Multiplexed Application Substrate over QUIC Encryption
Internet-Draft
Intended status: Standards Track
Expires: 27 October 2025

Y. Rosomakho
Zscaler

25 April 2025

Reverse HTTP CONNECT for TCP and UDP
draft-rosomakho-masque-reverse-connect-00

Abstract

This document specifies an extension to the HTTP CONNECT method, enabling a proxy client to accept inbound TCP and UDP sessions proxied through HTTP/1.1, HTTP/2, or HTTP/3. This mechanism allows the client to dynamically advertise available local or internal network services and expose them through a HTTP proxy without reliance on IP routing.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://yaroslavros.github.io/draft-masque-reverse-connect/draft-rosomakho-masque-reverse-connect.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-rosomakho-masque-reverse-connect/.

Discussion of this document takes place on the Multiplexed Application Substrate over QUIC Encryption mailing list (mailto:masque@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/masque/. Subscribe at https://www.ietf.org/mailman/listinfo/masque/.

Source for this draft and an issue tracker can be found at https://github.com/yaroslavros/draft-masque-reverse-connect.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Reverse HTTP CONNECT for TCP [TCP] and UDP [UDP] extends the traditional CONNECT method (see Section 9.3.6 of [HTTP]) by enabling a proxy client to accept inbound TCP and UDP sessions proxied through HTTP/1.1 [HTTP/1.1], HTTP/2 [HTTP/2], or HTTP/3 [HTTP/3]. In contrast to the traditional CONNECT method, which establishes outbound sessions to remote servers, this extension facilitates inbound sessions to the proxy client, allowing access to local or internal services.

Unlike Proxying IP in HTTP [CONNECT-IP], this approach simplifies deployment in dynamic or constrained network environments by removing reliance on IP routing. By eliminating the need for routing configurations, this approach reduces operational complexity and allows for easier integration in scenarios where traditional IP routing is impractical. On top of that, Reverse HTTP CONNECT reduces overhead as it does not carry IP or transport layer headers.

Since Reverse HTTP CONNECT is built on top of existing HTTP transport, it can be efficiently combined with outbound CONNECT and other HTTP communications.

The primary use cases for this extension include:

* *Access to Local Services*: A proxy client can expose its own local services by accepting inbound TCP or UDP sessions from an HTTP server. For example, this can enable remote management or access to local application servers that can only be accessed through an outbound connection to the proxy.

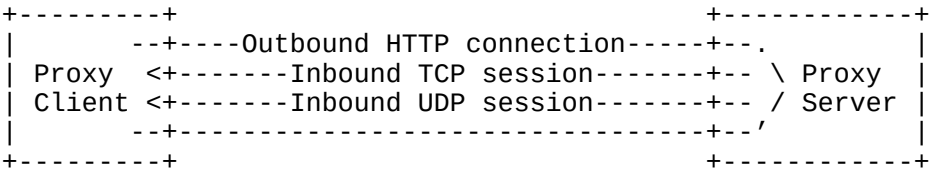

Figure 1: Accessing local client services

* *Gateway to Internal Networks*: A proxy client acts as a gateway, facilitating access to internal network services provided over TCP or UDP sessions. In this scenario, the proxy client forwards incoming session originating from an HTTP server to specific internal services, enabling secure communication with private network resources.

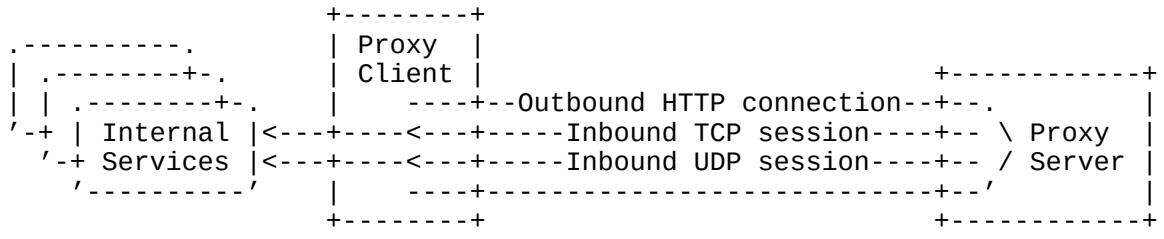

Figure 2: Accessing internal services through client

As explained in the specification both use cases can be combined on the same proxy client.

