Vanadium Proxy Protocol

suharshs@google.com
March 2, 2016

Abstract

Vanadium assumes an environment composed of various *networks*, e.g. public internet, private intranets, NFC, bluetooth, etc. Communication between endpoints within a single network can always be initiated, but this is not true for endpoints in different networks. Vanadium solves this issue with Proxies.
Servers can make connections to proxies that live outside of the Server's network. The Proxy will route incoming connections from Clients to the Server, making the Server *accessible* outside of its network. Proxies enable end-to-end connections between Servers and Clients while keeping the data incomprehensible to the proxy.
There are cases where we want to have multiple Proxies between a dialing and accepting process (i.e. to traverse multiple network/protocol boundaries).
This doc describes the implementation of the Vanadium proxy which addresses the following issues:

- Secure end-to-end connections between servers and clients, where malicious proxies cannot "spy" on RPCs.
- Support for multiple proxy hops.
- Rendezvous listening: as proxies gain and lose new endpoints (due to connectivity issues, etc.), they communicate these changes to the server, or intermediate proxies, so that the server can mount up to date endpoints.

Overview

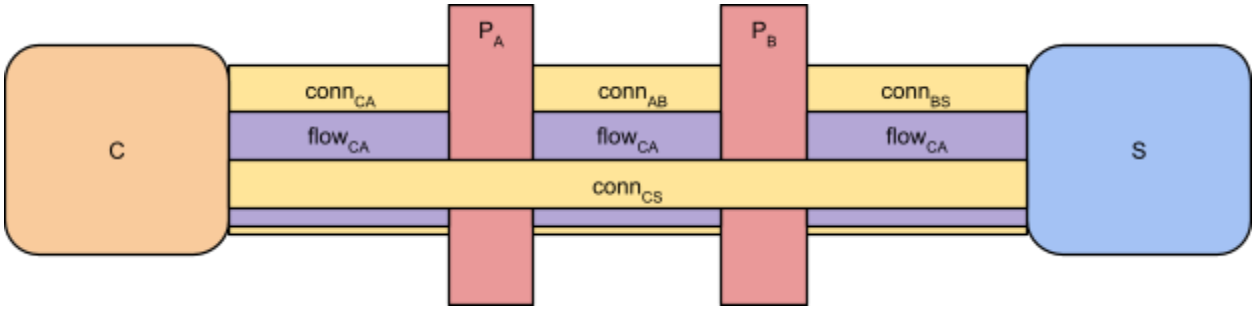
Definitions

A Conn is a bidirectional byte stream between two processes.
Flows are encrypted, authenticated byte streams between two Vanadium processes. Flows are multiplexed on Conns.
Flow Managers that manage the creations of Conns and Flows. Each Vanadium process has a Flow Manager with a globally unique id. This id is called a RoutingID.

RoutingIDs are self-selected by Flow Managers. Nothing prevents two Flow Managers from using the same RoutingID. In the case that this happens, the security model will prevent information leaks.
A local route is a local routing key used between intermediate proxies.
And endpoint contains all the information needed to connect to a peer (plus some security information, which we will ignore in this doc). It has the form
`host:port@route`$_0$`...route`$_N$`@RoutingID.`
Code

Encapsulated Flows

The proxy protocol is implemented by encapsulating Flows on other Flows. See below:



In the image above there are four processes: Client C, Server S, and Proxies P_A and P_B. C wants to create a Flow to S, but S is only accessible through both proxies P_A and P_B. Flows are created between adjacent processes, (C,P_A), (P_A,P_B), (P_BS). The proxies copy bytes from incoming flows to outgoing connections.

Protocol

Messages

The proxy protocol consist of the following three messages sent on Flows.

