less than 128 characters (which can be as long as 763 bytes), and MUST have been processed using SASLprep [RFC4013].

Presence of the REALM attribute in a request indicates that long-term credentials are being used for authentication. Presence in certain error responses indicates that the server wishes the client to use a long-term credential for authentication.

15.8. NONCE

The NONCE attribute may be present in requests and responses. It contains a sequence of qdtext or quoted-pair, which are defined in RFC 3261 [RFC3261]. Note that this means that the NONCE attribute will not contain actual quote characters. See RFC 2617 [RFC2617], Section 4.3, for guidance on selection of nonce values in a server.

It MUST be less than 128 characters (which can be as long as 763 bytes).

15.9. UNKNOWN-ATTRIBUTES

The UNKNOWN-ATTRIBUTES attribute is present only in an error response when the response code in the ERROR-CODE attribute is 420.

The attribute contains a list of 16-bit values, each of which represents an attribute type that was not understood by the server.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Attribute 1 Type         |      Attribute 2 Type         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Attribute 3 Type         |      Attribute 4 Type    ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 8: Format of UNKNOWN-ATTRIBUTES Attribute

Note: In [RFC3489], this field was padded to 32 by duplicating the last attribute. In this version of the specification, the normal padding rules for attributes are used instead.

15.10. SOFTWARE

The SOFTWARE attribute contains a textual description of the software being used by the agent sending the message. It is used by clients and servers. Its value SHOULD include manufacturer and version number. The attribute has no impact on operation of the protocol, and serves only as a tool for diagnostic and debugging purposes. The value of SOFTWARE is variable length. It MUST be a UTF-8 [RFC3629] encoded sequence of less than 128 characters (which can be as long as 763 bytes).

15.11. ALTERNATE-SERVER

The alternate server represents an alternate transport address identifying a different STUN server that the STUN client should try.

It is encoded in the same way as MAPPED-ADDRESS, and thus refers to a single server by IP address. The IP address family MUST be identical to that of the source IP address of the request.

16. Security Considerations

16.1. Attacks against the Protocol

16.1.1. Outside Attacks

An attacker can try to modify STUN messages in transit, in order to cause a failure in STUN operation. These attacks are detected for both requests and responses through the message-integrity mechanism, using either a short-term or long-term credential. Of course, once detected, the manipulated packets will be dropped, causing the STUN transaction to effectively fail. This attack is possible only by an on-path attacker.

An attacker that can observe, but not modify, STUN messages in-transit (for example, an attacker present on a shared access medium, such as Wi-Fi), can see a STUN request, and then immediately send a STUN response, typically an error response, in order to disrupt STUN processing. This attack is also prevented for messages that utilize MESSAGE-INTEGRITY. However, some error responses, those related to authentication in particular, cannot be protected by MESSAGE-INTEGRITY. When STUN itself is run over a secure transport protocol (e.g., TLS), these attacks are completely mitigated.

Depending on the STUN usage, these attacks may be of minimal consequence and thus do not require message integrity to mitigate. For example, when STUN is used to a basic STUN server to discover a server reflexive candidate for usage with ICE, authentication and message integrity are not required since these attacks are detected during the connectivity check phase. The connectivity checks themselves, however, require protection for proper operation of ICE overall. As described in Section 14, STUN usages describe when authentication and message integrity are needed.

Since STUN uses the HMAC of a shared secret for authentication and integrity protection, it is subject to offline dictionary attacks. When authentication is utilized, it SHOULD be with a strong password that is not readily subject to offline dictionary attacks. Protection of the channel itself, using TLS, mitigates these attacks. However, STUN is most often run over UDP and in those cases, strong passwords are the only way to protect against these attacks.

16.1.2. Inside Attacks

A rogue client may try to launch a DoS attack against a server by sending it a large number of STUN requests. Fortunately, STUN requests can be processed statelessly by a server, making such attacks hard to launch.