2. Conventions and Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Client Configuration

To use a Reverse Connect proxy HTTP clients are configured with two URI Templates [TEMPLATE]:

* Listener Control Channel described in Section 4.1

* Accepting Connection Requests described in Section 5.2.

4. Listener Control Channel

The Listener Control Channel enables the HTTP server to request the establishment of inbound TCP or UDP sessions on the proxy client. This is necessary because the HTTP protocol inherently expects the client to initiate connections. To address this, the Listener Control Channel uses the Capsule Protocol (see Section 3.2 of [HTTP-DGRAM]) to carry connection requests from the server to the client.

Additionally, the Listener Control Channel can carry optional Service Discovery Capsules, which allow the proxy client to advertise available local or internal services to the HTTP server. This feature facilitates dynamic service discovery and ensures that the server has up-to-date information about the services accessible through the proxy client.

4.1. Listener Control Channel URI Template

The Listener Control Channel template is similar to the one defined in Section 3 of [CONNECT-IP] and it MAY contain two variables: "target" and "ipproto". These variables are used to define the scope of network services that the client accepts connectivity for.

Examples are shown below:

```
https://example.org/.well-known/masque/listen/{target}/{ipproto}/
https://proxy.example.org:4443/masque/listen?t={target}&i={ipproto}
https://proxy.example.org:4443/masque/listen{?target,ipproto}
https://masque.example.org/?user=bob
```

Figure 3: Listener Control Channel URI Template Examples

All template requirements listed in Section 3 of [CONNECT-IP] apply here.

If set, the variable "target" MUST contain one of the following values:

* A hostname according to Section 4.6 of [CONNECT-IP]
* IPv4 or IPv6 prefix according to Section 4.6 of [CONNECT-IP]
* "*" that does not limit scope
* A wildcard: a domain name prefixed by "*.". This implies that client MAY accept inbound sessions for any host in a given domain
* "." which means that the client will accept sessions to local services only and will not forward to other hosts

If set, the variable "ipproto" MUST contain one of the following values:

* "6" which means that the client will only accept TCP sessions.
* "17" which means that the client will only accept UDP sessions.
* "*" which means that the client will accept both TCP and UDP sessions.

As with [CONNECT-IP], some client configurations for Reverse Connect proxies will only allow the user to configure the proxy host and proxy port. Clients with such limitations MAY attempt to access proxying capabilities using the default template, which is defined

as: "https://$PROXY_HOST:$PROXY_PORT/.well-known/masque/listen/{target}/{ipproto}/", where $PROXY_HOST and $PROXY_PORT are the configured host and port of the proxy, respectively. Reverse Connect proxy deployments SHOULD offer service at this location if they need to interoperate with such clients.

4.2. Establishing Listener Control Channel over HTTP/1.1

When establishing the Listener Control Channel using HTTP/1.1 [HTTP/1.1], the client follows the requirements from Section 4.2 of [CONNECT-IP] but uses "connect-listen" as the value for the Upgrade header.

For example, the client configured with the default URI Template "https://example.org/.well-known/masque/listen/{target}/{ipproto}/" can establish the Listener Control Channel for local TCP and UDP services by sending the following request:

```
GET https://example.org/.well-known/masque/listen/./*/ HTTP/1.1
Host: example.org
Connection: Upgrade
Upgrade: connect-listen
Capsule-Protocol: ?1
```

Figure 4: Example HTTP/1.1 Request

The server confirms successful registration of the client as described in Section 4.3 of [CONNECT-IP], but with "connect-listen" for the value for Upgrade header.

