dispatch
Internet-Draft
Intended status: Standards Track
Expires: April 28, 2011

J.R. Rosenberg
jdrosen.net
C. Jennings
Cisco
M. Petit-Huguenin
Stonyfish
October 25, 2010

Verification Involving PSTN Reachability: Requirements and Architecture Overview
draft-rosenberg-dispatch-vipr-overview-04

Abstract

The Session Initiation Protocol (SIP) has seen widespread deployment within individual domains, typically supporting voice and video communications. Though it was designed from the outset to support inter-domain federation over the public Internet, such federation has not materialized. The primary reasons for this are the complexities of inter-domain phone number routing and concerns over security. This document reviews this problem space, outlines requirements, and then describes a new model and technique for inter-domain federation with SIP, called Verification Involving PSTN Reachability (ViPR). ViPR addresses the problems that have prevented inter-domain federation over the Internet. It provides fully distributed inter-domain routing for phone numbers, authorized mappings from phone numbers to domains, a new technique for automated VoIP anti-spam, and privacy of number ownership, all while preserving the trapezoidal model of SIP.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering

Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 28, 2011.

Copyright Notice

Table of Contents

1. Introduction

The Session Initiation Protocol (SIP) was originally published as RFC 2543 [RFC2543] in May of 1999. This was followed by subsequent publication of RFC 3261 [RFC3261], which brought the protocol to sufficient maturity to enable large scale market adoption.

And indeed, it has seen large scale market adoption. SIP has seen hundreds of implementations, spanning consumer products, enterprise servers, and large scale carrier equipment. It carries billions and billions of minutes of calls, and has become the lingua franca of interconnection between products from different vendors. If one measures success in deployment, then clearly SIP is a success.

However, in other ways, it has failed. SIP was designed from the ground up to enable communications between users in different domains, all over the public Internet. The intention was that real-time communications should be no different than email or the web, with the same any-to-any connectivity that has fueled the successes of those technologies. Though SIP is used between domains, it is typically through private federation agreements. The any-to-any Internet federation model envisioned by SIP has not materialized at scale.

This document introduces a new technology, called Verification Involving PSTN Reachability (ViPR), that enables us to break down the barriers that have prevented inter-domain VoIP. By stepping back and changing some of the most fundamental assumptions about federation, ViPR is able to address the key problems preventing its deployment. ViPR focuses on incremental deployability over the unrealizable nirvana. At the same time, ViPR ensures that SIP’s trapezoidal model - direct federation between domains without any intermediate processing beyond IP transport - is realized. That model is required in order to allow innovative new services to be deployed.

2. Problem Statement

The first question that must be asked is this - why haven’t we seen widespread adoption of inter-domain SIP federation?

There are many reasons for it. They are - in order of importance - the phone number routing problem, the open pinhole problem, the quality of service problem, and the troubleshooting problem. The two former ones are the most significant.

2.1. The Phone Number Routing Problem

Inter-domain federation requires that the sending domain determine the address of the receiving domain, in the form of a DNS name (example.com) or one or more IP addresses that can be used to reach the domain. In email and in the web, this is easy. The identifiers used by those services - the email address and web URL respectively - embed the address of the receiving domain. A simple DNS lookup is all that is required to route the connection. SIP was designed to use the same email-style identifiers.

However, most SIP deployments utilize phone numbers, and not email-style SIP URI. This is due to the huge installed base of users that continue to exist solely on the public switched telephone network (PSTN). In order to be reached by users on the PSTN, and in order to reach them, users in SIP deployments need to be assigned a regular PSTN number. Users in SIP deployments need to place that PSTN number on business cards, use it in their email signatures, and in general, give it out to their friends and colleagues, in order to be reached. While those users could additionally have an email style SIP URI, the PSTN number serves as a single, global identifier that works for receiving calls from users on the PSTN as well as users within the same SIP domain. Why have two identifiers when one will suffice? The universality of PSTN numbers is the reason why most SIP deployments continue to use them - often exclusively.

Another reason is that many SIP deployments utilize hardphones or telephony adaptors, and the user interfaces on these devices - patterned after existing phones - only allow phone-number based dialing. Consequently, these users are only allocated PSTN numbers, and not email-style SIP URI.

Finally, a large number of SIP deployments are in domains where the endpoints are not IP. Rather, they are circuit based devices, connected to a SIP network through a gateway. SIP is used within the core of the network, providing lower cost transit, or providing add-on services. Clearly, in these deployments, only phone numbers are used.

Consequently, to make inter-domain federation incrementally deployable and widely applicable, it needs to work with PSTN numbers rather than email-style SIP URI. Telephone numbers, unlike email addresses, do not provide any indication of the address of the domain which "owns" the phone number. Indeed, the notion of phone number ownership is somewhat cloudy. Numbers can be ported between carriers. They can be assigned to a user or enterprise, and then later re-assigned to someone else. Numbers are granted to users and enterprises through a complex delegation process involving the ITU,

governments, and telecommunications carriers, often involving local regulations that vary from country to country.

Therefore, in order to deploy inter-domain federation, domains are required to utilize some kind of mechanism to map phone numbers to the address of the domain to which calls should be routed. Though several techniques have been developed to address this issue, none have achieved large-scale Internet deployments.

2.2. The Open Pinhole Problem

The inter-domain federation mechanism built into SIP borrows heavily from email. Each domain runs a SIP server on an open port. When one domain wishes to contact another, it looks up the domain name in the DNS, and connects to the that server on the open port. Here, "open" means that the server is reachable from anywhere on the public Internet, and is not blocked by firewalls.

This simple design worked well in the early days of email. However, the email system has now become plagued with spam, to the point of becoming useless. Administrators of SIP domains fear - rightfully so - that if they make a SIP server available for anyone on the Internet to contact, it will open the floodgates for VoIP spam, which is far more disruptive than email-based spam [RFC5039]. Administrators also worry - rightfully so - that an open server will create a back-door for denial-of-service and other attacks that can potentially disrupt their voice service. Administrators are simply not willing to take that risk; rightly or wrongly, voice deployments demand higher uptimes and better levels of reliability than email, especially for enterprises.

Fears around spam and denial-of-service attacks, when put together, form the "open pinhole problem" - that domains are not willing to enable SIP on an open port facing the Internet.

To fix this, a new model for federation is needed - a model where these problems are addressed as part of the fundamental design, and not as an after-thought.

2.3. Quality of Service Problem

The Internet does not provide any QoS guarantees. All traffic is best effort. This is not an issue for data transaction services, like web and email. It is, however, a concern when using real-time services, such as voice and video.

That said, there are a large number of existing VoIP deployments that run over the Internet. Though the lack of QoS is a concern, it has

not proven a barrier to deployment. We believe that, if the more fundamental issues - the phone number routing and open pinhole problems - can be addressed, the QoS problem will sort itself out. As such, we do not discuss this issue further here.

2.4. Troubleshooting Problem

The final problem that is stopping large scale inter-domain federation is the troubleshooting problem. When connecting calls between domains, problems will happen. Calls will get blocked. Calls will get misdelivered. Features won't work. There will be one-way media or no media at all. The video won't start. Call quality will be poor.

These problems are common in VoIP deployments, and they are tough to troubleshoot even within a single administrative domain. When real-time services extend inter-domain, the problem becomes worse. A new angle is introduced: the first step is identifying who is at fault.

Fortunately, work is underway to improve the ability for network administrators to diagnose VoIP problems. Common log formats [CLF-SYNTAX] and consistent session IDs [SESSION-ID], for example, can help troubleshoot interdomain calls.

In addition to these, any new technology that facilitates inter-domain federation needs to have troubleshooting built-in, so that it is not a barrier to deployment.

3. Summary of Existing Solutions

Given the value that inter-domain SIP federation brings, it is no surprise that many attempts have been made at solving it. Indeed, these have all been deployed to varying degrees. However, all of them have fundamental limitations that have inhibited widespread deployment.

3.1. Domain Routing

The first solution that has been proposed for SIP inter-domain federation is built into SIP itself - domain routing. In this technique, users utilize email-style SIP URI as identifiers. By utilizing the DNS lookup mechanism defined in [RFC3263], SIP enables calls to be routed between domains in much the same way email is routed between domains.

This technique works well in theory, but it has two limitations which have limited its deployment:

1. The majority of SIP deployments utilize phone numbers, often exclusively. In such a case, domain routing cannot be used.
2. Domain federation brings with it the possibility (and strong likelihood) of the same levels of spam and DoS attacks that have plagued the email system.

These issues have already been discussed above.

3.2. Public ENUM

Public ENUM, defined in [RFC3761], tries to address the phone number routing problem by cleverly placing phone numbers into the public DNS. Clients can then perform a simple DNS lookup on a phone number, and retrieve a SIP URI which can be used to route to that phone number.

Unfortunately, public ENUM requires that the entries placed into the DNS be populated following a chain of responsibility that mirrors the ownership of the numbers themselves. This means that, in order for a number to be placed into the DNS, authorization to do so must start with the ITU, and from there, move to the country, telecom regulator, and ultimately the end user. The number of layers of bureaucracy required to accomplish this is non-trivial. In addition, the telecom operators - which would be partly responsible for populating the numbers into the DNS - have little incentive to do so. As a consequence, public ENUM is largely empty, and is likely to remain so for the foreseeable future.

Instead, ENUM has morphed into a technique for federation amongst closed peering partners, called private ENUM or infrastructure ENUM [RFC5067]. While there is value in this technology, it does not enable the open federation that public ENUM was designed to solve.

It is clear from the legacy of ENUM deployments, that any kind of phone number routing solution should not rely on government or telecom processes for population of the databases.

3.3. Private Federations

Private federations are a cooperative formed amongst a small number of participating domains. The cooperative agrees to use a common technique for federation, and through it, is able to connect to each other. There are many such federations in use today.

Some of these federations rely on a central database, typically run by the federation provider, that can be queried by participating domains. The database contains mappings from phone numbers to domains, and is populated by each of the participating domains, often

manually. Each domain implements an agreed-upon query interface that can be used to access the database when a number is called. Sometimes ENUM is used for this interface (called private ENUM), other times, a SIP redirection is used. Some federations also utilize private IP networks in order to address QoS problems. "SIP trunking" - a service being offered by many telecom operators as a SIP-based PRI replacement - is a form of private federation.

Private federations work, but they have one major limitation: scale. As the number of participating domains grows, several problems arise. Firstly, the size of the databases become unruly. Secondly, the correctness of the database becomes an issue, since the odds of misconfigured numbers (either intentionally or accidentally) increases. As the membership grows further, the odds increase that "bad" domains will be let in, introducing a source of spam and further problems. The owner of the federation can - and often does - assume responsibility for this, and can attempt to identify and shut down misbehaving participants. Indeed, as the size of the federations grow, the owner of the federation needs to spend increasing levels of capital on maintaining it. This, in turn, requires them to charge money for membership, and this can be a barrier to entry.

4. Key Requirements

From the discussion on the problems of inter-domain federation and the solutions that have been attempted so far, several key requirements emerge:

REQ-1: The solution should allow for federation between any number of domains.
REQ-2: The solution must enable users in one domain to identify users in another domain through the use of their existing E.164 based phone numbers.
REQ-3: The solution must work with deployments that utilize any kind of endpoint, including non-IP phones connected through gateways, IP softphones and hardphones.
REQ-4: The solution should not require any change in user behavior. The devices and techniques that users have been using previously to make inter-domain calls should continue to work, but now result in inter-domain IP federation.
REQ-5: The solution should work worldwide, for any domain anywhere.
REQ-6: The solution should not require any new services from any kind of centralized provider. A domain should be able, of its own free-will and accord, to deploy equipment and connect to the federation.

REQ-7: The solution should not require any prior arrangement between domains in order to facilitate federation between those domains. Federation must occur opportunistically - connections established when they can be.
REQ-8: The solution must work for domains of any size - starting at a single phone to the largest telecom operator with tens of millions of numbers.
REQ-9: The solution must have built-in mechanisms for preventing spam and DoS attacks. This mechanism must be fully automated.
REQ-10: The solution must not require any processing whatsoever by SIP or RTP intermediaries. It must be possible for a direct SIP connection to be established between participating domains.

These requirements, when put together, appear to be mutually unsolvable. And indeed, they have been - until now.

5. Executive Overview

Verification Involving PSTN Reachability (ViPR) is a new technology that is aimed at solving the problems that have prevented large-scale Internet-based SIP federation of voice and video. ViPR solves these problems by creating a hybrid of three technologies - the PSTN itself, a Peer to Peer (P2P) network, and SIP. By combining all three, ViPR enables an incrementally deployable solution to federation.

5.1. Key Properties

ViPR has several important properties that enable it to solve the federation problem:

Works With Numbers: ViPR enables federation for existing PSTN numbers. It does not require users or administrators to know or configure email-style identifiers. It does not require the allocation of new numbers. It does not require a change in user behaviors. Whatever way users were dialing numbers yesterday, works with ViPR tomorrow.
Works with Existing Endpoints: ViPR does not require any changes to endpoints. Consequently, it works with existing SIP endpoints, or with non-IP endpoints connected through gateways.
Fully Distributed: ViPR does not require any kind of central authority or provider. A domain wishing to utilize ViPR just deploys it on their own. ViPR utilizes the existing PSTN and existing Internet connectivity the domain already has, and by combining them, achieves inter-domain federation. Domains do not need to wait for their service providers to roll out any kind of new features, databases, or functionality.

Verified Mappings: The biggest issue in mapping from a phone number to a domain or IP address, is determining whether the mapping is correct. Does that domain really own the given phone number? While solutions like ENUM have solved this problem by relying on centralized delegations of authorization, ViPR provides a secure mapping in a fully distributed way. ViPR guarantees that phone calls cannot be misrouted or numbers stolen.

Worldwide: ViPR works worldwide. Any domain that is connected to both the PSTN and the Internet can participate. It doesn't matter whether the domain is in Africa, the Americas, or Australia. Since ViPR does not depend on availability of any regional services beyond IP and PSTN access - both of which are already available globally - ViPR itself is globally available.

Unlimited Scale: ViPR has nearly infinite scale. Any number of domains can participate.

Self-Scale: ViPR self-scales. This means that the amount of computation, memory, and bandwidth that a domain must deploy scales in direct proportion to the size of their own user base.

Self-Learning: ViPR is completely automated. A domain never, ever has to configure any information about another domain. It never has to provision IP addresses, domain names, certificates, phone number prefixes or routing rules. Without any prior coordination, ViPR enables one domain to connect to a different domain.

Automated Anti-Spam ViPR comes with a built-in mechanism for preventing VoIP spam. This mechanism is new, and specific to VoIP. In this way, it is fundamentally different from existing VoIP anti-spam techniques which borrow from email [RFC5039]. This new technique is fully automated, and requires no configuration by administrators and no participation from end users. Though it is not a 100% solution to the problem, it brings substantial economic and legal ammunition to the table to act as a good deterrent for a long while.

Feature Velocity: ViPR enables direct SIP connections between two domains seeking to federate. There are no SIP intermediaries of any sort between the two. This means that domains have no dependencies on intermediaries for deployment of new features.

Designed for the Modern Internet: ViPR is built to run on the modern Internet. It assumes the worst from everyone. It assumes limited connectivity. It assumes network failures. It assumes there are attackers seeking to eavesdrop calls. Security is built-in and cannot be disabled.

Reliable: ViPR is reliable. Through its hybridization of the PSTN and the Internet, it makes sure that calls always go through. Indeed, to route a call between domains A and B, ViPR never depends on a server or service anywhere outside of domains A and B (besides vanilla PSTN and IP access) being operational.

At first glance, these properties seem impossible to realize. And

indeed, given the assumptions that have traditionally been made about how federation has to work, these properties are impossible to realize. It is only by stepping back, and rethinking these fundamental assumptions, that a solution can be found.

5.2. Challenging Past Assumptions

Two unstated assumptions of SIP federation are challenged by ViPR.

The first assumption that federation solutions have made is this:

> The purpose of SIP federation is to eliminate the PSTN, and consequently, we cannot assume the PSTN itself as part of the solution.

Though unstated, this assumption has clearly been part of the design of existing solutions. SIP federation based on email-style URIs, as defined in RFC 3261, doesn't utilize or make mention of the PSTN. Solutions like ENUM, or private registries, do not utilize or make mention of the PSTN. In one sense, it's obvious that they shouldn't - after all, the purpose is to replace the PSTN. However, such an approach ignores an incremental solution - a solution which utilizes the PSTN itself to solve the hard problems in SIP federation.

After all, the PSTN has accomplished a great deal. It reaches worldwide. It provides a global numbering translation service that maps phone numbers to circuits. It is highly reliable, and provides QoS. It has been built up over decades to achieve these goals. This begs the question - can we build upon the capabilities already provided by the PSTN, and use them to solve the problems that plague SIP federation?

Indeed, the answer is yes once another assumption is challenged. This second assumption is:

> A federation solution must be the same as the final target federation architecture, and not just a step towards it.

Though unstated, this assumption has also been true. SIP's email-style federation was a pure 'target architecture' - the place we want to get to. ENUM was the same - a worldwide global DNS database with everyone's phone numbers - an unrealizable nirvana of open connectivity.

Historically, technologies are more successful when they are incrementally deployable. Indeed, in many cases, the target architecture is unrealizable because there is no obvious way to get there. As such, the focus needs to be on the next incremental step that we can take, and that step in turn creates the technological and market pressures that will drive the next step. In the end, the target may not be the perfect nirvana we all imagined, but we've at least arrived.

As such, ViPR is very much focused on incremental deployability. It is not the end of the federation story, it is the beginning. It discards the nirvana of perfect IP federation for a solution that federates most, but not all calls, by relying on the PSTN to fill in the gaps. ViPR's philosophy is not to let the perfect be the enemy of the good.

5.3. Technical Overview

A high level view of the architecture is shown in Figure 1. The figure shows four different domains, a.com, b.com, c.com and d.com, federating using ViPR technology. Each domain is connected to both the public Internet and to the traditional PSTN.

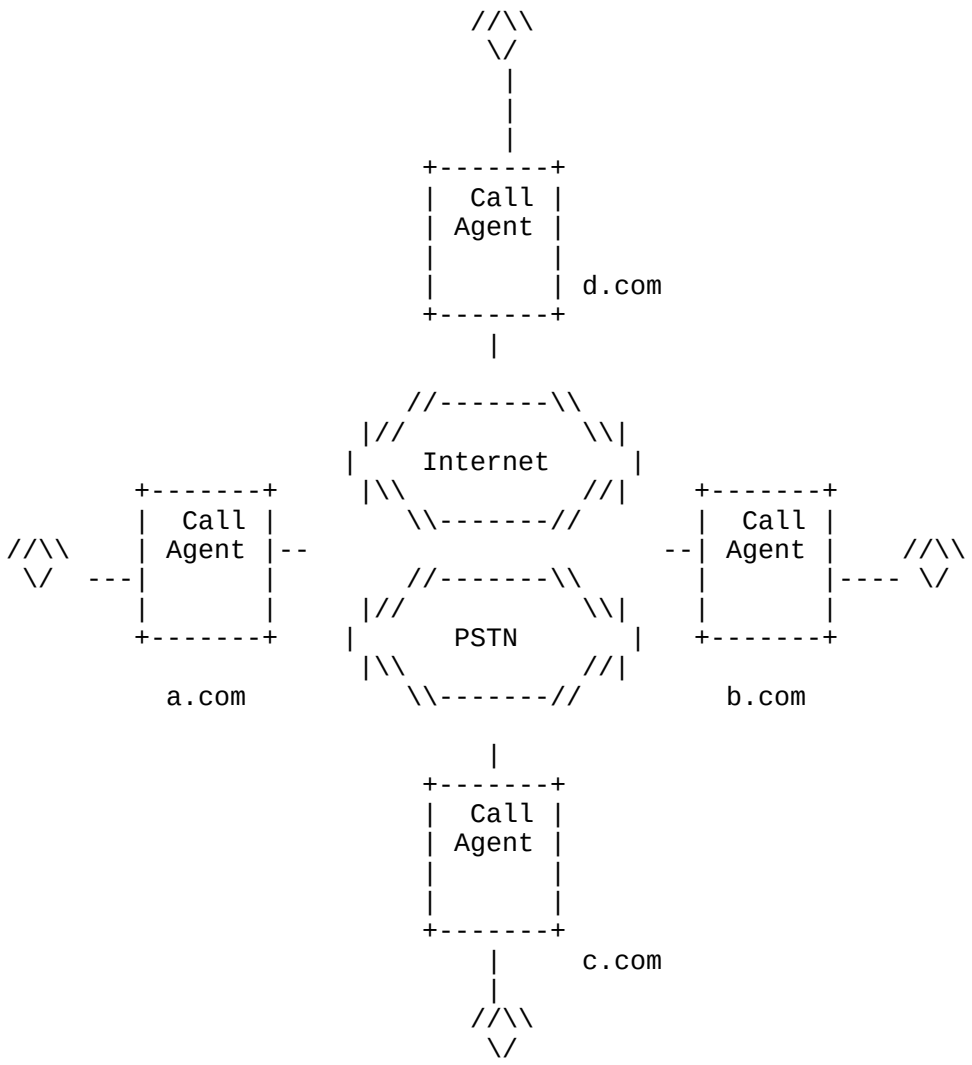


Figure 1: High Level Architecture

For purposes of explanation, it is easiest to think of each domain as having a single call agent which participates in the federation solution. In actuality, the functionality is decomposed into several sub-components, and this is discussed in more detail below. The call agent is connected to one or more phones in the domain, and is responsible for routing calls, handling features, and processing call state. The call agent is stateful, and is aware of when calls start and stop.

Assume that all four domains have a 'fresh' installation of ViPR, and that domain b.com 'owns' +1 408 555 5..., a block of 1000 numbers allocated by its PSTN provider.

The ViPR mechanism can be broken into four basic steps: storage of phone numbers, PSTN first call, validation and caching, and SIP call.