A rogue client may use a STUN server as a reflector, sending it requests with a falsified source IP address and port. In such a case, the response would be delivered to that source IP and port. There is no amplification of the number of packets with this attack (the STUN server sends one packet for each packet sent by the client), though there is a small increase in the amount of data, since STUN responses are typically larger than requests. This attack is mitigated by ingress source address filtering.

Revealing the specific software version of the agent through the SOFTWARE attribute might allow them to become more vulnerable to attacks against software that is known to contain security holes. Implementers SHOULD make usage of the SOFTWARE attribute a configurable option.

16.2. Attacks Affecting the Usage

This section lists attacks that might be launched against a usage of STUN. Each STUN usage must consider whether these attacks are applicable to it, and if so, discuss counter-measures.

Most of the attacks in this section revolve around an attacker modifying the reflexive address learned by a STUN client through a

Binding request/response transaction. Since the usage of the reflexive address is a function of the usage, the applicability and remediation of these attacks are usage-specific. In common situations, modification of the reflexive address by an on-path attacker is easy to do. Consider, for example, the common situation where STUN is run directly over UDP. In this case, an on-path attacker can modify the source IP address of the Binding request before it arrives at the STUN server. The STUN server will then return this IP address in the XOR-MAPPED-ADDRESS attribute to the client, and send the response back to that (falsified) IP address and port. If the attacker can also intercept this response, it can direct it back towards the client. Protecting against this attack by using a message-integrity check is impossible, since a message-integrity value cannot cover the source IP address, since the intervening NAT must be able to modify this value. Instead, one solution to preventing the attacks listed below is for the client to verify the reflexive address learned, as is done in ICE [MMUSIC-ICE]. Other usages may use other means to prevent these attacks.

16.2.1. Attack I: Distributed DoS (DDoS) against a Target

In this attack, the attacker provides one or more clients with the same faked reflexive address that points to the intended target. This will trick the STUN clients into thinking that their reflexive addresses are equal to that of the target. If the clients hand out that reflexive address in order to receive traffic on it (for example, in SIP messages), the traffic will instead be sent to the target. This attack can provide substantial amplification, especially when used with clients that are using STUN to enable multimedia applications. However, it can only be launched against targets for which packets from the STUN server to the target pass through the attacker, limiting the cases in which it is possible.

16.2.2. Attack II: Silencing a Client

In this attack, the attacker provides a STUN client with a faked reflexive address. The reflexive address it provides is a transport address that routes to nowhere. As a result, the client won't receive any of the packets it expects to receive when it hands out the reflexive address. This exploitation is not very interesting for the attacker. It impacts a single client, which is frequently not the desired target. Moreover, any attacker that can mount the attack could also deny service to the client by other means, such as preventing the client from receiving any response from the STUN server, or even a DHCP server. As with the attack in Section 16.2.1, this attack is only possible when the attacker is on path for packets sent from the STUN server towards this unused IP address.

16.2.3. Attack III: Assuming the Identity of a Client

This attack is similar to attack II. However, the faked reflexive address points to the attacker itself. This allows the attacker to receive traffic that was destined for the client.

16.2.4. Attack IV: Eavesdropping

In this attack, the attacker forces the client to use a reflexive address that routes to itself. It then forwards any packets it receives to the client. This attack would allow the attacker to observe all packets sent to the client. However, in order to launch the attack, the attacker must have already been able to observe packets from the client to the STUN server. In most cases (such as when the attack is launched from an access network), this means that the attacker could already observe packets sent to the client. This attack is, as a result, only useful for observing traffic by attackers on the path from the client to the STUN server, but not generally on the path of packets being routed towards the client.

16.3. Hash Agility Plan

This specification uses HMAC-SHA-1 for computation of the message integrity. If, at a later time, HMAC-SHA-1 is found to be compromised, the following is the remedy that will be applied.