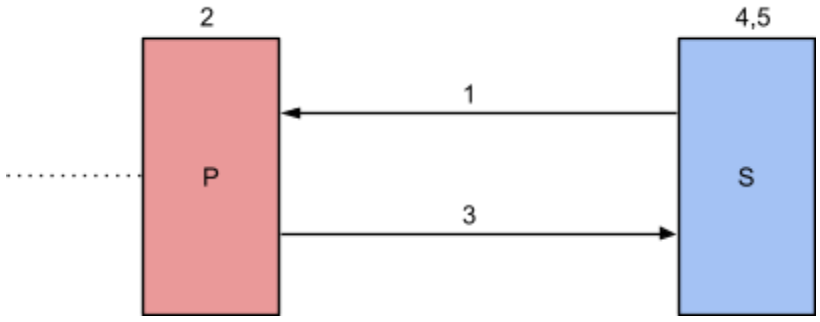
```
// ProxyServerRequest is sent when a server wants to listen through a proxy.
type ProxyServerRequest struct{}

// MultiProxyRequest is sent when a proxy wants to accept connections from
another proxy.
type MultiProxyRequest struct{}

// ProxyResponse is sent by a proxy in response to a ProxyServerRequest or
// MultiProxyRequest. It notifies the server of the endpoints it should publish
// or notifies a requesting proxy of the endpoints it accepts connections from.
// Subsequent ProxyResponse messages are sent as changes to the endpoints occur
type ProxyResponse struct {
      Endpoints []naming.Endpoint
```

}
Code

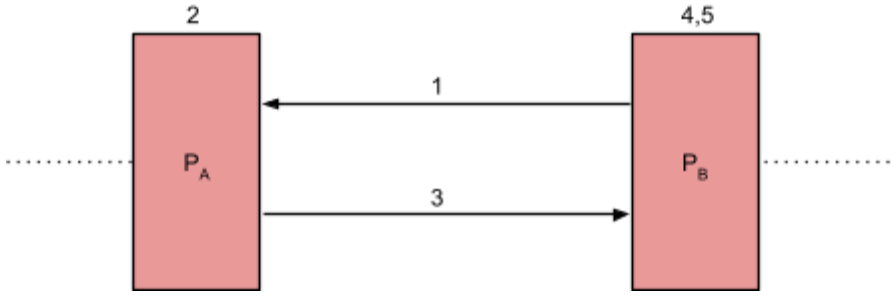
Servers listening through proxies.



In the image above S wants to be proxied through P. This is what takes place:

1. S dials a Flow to P. P and S are authorized here. S sends a ProxyServerRequest to P on this Flow.
2. P's Flow Manager stores the Conn from S in its cache, keyed by the RoutingID of S. *Note that we don't use a local route in the endpoint here.*
3. P sends S the a ProxyResponse with encoded endpoint for S on the Flow. If P is also proxied, this will be of the form $host_X$:$port_X$@...$route_P$@$RoutingID_S$. Otherwise the endpoint will be of the form $host_P$:$port_P$@@$RoutingID_S$.
4. S can now publish its Client accessible endpoint.
5. S will continue to listen on the dialed Flow for any new ProxyResponse messages and update its published endpoints accordingly.

Proxies listening through proxies



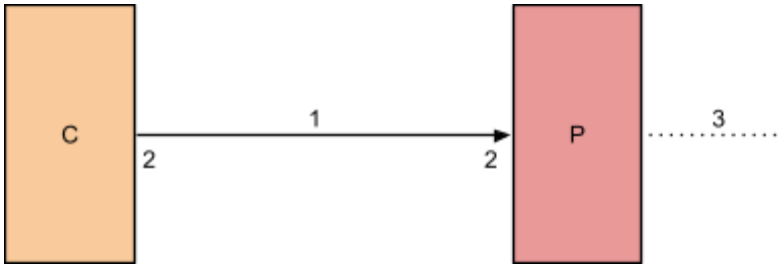
In the image above P_B wants to be proxied through P_A. This is what takes place:

1. P_B dials a Flow to P_A. P_A and P_B are authorized here. P_B sends a MultipleProxyRequest on this Flow.
2. P_A's Flow Manager stores the Conn from P_B in its cache, keyed by the RoutingID of P_B. P_A allocates a local route to identity P_B. The RoutingID of P_B is now stored in a map in P_A keyed by local route. This local route is much smaller than the global ID, to reduce the size of the resulting Server endpoint.

3. P_A sends a Proxy Response to P_B with the encoded endpoint for P_B on the Flow. If P_A is also proxied, this will be of the form $host_X$:$port_X$@...$route_A$$route_B$. Otherwise the endpoint will be of the form $host_A$:$port_A$@$route_B$.
4. Now P_B can handle any proxy requests it has.
5. P_B will continue to listen on the dialed Flow for any new ProxyResponse message from P_A and send ProxyResponse update messages to any proxied Servers accordingly. This allows Proxies to start up in any order and the server to eventually get all of its accessible endpoints.

Clients connecting through a proxy

Client connects to Proxy.