5.3.1. Storage of Phone Numbers

The first step is that the call agents form a single, worldwide P2P network, using RELOAD [P2PSIP-BASE] with the Chord algorithm. This P2P network forms a distributed hash table (DHT) running amongst all participating domains. A distributed hash table is like a simple database, allowing storage of key-value pairs, and lookup of objects by key. Unlike a normal hash table, which resides in the memory of a single computer, a distributed hash table is spread across all of the servers which make up the P2P network. In this case, it is spread across all of the domains participating in the ViPR federation.

The neat trick solved by Chord (and by other DHT algorithms), is an answer to the following: given that the desired operation is to read or write an object with key K, which node in the DHT is the box that currently stores the object with that key? Chord provides a clever algorithm which routes read and write operations through nodes in the DHT until they eventually arrive at the right place. With Chord, this will take no more than log2N hops, where N is the number of nodes in the DHT. Consequently, for a DHT with 1024 nodes, 10 hops are required in the worst case. For 2048, 11 hops. And so on. The logarithmic factor allows DHTs to achieve incredible scale and to provide enormous storage summed across all of the nodes that make up the DHT.

This logarithmic hopping behavior also means that each node in the DHT does not need to establish a TCP/TLS connection to every other node. Rather, connections are established to a smaller subset - just log(N) of the nodes.

In DHTs, each participating entity is identified by a Node-ID. The Node-ID is a 128 bit number, assigned randomly to each entity. They have no inherent semantic meaning; they are not like domain names or IP addresses.

In the case of ViPR, each call agent is identified by one or more Node-IDs. For purposes of discussion, consider the case where the call agent has just one. Each participating domain, including b.com in our example, uses the DHT to store a mapping from each phone number that it owns, to its own Node-ID. In the case of b.com, it would store 1000 entries into the DHT, each one being a mapping from

one of its phone numbers, to its own Node-ID. Furthermore, when the mappings are stored, the mapping is actually from the SHA-1 hash of the phone number, to the Node-ID of the call agent which claims ownership of that number.

Pretending that the Node-ID of the call agent in domain b.com is 0x1234 (a shorter 16 bit value to simplify discussion), the entries stored into the DHT by b.com would be:

Key	Value
SHA1(+14085555000)	0x1234
SHA1(+14085555001)	0x1234
SHA1(+14085555002)	0x1234
.....	
SHA1(+14085555999)	0x1234

Figure 2: DHT Contents

It is important to note that the DHT does not contain phone numbers (it contains hashes of them), nor does it contain IP addresses or domain names. Instead, it is a mapping from the hash of a phone number (in E.164 format) to a Node-ID.

b.com will store this mapping when it starts up, or when a new number is provisioned. The information is refreshed periodically by b.com. The actual server on which these mappings are stored depends on the Chord algorithm. Typically, the entries will be uniformly distributed amongst all of the call agents participating in the network.

5.3.2. PSTN First Call

At some point, a user (Alice) in a.com makes a call to +1 408 555 5432, which is her colleague Bob. Even though both sides have ViPR, the call takes place over the plain old PSTN. Alice talks to Bob for a bit, and they hang up.

At a random point of time after the call has completed, the call agent in a.com "wakes up" and says to itself, "that’s interesting, someone in my domain called +1 408 555 5432, and it went over the PSTN. I wonder if that number is reachable over IP instead?". To make this determination, it hashes the called phone number, and looks it up in the DHT. It is important to note that this lookup is not at the time of an actual phone call - this lookup process happens outside of any phone call, and is a background process.

The query for +1 408 555 5432 will traverse the DHT, and eventually arrive at the node that is responsible for storing the mapping for that number. Typically, that node will not be b.com, but rather one of the other nodes in the network (for example. c.com). In many cases, the called number will not find a matching mapping in the DHT. This happens when the number that was dialed is not owned by a domain participating in ViPR. When that happens, a.com takes no further action. Next time there is another call to the same number, it will repeat the process and check once more whether the dialed number is in the DHT.

In this case, there is a match in the DHT, and a.com learns the Node-ID of b.com. It then proceeds to the validation step. It is also possible that there are multiple matches in the DHT. This can happen if another domain - d.com for example - also claims ownership of that number. When there are multiple matching results, a.com learns all of them, and performs the validation step with each.

5.3.3. Validation and Caching

Why not just store the domain in the DHT, instead of the Node-ID? In that case, once a.com performed the lookup, it would immediately learn that the number maps to b.com, and could then make a direct SIP call next time.

The main reason this doesn’t work is security. The information in the DHT is completely untrusted. There is nothing so far that enables a.com to know that b.com does, in fact, own the phone number in question. Indeed, if multiple domains make a claim on the number, it has no way to know which one (if any) actually owns it.

To address this critical problem, ViPR utilizes a technique called phone number validation. Phone number validation is the key concept in ViPR. The essential idea is that a.com will connect to the b.com server, by asking the DHT to form a connection to b.com’s Node-ID. Once connected, a.com demands proof of ownership of the phone number. This proof comes in the form of demonstrated knowledge of the previous PSTN call. When a call was placed from a.com to +1 408 555 5432, the details of that call - including its caller ID, start time, and stop time, create a form of shared secret - information that is only known to entities that participated in the call. Thus, to obtain proof that b.com really owns the number in question, a.com will demand a knowledge proof - that b.com is aware of the details of the call. The only way that b.com could know these details is if it had received the call, and the only way it could have received the call is if it owned the phone number.

There are a great many details required for this validation protocol

to be secured. It needs to handle the fact that call start and stop times won't exactly match on both sides. It needs to deal with the fact that many calls start on the top of the hour. It needs to deal with the fact that caller ID is not often delivered, and when it is delivered, is not reliable. It needs to deal with the fact that a.com may in fact be the attacker, trying to use the validation protocol to extract the shared secret from b.com. All of this is, in fact, handled by the protocol. The protocol is based on the Secure Remote Password for TLS Authentication (SRP-TLS) [RFC5054], and is described more fully in [VIPR-PVP].

At the end of the validation process, both a.com and b.com have been able to ascertain that the other side did in fact participate in the previous PSTN call. At that point, a.com sends its domain name to b.com (this is described in more detail below), and b.com sends to a.com - all over a secured channel - a SIP URL to use for routing calls to this number, and a ticket. The ticket is a cryptographic object, opaque to a.com, but used by b.com to allow incoming SIP calls. It is similar in concept to kerberos tickets - it is a grant of access. In this case, it is a grant of access for a.com to call +1 408 555 5432, and only +1 408 555 5432.

The a.com call agent receives the SIP URI and ticket, and stores both of them in an internal cache. This cache builds up slowly over time, containing the phone number, SIP URI, and ticket, for those numbers which are called by a.com and validated using ViPR. Because the cache entries are only built for numbers which have actually been called by users in the enterprise, the size of the cache self-scales. A call agent supporting only ten users will build up a cache proportional to the volume of numbers called by ten people, whereas a call agent supporting ten thousand users will build up a cache which is typically a thousand times larger.

5.3.4. SIP Call

At some point in the future, another call is made to +1 408 555 5432. The caller could be Alice, or it could be any other user attached to the same call agent. This time, the call agent notes that it has a cached route for the number in question, along with a SIP URI that can be used to reach that route. It also has a ticket.

The a.com call agent attempts to contact the SIP URI by establishing a TCP/TLS connection to the SIP URI it learned. If this connection cannot be made, it proceeds with the call over the PSTN. This ensures that, in the event of an Internet failure or server failure, the call can still proceed. Assuming the connection is established, the a.com call agent sends a traditional SIP INVITE to the terminating call agent, over this newly formed secure connection.

The SIP call setup request also contains the ticket, placed into a new SIP header in the message.

When this call setup request arrives at the b.com call agent, it extracts the ticket from the new SIP header. This ticket is an object, opaque to a.com, that was previously generated by the b.com call agent. Figure 3 illustrates how this ticket is generated and used.

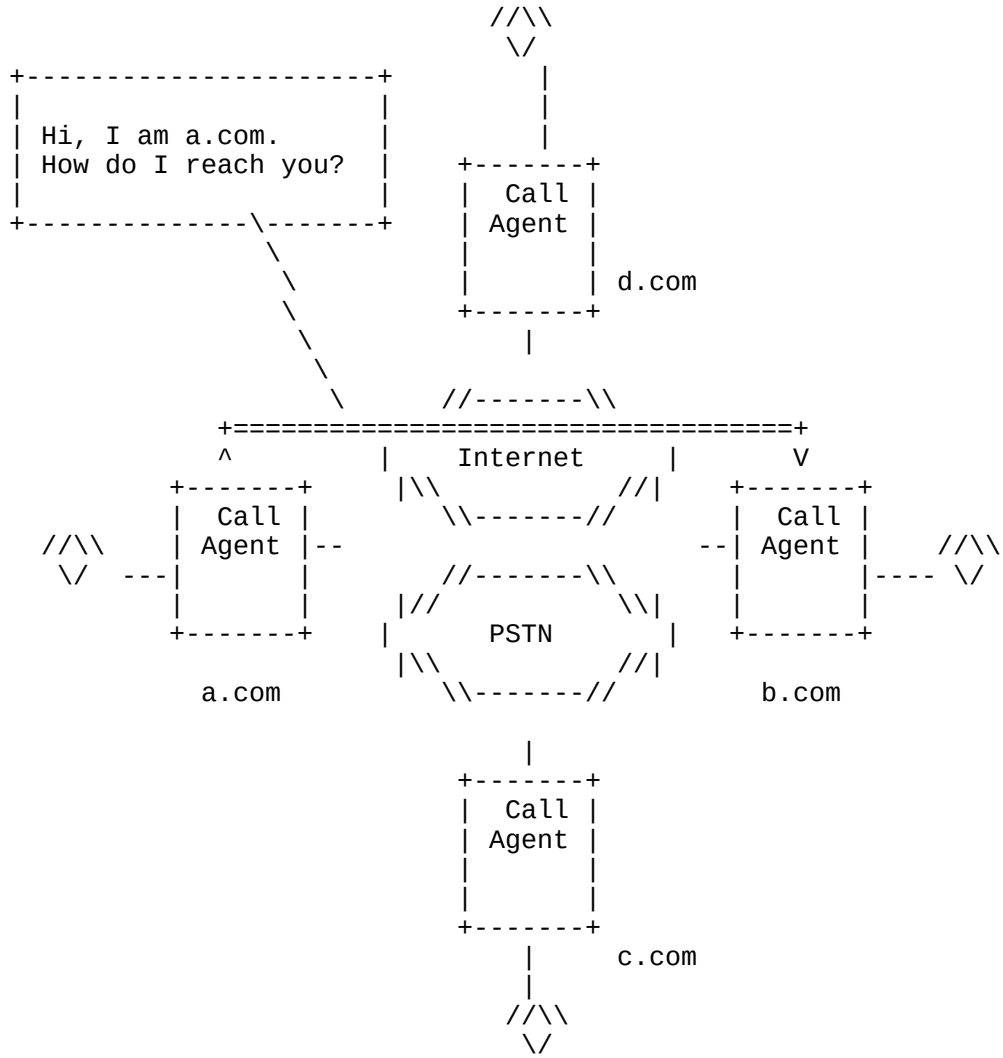


Figure 3: Ticket Validation Step 1

Towards the end of the validation process, domains a.com and b.com had determined that each was, in fact in possession of the shared secret information about the prior PSTN call. However, neither side has any information about the domain names of the other side. The originating domain - a.com - tells b.com that its domain name is a.com. It offers no proof of this assertion at this time.

Next, the b.com domain generates the ticket. The ticket has three fundamental parts to it:

1. The phone number that was just validated - in this case, +1 408 555 5432.
2. The domain name that the originating side claims it has - a.com in this case.
3. A signature generated by b.com, using a key known to itself only, over the other two pieces of information.

This ticket is then sent back to a.com at the end of the validation process, as shown in Figure 4.

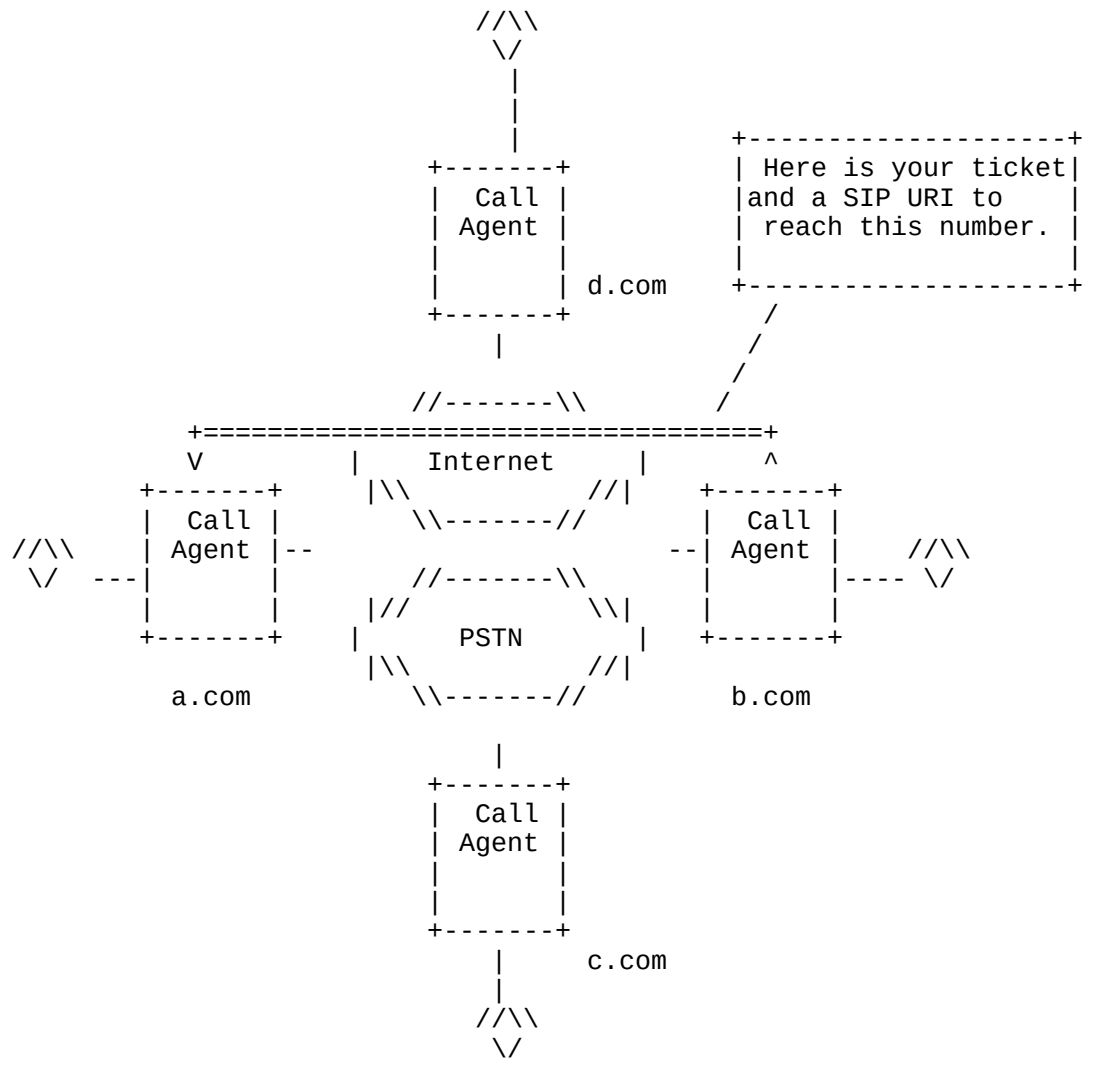


Figure 4: Ticket Validation Step 2

When a.com generates a SIP INVITE, it will contain this ticket. The INVITE arrives at the b.com call agent over the mutually authenticated TLS connection established between the domains.

The b.com call agent looks for the SIP header field in the INVITE that contains the ticket. First, it verifies the signature over the ticket. Remember that the b.com agent is the one that generated the ticket in the first place; as such, it is in possession of the key

required to validate the signature. Once validated, it performs two checks:

1. It compares the phone number in the call setup request (the Request URI) against the phone number stored in the ticket.
2. It compares the domain name of the calling domain, learned from the certificates in the mutual TLS exchange, against the domain name stored in the ticket.

If both match, the b.com call agent knows that the calling party is in fact the domain they claimed previously, and that they had in fact gone through the validation process successfully for the number in question. A consequence of this is that the following property is maintained:

> A domain can only call a specific number over SIP, if it had previously called that exact same number over the PSTN.

This property is key in fighting spam and denial-of-service attacks. Because calling numbers on the PSTN costs money - especially international calls - ViPR creates a financial disincentive for spammers. For a spammer to ring every phone in a domain with a SIP call, it must have previously called every number in the domain with a PSTN call, and had a successfully completed call to each and every one of them. Of course, once that PSTN call had been placed, the spammer would have already achieved their goals, and at cost. The additional VoIP call is not so exciting.

This property also means that, in order for an attacker to spam call numbers on VoIP, it must have already spam-called those same numbers on the PSTN. This means that the attacker would clearly be subject to regulations and laws governing usage of the PSTN for calling. As an example, a spammer in the United States would have already violated U.S. do-not-call rules by initiating the spam calls to the PSTN numbers.

It is important to note that ViPR does not completely address the spam problem. A large spamming clearing house organization could actually incur the costs of launching the PSTN calls to numbers, and then, in turn, act as a conduit allowing other spammers to launch their calls to those numbers for a fee. The clearinghouse would actually need to transit the signaling traffic (or, divulge the private keys to their domain name), which would incur some cost. As such, while this is not an impossible situation, the barrier is set reasonably high to start with - high enough that it is likely to deter spammers until it becomes a highly attractive target, at which point other mechanisms can be brought to bear. This is, again, an example of the incremental deployability philosophy that ViPR takes -

let not the perfect be the enemy of the good.

6. Architecture Components and Functions

The architecture in Figure 1 is overly simplistic. ViPR allows the functionality embedded within the call agent can be split up into three components, as shown in Figure 5:

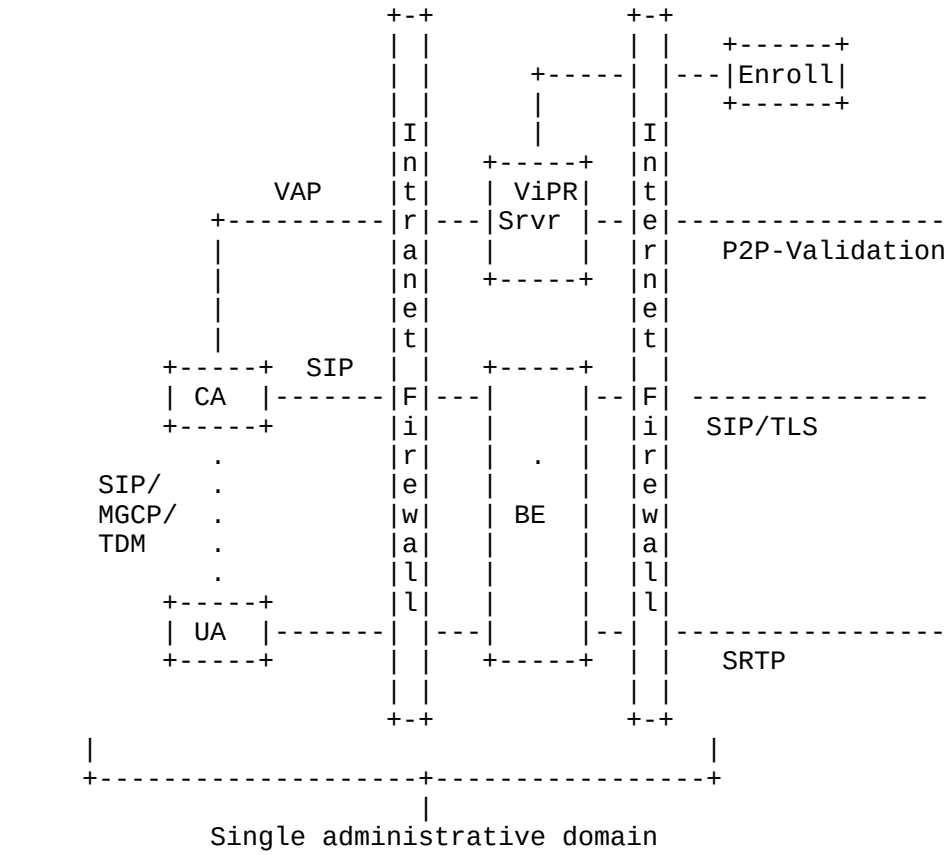


Figure 5: Architecture

Within each domain, there are three components that are ViPR-aware. These are the ViPR server, the call agent (CA), and the border element (BE). Outside of the domain, there is a P2P network and an enrollment server. A domain will typically have firewalls - an Internet firewall and an intranet firewall.

The sections which follow describe the roles and responsibilities of

each component in more detail.

6.1. ViPR Server

The ViPR server is the heart of the system. It performs several key functions:

1. It implements the P2P protocol, acting as one or more nodes in the DHT. By placing this function separate from the call agent, it allows the call agent to be isolated from the traffic and security concerns that are often associated with a P2P network.
2. It implements the validation mechanism. It is informed of call events by the call agent, and sometime after the call, looks up the number in the DHT, and if found, attempts to connect to the node claiming ownership of the number, and then validates it.
3. It pushes newly learned routes to the call agent once validation has occurred. The ViPR server does not hold the call routes; this eliminates the need for an off-box query to perform call routing logic.
4. It stores numbers into the DHT. The call agent informs the ViPR servers of numbers to be published, and the ViPR server places them into the P2P network. Refreshing the stored numbers (by asking the ViPR server to restore them) is the responsibility of the call agent.
5. It implements a distributed quota enforcement algorithm, ensuring that malicious ViPR servers cannot store excessive data into the network.
6. It implements a policing function, pacing its store and fetch requests into the DHT to ensure that the network is not overwhelmed.