We will define a STUN extension that introduces a new message-integrity attribute, computed using a new hash. Clients would be required to include both the new and old message-integrity attributes in their requests or indications. A new server will utilize the new message-integrity attribute, and an old one, the old. After a transition period where mixed implementations are in deployment, the old message-integrity attribute will be deprecated by another specification, and clients will cease including it in requests.

It is also important to note that the HMAC is done using a key that is itself computed using an MD5 of the user's password. The choice of the MD5 hash was made because of the existence of legacy databases that store passwords in that form. If future work finds that an HMAC of an MD5 input is not secure, and a different hash is needed, it can also be changed using this plan. However, this would require administrators to repopulate their databases.

17. IAB Considerations

The IAB has studied the problem of Unilateral Self-Address Fixing (UNSAF), which is the general process by which a client attempts to determine its address in another realm on the other side of a NAT through a collaborative protocol reflection mechanism (RFC3424 [RFC3424]). STUN can be used to perform this function using a Binding request/response transaction if one agent is behind a NAT and the other is on the public side of the NAT.

The IAB has mandated that protocols developed for this purpose document a specific set of considerations. Because some STUN usages provide UNSAF functions (such as ICE [MMUSIC-ICE]), and others do not (such as SIP Outbound [SIP-OUTBOUND]), answers to these considerations need to be addressed by the usages themselves.

18. IANA Considerations

IANA has created three new registries: a "STUN Methods Registry", a "STUN Attributes Registry", and a "STUN Error Codes Registry". IANA has also changed the name of the assigned IANA port for STUN from "nat-stun-port" to "stun".

18.1. STUN Methods Registry

A STUN method is a hex number in the range 0x000 - 0xFFF. The encoding of STUN method into a STUN message is described in Section 6.

The initial STUN methods are:

0x000: (Reserved)
0x001: Binding
0x002: (Reserved; was SharedSecret)

STUN methods in the range 0x000 - 0x7FF are assigned by IETF Review [RFC5226]. STUN methods in the range 0x800 - 0xFFF are assigned by Designated Expert [RFC5226]. The responsibility of the expert is to verify that the selected codepoint(s) are not in use and that the request is not for an abnormally large number of codepoints. Technical review of the extension itself is outside the scope of the designated expert responsibility.

18.2. STUN Attribute Registry

A STUN Attribute type is a hex number in the range 0x0000 - 0xFFFF. STUN attribute types in the range 0x0000 - 0x7FFF are considered comprehension-required; STUN attribute types in the range 0x8000 - 0xFFFF are considered comprehension-optional. A STUN agent handles unknown comprehension-required and comprehension-optional attributes differently.

The initial STUN Attributes types are:

Comprehension-required range (0x0000-0x7FFF):
  0x0000: (Reserved)
  0x0001: MAPPED-ADDRESS
  0x0002: (Reserved; was RESPONSE-ADDRESS)
  0x0003: (Reserved; was CHANGE-ADDRESS)
  0x0004: (Reserved; was SOURCE-ADDRESS)
  0x0005: (Reserved; was CHANGED-ADDRESS)
  0x0006: USERNAME
  0x0007: (Reserved; was PASSWORD)
  0x0008: MESSAGE-INTEGRITY
  0x0009: ERROR-CODE
  0x000A: UNKNOWN-ATTRIBUTES
  0x000B: (Reserved; was REFLECTED-FROM)
  0x0014: REALM
  0x0015: NONCE
  0x0020: XOR-MAPPED-ADDRESS

Comprehension-optional range (0x8000-0xFFFF)
  0x8022: SOFTWARE
  0x8023: ALTERNATE-SERVER
  0x8028: FINGERPRINT

STUN Attribute types in the first half of the comprehension-required range (0x0000 - 0x3FFF) and in the first half of the comprehension-optional range (0x8000 - 0xBFFF) are assigned by IETF Review [RFC5226]. STUN Attribute types in the second half of the comprehension-required range (0x4000 - 0x7FFF) and in the second half of the comprehension-optional range (0xC000 - 0xFFFF) are assigned by Designated Expert [RFC5226]. The responsibility of the expert is to verify that the selected codepoint(s) are not in use, and that the request is not for an abnormally large number of codepoints. Technical review of the extension itself is outside the scope of the designated expert responsibility.