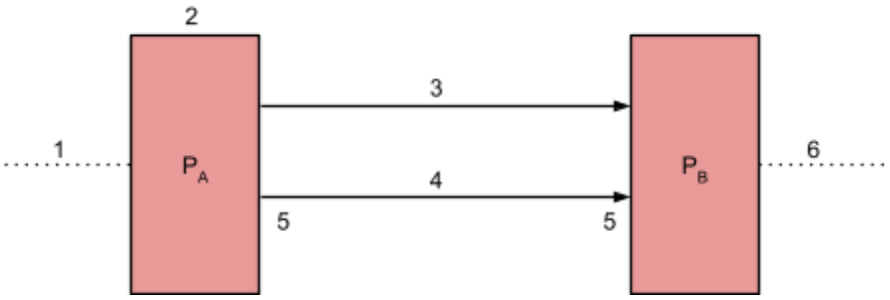


In the image above, C wants to connect to a Server that is proxied through P. This is what takes place between C and P:

1. C has to dial a Server with endpoint $host_P$:$port_P$@$route_x$,...,$route_z$@$RoutingID_S$. It connects to $host_P$:$port_P$, creating a Flow to P. P is authorized by C here. C sends a Setup message to the end server on the Flow just established.
2. Now, both sides turn off C/P encryption/decryption on the Flow. This is because all data messages sent and read by C after this point will be encrypted between C and S.
3. P handles the situation in the two cases below.

Proxy connects to Proxy.

If P needs to connect to another Proxy to access S (i.e. if there are local routes in the endpoint).

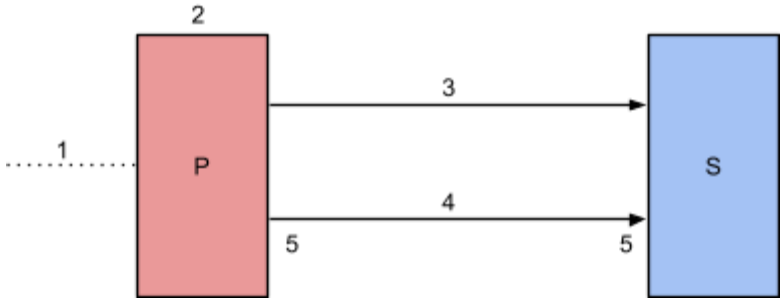


In the image above, P_A receives message that needs to be routed to P_B. This is how it happens:

1. P_A receives an incoming Setup message (see above) on the incoming Flow.
2. P_A reads the destination endpoint $host_X$:$port_X$@$route_B$…@$RoutingID_S$ from the Setup message. Since there is are routes in the endpoint, P_A looks up the first route, $route_B$ in its local route cache. It finds global $RoutingID_B$.

3. P_A uses $RoutingID_B$ to dial a Flow to P_B on the Conn cached by the Flow Manager on P_A.
4. P_A alters the Setup message by remote its $route_B$ from the endpoint and sends the new Setup message to P_B on the Flow it just created.
5. Both sides turn off A/B encryption/decryption on the Flow. P_A now starts two loops that copy bytes between the incoming Flow to the Flow to P_B.
6. P_B either repeats these steps if forwarding to another proxy, or follows the steps below.

Proxy connects to Server.
If P can directly access S (i.e. There are no routes in the endpoint, just a RoutingID)



In the image above, P receives a message that needs to be routed to end-server S. This is how it happens:

1. P receives an incoming Setup message (see above).
2. P reads the destination endpoint $host_X$:$port_X$@@$RoutingID_S$ from the Setup message. Since there are no routes in the endpoint, P reads the $RoutingID_S$ from the endpoint.
3. P uses $RoutingID_s$ to dial a Flow to S on the Conn cached by the Flow Manager on P.
4. P forwards the Setup message as-is to S on the Flow just created.
5. Both sides turn off P/S encryption/decryption on the Flow. P now starts two loops that copy bytes between the incoming Flow to the outgoing Flow to S.

At this point, the Client and Server have a unencrypted bytes stream connecting them that spans multiple proxies. The client proceeds to send a Setup message intended for S on this byte stream. The Flow creation process now just follows as usually.
Additionally since the Server's RoutingID is in the endpoint that the Client used to dial it, the Client can use the RoutingID to cache and reuse this Conn.

Implementation

The implementation of the proxy can be found here.