In order to join the P2P network and be able to receive incoming validation requests, the ViPR server must have open access to the public Internet. For this reason, it is typically placed into the DMZ. The Internet firewall will require two pinholes to be opened towards the ViPR server: one for the P2P protocol, and one for the validation protocol.

It is important to understand that the ViPR server does not perform any call processing. It does not process SIP or RTP traffic. It is a non-real-time server that performs validation processing in the background, outside of actual call attempts.

The ViPR server needs to connect with the call agent. This is done through the ViPR Access Protocol (VAP). VAP is described in more detail below.

6.2. Call Agent

The call agent is a box within the domain which performs call processing on behalf of one or more phones within the domain. ViPR can work with a wide variety of call agents, as long as they meet some specific criteria:

- The call agent must be know of the start time, stop time, caller ID, and called numbers of calls placed from phones towards the PSTN.
- The call agent must be capable of making routing decisions for outbound calls from phones that would otherwise go to the PSTN, directing them towards the PSTN or towards other domains (based on ViPR routing rules).

Based on this definition, many different types of products typically found within a domain could act as the call agent. An IP PBX or TDM PBX with a SIP interface can be the call agent. A Session Border Controller (SBC) that connects calls from a PBX to the PSTN, can act as the call agent. An IMS application server can act as the call agent. A PSTN gateway, used for all calls egressing a domain from a set of phones, can act as a call agent.

A SIP proxy can act as a call agent; as long as it is capable of stashing the relevant call information into Record-Route headers for usage at the end of the call, it can even operate without retaining call state.

A single phone can also act as the call agent, representing itself and its own phone number.

In ViPR, the call agent performs several key functions specific to ViPR:

- It informs the ViPR server of the phone numbers to be stored in the DHT for its domain.
- It refreshes those numbers in the DHT, redoing the storage operation periodically.
- At the end of a call, the call agent sends a ViPR Call Record (VCR) to the ViPR server, containing the start time, stop time, caller ID and called party number.
- It learns validated routes from the ViPR server. These routes consist of a phone number, a SIP URI to utilize when contacting that phone number, and a corresponding ticket. The call agent is responsible for storing those routes.
- When a call is to be made towards a PSTN number, the call agent is responsible for checking whether there is a route for that number, learned via a prior notification from the ViPR server. If so, it

is responsible for sending the INVITE towards the learned SIP URI, and for including the ticket the X-Cisco-ViPR-Ticket header field.

Those functions which require communications with the ViPR server are done by implementing VAP. VAP is a client-server protocol, with the call agent acting as the client, and the ViPR server acting as the server. For this reason, the call agent is sometimes called the VAP client or ViPR client.

6.3. Border Element

The border element is responsible for the SIP layer perimeter security functions. In particular:

- The border element ensures that all egress SIP traffic is carried over TLS. Border elements must reject any incoming SIP requests which are not over TLS. SIP over TLS is mandatory-to-use in ViPR, and it must be performed using mutual TLS.
- The border element ensures that all egress RTP traffic is actually carried using SRTP. If the traffic originated by the UA in the domain is inherently SRTP, the criteria is met. However, many domains do not utilize SRTP internally, and if it is not used internally, the border element must convert to SRTP. Similarly, the border element is responsible for rejecting any incoming SIP calls that are not set up with SRTP. SRTP is mandatory in ViPR.
- The border element ensures that ingress and egress SIP traffic is 'fixed up' so that it can pass through the Internet firewall successfully. Typically, this is done using a traditional SBC/ALG function.
- The border element inspects all incoming SIP INVITEs, and performs ticket verification. In this process, it looks for the X-Cisco-ViPR-Ticket header field in the INVITE. If not present, it discards the request. If present, it verifies the signature, and then compares the called number and remote TLS domain against the contents of the ticket. If they do not match, the border element discards the INVITE.

The border element can perform other, non-ViPR tasks, as is common for border elements. These include header inspection and validation, anti-virus checks on embedded content, SIP state machine conformance, policy checks on various services, and so on.

The role of the border element can be fulfilled by any number of products typically found within domains. These include Session Border Controllers and firewalls. Indeed, the border element function can be embedded directly in the Internet firewall.

The border element is connected to the call agent via SIP, and to the

user agent (UA) via RTP. The border element has no direct connection to the ViPR server. However, in order for ticket processing to work in this model, the ViPR server and border element must share a secret that is used to create the tickets. This is discussed in more detail below.

6.4. Enrollment Server

P2P protocols - including RELOAD - require the usage of an enrollment server in order to obtain the certificates that are used to secure the network. ViPR uses, and indeed requires, that all RELOAD traffic be over TCP/TLS with mutual authentication. The certificates used are obtained through an enrollment process. The details on how P2P enrollment are done are beyond the scope of this document.

6.5. P2P Network

The collection of ViPR servers form a single, worldwide, P2P network utilizing RELOAD and the Chord algorithm.

It is very important to understand that the DHT is never accessed in real-time. It is not queried at call setup time. This is because the DHT is slow, involving many hops. Queries could take seconds. Furthermore, we don’t want to rely on proper operation of the DHT to actually make calls.

7. Protocols

The overall ViPR solution utilizes several protocols, each performing a different function.

7.1. P2P: RELOAD

ViPR utilizes the RELOAD protocol [P2PSIP-BASE] to run amongst each of the ViPR servers. Each ViPR server acts as one or more nodes in the DHT. The number of nodes that the ViPR server implements directly determines the quota allocated to that ViPR server, and in turn, the amount of work it must perform storing data.

ViPR, however, does not implement the SIP usage that has been defined for RELOAD [P2PSIP-SIP]. That is because the DHT is not used as a traditional distributed registrar. Instead, it implements a new usage - the ViPR usage - which stores phone numbers. It also utilizes the DHT for storage of certificates, using a certificate usage.

7.1.1. ViPR Usage

The ViPR usage is described in detail in [VIPR-RELOAD-USAGE]. This section provides a brief overview.

The ViPR usage makes use of the dictionary type. Each resource-ID is a key, computed by taking the SHA1 hash of an E.164 formatted phone number. The value stored at this resource-ID is a dictionary. The dictionary entries are the set of virtual ViPR servers which claim ownership of those numbers.

Since a ViPR server might support a multiplicity of call agents from different domains, it is necessary to logically segment a ViPR server so that - from a security perspective - it operates logically like different virtual ViPR servers, one for each call agent. Each virtual instance of a ViPR server is called a VService. Thus, the entries in the dictionary are key value pairs whose key is the concatenation of the Node-ID and an identifier for the VService within that node. The value at each key is the Node-ID to contact for validation.

When a node in the DHT receives a Store request, and it is the responsible node for the resource-ID, it will verify that the Node-ID in both the key and value of the dictionary entry match the Node-ID in the certificate it presents. This ensures that one ViPR server can never overwrite data from another ViPR server.

The ViPR usage also specifies a quota mechanism. Unlike the SIP usage, where there are very specific rules about what resource-IDs a node may store into the DHT, with ViPR, there is no way to restrict what resource-IDs may be stored by a ViPR server. This is because, in ViPR, the resource-IDs are derived from phone numbers, and at the time of storage, there is no way to know whether the node performing the store actually owns this phone number. Consequently, a responsible node will accept stores from any node for any resource-ID. However, to limit malicious users from consuming all of the resources of the DHT, the ViPR usage imposes a quota on storage. Each node performing a store is allocated a fixed quota on the number of records it can place into the DHT. A probabilistic enforcement model is utilized at each responsible node based on the fraction of the hashspace owned by that responsible node. Roughly speaking, if the system quota is 10,000 phone numbers per Node-ID, if a responsible node owns 10% of the DHT, it will accept an average of 1000 phone numbers from any one single Node-ID.

7.1.2. Certificate Usage

Further details pending.

7.2. ViPR Access Protocol (VAP)

The ViPR Access Protocol (VAP) is documented in [VIPR-VAP].

VAP is a client-server protocol that runs between the call agent and the ViPR server. VAP is a simple, binary based, request/response protocol. It utilizes the same syntactic structure and transaction state machinery as STUN [RFC5389], but otherwise is totally distinct from it. VAP clients initiate TCP/TLS connections towards the ViPR server. The ViPR server never opens connections towards the call agent. This allows the ViPR servers to run on the public side of NATs and firewalls.

Once the connections are established, the call agent sends a Register message to the ViPR server. This register message primarily provides authentication and connects the client to the ViPR server. VAP provides several messages for different purposes:

- Publish: The Publish message informs the ViPR server of service information. There are two types of Publishes supported in ViPR. The first is the ViPR Service (VService). This informs the ViPR server of the SIP URIs on the call agent and black and white lists used by the ViPR server to block validations. The ViPR server stores that information locally and uses it during the validation process, as described above. The second Publish is the ViPR number service. The ViPR server, upon receiving this message, performs a Store operation into the DHT.
- UploadVCR: This message comes in two flavors - an originating and terminating message. An originating UploadVCR comes from a call agent upon completion of a non-ViPR call to the PSTN. A terminating UploadVCR comes from an agent upon completion of a call received FROM the PSTN. The ViPR server behavior for both messages is very different. For Originating UploadVCR, the ViPR server will store these, and at a random time later, query the DHT for the called number and attempt validation against the ViPR servers that are found. For a terminating UploadVCR, the ViPR server will store these, awaiting receipt of a validation against them.
- Subscribe: Call agents can subscribe for information from the ViPR server. There is one service that the call agent can subscribe for: number Service. When a new number is validated, the ViPR server will send a Notify to the call agent, containing the validated number, the ticket, and a set of SIP trunk URIs.

- Notify: The ViPR server sends this message to the call agent when it has an event to report for a particular subscription.

The VAP protocol provides authentication by including an integrity object in each message. This integrity message is the hash of the contents of the message and a shared secret between the ViPR server and the client. VAP can also be run over TLS, which enhances security further.

The P2P network introduces rate limits for the purposes of performance management and limiting denial of service attacks. Each node in the DHT comes with it a limit on the amount of stores per second, reads per second, and total amount of data it can store in the DHT. The ViPR server rigorously follows those limits.

As a consequence, when numbers are stored into the DHT, they are written in slowly based on the rate limits. The call agent will send a Publish operation for each individual number. The ViPR server will perform the store in a rate-limited fashion. When the store is complete, the ViPR server responds to the Publish, and the call agent can move to the next DID to publish. Thus, it may take hours or even days to fully store the set of numbers into the DHT. The process then repeats several days later in order to refresh the data in the DHT.

7.3. Validation Protocol

The core of ViPR is the validation protocol. The validation protocol is used by one ViPR server to connect to another, demand proof-of-knowledge of a previous PSTN call, and once proven, securely learn a SIP URI and ticket for usage in future SIP calls between domains.

The validation protocol is documented in [VIPR-PVP].

The validation protocol is built using TLS-SRP [RFC5054]. TLS-SRP creates a secure TLS connection, but instead of using certificates, utilizes a password. TLS-SRP was designed for cases where the passwords are relatively weak. In the case of the validation protocol, the passwords are formed from parameters of a previous PSTN call. Once a secure TLS connection is formed, a simple request/response protocol is run over it. The request contains the domain name of the originating ViPR server, and the response contains the SIP URI and ticket for that number.

The validation protocol properly handles time offsets between the two domains for the start and stop times of the calls, the relatively weak entropy of a single phone call, the grand chessmaster attack, and non-delivery or inaccurate delivery of caller-ID, amongst other

issues. The validation protocol can be tuned by administrators to allow for arbitrary levels of security, measured in terms of equivalent entropy. The equivalent entropy is the number of bits of entropy that must be demonstrated, as if the domains were authenticating each other using a password with that amount of entropy. This gives domains a 'nerd knob' they can turn to trade off security for performance.

Because the validation protocol utilizes TLS-SRP, it does not run directly through the DHT. This is why a ViPR server requires a separate pinhole to be opened for the validation protocol.

7.4. SIP Extensions

The connection between the call agents in different domains is SIP. ViPR requires that the inter-domain connections run over TLS, and furthermore, utilize SRTP keyed with Sdescriptions.

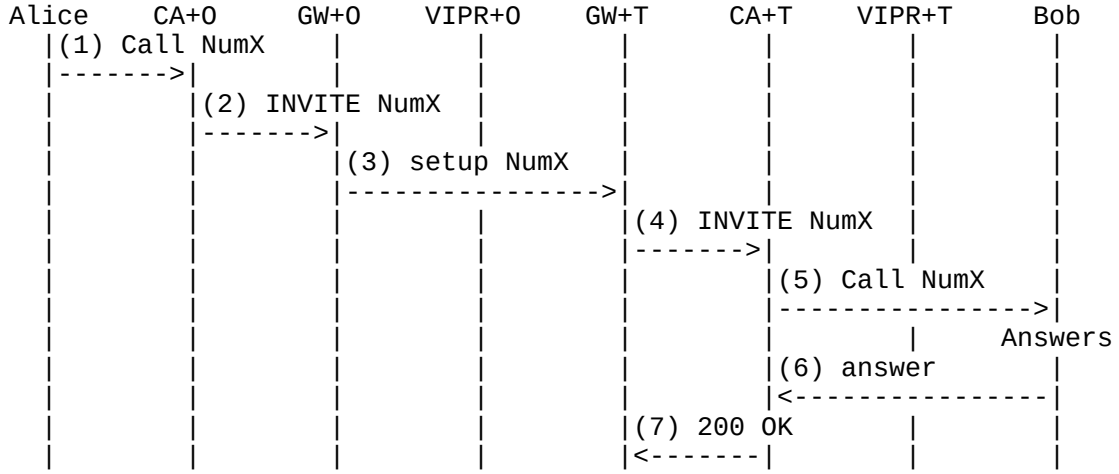
ViPR extends SIP with its anti-spam mechanism. This takes the form of a ticket, present in a SIP header field. [VIPR-SIP-ANTISPAM] defines this header field and the format of the ticket it contains.

8. Example Call Flows

This section provides call flows for the key use cases.

8.1. PSTN Call and VCR Upload

A call flow for the initial PSTN call and VCR upload is shown in Figure 6.



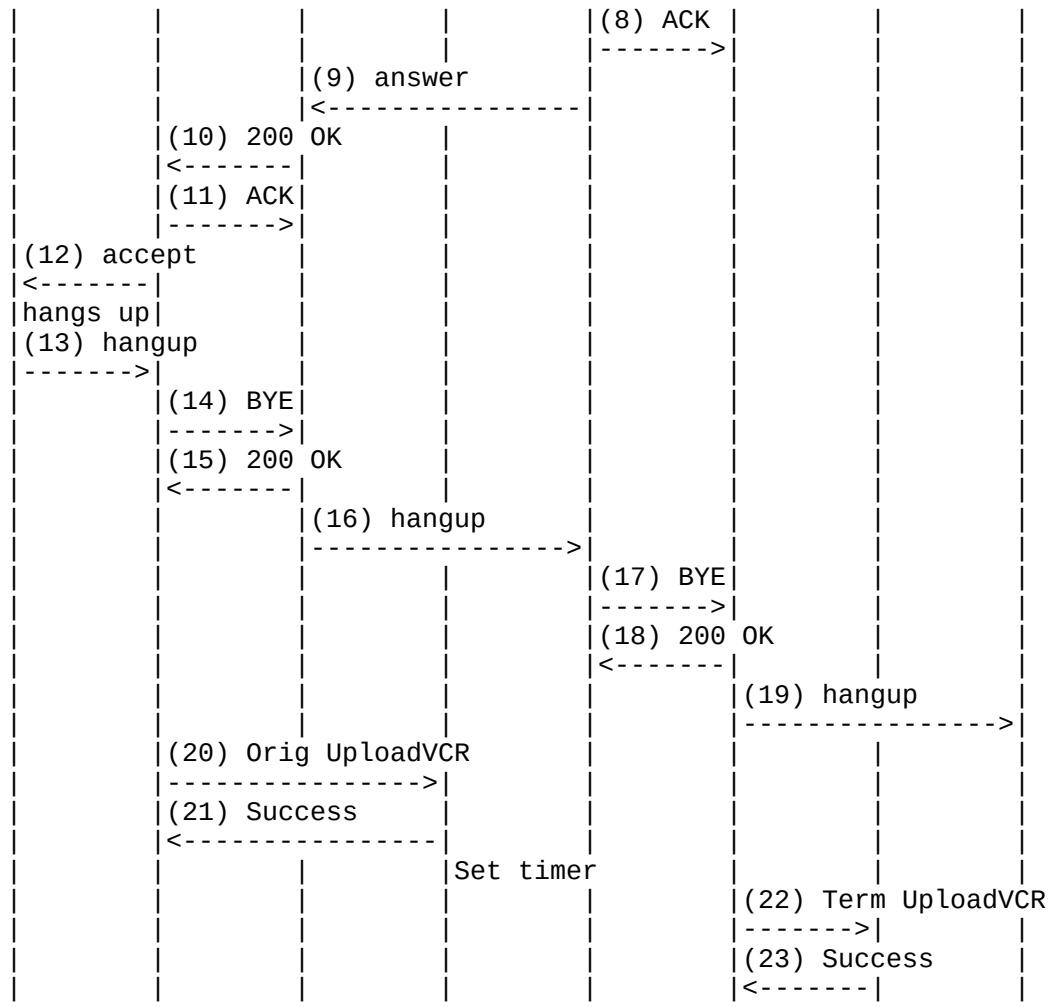


Figure 6: PSTN Call and Upload

In message 1, Alice calls the number of her colleague, Bob. This is NumX. This call is routed over the PSTN, through the terminating call agent, and rings Bob's phone (messages 1-5). Bob answers the phone, and this is propagated back to Alice (messages 6-12). Bob and Alice talk for a while, and then Alice hangs up. This hangup is propagated to Bob, and the call is terminated (messages 13-19).

The originating call agent notes that this call went to the PSTN, and might be a candidate for a future SIP call. It sends an UploadVCR message to its ViPR server (message 20), containing the start time,

stop time, callerID and called party number. The ViPR server acknowledges this (message 21), and then sets a timer for a random time into the future, at which point it will attempt validation. The terminating side is similar; it sends an UploadVCR to its ViPR server (message 22), which is acknowledged (message 23). The terminating side does not set a timer; it waits for a possible validation attempt which may or may not arrive in the future.

8.2. DHT Query and Validation

This section provides the call flow for what happens on the originating ViPR server when the timer fires, in Figure 7.

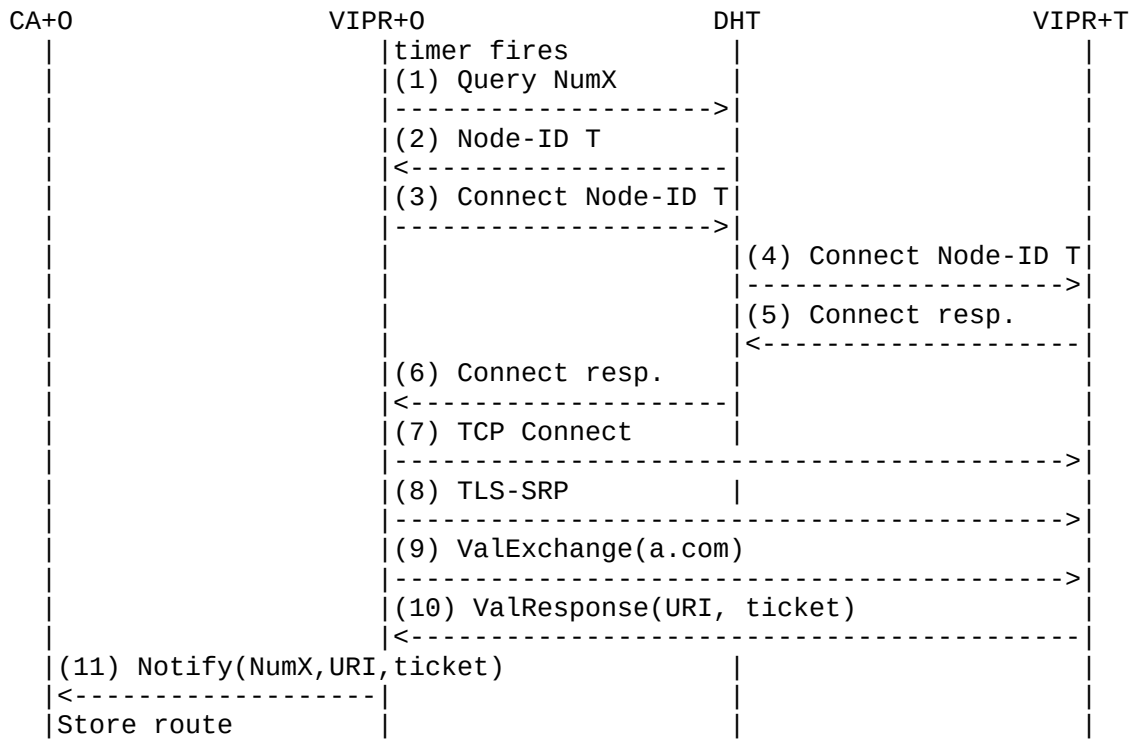


Figure 7: Validation Flow

First, the timer that was set by the originating ViPR server in Figure 6 fires. When it fires, the ViPR server examines the called party number from the VCR. It performs a query into the DHT, to see if this number has been stored by any domain (message 1). In this case, it has, and the DHT returns with a successful query response (message 2). This response indicates that the terminating ViPR

server, with node-ID T, claims ownership of the number.

The originating ViPR server asks the DHT to form a connection between itself and the terminating ViPR server. This message exchanges IP addresses and ports through which a TCP connection can be attempted; details are omitted (messages 3-6). Now, the originating ViPR server can establish a TCP connection to the terminating ViPR server (message 7). Next, the originating ViPR server begins negotiation of a TLS-SRP connection. The TLS-SRP uses the caller ID and called number as a "username" for this exchange, and the start time and stop time of the call as a password. As both sides share the same values for this secret, the secure connection is established. This is now a TLS connection between the two ViPR servers.