18.3. STUN Error Code Registry

A STUN error code is a number in the range 0 - 699. STUN error codes are accompanied by a textual reason phrase in UTF-8 [RFC3629] that is intended only for human consumption and can be anything appropriate; this document proposes only suggested values.

STUN error codes are consistent in codepoint assignments and semantics with SIP [RFC3261] and HTTP [RFC2616].

The initial values in this registry are given in Section 15.6.

New STUN error codes are assigned based on IETF Review [RFC5226]. The specification must carefully consider how clients that do not understand this error code will process it before granting the request. See the rules in Section 7.3.4.

18.4. STUN UDP and TCP Port Numbers

IANA has previously assigned port 3478 for STUN. This port appears in the IANA registry under the moniker "nat-stun-port". In order to align the DNS SRV procedures with the registered protocol service, IANA is requested to change the name of protocol assigned to port 3478 from "nat-stun-port" to "stun", and the textual name from "Simple Traversal of UDP Through NAT (STUN)" to "Session Traversal Utilities for NAT", so that the IANA port registry would read:

```
stun   3478/tcp   Session Traversal Utilities for NAT (STUN) port
stun   3478/udp   Session Traversal Utilities for NAT (STUN) port
```

In addition, IANA has assigned port number 5349 for the "stuns" service, defined over TCP and UDP. The UDP port is not currently defined; however, it is reserved for future use.

19. Changes since RFC 3489

This specification obsoletes RFC 3489 [RFC3489]. This specification differs from RFC 3489 in the following ways:

- Removed the notion that STUN is a complete NAT traversal solution. STUN is now a tool that can be used to produce a NAT traversal solution. As a consequence, changed the name of the protocol to Session Traversal Utilities for NAT.

- Introduced the concept of STUN usages, and described what a usage of STUN must document.

- Removed the usage of STUN for NAT type detection and binding lifetime discovery. These techniques have proven overly brittle due to wider variations in the types of NAT devices than described in this document. Removed the RESPONSE-ADDRESS, CHANGED-ADDRESS, CHANGE-REQUEST, SOURCE-ADDRESS, and REFLECTED-FROM attributes.

- Added a fixed 32-bit magic cookie and reduced length of transaction ID by 32 bits. The magic cookie begins at the same offset as the original transaction ID.

- Added the XOR-MAPPED-ADDRESS attribute, which is included in Binding responses if the magic cookie is present in the request. Otherwise, the RFC 3489 behavior is retained (that is, Binding response includes MAPPED-ADDRESS). See discussion in XOR-MAPPED-ADDRESS regarding this change.

- Introduced formal structure into the message type header field, with an explicit pair of bits for indication of request, response, error response, or indication. Consequently, the message type field is split into the class (one of the previous four) and method.

- Explicitly point out that the most significant 2 bits of STUN are 0b00, allowing easy differentiation with RTP packets when used with ICE.

- Added the FINGERPRINT attribute to provide a method of definitely detecting the difference between STUN and another protocol when the two protocols are multiplexed together.

- Added support for IPv6. Made it clear that an IPv4 client could get a v6 mapped address, and vice versa.

- Added long-term-credential-based authentication.

- Added the SOFTWARE, REALM, NONCE, and ALTERNATE-SERVER attributes.

- Removed the SharedSecret method, and thus the PASSWORD attribute. This method was almost never implemented and is not needed with current usages.

- Removed recommendation to continue listening for STUN responses for 10 seconds in an attempt to recognize an attack.

- Changed transaction timers to be more TCP friendly.