An example of the server confirming successful registration of the client is provided below:

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: connect-listen
Capsule-Protocol: ?1
```

Figure 5: Example HTTP/1.1 Response

4.3. Establishing Listener Control Channel over HTTP/2 and HTTP/3

When establishing Listener Control Channel using HTTP/2 [HTTP/2] or HTTP/3 [HTTP/3], the client follows requirements from Section 4.4 of [CONNECT-IP], but uses "connect-listen" as the value for :protocol pseudo-header.

For example, the client configured with the default URI Template "https://example.org/.well-known/masque/listen/{target}/{ipproto}/" can establish the Listener Control Channel for local TCP and UDP services by sending the following request:

```
HEADERS
:method = CONNECT
:protocol = connect-listen
:scheme = https
:path = /.well-known/masque/listen/./*/
:authority = example.org
capsule-protocol = ?1
```

Figure 6: Example HTTP/2 or HTTP/3 Request

The server indicates a successful registration of the client as described in Section 4.5 of [CONNECT-IP].

Example of server confirming successful registration of the client is provided below:

```
HEADERS
:status = 200
capsule-protocol = ?1
```

Figure 7: Example HTTP/2 or HTTP/3 Response

4.4. Service Discovery

The Reverse Connect client MAY advertise offered TCP and UDP services to the proxy server through the AVAILABLE_SERVICES Capsule over the established Listener Control Channel. This information serves as a hint for the proxy server and does not constrain the server from attempting to establish sessions with services that were not advertised. The presence of a service in the AVAILABLE_SERVICES list does not guarantee actual service availability, as the client may not be able to track the health of the service.

Capsule format for AVAILABLE_SERVICES is the following:

```
AVAILABLE_SERVICES Capsule {
  Type (i) = 0xTBD,
  Length (i),
  Service (..) ...,
}
```

Figure 8: AVAILABLE_SERVICES Capsule Format

The AVAILABLE_SERVICES capsule contains a sequence of zero or more Services.

```
Service {
  Destination Type (8),
  Destination (..),
  Protocol (8),
  Port (16),
}
```

Figure 9: Service Format

Each service record represents potential connection destination and contains the following fields:

Destination Type: A single byte value defining type and format of Destination field. Valid destination types are:

* 0: Destination is local for the client.

* 1: Destination is specified by a hostname.

* 4: Destination is specified by an IPv4 address.

* 6: Destination is specified by an IPv6 address.

Destination: Content of the Destination field is determined by the Destination Type. For local destination type destination is omitted. IPv4 destination carries 32 bits of IPv4 address. IPv6 destination carries 128 bits of IPv6 address. Hostname destination (destination type 1) is provided in the following format:

```
Hostname_Destination {
  Length (i),
  Hostname (..),
}
```

Figure 10: Hostname Destination Format

Length is encoded as a variable-length integer and Hostname contains the hostname string.

Protocol: A single byte value of 6 for TCP and 17 for UDP.

Port: Two bytes value of target TCP or UDP port.

AVAILABLE_SERVICES messages are not incremental. A new message completely overwrites the previous hint.

If any of the capsule fields are malformed upon reception, the server MUST follow the error-handling procedure defined in Section 3.3 of [HTTP-DGRAM].

5. Connection Lifecycle

5.1. Requesting Connection

To request a new outbound connection from the client for an inbound TCP or UDP session, the server sends a CONNECTION_REQUEST Capsule. The CONNECTION_REQUEST Capsule is defined as follows:

```
CONNECTION_REQUEST Capsule {
  Type (i) = 0xTBD,
  Length (i),
  Request ID (i),
  Service (..),
}
```

Figure 11: CONNECTION_REQUEST Capsule Format

Request ID: A unique request identifier, encoded as a variable-length integer. The Request ID MUST be unique for a given Listener Control Channel.

Service: Session destination as described in Section 4.4.

If any of the capsule fields are malformed upon reception, or if the Request ID has been previously used on the same Listener Control Channel, the client MUST follow the error-handling procedure defined in Section 3.3 of [HTTP-DGRAM].

If the client declines the connection request it responds with a CONNECTION_REQUEST_DECLINED Capsule in the following format:

```
CONNECTION_REQUEST_DECLINED Capsule {
  Type (i) = 0xTBD,
  Length (i),
  Request ID (i),
}
```

Figure 12: CONNECTION_REQUEST_DECLINED Capsule Format

If any of the capsule fields are malformed upon reception, or if the Request ID does not correspond to an outstanding CONNECTION_REQUEST on the same Listener Control Channel, the server MUST follow the error-handling procedure defined in Section 3.3 of [HTTP-DGRAM].

5.2. Accepting Connection

To accept the inbound session, the client sends a new outbound request to the server. This request uses an Accepting Connection URI Template [TEMPLATE]. The URI Template MUST contain "request_id" variable.

Examples are shown below:

```
https://example.org/.well-known/masque/accept/{request_id}/
https://proxy.example.org:4443/masque/accept?id={request_id}
https://proxy.example.org:4443/masque/accept{?request_id}
https://masque.example.org/?user=bob&request_id={request_id}
```

Figure 13: Accepting Connection URI Template Examples

The following requirements apply to the Accepting Connection URI Template:

* The URI Template MUST be a level 3 template or lower.
* The URI Template MUST be in absolute form and MUST include non-empty scheme, authority, and path components.
* The path component of the URI Template MUST start with a slash "/".
* All template variables MUST be within the path or query components of the URI.
* The URI Template MUST contain variable "request_id" and MAY contain other variables.
* The URI Template MUST NOT contain any non-ASCII Unicode characters and MUST only contain ASCII characters in the range 0x21-0x7E inclusive (note that percent-encoding is allowed; see Section 2.1 of [URI].
* The URI Template MUST NOT use Reserved Expansion ("+" operator), Fragment Expansion ("#" operator), Label Expansion with Dot-Prefix, Path Segment Expansion with Slash-Prefix, nor Path-Style Parameter Expansion with Semicolon-Prefix.

Clients SHOULD validate the requirements above; however, clients MAY use a general-purpose URI Template implementation that lacks this specific validation. If a client detects that any of the requirements above are not met by a URI Template, the client MUST reject its configuration and abort the request without sending it to the IP proxy.

As with [CONNECT-IP], some client configurations for Reverse Connect proxies will only allow the user to configure the proxy host and proxy port. Clients with such limitations MAY attempt to access proxying capabilities using the default template, which is defined as: "https://$PROXY_HOST:$PROXY_PORT/.well-known/masque/accept/{request_id}/", where $PROXY_HOST and $PROXY_PORT are the configured host and port of the proxy, respectively. Reverse Connect proxy deployments SHOULD offer service at this location if they need to interoperate with such clients.

5.2.1. Accepting Connection over HTTP/1.1

When accepting an inbound session over HTTP/1.1 [HTTP/1.1], the client sends to the proxy a new request as follows:

* The method SHALL be "GET"
* The request SHALL include a single "Host" header field containing the origin of the proxy.
* The request SHALL include a "Connection" header field with the value "Upgrade" (note that this requirement is case-insensitive as per Section 7.6.1 of [HTTP]).
* The request SHALL include an Upgrade header field with value "connect-accept".

A request that does not match the above restrictions is considered malformed. A proxy server receiving a malformed request MUST respond with an error and SHOULD use the 400 (Bad Request) status code. If request_id template variable is not matching an outstanding connection request for given client, proxy server MUST respond with an error and SHOULD use the 404 (Not Found) status code.

The proxy SHALL indicate a successful response by replying with the following requirements:

* The HTTP status code on the response SHALL be 101 (Switching Protocols).

* The response SHALL include a Connection header field with value "Upgrade" (note that this requirement is case-insensitive as per Section 7.6.1 of [HTTP]).

* The response SHALL include a single Upgrade header with value "connect-accept".

* The response SHALL meet the requirements of HTTP responses that start the Capsule Protocol according to Section 3.2 of [HTTP-DGRAM].

A response that does not match these requirements MUST be treated as failed and corresponding connection MUST be aborted by the client.

For example, the client configured with the URI Template "https://example.org/.well-known/masque/accept/{request_id}/" can accept the inbound Connection as a response to a connection request with Request ID 1:

```
GET https://example.org/.well-known/masque/accept/1/ HTTP/1.1
Host: example.org
Connection: Upgrade
Upgrade: connect-accept
Capsule-Protocol: ?1
```

Figure 14: Example HTTP/1.1 Request Accepting Connection

The proxy server could respond with the following:

```
HTTP/1.1 101 Switching Protocols
Connection: Upgrade
Upgrade: connect-accept
Capsule-Protocol: ?1
```

Figure 15: Example HTTP/1.1 Response Accepting Connection

5.2.2. Accepting Connection over HTTP/2 and HTTP/3

When accepting an inbound sessions over HTTP/2 [HTTP/2] or HTTP/3 [HTTP/3], the client sends to the proxy a request on a new Stream of the connection that carries the Listener Control Channel as follows:

* The :method pseudo-header field SHALL be "CONNECT".

* The :protocol pseudo-header field SHALL be "connect-accept".

* The :authority pseudo-header field SHALL contain the authority of the proxy.

* The :path and :scheme pseudo-header fields SHALL contain the path and scheme of the request URI derived from the Accepting Connection URI template for the proxy.

A request that does not match the above restrictions is considered malformed. A proxy server receiving a malformed request MUST respond with an error and SHOULD use the 400 (Bad Request) status code. If "request_id" template variable is not matching an outstanding connection request for given client, proxy server MUST respond with an error and SHOULD use the 404 (Not Found) status code.

The proxy SHALL indicate a successful response by replying with the following requirements:

* The HTTP status code on the response SHALL be in 2xx (Successful) range.

* The response SHALL meet the requirements of HTTP responses that start the Capsule Protocol according to Section 3.2 of [HTTP-DGRAM].

A response that does not match these requirements MUST be treated as failed and corresponding connection MUST be aborted by the client.

For example, the client configured with the URI Template "https://example.org/.well-known/masque/accept/{request_id}/" can accept the inbound session as a response to a connection request with Request ID 1:

```
HEADERS
:method = CONNECT
:protocol = connect-accept
:scheme = https
:path = /.well-known/masque/accept/1/
:authority = example.org
capsule-Protocol = ?1
```

Figure 16: Example HTTP/2 or HTTP/3 Request Accepting Connection

The proxy server could respond with the following:

```
HEADERS
:status = 200
capsule-Protocol = ?1
```

Figure 17: Example HTTP/2 or HTTP/3 Response Accepting Connection

5.3. Establishing Connection

Once the client receives a successful response from the proxy for the request accepting the connection it establishes the requested TCP or UDP socket to a local or internal destination. If the socket failed to establish, the client MUST immediately close the request stream.

For TCP flows, communicating parties carry TCP payload data in the payload of DATA Capsules over the established stream according to Section 8.3 of [Template-TCPCONNECT]. Either party MAY close the stream if TCP connection is terminated.

UDP data is carried in DATAGRAM capsules or encoded in QUIC DATAGRAM frames as explained in Section 5 of [CONNECT-UDP]. The lifetime of the UDP socket is tied to the request stream as described in Section 3.1 of [CONNECT-UDP].

6. Security Considerations

Proxy servers providing Reverse HTTP Connect services MUST restrict their services to authenticated users. An unauthorized user accepting an inbound session from the proxy server may cause availability risks and compromise the wider systems by misrouting communications. Clients that advertise services through the AVAILABLE_SERVICES Capsule may inadvertently expose sensitive or private services. Proxy servers MUST verify that advertised services comply with organizational security policies.

To avoid exposing local or internal services to unauthorized parties, clients MUST confirm identity of the proxy service that they connect to and allow inbound sessions only to services that should be accessible from the proxy.

Proxies providing Reverse HTTP Connect service over HTTP/1.1 MUST consistently authenticate clients on both Listener Control Channel and individual Connection Acceptance Requests. To minimize the risks of misrouting connection request to a rogue client, HTTP/1.1 Reverse Connect proxies SHOULD generate Request ID randomly instead of using a predictable sequence.

7. IANA Considerations

7.1. HTTP Upgrade Tokens

IF APPROVED, IANA is requested to add the following entries to the HTTP Upgrade Token Registry:

Value	Description	Reference
"connect-listen"	Listening for inbound connection requests	(This document)
"connect-accept"	Accepting an inbound connection request	(This document)

Table 1

7.2. New MASQUE Default Templates

IF APPROVED, IANA is requested to add the following entries to the MASQUE URI Suffixes Registry:

Value	Description	Reference
"listen"	Listening for inbound connection requests	(This document)
"accept"	Accepting an inbound connection request	(This document)

Table 2

7.3. New HTTP Capsule Types

IF APPROVED, IANA is requested to add the following entries to the HTTP Capsule Types Registry:

Value	Capsule Type
(TBD)	AVAILABLE_SERVICES
(TBD)	CONNECTION_REQUEST
(TBD)	CONNECTION_REQUEST_DECLINED

Table 3

All of these new entries use the following values for these fields:

Status: permanent Reference: (this document) Change Controller: IETF
Contact: MASQUE Notes: None

8. Normative References

[CONNECT-IP]
Pauly, T., Ed., Schinazi, D., Chernyakhovsky, A., K端hlewind, M., and M. Westerlund, "Proxying IP in HTTP", RFC 9484, DOI 10.17487/RFC9484, October 2023, <https://www.rfc-editor.org/rfc/rfc9484>.

[CONNECT-UDP]
Schinazi, D., "Proxying UDP in HTTP", RFC 9298, DOI 10.17487/RFC9298, August 2022, <https://www.rfc-editor.org/rfc/rfc9298>.

[HTTP]
Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[HTTP-DGRAM]
Schinazi, D. and L. Pardue, "HTTP Datagrams and the Capsule Protocol", RFC 9297, DOI 10.17487/RFC9297, August 2022, <https://www.rfc-editor.org/rfc/rfc9297>.

[HTTP/1.1]
Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <https://www.rfc-editor.org/rfc/rfc9112>.

[HTTP/2]
Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/rfc/rfc9113>.

[HTTP/3]
Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/rfc/rfc9114>.

[RFC2119]
Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/rfc/rfc2119>.

[RFC8174]
Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/rfc/rfc8174>.

[TCP] Postel, J., "Transmission Control Protocol", RFC 793, DOI 10.17487/RFC0793, September 1981, <https://www.rfc-editor.org/rfc/rfc793>.

[TEMPLATE] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/rfc/rfc6570>.

[Template-TCPCONNECT] Schwartz, B. M., "Template-Driven HTTP CONNECT Proxying for TCP", Work in Progress, Internet-Draft, draft-ietf-httpbis-connect-tcp-07, 17 March 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-connect-tcp-07>.

[UDP] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <https://www.rfc-editor.org/rfc/rfc768>.

[URI] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/rfc/rfc3986>.

Acknowledgments

TODO acknowledge.

Author’s Address

Yaroslav Rosomakho
Zscaler
Email: yrosomakho@zscaler.com