Over this secure connection, the originating ViPR server sends a ValExchange request. This request contains the domain name that is claimed by the originating ViPR server (this claim is not verified at this time) (message 9). This is received by the terminating ViPR server, which then creates a ticket for that domain and NumX, and passes the ticket and the SIP URI back to the originating ViPR server (message 10). The originating ViPR server sends this information to its call agent (message 11), which then stores it for usage in a future call.

8.3. DHT Query and No Match

In this case, after the PSTN call of Figure 6, the timer fires, but the originating ViPR server finds no match in the DHT. This is an alternative case to the flow in Figure 7.

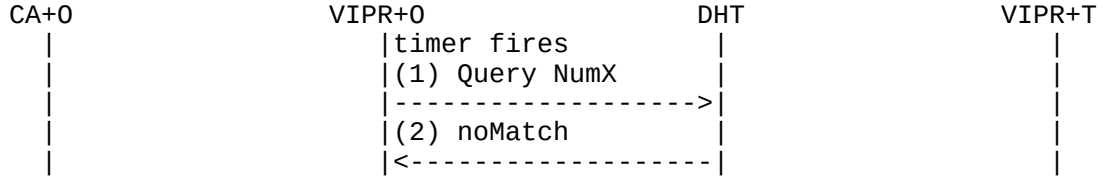


Figure 8: DHT No-Match

8.4. SIP Call

In this case, shown in Figure 9, a user makes a call to a number which has been learned via ViPR.

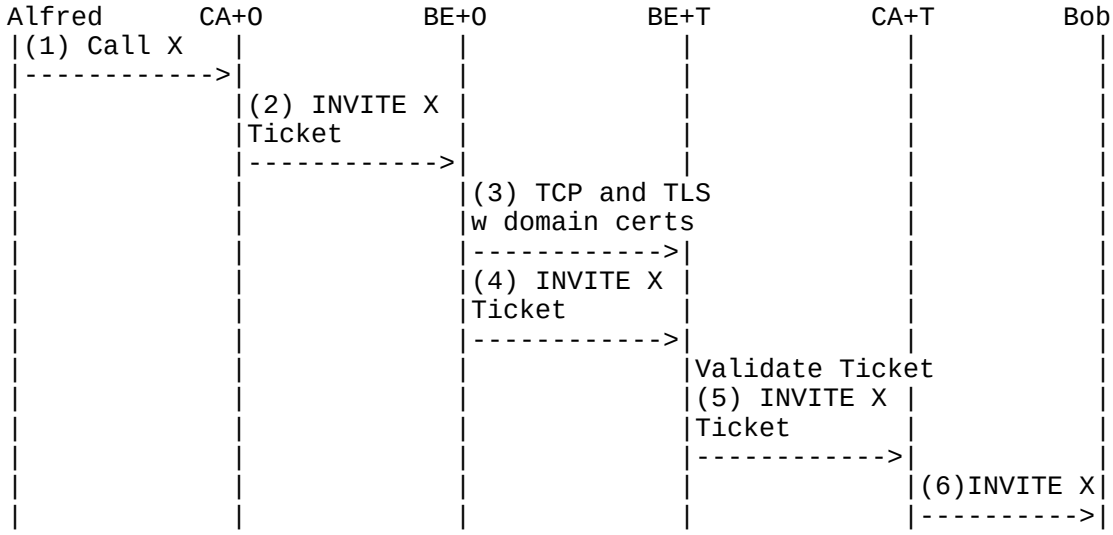


Figure 9: SIP Call

First, a user in the originating domain - Alfred - calls Bob's number (message 1). The originating call agent notes that it has a cached route for that number. It extracts the SIP URI, using it as the topmost Route header field, and then attaches the ticket to the X-Cisco-ViPR-Ticket header field. This INVITE is sent to a default next hop border element (message 2). The border element establishes a TCP/TLS connection with the domain in the Route header. It uses a traditional domain certification for this TLS connection (message 3). Once established, it sends the INVITE over the connection (message 4).

This arrives at the terminating call agent, which extracts the ticket and verifies it. To verify it, it checks the signature using the key that was used to create the ticket. Then, it compares the domain name in the ticket with the domain name from the TLS connection handshake. Finally, it compares the called party number in the Request-URI with the value from the ticket. Assuming they all match, the call is forwarded to the terminating call agent (message 5), where it is finally delivered to Bob (message 6).

9. Security Considerations

Security is incredibly important for ViPR. This section provides an overview of some of the key threats and how they are handled.

9.1. Attacks on the DHT

Attackers could attempt to disrupt service through a variety of attacks on the DHT.

Firstly, it must be noted that the DHT is never used at call setup time. It is accessed as a background task, solely to learn NEW numbers and routes that are not already known. If, by some tragedy, an attacker destroyed the P2P network completely, it would not cause a single call to fail. Furthermore, it would not cause calls to revert to the PSTN - calls to routes learned previously would still go over the IP network. The only impact to such a devastating attack, is that a domain could not learn *new* routes to new numbers, until the DHT is restored to service. This service failure is hard for users and administrators to even notice.

That said, ViPR prevents many of these attacks. The DHT itself is secured using TLS - its usage is mandatory. Quota mechanisms are put into place that prevent an attacker from storing large amounts of data in the DHT. Other attacks are prevented by mechanisms defined by RELOAD itself, and are not ViPR specific.

9.2. Theft of Phone Numbers

The key security threat that ViPR is trying to address is the theft of phone numbers. In particular, a malicious domain could store, into the DHT, phone numbers that it does not own, in an attempt to steal calls targeted to those numbers. This attack is prevented by the core validation mechanism, which performs a proof of knowledge check to verify ownership of numbers.

An attacker could try to claim numbers it doesn’t own, which are claimed legitimately by other domains in the ViPR network. This attack is prevented as well. Each domain storing information into the DHT can never overwrite information stored by another domain. As a consequence, if two domains claim the same number, two records are stored in the DHT. An originating domain will validate against both, and only one will validate - the real owner.

An attacker could actually own a phone number, use it for a while, validate with it, and build up a cache of routes at other domains. Then, it gives back the phone number to the PSTN provider, who allocates it to someone else. However, the attacker still claims ownership of the number, even though they no longer have it. This attack is prevented by expiring the learned routes after a while. Typically, operators do not re-assign a number for a few months, to allow out-of-service messages to be played to people that still have the old number. Thus, the TTL for cached routes is set to match the

duration that carriers typically hold numbers.

An attacker could advertise a lot of numbers, most of which are correct, some of which are not. ViPR prevents this by requiring each number to be validated individually.

An attacker could make a call so they know the call details of the call they made and use this to forge a validation for that call. They could then try to convince other users, which would have to be in the same domain as the attacker, to trust this validation. This is mitigated by not sharing validations inside of domains where the users that can originate call from that domain are not trusted by the domain.

9.3. Spam

Another serious concern is that attackers may try to launch VoIP spam (also known as SPIT) calls into a domain. ViPR prevents this by requiring that a domain make a PSTN call to a number before it will allow a SIP call to be accepted to that same number. This provides a financial disincentive to spammers. The current relatively high cost of international calling, and the presence of national do-not-call regulations, have prevented spam on the PSTN to a large degree. ViPR applies those same protections to SIP connections.

As noted above, ViPR still lowers the cost of communications, but it does so by amortizing that savings over a large number of calls. The costs of communications remain high for infrequent calls to many numbers, and become low for frequent calls to a smaller set of numbers. Since the former is more interesting to spammers, ViPR gears its cost incentives away from the spammers, and towards domains which collaborate frequently.

Of course, ViPR’s built-in mechanism is not a guarantee. A SPIT clearinghouse could shoulder the costs of the PSTN calls, and then re-sell its access for a fee. However, this still causes the clearinghouse to utilize non-trivial resources in its attack. Though these costs are less than the PSTN, they are more than zero, and should act as a deterrent for a long while.

9.4. Eavesdropping

Another class of attacks involves outsiders attempting to listen in on the calls that run over the Internet, or obtain information about the call through observation of signaling.

All of these attacks are prevented by requiring the usage of SIP over TLS and SRTP. These are mandatory to use.

10. IANA Considerations

This specification does not require any actions from IANA.

11. Acknowledgements

Thanks to Theo Zourzouvillys for pointing out the fifth thief of phone numbers attack.

12. References

12.1. Normative References

[P2PSIP-BASE]
Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD) Base Protocol", draft-ietf-p2psip-base-11 (work in progress), October 2010.

[P2PSIP-SIP]
Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "A SIP Usage for RELOAD", draft-ietf-p2psip-sip-05 (work in progress), July 2010.

[VIPR-RELOAD-USAGE]
Rosenberg, J., Jennings, C., and M. Petit-Huguenin, "A Usage of Resource Location and Discovery (RELOAD) for Public Switched Telephone Network (PSTN) Verification", draft-rosenberg-dispatch-vipr-reload-usage-03 (work in progress), October 2010.

[VIPR-SIP-ANTISPAM]
Rosenberg, J., Jennings, C., and M. Petit-Huguenin, "Session Initiation Protocol (SIP) Extensions for Blocking VoIP Spam Using PSTN Validation", draft-rosenberg-dispatch-vipr-sip-antispam-03 (work in progress), October 2010.

[VIPR-VAP]
Rosenberg, J., Jennings, C., and M. Petit-Huguenin, "Verification Involving PSTN Reachability: The ViPR Access Protocol (VAP)", draft-rosenberg-dispatch-vipr-vap-03 (work in progress), October 2010.

[VIPR-PVP]
Rosenberg, J., Jennings, C., and M. Petit-Huguenin, "The

Public Switched Telephone Network (PSTN) Validation Protocol (PVP)", draft-rosenberg-dispatch-vipr-pvp-03 (work in progress), October 2010.

12.2. Informative References

[RFC2543] Handley, M., Schulzrinne, H., Schooler, E., and J. Rosenberg, "SIP: Session Initiation Protocol", RFC 2543, March 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.

[RFC3263] Rosenberg, J. and H. Schulzrinne, "Session Initiation Protocol (SIP): Locating SIP Servers", RFC 3263, June 2002.

[RFC5039] Rosenberg, J. and C. Jennings, "The Session Initiation Protocol (SIP) and Spam", RFC 5039, January 2008.

[RFC3761] Faltstrom, P. and M. Mealling, "The E.164 to Uniform Resource Identifiers (URI) Dynamic Delegation Discovery System (DDDS) Application (ENUM)", RFC 3761, April 2004.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

[RFC5067] Lind, S. and P. Pfautz, "Infrastructure ENUM Requirements", RFC 5067, November 2007.

[RFC5054] Taylor, D., Wu, T., Mavrogiannopoulos, N., and T. Perrin, "Using the Secure Remote Password (SRP) Protocol for TLS Authentication", RFC 5054, November 2007.

[CLF-SYNTAX] Roach, A., "Binary Syntax for SIP Common Log Format", draft-roach-sipping-clf-syntax-01 (work in progress), May 2009.

[SESSION-ID] Kaplan, H., "A Session Identifier for the Session Initiation Protocol (SIP)", draft-kaplan-sip-session-id-02 (work in progress), March 2009.

Appendix A. Release notes

This section must be removed before publication as an RFC.

A.1. Modifications between rosenberg-04 and rosenberg-03

- Nits.
- Shorter I-Ds references.
- Changed phone numbers to follow E.123 presentation.
- Expanded P2P initialisms.
- Uses +1 408 555 prefix for phone numbers in examples.

Authors' Addresses

Jonathan Rosenberg
jdrosen.net
Monmouth, NJ
US

Email: jdrosen@jdrosen.net
URI: http://www.jdrosen.net

Cullen Jennings
Cisco
170 West Tasman Drive
MS: SJC-21/2
San Jose, CA 95134
USA

Phone: +1 408 421-9990
Email: fluffy@cisco.com

Marc Petit-Huguenin
Stonyfish

Email: marc@stonyfish.com

dispatch
Internet-Draft
Intended status: Standards Track
Expires: April 28, 2011

J.R. Rosenberg
jdrosen.net
C. Jennings
Cisco
M. Petit-Huguenin
Stonyfish
October 25, 2010

The Public Switched Telephone Network (PSTN) Validation Protocol (PVP)
draft-rosenberg-dispatch-vipr-pvp-03

Abstract

One of the main challenges in inter-domain federation of Session Initiation Protocol (SIP) calls is that many domains continue to utilize phone numbers, and not email-style SIP URI. Consequently, a mechanism is needed that enables secure mappings from phone numbers to domains. The main technical challenge in doing this securely is to verify that the domain in question truly is the "owner" of the phone number. This specification defines the PSTN Validation Protocol (PVP), which can be used by a domain to verify this ownership by means of a forward routability check in the PSTN.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference

material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 28, 2011.

Table of Contents

1. Introduction

The validation protocol is the key security mechanism in ViPR. It is used to couple together PSTN calls with IP destinations based on shared knowledge of a PSTN call. This document relies heavily on the concepts and terminology defined in [VIPR-OVERVIEW] and will not make sense if you have not read that document first.

The protocol assumes that two enterprises, the originating one (enterprise O) initiates a call on the PSTN to an E.164 number ECALLED that terminates on the terminating enterprise (enterprise T). Each enterprise has a ViPR server, acting as a P2P node. The node in enterprise O is PO, and the node in enterprise T is PT. This PSTN call completes successfully, and knowledge of this call is known to PO and PT. Later on, PO will query the P2P network with number ECALLED. It comes back with a Node-ID PCAND for a node. At this time, PO can't know for sure that PCAND is in fact PT. All it knows is that some node, PCAND, wrote an entry into the DHT claiming that it was the owner of number ECALLED. The objective of the protocol is for PO to determine that node PCAND can legitimately claim ownership of number ECALLED, by demonstrating knowledge of the previous PSTN call. It demonstrates that knowledge by demonstrating it knows the start time, stop timer, and possibly caller ID for the PSTN call made previously.

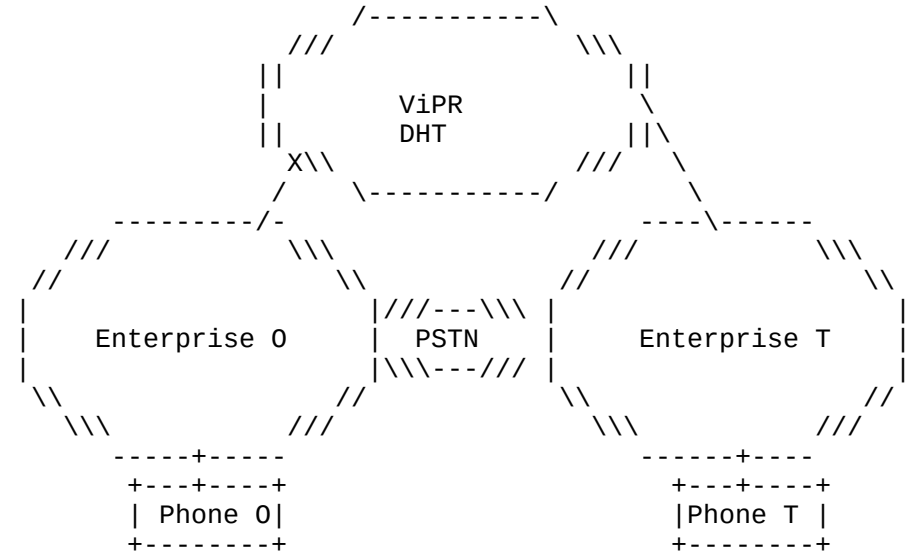


Figure 102: Validation Model

If node PCAND can demonstrate such knowledge, then enterprise O can assume that node PCAND had in fact received the call, which could only have happened if it had knowledge of the call to number ECALLED, which could only have happened if PCAND is in enterprise T, and thus it is PT. This is because PSTN routing is assumed to be "secure", in that, if someone calls some number through the PSTN, it will in fact reach a terminating line (whether it be analog, PRI, or other) which is the rightful "owner" of that number. If enterprise T was not the owner of the number, if would not have received the call, would not know its start/stop/caller ID, not be able to provide that information to PT, and not be able to satisfy the knowledge proof. This basic approach is shown in Figure 102.

A first question commonly asked is, why not just do regular authentication? What if we give each node a certificate, and then have the nodes authenticate each other? The answer is that a certificate certifies that a particular node belongs to a domain - for example, that node PT is part of example.com. A certificate does not assert that, not only is PT example.com, but example.com owns the following phone numbers. Therefore simple certificate authentication does not provide any guarantee over ownership of phone numbers.

In principle, it might be possible to ask certificate authorities, such as Verisign, to assert just that. However, traditionally, certificate authorities have been extremely hesitant to certify much at all. The reason is, the certifier needs to be able to assure that the information is correct. How can a certifier like Verisign verify that, in fact, a particular enterprise owns phone numbers? It could make a few test calls, perhaps, to check if they look right. However, these test calls are disruptive to users that own the numbers (since their phones will ring!). If the test calls are done for a subset of the numbers, it is not secure. If the certifier simply required, as part of the business agreement, that the enterprises provided correct information, the certifier might avoid legal liability, but the legitimacy of the service will be compromised and customers will stop using it. Furthermore, it has proven incredibly hard to do this kind of certification worldwide with a single certificate authority.

ViPR has, as a goal, to work anywhere in the world and do guarantee correct call routing with five nines of reliability. Consequently, traditional certificates and authentication do not work. It turns out to be quite hard to design a secure version of this validation protocol. To demonstrate this, we will walk through some initial attempts at it, and show how they fail.

2. The Wrong Way

The first attempt one might make is the following. PO takes the caller ID for the call, ECALLING and called number ECALLED for the call, and sends them to candidate node PCAND. These two identifiers - the called number E and the caller ID, form a unique handle that can be used to identify the call in question. Node PCAND looks at all of the ViPR Call Records (VCRs) of the calls over the last 48 hours, and takes those with the given called party number and calling party number. If there is more than one match, the most recent one is used. We now have a unique call.

Now, node PCAND demonstrates knowledge of this call by handing back the start and stop times for this call in a message back to PO. This approach is shown in Figure 103.

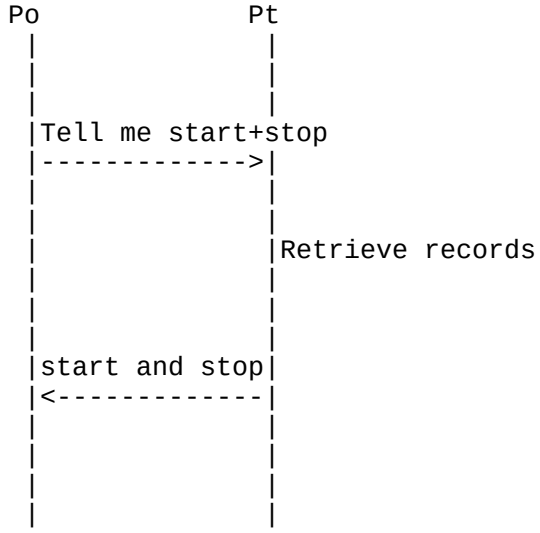


Figure 103: Incorrect Validation Protocol: Take 1

Unfortunately, this method has a major problem, shown in Figure 104.

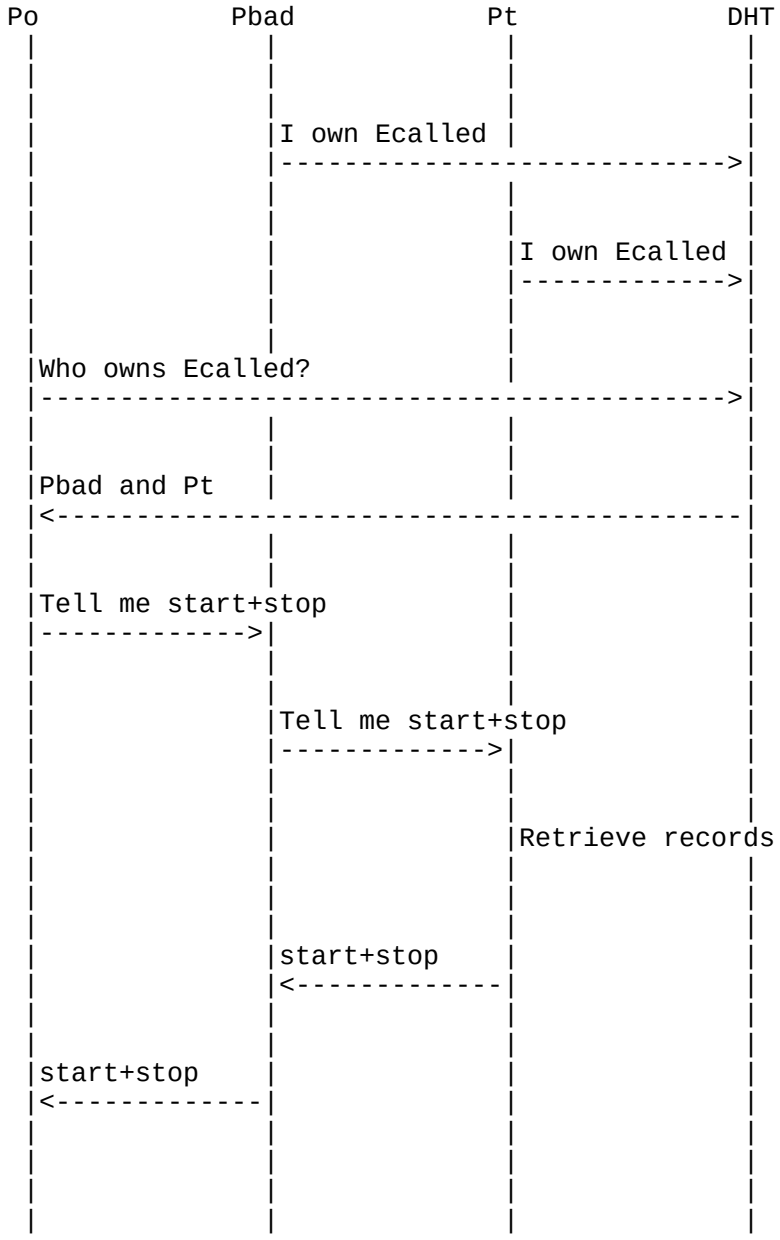


Figure 104: Attack for Incorrect Validation Protocol

Consider an attacker BadGuy PBAD. PBAD joins the P2P network, and advertises a number prefix they do NOT own, but which is owned by

enterprise T and node PT. Now, when PO queries the DHT with number ECALLED, it comes back with two results - the one from PBAD and the one from node PT. Details of querying the DHT are provided in [VIPR-RELOAD-USAGE]. It begins validation procedures with both. PBAD will now be asked to show the start and stop times for the call, given ECALLED and ECALLING. It doesn't know that information. However, node PT does. So now, PBAD, acting as if it where the originating party, begins the validation protocol with node PT. It passes the calling and called numbers sent by PO. PT finds a match and returns the call start and stop times to PBAD. PBAD, in turn, relays them back to PO. They are correct, and as a consequence, PO has just validated PBAD!