- Removed the STUN example that centered around the separation of the control and media planes. Instead, provided more information on using STUN with protocols.

- Defined a generic padding mechanism that changes the interpretation of the length attribute. This would, in theory, break backwards compatibility. However, the mechanism in RFC 3489 never worked for the few attributes that weren't aligned naturally on 32-bit boundaries.

- REALM, SERVER, reason phrases, and NONCE limited to 127 characters. USERNAME to 513 bytes.

o Changed the DNS SRV procedures for TCP and TLS. UDP remains the same as before.

20. Contributors

Christian Huitema and Joel Weinberger were original co-authors of RFC 3489.

21. Acknowledgements

The authors would like to thank Cedric Aoun, Pete Cordell, Cullen Jennings, Bob Penfield, Xavier Marjou, Magnus Westerlund, Miguel Garcia, Bruce Lowekamp, and Chris Sullivan for their comments, and Baruch Sterman and Alan Hawrylyshen for initial implementations. Thanks for Leslie Daigle, Allison Mankin, Eric Rescorla, and Henning Schulzrinne for IESG and IAB input on this work.

22. References

22.1. Normative References

[ITU.V42.2002] International Telecommunications Union, "Error-correcting Procedures for DCEs Using Asynchronous-to-Synchronous Conversion", ITU-T Recommendation V.42, March 2002.

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC2988] Paxson, V. and M. Allman, "Computing TCP's Retransmission Timer", RFC 2988, November 2000.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.

22.2. Informative References

[BEHAVE-NAT] MacDonald, D. and B. Lowekamp, "NAT Behavior Discovery Using STUN", Work in Progress, July 2008.

[BEHAVE-TURN] Rosenberg, J., Mahy, R., and P. Matthews, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)", Work in Progress, July 2008.

[KARN87] Karn, P. and C. Partridge, "Improving Round-Trip Time Estimates in Reliable Transport Protocols", SIGCOMM 1987, August 1987.

[MMUSIC-ICE] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", Work in Progress, October 2007.

[MMUSIC-ICE-TCP] Rosenberg, J., "TCP Candidates with Interactive Connectivity Establishment (ICE)", Work in Progress, July 2008.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.

[RFC3424] Daigle, L. and IAB, "IAB Considerations for UNilateral Self-Address Fixing (UNSAF) Across Network Address Translation", RFC 3424, November 2002.

[RFC3489] Rosenberg, J., Weinberger, J., Huitema, C., and R. Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)", RFC 3489, March 2003.

[RFC4107] Bellovin, S. and R. Housley, "Guidelines for Cryptographic Key Management", BCP 107, RFC 4107, June 2005.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[SIP-OUTBOUND] Jennings, C. and R. Mahy, "Managing Client Initiated Connections in the Session Initiation Protocol (SIP)", Work in Progress, June 2008.

Appendix A. C Snippet to Determine STUN Message Types

Given a 16-bit STUN message type value in host byte order in msg_type parameter, below are C macros to determine the STUN message types:

```
#define IS_REQUEST(msg_type)       (((msg_type) & 0x0110) == 0x0000)
#define IS_INDICATION(msg_type)    (((msg_type) & 0x0110) == 0x0010)
#define IS_SUCCESS_RESP(msg_type)  (((msg_type) & 0x0110) == 0x0100)
#define IS_ERR_RESP(msg_type)      (((msg_type) & 0x0110) == 0x0110)
```

Authors' Addresses

Jonathan Rosenberg
Cisco
Edison, NJ
US

EMail: jdrosen@cisco.com
URI: http://www.jdrosen.net

Rohan Mahy
Unaffiliated

EMail: rohan@ekabal.com

Philip Matthews
Unaffiliated

EMail: philip_matthews@magma.ca

Dan Wing
Cisco
771 Alder Drive
San Jose, CA 95035
US

EMail: dwing@cisco.com

Full Copyright Statement

Intellectual Property