Typically, the first response to this is, "Well the problem is, you let two separate people write the same number into the DHT. Why don't you make sure on the right one is allowed to write it in?". That is not possible, since there is no mechanism by which an arbitrary node in the DHT can determine who is the rightful owner of this number. "OK", the reader responds, "So instead, why don't you define a rule that says, if there are two entries in the DHT for a particular number, consider this an attack and don't try to validate the number". That would prevent the attack above. However, it introduces a Denial of service attack. An attacker can pick a target number, write it into the DHT, and prevent successful validation from happening towards that number. They can't misroute calls, but they can stop ViPR from working for targeted numbers. That is not acceptable. ViPR has to be immune from attacks like this; it should not be possible, through simple means such as configuration, for an attacker to cause a targeted number to never be validated.

One might be tempted to add a signature over the call start and stop times, but it does not help. BadGuy can just resign them and relay them on.

In essence, this simple approach is like a login protocol where the client sends the password in the clear. Such mechanisms have serious security problems.

Realizing the similarities between the validation protocol and a login protocol, a next attempt would be to use a much more secure login mechanism - digest authentication. To do this, domain O takes the called number E and the caller ID, and send them to node P. Node P treats these as a "username" of sorts - an index to find a single matching call. The start time and stop times of the call become the "password". Enterprise O also sends a big random number - a nonce - to node P. Node P then takes the random number, takes the password, hashes them together, and sends back the hash. All of this is done over a TLS connection between enterprise O and node P. Digest over

TLS is very secure, so surely this must be secure too, right? Wrong!

It is not. Indeed it is susceptible to EXACTLY the same attack described previously. This is shown in Figure 105.

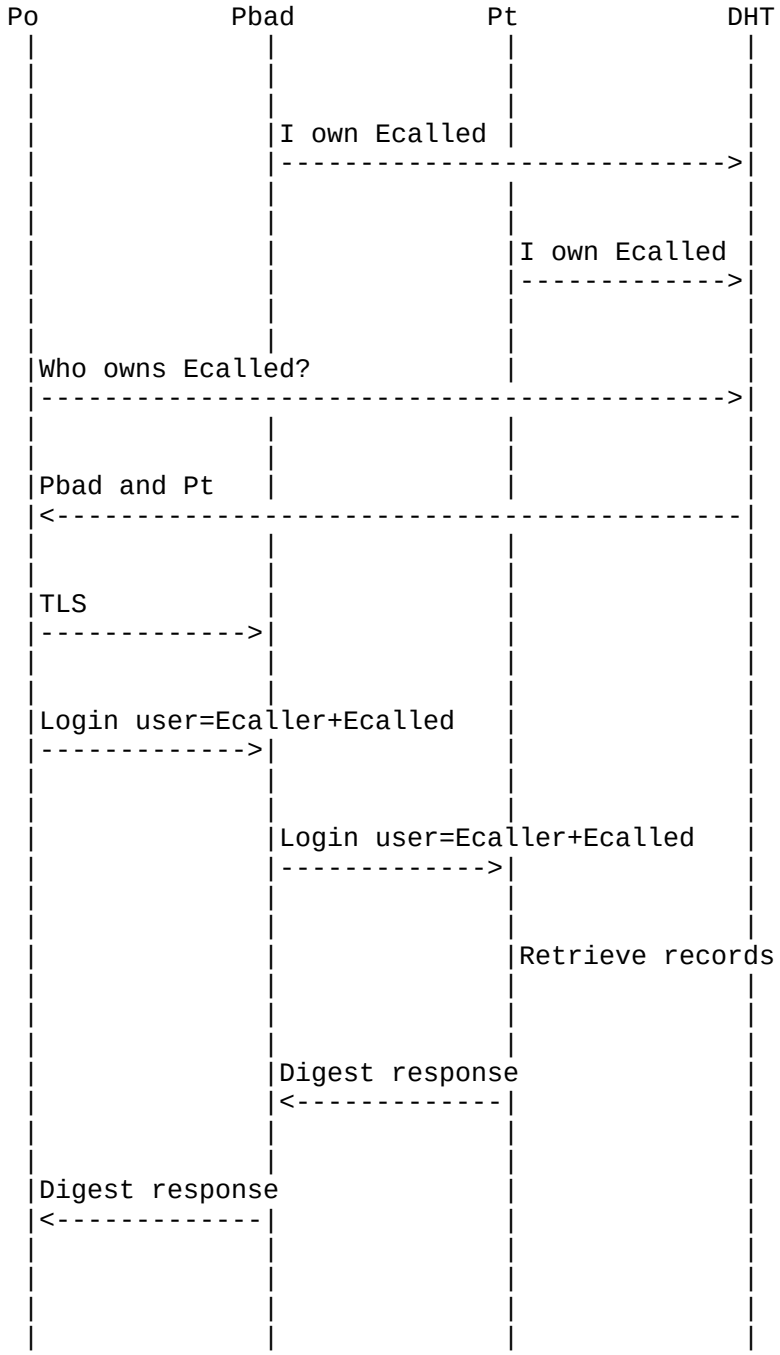

Po
Pbad
Pt
DHT
I own Ecalled
I own Ecalled
Who owns Ecalled?
Pbad and Pt
TLS
Login user=Ecaller+Ecalled
Login user=Ecaller+Ecalled
Retrieve records
Digest response
Digest response

Figure 105: Trying Digest for Validation

In a similar attack, PBAD could pick a random called number it is interested in, query the P2P network for it, find node PT. Then, provide node PT the number ECALLED to attack, and ECALLING, assuming it can guess a likely caller ID. It then takes the received digest response, and goes through every possible start/stop time over the last 24 hours, running them through the hash function. When the hash produces a match, the PBAD has just found a full VCR for node PT. It can then write into the DHT using number E as a key, pointing to itself, and satisfy validation requests against it, without even needing to ask node P again. Our first attempt is susceptible to this attack too.

The problem here is that the call start and stop times have "low entropy" - they are not very random and are easily guessable, just like a poorly chosen password.

What we really want to do here is have a "login" protocol that creates a secure connection between a client and a server, where we use the called number and caller ID as a "username" to identify a PSTN call, and then use the start and stop times as a "password". But our login protocol has to have some key features:

1. Someone posing as a server, but which does not have the username and password, cannot determine the username and password easily as a consequence of an authentication operation started by a valid client, aside from successfully guessing in the one attempt it is given on each connection attempt.
2. Someone posing as a client, but which does not have the username and password, cannot determine the username and password as a consequence of an authentication operation started against a valid server, aside from guessing in the one attempt it is given on each TLS connection attempt.
3. An active MITM, who is explicitly on the path of the exchanges and has visibility and the ability to modify messages, cannot obtain the shared secret, nor can it observe or modify information passed between the client and real server.
4. It is impossible for a passive observer to view the exchange and obtain the shared secret or any of the material that is exchanged.
5. It is impossible for a rogue client or rogue server to participate in a login with a legitimate peer, and then take the messages exchanged, and run an offline dictionary attack to work through every possible combination of start and stop times. Fortunately, these properties are provided by a class of password authentication protocols called Encrypted Key Exchange or EKE protocols.

3. EKE Protocols

EKE protocols were proposed in 1992 by Steve Bellovin. Since their proposal, numerous variations have been defined. One of them, the Secure Remote Password protocol, was standardized by the IETF in RFC 2945 [RFC2945]. A TLS mode of SRP was later defined in RFC 5054 [RFC5054]. It is the latter protocol which is actually used by ViPR. A high level overview of EKE protocols is shown in Figure 106. Alice and Bob share a shared secret P. Alice generates a public/private keypair. She then takes her public key, and encrypts it using her password as a symmetric encryption key. She sends this encrypted key to Bob. Bob, who shares the password, uses it as a symmetric key and decrypts the message, obtaining Alice’s new public key. Bob then constructs a big random number R, which is to be used as a session key. Bob then encrypts R with the public key he just got from Alice, and sends that to Alice. Now, Alice, using her public key, decrypts the message and obtains the session key R.

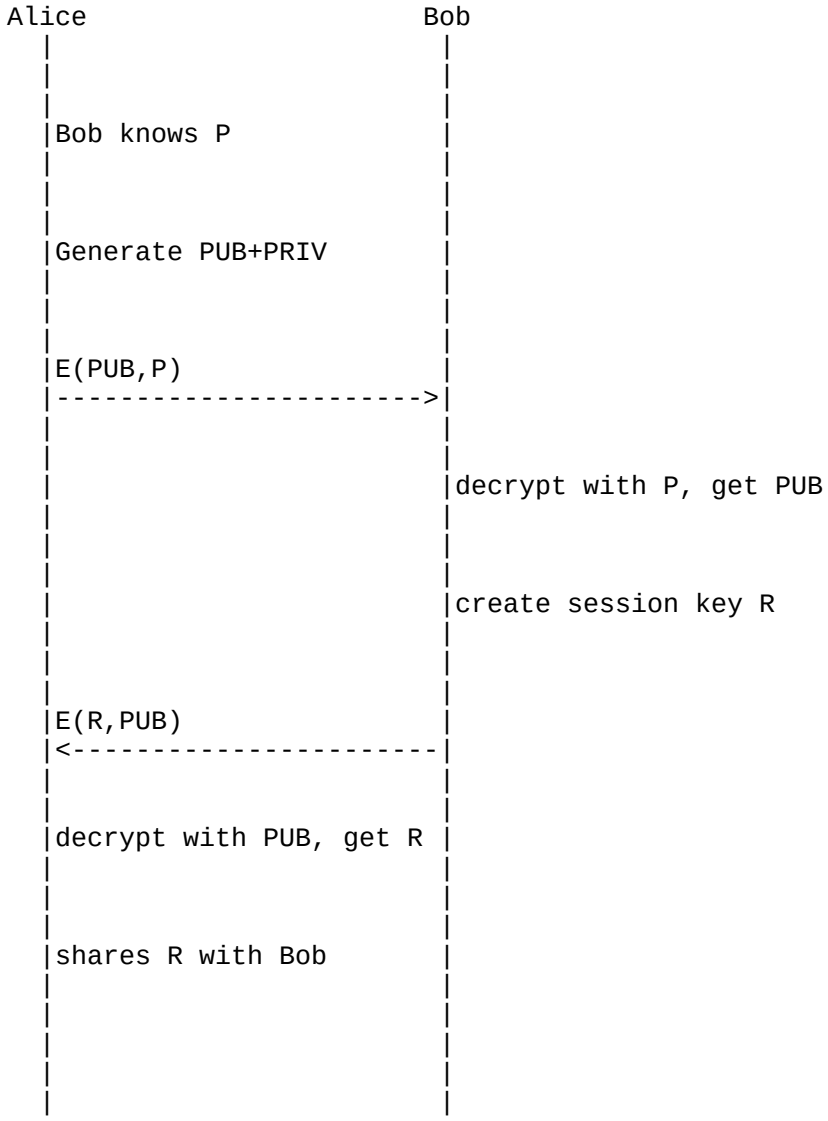
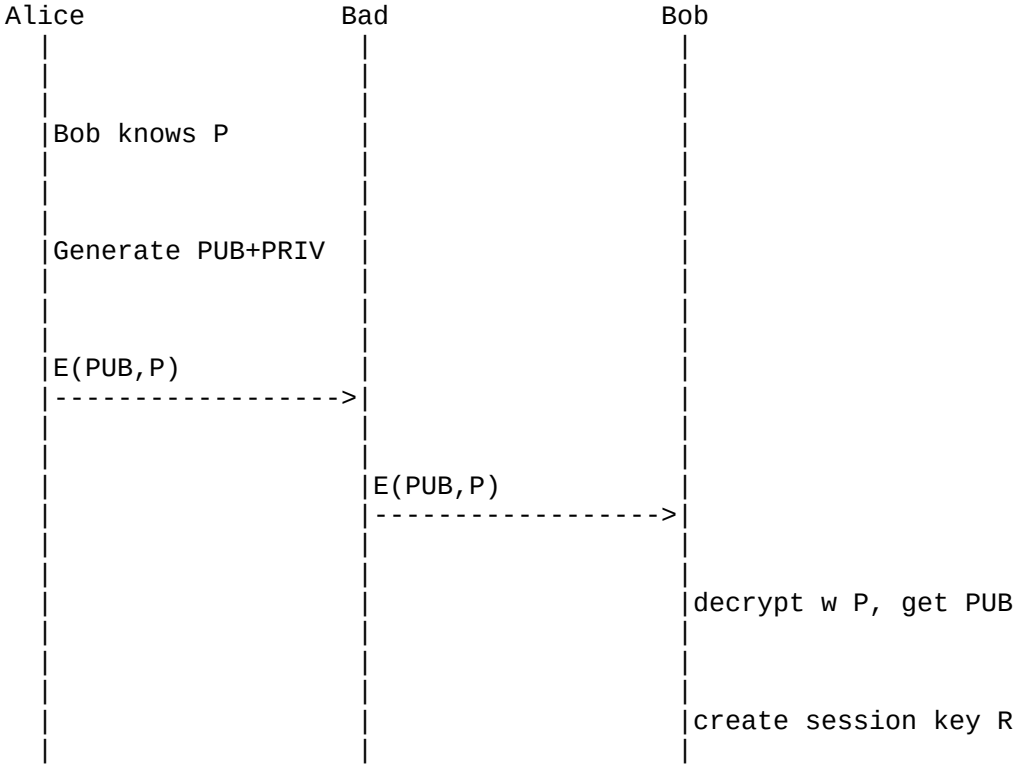


Figure 106: High Level EKE Model

At this point Alice and Bob share a session key R which can be used for authentication (by having Alice and Bob prove to each other that they have the same value for R) or for encrypting data back and forth. How does this help? Consider our man-in-the-middle attack again, in Figure 107. Once again, Alice shares a password with legitimate user Bob. However, she begins the "login" process with

BadGuy. She passes E(PUB,P) to BadGuy. BadGuy doesn't know P, so he can't decrypt the message. More importantly, he can't run through each possible password P and decrypt the message. If he did, he wouldn't be able to tell if he got it right, since PUB appears random; the decryption process would produce a random string of bits whether it was successful or not. So for now, BadGuy can only pass it on. BadGuy now intercepts E(R,PUB). Now, BadGuy can try the following. He can run through each P, decrypt E(PUB,R), obtain PUB. However, since we are using asymmetric encryption (i.e., public key encryption), even with PUB he cannot DECRYPT E(R,PUB)! BadGuy does not have the private key, which he needs to decrypt. Given a public key, he cannot guess the private key either. That is how public/ private keying systems work. That is the secret here to making this work. So, once again, BadGuy has no choice but to pass the message on. Now, Alice and Bob share R but it is unknown to BadGuy. Bob now takes his Node-ID, encrypts it with R, and sends to Alice. Once again, BadGuy doesn't have R and can't get it, so he has no choice but to pass it on. Alice decrypts this Node-ID with R, and now knows that she is actually talking to Bob - since she has Bob's Node-ID. Other data can be substituted for the Node-ID, and indeed this is what happens in the actual validation protocol.



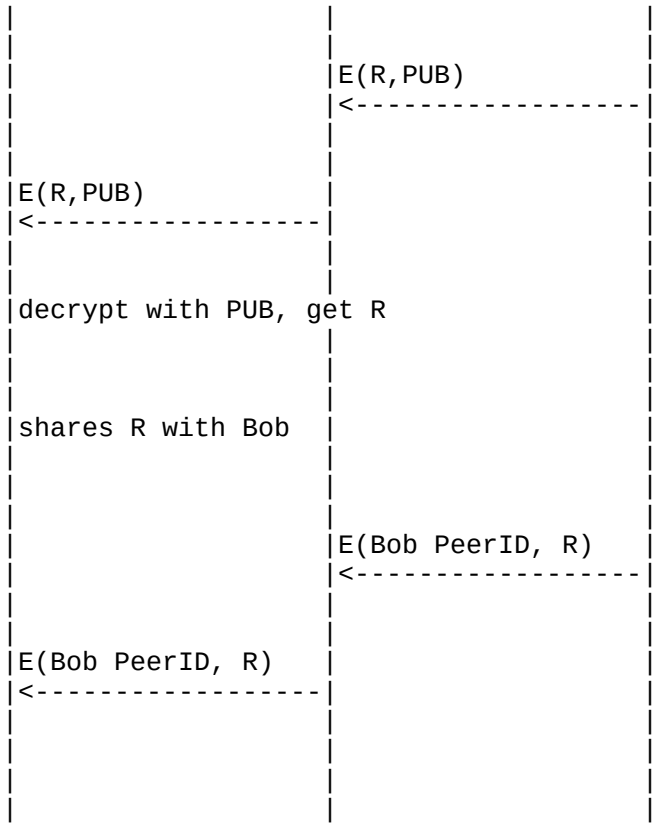


Figure 107: Attacking EKE Protocols

However, the main point of this exercise is to demonstrate that EKE protocols have the desired properties.

4. Protocol Overview

The validation protocol begins with the following assumptions:

1. Node PO wishes to validate with node PCAND, and has its Node-ID (which it obtained via the DHT) and VServiceID (which it also obtained via the DHT Fetch).
2. Node PO and PCAND have a series of call records over the last 48 hours, uploaded by their call agents. Each call record contains an E.164 calling and called party number, and a start and stop time in NTP time. On the terminating side, each call record is also associated with a VServiceID.

3. Node PO is seeking to validate a call to called number ECALLED with caller ID ECALLING.

The validation protocol operates by having the originating node make a series of attempts to connect to, and "login" to the terminating node. Each "login" attempt consists of establishment of a TCP connection, and then execution of TLS-SRP procedures over that connection. TLS-SRP[RFC5054] relies on a shared secret - in the form of a username and password - in order to secure the connection. In ViPR, the username and password are constructed by using information from a target VCR along with the VServiceID learned from the DHT. The "username", instead of identifying a user, identifies a (hopefully) unique VCR shared between the originating and terminating nodes. The "password" is constructed from the VCR such that it knowledge of the information is unique to knowledge of the VCR itself.

Unfortunately, it is difficult to construct usernames and passwords that always uniquely identify a VCR. To deal with this, the validation protocol requires the originator to construct a series of usernames and passwords against a series of different nodes and their corresponding IP addresses and ports, and then run through them until a connection is securely established.

5. Username and Password Algorithms

ViPR provides two different algorithms for mapping from a particular VCR to a username and password:

1. Method A: This method makes use of the called party and calling party number to form the username, and the start and stop times of the call to form the password.
2. Method B: This method makes use of the called party number, along with a point in time in the middle of the call, as the username, and then the start and stop times to form the password.

The originating node will first try validations with method A, and if those all fail, then try with method B. The method itself, along with necessary information on how to use the method, is encoded into the username itself. The format of the username is (using ABNF [RFC4234]):

```
username = method-a / method-b / future-method
future-method = method ":" method-data
method = 1ALPHA
method-data = 1*(alphanum / method-unreserved)

method-a = "a:" vserviceid originating-number terminating-number
           rounding-time
method-b = "b:" vserviceid terminating-number timekey rounding-time

vserviceid = "vs=" 1*32HEXDIG ";"
originating-number = "op=+" 1*15DIGIT ";"
terminating-number = "tp=+" 1*15DIGIT ";"
timekey = "tk=" 1*16DIGIT "." 1*16DIGIT ";"
rounding-time = "r=" 1*6DIGIT ";"
```

This format starts with the method, followed by a colon, followed by a sequence of characters that are specific to the method. Both methods a and b rely on conveyance of information attributes that make up the username. Each attribute follows a specific format.

Examples include:

```
a:vs=7f5a8630b6365bf2;op=+17325552496;tp=+14085553084;r=1000;
b:vs=7f5a8630b6365bf2;tp=+14085553084;tk=172636364.133622;r=1000;
```

Both methods use a rounding factor R. This is used to round the start and stop times in the password to a specific nearest multiple of R (which is in milliseconds). This rounding is done because the passwords need to be bit exact and we need to compensate for different measured values.

If we will fallback to method B (which works more often), why have both? There are two answers:

1. The caller ID mechanism (method A) will work, and the non-caller ID (method B) won’t work, for numbers like 8xx.
2. Method A has much higher entropy (see analysis in Section 9.1). Validating with it provides greater confidence in the validity of the number. In this phase, nothing is done with this "confidence". However, in later phases, it is anticipated that low-confidence numbers will require multiple validations for different calls to occur before they are trusted. To allow for this feature to be added later, validation with both methods must be present in the initial release.

The sections below detail precisely how these are constructed.

6. Originating Node Procedures

Most of the work for validation is on the side of the originator. It establishes connections and performs a series of validation checks.

6.1. Establishing a Connection

The first step in the process is to establish a TCP connection to PCAND. To do that, PO sends a DHT PING message targeted towards PCAND. This will return one or two IP addresses and ports. This provides one or two targets to which a connection attempt is made. An attempt is made first to the public address, then if that connection times out, to the private. Once connected, TLS-SRP is run over the connection.

6.2. Constructing a Username and Password

When a terminating node receives a username in a format it doesn’t understand, it fails the validation. This allows for graceful upgrade to new mechanisms in the future.

6.2.1. Method A

The PO examines the VCR it is using for validation. It extracts the calling and called party numbers, both of which are E.164 based. This VCR will have been uploaded at a previous point in time. PO then examines the VCRs posted in the time since this one was uploaded, and looks for any more recent VCRs with the same calling and called party numbers regardless of VService. If it finds one or more, it takes the most recent one (as measured by its end time). If it finds no more recent, it uses the VCR which triggered the validation in the first place.

Why do this? This deals with the following case. User A calls user B, causing a VCR to be uploaded. The originating node sets a timer, which fires 12 hours later. However, within that 12 hour period, A called B again. If node A provides the caller ID and called party numbers as the "key" to select a VCR, it will match multiple records over the past day. We need to pick one, so the most recent is always used. This requires the originating node to know and use the most recent VCR. Furthermore, we most choose the most recent VCR regardless of its VService, because the originating Upload VCRs are sent using an arbitrary VService. Thus, the more recent call may have been done using a different VService than the one which triggered the validation. Since the actual Vservices are not common between originating and terminating sides, we must choose the VCR on the originating side regardless of VService. The username is constructed by using the syntax for method A described above. The

calling party number is set to "op", and the called party number is set to "tp", and "r" is the value of Tr as an integral number of milliseconds. The VServiceID learned from the dictionary entry is used as the value of "vs".

This username will select the identical VCR at the terminating node, under the following conditions:

1. PT is aware of all calls made to the called party number. This property is true so long as each incoming number is handled by a single call agent within a domain, and furthermore, the VCR for calls to that number is always posted to a ViPR server which advertises that number into the DHT. These properties are readily met by ViPR for typical single user numbers. For 8xx numbers, which are translated within the PSTN and may route to a multiplicity of non-8xx numbers, it is more difficult. ViPR will only work with 8xx numbers if all calls to those numbers get sent to agents which share the same ViPR server.
2. PO is aware of all calls made to the called party number with the given caller ID. This property can be hard to meet. If the caller ID for a call is set to the number of the calling phone, and all VCRs made from that phone are posted to the same ViPR server, that server will know about all calls made by the domain with the given DID in the caller ID. However, in domains that set the PSTN caller ID to the attendant line number, it is possible that there would be two separate agents, each utilizing different ViPR servers. A user in each agent calls the same number, and the same PSTN caller ID is used. However, one ViPR server knows about one of the calls, and a different ViPR server knows about the other call. However, PT knows about both. In that case, validation from one of the ViPR servers will fail, and from the other, succeed.
3. There were no calls on the PSTN to the called party which spoofed the caller ID to match the caller ID used by the valid enterprise. In that case, PT will have a VCR for a call with a matching calling/called party number, but this VCR is unknown to PO since the call was not actually made by the originating enterprise. This attack is described in more detail in XXXX.

Next, the password is selected. The password is basically the start and stop times for the call. However, the SRP protocol requires a bit exact agreement on the password. Unfortunately, the calling and called parties will not have the same values for the start and stop times, for several reasons:

1. The call start time at the originating and terminating ends will differ by the propagation delay of the call acceptance message through the PSTN. This can be several hundreds of milliseconds.

2. The clocks at the originating and terminating ends may not be synchronized, which can also introduce different values for the start and stop times.
3. The call termination time at the originating and terminating ends will also differ by the propagation time; this propagation time may in fact be different for the call acceptance and termination.

It is also important to note that agreement on a call acceptance and termination time assumes an explicit signaling message is sent for these two events. In the case of analog FXO ports, there is no signaling at all, and consequently, these points in time cannot be measured. It is possible to agree upon other call characteristics when analog lines are in use, but they have much worse accuracy and consequently much, much lower entropy. For this reason, this specification of ViPR only works in telephony systems with explicit messaging for call acceptance and termination, which includes PRI, SS7, BRI, analog trunks with answer and disconnect supervision, and CAS trunks.

To deal with these inaccuracies in timing, the start and stop times need to be rounded. Let Tr be the rounding interval, so that each time is rounded to the value of N*Tr for integral N, such that N*Tr is less than the start or stop time, and (N+1)*Tr is greater than it. In other words, "round down". If Tr=1 second, this would round down to the nearest second.

Unfortunately, rounding doesn’t fully help. Lets say that the difference between the start times on the originating and terminating nodes is delta. We can still have different values for the start time if one side rounds to one value, and the other side to a different value. If delta=100ms and Tr=1s, consider a start time of 10.08 seconds on one side, and 9.98 on the other side. One side will round to 10 seconds and the other to nine seconds. The probability of this happening is approximately delta/Tr. We could just make Tr really large to compensate, but this reduces the entropy of the system (see below).

To deal with this, the originating node will actually compute FOUR different passwords. For the start time and stop time both, the originating node will round down as follows. Let T be the time in question. Let N be the value such that N*Tr <= T < (N+1)*Tr. In other words, N*Tr is the nearest round-down value, and (N+1)*Tr is the nearest round up. Let T1 and T2 be the two rounded values of T. We have:

```
if (T >= ((2N+1)/2) * Tr) then:
                          T1 = N*Tr
                          T2 = (N+1)*Tr
if (T <  ((2N+1)/2) * Tr) then:
                          T1 = N*Tr
                          T2 = (N-1)*Tr
```

In other words, if T is in the top half of the rounding interval, we try the rounded values above and below. If T is in the bottom half, we try the rounded values below, and below again. Pictorially:

[[TBD]]

Figure 108: Rounding mechanism for validation protocol

These are tried in the following sequence:

1. Try Tstart-1 and Tend-1.
2. Try Tstart-2 and Tend-1.
3. Try Tstart-1 and Tend-2.
4. Try Tstart-2 and Tend-2.

For example, if the originating side has a start time of 10.08 and a stop time of 30.87, the four start and stop times with Tr=1s are:

Start	Stop
10	30
9	30
10	31
9	31

Each of these times is represented in 64 bit NTP time (Tr can be configured to less than 1s in which case there will be non-zero values in the least significant 32 bits). Each password is then computed by taking the 64 bit start time, followed by the 64 bit end time, resulting in a 128 bit word. This word is base64 encoded to produce an ASCII string representation of 21 characters. To perform the caller ID based validation, the SRP-TLS procedure is done four times, once with each of the four username/password combinations (of course the username is identical in all four cases). As long as delta is less than Tr/2, one of this is guaranteed to work.

6.2.2. Method B

Unfortunately, in many cases caller ID cannot be used as an identifier for the VCR. This is because:

1. CallerID is frequently suppressed in the PSTN, and not delivered. This is especially true in international cases.
2. CallerID is sometimes munged by the PSTN, and delivered, but with a different value than was sent by the originator. This happens in certain arbitrage interexchange carriers.

Consequently, if no caller ID was delivered at all, the terminating side will not have a matching record. In that case, it informs the calling side that it should abort and revert to method B. If munged, it will also abort for the same reason.

If the caller ID attempt aborts, PO now tries a different approach. In this approach, the "username" is the combination of the called party number and a point during the call, selected at random. The password is equal to the start and stop times of the call. This method uses the method-tag "B" in the username.

Unlike method A, with method B, the VCR which triggered the validation is used, regardless of whether there were other, more recent, calls to the same calledparty number! This is because, in method B (unlike method A), the time itself is used as a key to select a VCR. Furthermore, using a more recent VCR does not interact properly with multi-tenancy. The called number and point during that call will select an identical VCR on the terminating side if the following conditions are met:

1. For the called party number, there was not more than one call in progress made to that number at the same time. This is generally true for numbers for a single user; typically there is only one active call at a time. Of course, it is possible a user receives a call, and then gets another. It then puts the first on hold while the second call is taken. In these cases, it is possible that the "username" will select a different VCR on PT, in which case the validation fails. More troubling are numbers representing call centers, conference bridges, 8xx numbers, and attendant numbers, all of which frequently have multiple calls in progress to them at the same time. As a consequence, for these types of called numbers, validation is typically only going to work if caller ID is delivered. Fortunately, 8xx numbers are only national in the first place, so it is likely that this will work.

2. PO is aware of all calls made from within its enterprise to ECALLED. This can fail if there are multiple ViPR servers serving different agents, and a call is made from one agent, sent to one ViPR server, and a call to that same number is made on a different agent, send to a different ViPR server. As in the caller ID case, this will still be OK in many cases - the validation from one ViPR server succeeds, the other fails.
3. PT is aware of all calls made to ECALLED. The same caveats as described above for the caller ID mechanism apply. PO takes the VCR, and chooses a time Tkey which is uniformly distributed between Tstart+Tr and Tstop-Tr. The usage of the Tr here is to make sure that Tkey is squarely inside of the call start and stop for PT as well. Note that, because Tkey is not a password, it is sent in the clear and does not need to be rounded.

The username encodes the called party number, Tkey, the DHT, and the VServiceID learned from the DHT query. The password is computed identically to method A.

6.3. Requesting Validation

Once the SRP-TLS connection is up, data is exchanged. This is done through a single VAP transaction initiated by PO. This transaction is only VAP in the sense that it utilizes the basic syntax (the header and TLV attribute structure), and its request/response model. Other than that, it is effectively a different protocol - the validation protocol.

PO sends a VAP request with method ValExchange (0x00d). It contains one attribute, Domain. The originating ViPR server obtains this domain by looking at the VService of the VCR that was eventually used for the validation. Note that, in cases where the VCR which triggered the validation, is different than the one actually used for validation (because a more recent VCR to the same number was found), it is important to use the VService associated with the VCR which was actually used for validation, and NOT the VService associated with the VCR which triggered the attempt. Multi-tenancy does not work properly without this. The domain from the VService is placed into the message. This is basically the domain name of the originating enterprise. It is included since it is needed by PT to compute the ticket.

PO will then receive a response. If it never receives a response within a timeout, it considers the validation to have failed, and continues to the next choice. If it receives any kind of error response, including a rejection due to a blacklisted domain, it considers validation to have failed, and continues to the next choice. If it is a success response, it will contain one attribute -

ServiceContent, which contains a ValInfo XML object. ValInfo is an XML object which contains the SIP URIs and the ticket. The ViPR server must parse the ValInfo XML object and perform verification on it to avoid attacks. The following checks are done:

1. Extract the <number> element. This will contain a single number. That value is compared with the E.164 called party number which was just validated. If they do not match, this is a potential attack, and the XML is discarded and the ViPR server acts as if validation failed. However it does not generate an alarm.
2. Remove any extensions to the XML which are not supported by the ViPR server (no extensions defined, so in this version, any elements except for the <ticket>, <number>, <route>s and their embedded <SIPURI> are removed.
3. Verify that the <route> element contains a single element, <SIPURI>.
4. Verify that the SIP URI is not larger than 614 characters, contains a domain name that is a valid set of domain name characters, contains a user part that is a valid set of characters, if it contains maddr, that the maddr is a valid domain or IP and less than 255 characters, and if there is a port, it is within 0-65535. This is for security purposes; to make sure a malicious ViPR server on the terminating side cannot send invalid URI and attack the call agent.
5. Verify that each SIP URI contains the same domain name. Once the checks and fixes are done, the patched XML is passed to subscribers in a Notify as described in [VIPR-VAP].

7. Terminating Node Procedures

7.1. Waiting for SRP-TLS

PT will listen on its configured port for TCP connections, and once one is received, it begins waiting for SRP-TLS. The TLS messaging will provide PT with a username.

It parses the username and determines the method. If the value of the method is not "a" or "b", this is a new method not supported by the node. The SRP-TLS procedures should be failed. If the method is "a", it is the caller ID mechanism. The called number, calling number, VService, and rounding time are extracted. PT then searches through its VCRs over the last 48 hours for one with a matching called number and caller ID and VService whose VServiceID matches the one from the username:

1. If none are found, PT proceeds with the SRP-TLS exchange, but using a fake username and password. This will cause the validation to eventually fail.
2. If one is found, it is used.
3. If more than one is found, the one with the most recent end time is used.

The start and stop times from the selected VCR are taken. Using the value of Tr from the username, both times are rounded down to the nearest multiple of Tr. Note that, this rounding is different than the one used on the originating side. The values are ALWAYS rounded down. So if the stop time is 10.99 and Tr is one second, the rounded down value of 10 is used. The start and stop times are then represented as 64 bit NTP times (after rounding), concatenated, and base64ed to produce a 21 character password. This is the password used with SRP-TLS.

Note that, the originating node will try up to four different password combinations. One of these should work, the others will cause SRP-TLS to fail due to differing shared secrets. However, it is the job of the originator to perform these four; to the terminating node, they are four separate attempts. Processing of SRP-TLS login attempts is stateless on the terminating side. This means that each attempt is treated independently by PT. It performs identical processing on each SRP-TLS attempt - examine the username, find a matching VCR, extract password, and fail the attempt or continue to success. The originating side has the main burden of sequencing through the various mechanisms.

If the method is "b", PT uses the extracted called party ID and a time in the middle of the call. It searches through all VCR records whose called number matches and whose VServiceID matches, and of those, takes the ones where Tkey is between Tstart and Tstop. Of those, if more than one match, the one with the most recent Tstop is used. Tstart and Tstop for that VCR are extracted, and converted to a password just as is done for the PO. The resulting SRP-TLS procedure will then either succeed or fail. Note that, if a domain has multiple Vservices that contain the same number, there will be multiple VCRs for calls to that number, and there will be multiple validation attempts, one for each of the Vservices.

7.2. Receiving Validation Requests

PT listens for incoming VAP/validation requests once the TLS connection is up. It rejects anything but a ValExchange method with a 400 response. This allows for future extensibility of the validation protocol. If the request was ValExchange, it extracts the domain name. This will be something like "example.com". PT knows

the VCR against which validation succeeded. That VCR is associated with a VService. The ViPR server checks the domain in the ValExchange request against the black/white list associated with that VService. If no VService is currently active, the ValExchange is rejected with a 403. If there is one active, and if the domain appears on the black list, or does not appear in the white list, the ViPR server rejects the ValExchange request with a 403 error response, indicating that this domain is not allowed to call.

If the domain was in the whitelist or not in the blacklist, or there was no whitelist/blacklist, PT constructs a successful response to the ValExchange request. It contains one attribute: ServiceContent. It has a ValInfo XML object, which contains a number, a ticket, and a series of routes.

The number is always the E.164 number which was just validated, including the plus sign. Note that this will also appear in the ticket. The route element is the sequence of route elements for each instance associated with the vservice.

Details of the ticker are provided in [VIPR-SIP-ANTISPAM] but the ticket attribute is constructed as follows:

1. A ticket unique ID TLV is created, containing a randomly chosen 128 bit value as the ticket ID. That is the first TLV in the ticket.
2. A salt TLV is created, containing a random 32 bit value. This is the second TLV in the ticket.
3. The validity has the start time set using the current time as the start time, and the current time + the ticket lifetime as the end time. The ticket lifetime is a per-DHT configurable parameter. The terminating ViPR server will have performed the validation using a particular VService; the DHT for that VService is used to find the right value for this parameter.
4. Number: This is the terminating number, in E.164 format, which was just validated.
5. Granting node: this is set to one of the Node-IDs associated with this ViPR server. Any will do.
6. Granting domain: This value is taken from the domain part of the SIP URI associated with the VService in which the validated VCR was found.
7. Granted-To domain: This is formed using the Domain sent in the ValExchange request.
8. Epoch: This is the current epoch associated with the password.
9. Integrity: Using the current password, this is computed from the rest of the Ticket.

The resulting sequence of TLVs is base64 encoded and that is placed into the ticket element in the ServiceContent attribute in the

ValExchange response.

8. Syntax Details

This section enumerates the methods and attributes used by VAP. The methods defined in VAP, and their corresponding method values, are:

Method	Value
ValExchange	0x00d

Figure 1: PVP Methods

The attribute names and corresponding types are:

Attribute Name	Type
Domain	0x3001

Figure 2: PVP Attributes

9. Security Considerations

[[This section is mostly missing and needs to be done.]]

9.1. Entropy

[[The entropy obtained in the information from the PSTN calls significantly impacts the security of this protocol. This section needs to provide an analysis of how much entropy actually exists in this information.]]

[[Defines the worst case of conference calls and resulting entropy]]

[[Describe the idea of doing multiple validations to aggregate entropy]]

9.2. Forward Routing Assumptions

[[Discuss forward routing security in PSTN and explain how this protocol is reliant on that.]]

10. IANA Considerations

[[TBD Define ports used.]]

11. Acknowledgements

Thanks to Patrice Bruno for his comments, suggestions and questions that helped to improve this document.

12. References

12.1. Normative References

[RFC4234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 4234, October 2005.

[RFC5054] Taylor, D., Wu, T., Mavrogiannopoulos, N., and T. Perrin, "Using the Secure Remote Password (SRP) Protocol for TLS Authentication", RFC 5054, November 2007.

[VIPR-SIP-ANTISPAM] Rosenberg, J., Jennings, C., and M. Petit-Huguenin, "Session Initiation Protocol (SIP) Extensions for Blocking VoIP Spam Using PSTN Validation", draft-rosenberg-dispatch-vipr-sip-antispam-03 (work in progress), October 2010.

[VIPR-VAP] Rosenberg, J., Jennings, C., and M. Petit-Huguenin, "Verification Involving PSTN Reachability: The ViPR Access Protocol (VAP)", draft-rosenberg-dispatch-vipr-vap-03 (work in progress), October 2010.

[VIPR-OVERVIEW] Rosenberg, J., Jennings, C., and M. Petit-Huguenin, "Verification Involving PSTN Reachability: Requirements and Architecture Overview", draft-rosenberg-dispatch-vipr-overview-04 (work in progress), October 2010.

12.2. Informative References

[RFC2945] Wu, T., "The SRP Authentication and Key Exchange System", RFC 2945, September 2000.

[VIPR-RELOAD-USAGE] Rosenberg, J., Jennings, C., and M. Petit-Huguenin, "A Usage of Resource Location and Discovery (RELOAD) for Public Switched Telephone Network (PSTN) Verification", draft-rosenberg-dispatch-vipr-reload-usage-03 (work in progress), October 2010.

Appendix A. Release notes

This section must be removed before publication as an RFC.

A.1. Modifications between rosenberg-03 and rosenberg-02

- Nits.
- Shorter I-Ds references.
- Removed sentence saying that Tkey is converted to base64.
- Added ValExchange method and Domain attribute definitions.
- Fixed the last sentence of 7.2 - the ticket goes into the ticket element in the ServiceContent attribute.
- Expanded first usage of VCR initialism.
- Replaced any insteance of peerID by Node-ID.
- Rewrote the ABNF.

Authors' Addresses

Jonathan Rosenberg
jdrosen.net
Monmouth, NJ
US

Email: jdrosen@jdrosen.net
URI: http://www.jdrosen.net

Cullen Jennings
Cisco
170 West Tasman Drive
San Jose, CA 95134
USA

Phone: +1 408 421-9990
Email: fluffy@cisco.com

Marc Petit-Huguenin
Stonyfish

Email: marc@stonyfish.com

dispatch
Internet-Draft
Intended status: Standards Track
Expires: April 28, 2011

J.R. Rosenberg
jdrosen.net
C. Jennings
Cisco
M. Petit-Huguenin
Stonyfish

October 25, 2010

A Usage of Resource Location and Discovery (RELOAD) for Public Switched Telephone Network (PSTN) Verification
draft-rosenberg-dispatch-vipr-reload-usage-03

Abstract

Verification Involving PSTN Reachability (ViPR) is a technique for inter-domain SIP federation. ViPR makes use of the RELOAD protocol to store unverified mappings from phone numbers to RELOAD nodes, with whom a validation process can be run. This document defines the usage of RELOAD for this purpose.

Table of Contents

1. Introduction

This document relies heavily on the concepts and terminology defined in [VIPR-OVERVIEW] and will not make sense if you have not read that document first. As it defines a usage for RELOAD [P2PSIP-BASE], it assumes the reader is also familiar with that specification. The same DHT can also be used for a RELOAD SIP usage [P2PSIP-SIP].

2. ViPR Usage

The ViPR usage defines details for how the DHT is used for ViPR operations.

The ViPR usage defines Kind-ID 0x00000001. This Kind-ID is a dictionary entry. Its Resource-ID is defined through a transformation which takes an E.164 based number, and computes a Resource-ID as the least significant 128 bits of the SHA1 hash of the following string: Cat(CHOICE(null, "COPY", "COPY2"), number) That is, the Resource-ID is the hash of a string which is the concatenation of the number, prefixed with nothing, or the words "COPY1" or "COPY2".

For example, for number +14085555432:

Resource-ID = least128(SHA1("+14085555432"))

or

Resource-ID = least128(SHA1("COPY1+14085555432"))

or

Resource-ID = least128(SHA1("COPY2+14085555432"))

The object stored at this resource ID is a dictionary entry, which has a key and a value:

Object = {key,value}

Here, the key is formed by taking the Node-ID of the storing node in hex format, without the "0x", appending a "+", followed by the VServiceID in hex format, without the "0x". For example, if a peer with Node-ID

0x8f60f5eab753037e64ab6c53947fd532

receives a Publish with a VServiceID of

0x7eeb6a7036478351

The resulting key is:

8f60f5eab753037e64ab6c53947fd532+7eeb6a7036478351

Both parts of this key are important. Using the Node-ID of the node performing the store basically segments the keyspace of the dictionary so that no two peers ever store using the same key. Indeed, the responsible node will verify the signature over the stored data and check the Node-ID against the value of the key, to make sure that a conflict does not take place. The usage of the VService allows for a single ViPR server to service multiple clusters, and to ensure that numbers published by one cluster (using one VServiceID) do not clobber or step on numbers published by another cluster (using a different VServiceID). The responsible node does not verify or check the VServiceID.

When a node receives a Store operation for this usage, the data itself has a signature. The node responsible for storing the data must verify this signature; the certificate will always be included in the data and indicate which Node-ID is used. The responsible node must check that this Node-ID is included in the cert. If the signature verifies, the responsible node checks that the data model is a dictionary entry. The key must meet the format above. The responsible node must check that it is a 32 character sequence of numbers and letters a-f, followed by a +, followed by a 16 character sequence of numbers and letters a-f. If this checks, the key is split in half along the plus. The first 32 characters are considered a hex value and compared with the Node-ID used for the signature. If they match, it is good. Otherwise the Store operation is rejected. If they did match, next the responsible node checks the value. It must be a TLV, with the same format used by VAP, and it must contain a single Node-ID attribute. The Node-ID must match that used for the signature. If they don’t match, the Store operation is rejected. If they do match, the next step is a quota check.

For each peer that the responsible node is storing data for, it must maintain a count of the number of unique dictionary entries being stored for that Node-ID. For each resourceID, each key constitutes a unique dictionary entry. So if a peer is storing 5 resourceIDs, and at each of those 5, there are two keys whose first 32 bits correspond to a particular Node-ID, it means this node is currently storing 10 unique dictionary entries for that Node-ID.

It takes the StorageQuota configuration parameter for this DHT, which measures the amount of numbers a particular node can store. That value is multiplied by nine (a 3x factor to account for the

application-layer copies (COPY1 and COPY2), and another 3x factor for replicas). Then, an addition 3x factor is added for rounding to make sure that the probability is low that a rejection occurs due to imperfect distribution of resourceIDs across the ring. (Open Issue: need to adjust this multiplier - basically birthday problem!) and then divided by the fraction of the hashspace owned by this ViPR server. If the result is less than one, it is rounded up to two. This is the max number of unique entries that can be stored for this storing peer ID. If the ViPR server is not yet storing this many entries for that peer ID, the store is allowed.

The method for merging data after a partition follows the normal RELOAD rules around temporal ordering.

3. PeerID Shim

Because the ViPR implementation of RELOAD protocol makes use of the concept of multiple Node-ID on the same physical box, utilizing a single cert, the TLS handshakes alone are not sufficient to determine the entity on both sides of the TLS connection. As such, we will have a small "shim" type of protocol, which runs after TLS, but is not formally part of RELOAD.

When a node initiates a TLS connection towards another node, after the TLS completes, it sends this message. The message contains the Node-ID associated with this connection. The recipient gets this, and sends back a similar message, containing its Node-ID. Both sides will verify that, the Node-ID sent by the other side, are amongst the Node-IDs listed in the certificate. The connections are then stored in the connection tables, indexed by this Node-ID.

Furthermore, if, after this exchange, a node determines that it already has a connection in its connection table with that Node-ID on the far side, the older connection is closed. This is actually a critical security function! Without this, a user could clone ViPR servers utilizing the same certs, and each one can join the network.

Finally, once the exchange has taken place, the node compares the Node-ID from its peer with the current set of blacklisted Node-ID from the ACL that is distributed through the DHT. If the remote Node-ID appears on the list, the node closes the TCP/TLS connection immediately.

The reason we are using a non-reload message for this, is that we need to be 100% sure that this never propagates. It is strictly over a single connection and should never be routed. Indeed, had we not had this idea of multiple Node-ID in a single cert, this would have

effectively been accomplished through TLS. Alternatively, there is a TLS command for telling the other side who I expect them to be; however this is not implemented in older versions of OpenSSL, and so our shim forms an alternative to that which can be run on top of OpenSSL.

4. Security Considerations

TBD

5. IANA Considerations

TBD. Need to register items in IANA registries created by RELOAD.

6. References

6.1. Normative References

[P2PSIP-BASE]
Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD) Base Protocol", draft-ietf-p2psip-base-11 (work in progress), October 2010.

[VIPR-OVERVIEW]
Rosenberg, J., Jennings, C., and M. Petit-Huguenin, "Verification Involving PSTN Reachability: Requirements and Architecture Overview", draft-rosenberg-dispatch-vipr-overview-04 (work in progress), October 2010.

6.2. Informative References

[P2PSIP-SIP]
Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "A SIP Usage for RELOAD", draft-ietf-p2psip-sip-05 (work in progress), July 2010.

Appendix A. Release notes

This section must be removed before publication as an RFC.

A.1. Modifications between rosenberg-03 and rosenberg-02

- Nits.
- Shorter I-Ds references.
- Fixed the peerID and VServiceID to be hexadecimal.
- Fixed the description of the dictionary entry
- Fixed the description of the TLV.
- Used +1 408 555 prefix for phone numbers in examples.
- Replaced peerId by Node-ID
- Replaced resourceID by Resource-ID

Authors' Addresses

Jonathan Rosenberg
jdrosen.net
Monmouth, NJ
US

Email: jdrosen@jdrosen.net
URI: http://www.jdrosen.net

Cullen Jennings
Cisco
170 West Tasman Drive
San Jose, CA 95134
USA

Phone: +1 408 421-9990
Email: fluffy@cisco.com

Marc Petit-Huguenin
Stonyfish

Email: marc@stonyfish.com

dispatch
Internet-Draft
Intended status: Standards Track
Expires: April 28, 2011

J.R. Rosenberg
jdrosen.net
C. Jennings
Cisco
M. Petit-Huguenin
Stonyfish
October 25, 2010

Session Initiation Protocol (SIP) Extensions for Blocking VoIP Spam Using PSTN Validation
draft-rosenberg-dispatch-vipr-sip-antispam-03

Abstract

Verification Involving PSTN Reachability (ViPR) is a new technique for inter-domain federation of SIP calls. ViPR makes use of the PSTN as an introduction mechanism to verify the correctness of mappings from phone numbers to domains. The PSTN introduction mechanism can also be used as a technique for blocking spam - a SIP caller is only authorized when its calling domain has previously called that same number over the PSTN. This document describes an extension to SIP which enables authorization of SIP calls based on a prior PSTN introduction.

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 28, 2011.

Copyright Notice

Table of Contents

1. Introduction

The anti-spam tickets described in this specification are the key security mechanism in ViPR for mitigation of SPAM. The domain originating a call inserts a ticket in the SIP INVITE sent to the other domain. The Border Element in the domain receiving the call (see Figure 1) can check the ticket to ensure that this originating domain has been authorized by the terminating domain. This document relies heavily on the concepts and terminology defined in [VIPR-OVERVIEW] and will not make sense if you have not read that document first.

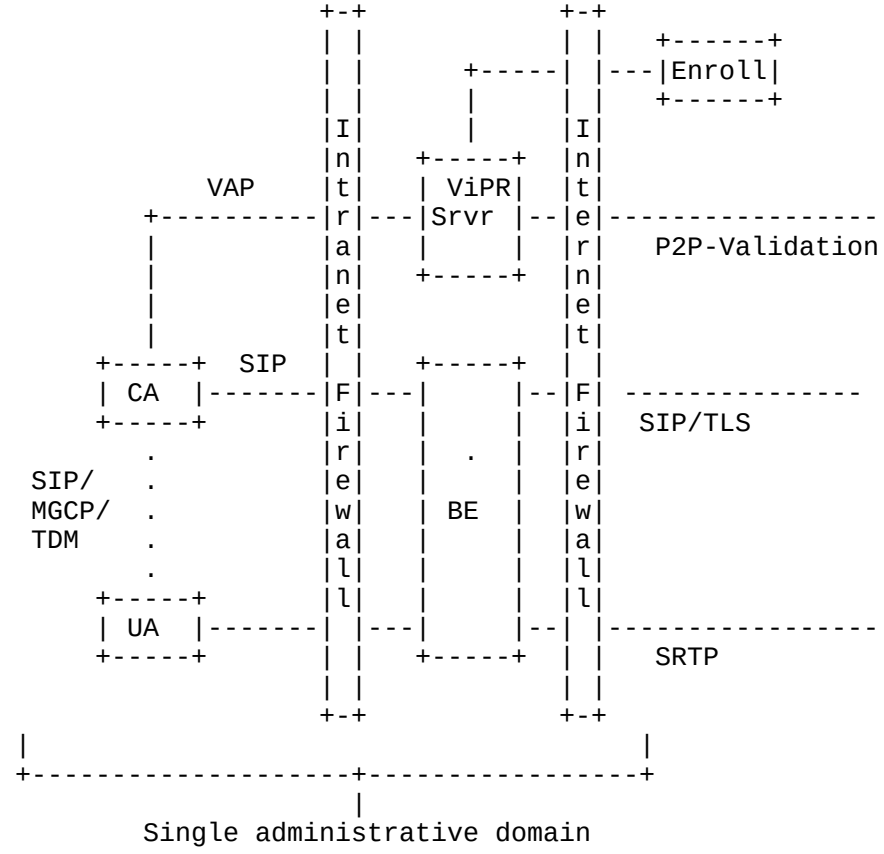


Figure 1: Architecture

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

3. Terminating Side Procedures

The Border Element will receive the TLS ClientHello which begins the TLS handshake. The Border Element will present its own configured cert. Once TLS handshaking is complete, the Border Element notes the domain from the SubjectAltName on the other side of the TLS connection, and associates it with that connection.

Next, the Border Element will receive an INVITE. This INVITE will contain a ticket in the X-Cisco-ViPR-Ticket header field value. The Border Element extracts this header field. This call flow assumes it is present. The Border Element parses it, and obtains the epoch value encoded in the ticket. This is matched against the current epoch value for the configured password. If they match, processing continues. The Border Element verifies the signature is correct. Next, it examines the start and stop time of the validity. If the current time is between the start and stop times, the check is passed. Next, the Border Element checks the granted-to domain in the ticket. It compares that domain against the domain name in the SubjectAltName of the peer on the other side of the TLS connection, as noted above. Next, it takes the Request-URI of the SIP INVITE. That will be of the form sip:+number@domain. If it is not in that form, and if the number does not begin with a plus, the request is dropped. The value, including the plus, is then compared to the number in the ticket. If it is equal, the check has passed. The Border Element leaves the header field in the request, but forwards to the Call Agent.

In addition, the Border Element will typically be configured to apply its SIP message validation logic, and enforce restrictions on the sizes of various SIP header fields. This provides an additional layer of security in case malicious SIP messages are sent.

The Border Element will also apply port forwarding in the case of NAT, so that the incoming request is forwarded to the appropriate Call Agent node.

The Call Agent will receive incoming SIP INVITEs. The Request-URI of the INVITE will contain an E.164 number as indicated by a leading plus. If the Request-URI is not an E.164, the request must be rejected with a 403. Only E.164 numbers can be accepted on a ViPR

trunk.

4. Originating Side Procedures

The routes stored to other domains in the Call Agent will each store a ticket to utilize with calls to that route. The Call Agent learns about these routes and the information needed to construct the ticket from the VAP protocol [VIPR-VAP]. When sending a SIP request to one of these domains, the Call Agent MUST include the ticket in any dialog forming request or request that is not in an existing dialog.

5. Tickets

This ticket is a sequence of characters. These MUST be placed into a X-Cisco-ViPR-Ticket SIP header field value. Consequently the format for this header field is:
Ticket = "X-Cisco-ViPR-Ticket" HCOLON ticket-val
ticket-val = 1*(alphanum / "-" / "_" / ".")

This header field MUST be utilized in all dialog forming requests and all out-of-dialog requests. It is not utilized in responses. The ticket-value is a modified base64 encoded version of an object that is composed of a series of TLVs. Each TLV is a 16 bit type, a 16 bit length, and a variable length value. The length field refers to the length of the value portion of the TLV, measured in bytes. The following TLV types are defined:

1. Ticket Unique ID: This TLV has a type of 0x0001. It contains a 128 bit ID that has a unique identifier for this ticket. The value MUST contain a 128 bit UUID defined by [RFC4122]. This TLV MUST be present. However at this time it is used for diagnostic purposes only.
2. Salt: This TLV has a type of 0x0002. It contains a value which MUST be at least 32 bits, and contains a random number. Its presence ensures that each ticket contains sufficient randomness. This TLV MUST be present.
3. Validity: This TLV has a type of 0x0003. It contains two 64 bit NTP times. The first is the start of the validity of the ticket, the next is the end time for the validity of the ticket. This TLV MUST be present.
4. Number: This TLV has a type of 0x0004. It contains a string which has an E.164 number, included the "+", which may be called using this ticket. The TLV has variable length. This TLV MUST be present.

5. Granting Node: This TLV has a type of 0x0005. It contains a 128 bit value which is the Node-ID of the node which granted the ticket. This TLV MUST be present.
6. Granting Domain: This TLV has a type of 0x0006. The domain which granted the ticket. A string, up to 256 characters, each of which must be a valid domain name character. The TLV has variable length. This TLV MUST be present.
7. Granted-To Domain: This TLV has a type of 0x0007. The domain to which the ticket is granted. A string, up to 256 characters, each of which must be a valid domain name character. The TLV has a variable length. This TLV MUST be present.
8. Epoch: This TLV has a type of 0x0008. It contains a 32 bit epoch value. It is used to select a key. This TLV MUST be present.
9. Integrity: This TLV has a type of 0x0009. It contains a 160 bit integrity value, computed using HMAC-SHA1. This TLV MUST be present and MUST be the last TLV in the object.

The base64 encoding uses the base64url encoding from RFC4648 [RFC4648], with the exception of the pad character, which is a "." instead of an "=". This ensures that the output is a valid SIP token.

To compute the MAC, the following is done. First, the key is obtained. The key is actually a 128 bit key, configured into the system. The key, P, is then used to compute Km:

Km = HMAC-SHA1(P, S | Epoch)

Based on PBKDF2 from PKCS #5 [RFC2898] with HMAC-SHA1 as PRF and iteration count of 1. Where S is the 32 bit salt and Epoch is the 32 bit Epoch, from the ticket. This produces a 160 bit Km. The MAC is then computed as another HMAC-SHA1, over the entire ticket up to but not including the Integrity itself, using Km as the key. This produces the 160 bit MAC.

6. Security Considerations

TBD

7. IANA Considerations

TBD - Register SIP Header

TBD - Form IANA registry for Ticket TLVs

8. Acknowledgements

Thanks to Patrice Bruno for his comments, suggestions and questions that helped to improve this document.

9. References

9.1. Normative References

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

[RFC2898] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification Version 2.0", RFC 2898, September 2000.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[VIPR-OVERVIEW] Rosenberg, J., Jennings, C., and M. Petit-Huguenin, "Verification Involving PSTN Reachability: Requirements and Architecture Overview", draft-rosenberg-dispatch-vipr-overview-04 (work in progress), October 2010.

9.2. Informative References

[VIPR-VAP] Rosenberg, J., Jennings, C., and M. Petit-Huguenin, "Verification Involving PSTN Reachability: The ViPR Access Protocol (VAP)", draft-rosenberg-dispatch-vipr-vap-03 (work in progress), October 2010.

Appendix A. Release notes

This section must be removed before publication as an RFC.

A.1. Modifications between rosenberg-03 and rosenberg-02

- Added terminology section.

- Nits
- Shorter I-Ds references.
- Changed issued-to to granted-to.
- Fixed the ABNF.
- The tickets is used in all dialog forming requests, not only INVITE.
- The Number TLV has a variable length.
- The Integrity TLV MUST be the last in the object.
- Fixed a discrepancy in the epoch length.

Authors' Addresses

Jonathan Rosenberg
jdrosen.net
Monmouth, NJ
US

Email: jdrosen@jdrosen.net
URI: http://www.jdrosen.net

Cullen Jennings
Cisco
170 West Tasman Drive
MS: SJC-21/2
San Jose, CA 95134
USA

Phone: +1 408 421-9990
Email: fluffy@cisco.com

Marc Petit-Huguenin
Stonyfish

Email: marc@stonyfish.com

dispatch
Internet-Draft
Intended status: Standards Track
Expires: April 28, 2011

J.R. Rosenberg
jdrosen.net
C. Jennings
Cisco
M. Petit-Huguenin
Stonyfish

October 25, 2010

Verification Involving PSTN Reachability: The ViPR Access Protocol (VAP)
draft-rosenberg-dispatch-vipr-vap-03

Abstract

Verification Involving PSTN Reachability (ViPR) is a technique for inter-domain SIP federation. ViPR hybridizes the PSTN, P2P networks, and SIP, and in doing so, addresses the phone number routing and VoIP spam problems that have been a barrier to federation. The ViPR architecture uses a server, the ViPR server, which performs P2P and validation services on behalf of call agents, which acts as clients to the server. Such an architecture requires a client/server protocol between call agents and the ViPR server. That protocol, defined here, is called the ViPR Access Protocol (VAP).

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference

material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 28, 2011.

Table of Contents

1. Introduction to ViPR

[VIPR-OVERVIEW] introduces a new technology, called Verification Involving PSTN Reachability (ViPR), which enables VoIP federation between domains, over the Internet. ViPR is a hybrid technology that combines together the PSTN, P2P networks, and SIP. In doing so, it addresses the phone number routing problem and anti-spam problems that have been the most significant barriers to widespread deployment of SIP inter-domain federation.

It is assumed that readers of this document have read and understood [VIPR-OVERVIEW].

One of the key protocols used in ViPR is the ViPR Access Protocol (VAP). VAP connects call agents, such as phones, SBCs and IP PBXs, to a ViPR server. This document defines the VAP protocol in detail.

2. Overview of VAP

A high level view on the ViPR architecture is shown in Figure 1. This architecture is discussed in more detail in [VIPR-OVERVIEW].

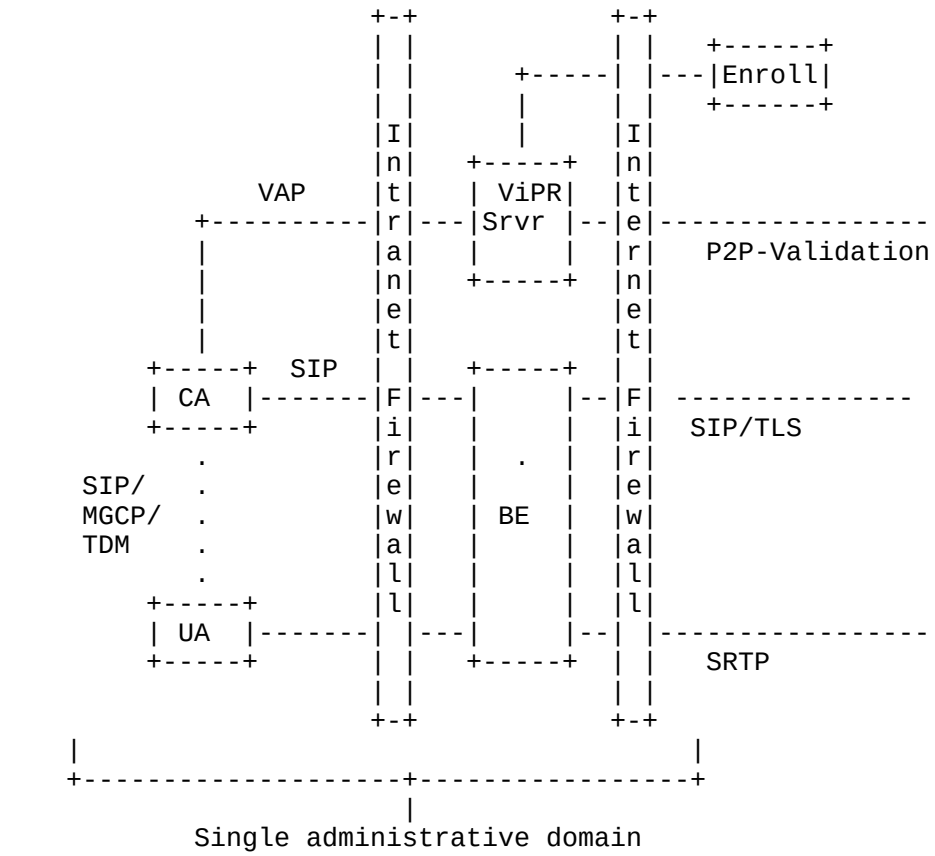


Figure 1: Architecture

A key component of this architecture is the ViPR server. The ViPR server is responsible for connecting to the P2P network, publishing phone numbers into that network, performing validation, and learning new routes. The ViPR server performs those functions on behalf of one or more call agents. This requires a protocol to run between the call agents and the ViPR server. This protocol is called VAP - the ViPR Access Protocol.

VAP is a client-server protocol that runs between the call agent and the ViPR server. VAP is a simple, binary based, request/response protocol. It utilizes the same syntactic structure and transaction state machinery as STUN [RFC5389], but otherwise is totally distinct from it. VAP clients initiate TCP/TLS connections towards the ViPR server. The ViPR server never opens connections towards the call agent. This allows the ViPR servers to run on the public side of NATs and firewalls.

Once the connections are established, the call agent sends a Register message to the ViPR server. This register message primarily provides authentication and connects the client to the ViPR server. VAP provides several messages for different purposes:

- Publish: The Publish message informs the ViPR server of service information. There are two types of Publishes supported in ViPR. The first is the ViPR Service (VService). This informs the ViPR server of the SIP URIs on the call agent and black and white lists used by the ViPR server to block validations. The ViPR server stores that information locally and uses it during the validation process, as described above. The second Publish is the ViPR Number Service. The ViPR server, upon receiving this message, performs a Store operation into the DHT.
- UploadVCR: This message comes in two flavors - an originating and terminating message. An originating UploadVCR comes from a call agent upon completion of a non-ViPR call to the PSTN. A terminating UploadVCR comes from an agent upon completion of a call received FROM the PSTN. The ViPR server behavior for both messages is very different. For originating UploadVCR, the ViPR server will store these, and at a random time later, query the DHT for the called number and attempt validation against the ViPR servers that are found. For a terminating UploadVCR, the ViPR server will store these, awaiting receipt of a validation against them.
- Subscribe: Call agents can subscribe for information from the ViPR server. There is one service that call agent can subscribe for: Number Service. When a new number is validated, the ViPR server will send a Notify to the call agent, containing the validated number, the ticket, and a set of SIP trunk URIs.
- Notify: The ViPR server sends this message to the call agent when it has an event to report for a particular subscription.

The VAP protocol provides authentication by including an integrity object in each message. This integrity message is the hash of the contents of the message and a shared secret between the ViPR server and the client. VAP can also be run over TLS, which enhances security further.

3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

4. VAP Message Structure

VAP messages follow the syntax and structure of Session Traversal Utilities for NAT (STUN) [RFC5389]. It also shares the same transaction model as STUN. However, aside from its common syntax and transaction model, STUN and VAP are unrelated.

VAP 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 RFC791 [RFC0791]. Unless otherwise noted, numeric constants are in decimal (base 10).

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

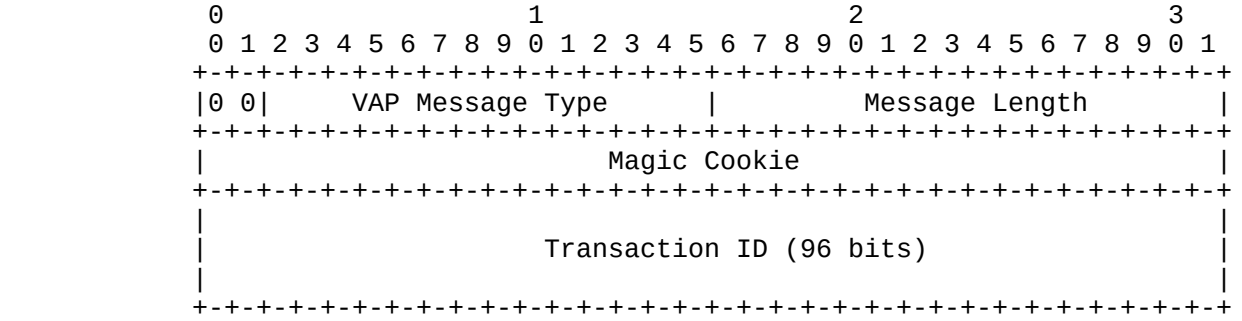


Figure 2: Format of VAP Message Header

The most significant two bits of every VAP message MUST be zeroes.

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

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

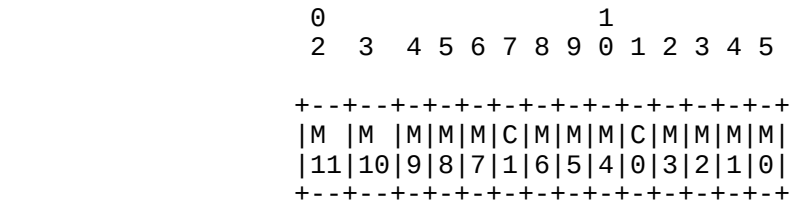


Figure 3: Format of VAP Message Type Field

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 0b10 is a success response, and a class of 0b11 is an error response. The method and class are orthogonal, so that for each method, a request, success response, error response and indication are defined for that method.

The magic cookie field MUST contain the fixed value 0x41666679 in network byte order (note that this is a different value than STUN).

The transaction ID is a 96 bit identifier, used to uniquely identify VAP transactions. For request/response transactions, the transaction ID is chosen by the VAP client for the request and echoed by the server in the response. The transaction ID MUST be uniformly and randomly chosen from the interval 0 .. 2**96-1, and SHOULD be cryptographically random. The client MUST choose a new transaction ID for new transactions. Success and error responses MUST carry the same transaction ID as their corresponding request.

The message length MUST contain the size, in bytes, of the message not including the 20 byte VAP header. Since all VAP attributes are padded to a multiple of four bytes, the last two bits of this field are always zero.

Following the VAP fixed portion of the header are zero or more attributes. Each attribute is TLV (type-length-value) encoded. The details of the attributes themselves is given in Section 10.3.

The methods defined in VAP, and their corresponding method values, are:

Method	Value
Register	0x001
Unregister	0x002
Publish	0x004
Unpublish	0x005
PublishRevoke	0x006
Subscribe	0x007
Unsubscribe	0x008
Notify	0x00a
UploadVCR	0x00b

Figure 4: VAP Methods

After the VAP header are zero or more attributes. Each attribute is TLV encoded, with a 16 bit type, 16 bit length, and variable value. Each attribute MUST end on a 32 bit boundary:

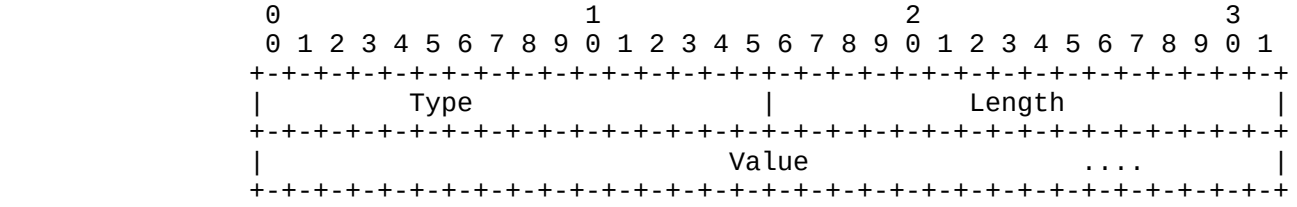


Figure 5: VAP Attributes

The Length refers to the length of the actual useful content of the Value portion of the attribute, measured in bytes. Since VAP 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 they are a multiple of 4 bytes. Such padding is only needed with attributes that take freeform strings, such as USERNAME. For attributes that contain more structured data, the attributes are constructed to align on 32 bit boundaries. The value in the Length field refers to the length of the Value part of the attribute prior to padding - i.e., the useful content. Consequently, when parsing messages, implementations will need to round up the Length field to the nearest multiple of four in order to find the start of the next attribute.

5. VAP Transactions

This section describes the general behavior of VAP transactions, regardless of the method.

5.1. Transport and Connection Management

VAP runs only over TCP. UDP is not supported. As a consequence, transactions are simple. For each transaction, the client sends a single request, and the server sends a response.

VAP can also be run over TLS. The server MUST implement TLS, and the client SHOULD utilize it. The TLS_RSA_WITH_AES_128_CBC_SHA ciphersuite MUST be implemented. The client MUST verify that the server certificate matches a configured value associated with the ViPR server that is to be used. The server MUST accept any certificate from the client. Client authentication is performed using a simple digest technique.

Reliability of VAP over TCP and TLS-over-TCP is handled by TCP itself, and there are no retransmissions at the VAP 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.

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 VAP requests to send over that connection, and;
- has no outstanding subscriptions

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). 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.

5.2. Requestor Procedures

Though VAP is a client/server protocol, the ViPR server can asynchronously send requests towards the client call agent. As such, this section defines transaction rules in terms of the requestor (the

entity sending the request) and the responder (the entity receiving the request).

5.2.1. Generating Requests

The requestor MUST construct a request message based on the syntax in Section 4. The message class MUST be a request. The method depends on the method of the request.

The requestor MUST add a MESSAGE-INTEGRITY, REALM and USERNAME attribute to the request message. The USERNAME contains a string which is the provisioned username identifying the client to the VAP server. The REALM attribute MUST have the value of "ViPR". The MESSAGE-INTEGRITY is computed as described in Section 10.3.3. That computation relies on a 16-byte key. The 16-byte key for MESSAGE-INTEGRITY HMAC 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, (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. Note that the password itself never appears in the message.

This format for the key was chosen so as to enable a common authentication database for SIP, which uses digest authentication as defined in RFC 2617 [RFC2617].

The request will contain other attributes depending on the method.

5.2.2. Receiving Responses

All responses MUST first be authenticated by the requestor. Authentication is performed by first comparing the Transaction ID of the response to an outstanding request. If there is no match, the requestor MUST discard the response. Then the requestor SHOULD check the response for a MESSAGE-INTEGRITY attribute. If not present, it MUST discard the response, except for error responses with response codes 431 and 436. If MESSAGE-INTEGRITY is present, the requestor computes the HMAC over the response. The key that is used MUST be same as used to compute the MESSAGE-INTEGRITY attribute in the request.

If the computed HMAC matches the one from the response, processing continues. If the response was discarded, in cases where the failure is due to an implementation error, this will cause timeout of the transaction.

If the response is an Error Response, the requestor checks the response code from the ERROR-CODE attribute of the response. For a

400 (Bad Request) response code, the requestor SHOULD generate an alarm (a notification here refers to some kind of indication, sent to the administrator of the system, indicating an error condition. Notification mechanisms include SNMP alarms, logs, syslog, and so on, and are a matter of local implementation) containing the reason phrase.

For a 431 (Integrity Check Failure) response code, this is typically caused by a mis-provisioning of the password. The requestor SHOULD generate an alarm and SHOULD NOT retry.

If the requestor receives a 436 (Unknown Username) response, it means that the username it provided in the request is unknown. This is typically due to a provisioning error, a consequence of a mismatched username. The requestor SHOULD generate an alarm.

The requestor MUST ignore any attributes from the response whose attribute type were not understood by the requestor.

5.3. Responder Behaviors

5.3.1. Receiving Requests

A responder will receive requests on an existing TCP connection, either one initiated by the client, or the one accepted by the ViPR server.

If a responder cannot process a request because the request does not meet the syntactic requirements necessary for the processing described below, the responder SHOULD reject the request with an error response and include an ERROR-CODE attribute with a response code of 400 (Bad Request). If the request is so malformed that a response cannot be generated, the request is just dropped. Error codes for specific failures are not provided, since these failures would not be seen in a functionally correct system. The protocol only provides error codes for errors that can arise due to misconfiguration or network error. Note, however, that a responder SHOULD NOT verify that a requestor has generated the request in full compliance to this specification; it should only validate what it needs to perform the processing described for handling the request.

First, the responder authenticates the request. The request will contain a USERNAME, REALM, and MESSAGE-INTEGRITY attribute. If the USERNAME is unknown, the responder generates an error response with an ERROR-CODE attribute with a response code of 436 (Unknown Username). The response MUST include the REALM, but MUST omit the MESSAGE-INTEGRITY attribute.

The responder computes the HMAC over the request. If the computed HMAC differs from the one from the MESSAGE-INTEGRITY attribute in the request, the responder MUST generate an error response with an ERROR-CODE attribute with a response code of 431 (Integrity Check Failure). This response MUST include a REALM but MUST omit the MESSAGE-INTEGRITY attribute.

The responder MUST ignore any attributes from the request whose attribute type were not understood by the responder.

5.3.2. Sending Responses

To construct the response the responder follows the message structure described in Section 4. The message type MUST indicate either a success response or error response class and MUST indicate the same method as the request. The responder MUST copy the transaction ID from the request to the response.

The attributes that get added to the response depend on the type of response.

When sending an error response, the server MUST add an ERROR-CODE attribute containing the error code. The reason phrase is not fixed, but SHOULD be something suitable for the error code.

All responses except for an error response with ERROR-CODE of 431 and 436 will contain a MESSAGE-INTEGRITY attribute. All responses will contain a REALM attribute. The computation of the message integrity is based on the same username value present in the request (along with its corresponding password); however the response SHOULD NOT contain the USERNAME attribute.

All responses MUST be sent on the same TCP connection on which the request was received. If this connection has closed, the responder MUST NOT open a new connection in order to try to send the response. The transaction is considered failed in this case.

6. State Model

The state model for VAP is shown in Figure Figure 6. This state is built up as a consequence of the primary messages which build state on the ViPR server: Register, Publish, UploadVCR and Subscribe.

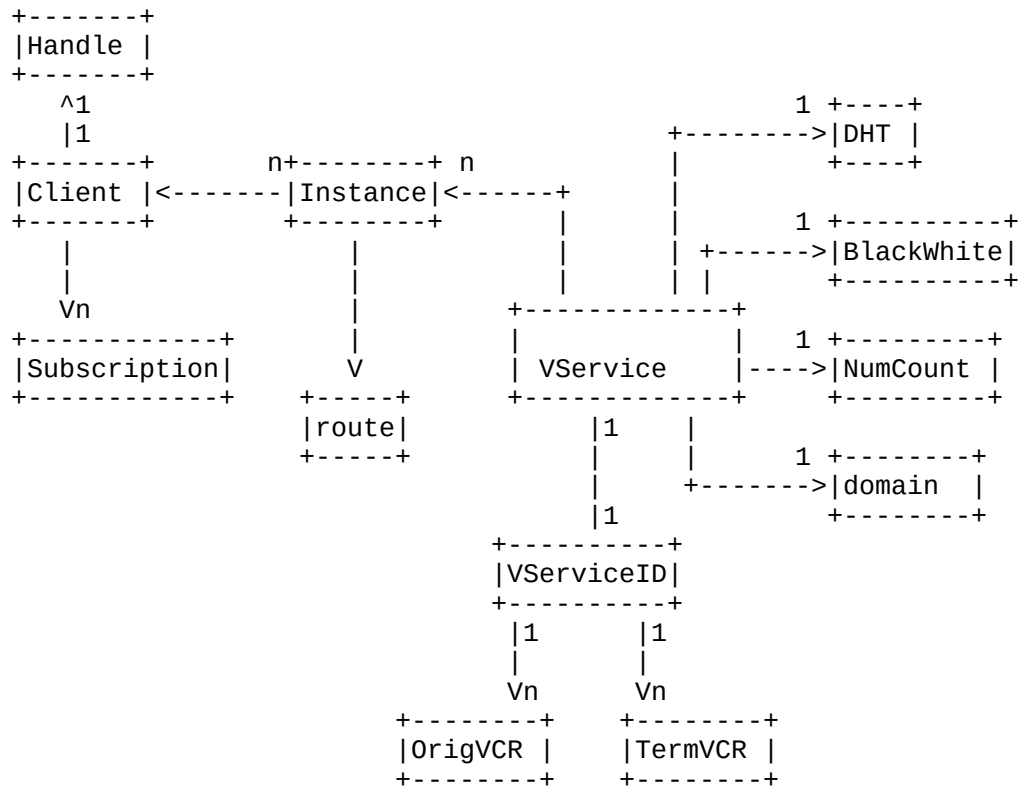


Figure 6: VAP State Model

It is important to understand that the ViPR client publishes two unique sets of information to the ViPR server:

1. The set of numbers that are reachable by the client through a particular ViPR service,

2. The set of ViPR services

Both of these are uploaded from the client to the ViPR server using a VAP Publish operation. The ViPR clients have the concept of a "ViPR Service" (not to be confused with ViPR server). A ViPR service is a unique instance of ViPR processing in a call agent - and is associated with a specific DHT, specific routes, specific domain, specific set of numbers to use, and specific set of policies governing operation. When a client publishes a number, it is always associated with a specific ViPR service, or VService. Multiple clients can publish the same VServices, and they will differ only in

the routes associated with that VService, as each client will have its own route to reach the same VService.

The ViPR server actively tracks the association of clients, VServices, routes, DHTs, BlackWhite lists, and VServiceIDs. Number publications and VService publications are differentiated from each other by different serviceID values in attributes in the Publish request. To be thoroughly confusing, this serviceID is not the same as a VServiceID. ServiceID refers to whether something is a VService publication or number publication, and is an enumerated value, whereas a VServiceID is an instance ID for a particular VService. The ViPR server only actually stores the VService publications; when receiving a Publish for a number service, the corresponding data is written directly to the DHT and then forgotten by the ViPR server. The ViPR server doesn't take any responsibility for removing the state or for keeping it fresh. All of this is the responsibility of the ViPR client. Consequently, VAP itself is not responsible for maintaining this information.

Firstly, when a client connects, it will Register to the ViPR server. That creates an instance of the client object, which is assigned a unique handle that identifies it. The client object is one of the key pieces of state (ViPR service being the other). All subsequent messaging from the client includes that Client-Handle, allowing the ViPR server to immediately determine the client associated with the messaging.

The client can issue subscriptions for services over its connection to the ViPR server. The ViPR server remembers the set of subscriptions from that client.

The VService publication builds the next large block of state. When a VService publication is received from a client, the ViPR server creates the VService object if it didn't have one yet for that VServiceID. Each distinct instance of a VService publication gets linked to it, and each distinct instance is, in turn, associated with one or more routes. Each route has a SIP URI, but the internal structure of the route is opaque to the ViPR server. It parses no deeper than the route element itself; the contents are not parsed, examined or checked by the ViPR server. This allows for future extensibility on how call routing is done. The VService itself has a numberCount, domain, BlackWhite list and DHT, all of which are learned from the VService publication. The VServiceID is 1-1 associated with each VService.

Finally, each UploadVCR, whether it is originating or terminating, contains a VServiceID as well. This binds it to a particular VService. It is important to note that, the linkage from VCRs to

VServices is indirect, through the VServiceID. This allows a temporary outage to break all client connections, which will delete the VService objects, but keep the VCRs and the VServiceIDs. When the clients reconnect, the VServices are rebuilt, along with their IDs, and once again can be linked to the VCRs.

When the VAP connection terminates, the client object and subscription state from the corresponding client is destroyed. Any instances of a VService from that client are destroyed. If there are no longer any instances of the VService left, the VService itself is destroyed. The VCRs are not affected by the termination of a connection from a client.

When the client TCP connection breaks or keepalives cease to be sent, the ViPR server will remove the registration, subscription and VServiceID to SIP trunk/DHT mappings. Similarly, on the client side, if the TCP connection breaks, the client must create a new TCP connection, register without a handle, subscribe and performs its VService publications.

The VAP state above is, in addition, utterly and completely orthogonal to the state of the DHT itself. That state is driven through number service publications, which cause storage operations into the DHT.

7. Protocol Versioning

Each version of VAP has a major and minor version number. This specification describes major version 1, minor version 0. It is anticipated that the protocol may require updating in the future.

If an update can be done such that an older client will work with a newer server, and an older server with a newer client, this MUST be done using an increase in the minor version number within the major version. This would typically include bug fixes and minor extensions. If a protocol change is such that it cannot be understood by previous servers and clients, this MUST be done using an increase in the major version number of the protocol.

This specification further requires that, in addition to the most recent version of the protocol they understand, a client MUST understand the previous major version number. For example, a client supporting version 2.1 would also need to support version 1.0.

The protocol version number is included in client register messages, and negotiation as part of that exchange.

This allows for a graceful upgrade procedure. When a new version of the protocol is to be rolled out, the clients are upgraded first, each in turn. When they are upgraded, they'll come back, but during registration, notices that the servers only support a previous major version. The clients thus switch to the previous version of the protocol. Once all of the clients are updated, the servers can be updated. When the clients connect to them, they will utilize the newest version of the protocol.

8. ViPR Client Procedures

8.1. Discovery

VAP provides no discovery mechanism. The client must be provisioned with the domain names and/or IP addresses and ports of its ViPR servers. Typically, a client will be provisioned with two servers - a primary and a backup.

8.2. Registration

Once a TCP connection is established, the client MUST perform a registration. This applies to all TCP connections held by the client for purposes of high availability.

The client constructs a Register request based on the basic client procedures in Section 5.2. In addition, the client MUST include the Client-Name attribute. This field is used strictly for debugging purposes and indicates the name of the client to the server.

If the client is registering for the first time towards this ViPR server, the registration MUST omit the Client-Handle attribute.

If the client is registering for the first time towards this ViPR server (and thus there was not Client-Handle attribute), the client MUST include a Protocol-Version attribute in the request. This includes the major and minor version number of the most recent version of the protocol supported by the client. For purposes of extensibility, in addition to their current version of the client protocol, a client MUST support the previous major version as well.

The client MUST include the Client-Label attribute in the request. However, it is not used and its contents are arbitrary.

Once constructed, the client sends the Register request to the ViPR server. The response is processed using the general techniques in Section 5.2. Assuming a success response is ultimately received, it indicates that the client has successfully registered. This response

will contain a Client-Handle attribute. The client MUST retain this handle and store it for the lifetime of the clients connection to the server. The response will also contain the Keepalive attribute, which tells the client how often it needs to keepalive its registration to the server.

If the response to the initial Register request (one without a Client-Handle) is an error response with an ERROR-CODE attribute with a response code of 478, it means that the server does not support the major protocol version signaled by the client. The client MUST extract the Protocol-Version attribute from the error response. This attribute indicates the major and minor versions supported by the server. Based on the principles in Section 7, the client will be able to support a version of the protocol that has a major protocol version matching the one in the Protocol-Version attribute of the error response. The client MUST switch to this version of the protocol, and then MUST generate a new Register request (without a Client-Handle), indicating a Protocol-Version equal to the new, lower version of the protocol.

If the response to the initial Register request (one without a Client-Handle) is an error response with an ERROR-CODE attribute with a response code of 477, it means that the server believes that the client has already registered on this connection. There has been a state synchronization error. The client SHOULD generate an alarm, and then tear down the TCP connection. It MUST open a new TCP connection, and then generate a fresh Register request (without a Client-Handle) over that connection.

If the Register message was for an existing connection (and thus a keepalive), and thus included the Client-Handle attribute in the request, but the response was a Register Error response with an ERROR-RESPONSE with a response code of 471, the client MUST consider this a failure of the connection. It SHOULD attempt a new connection and a new Register, but without a Client-Handle.

During an initial Register (one that omits Client-Handle), the client MUST NOT generate any subsequent requests until that Register transaction completes.

If the TCP connection fails, the client needs to reconnect and create a new registration without the handle, and furthermore, resubscribe and republish as needed. In other words, on the client side, the lifetime of the handle is equal to the lifetime of the TCP connection. The server also holds onto the handle as long as the connection is active. However, it will also watch for refreshes of registrations, and if it doesn’t see one fast enough, remove the client registration, the handle, and state received from that client,

as well.

8.3. Unregistering

A Client that wishes to terminate its connection gracefully does so using the Unregister request. This request is first constructed as described in Section 5.2. Once constructed, the client MUST add the Client-Handle attribute to the request, and send it to the ViPR server.

If the response was an error response and was of type 400, it means that the client did not construct the request properly. The client MUST NOT retry unless it changes the content or set of attributes in the request to match the requirements defined here.

If the response was an error response with an ERROR-RESPONSE attribute with a response code of 471, the client MUST consider this a failure of the connection. It indicates a synchronization error between client and server. The client SHOULD generate an alarm.

If the response was an error response and was of type 474, it means that the client sent an Unregister request on a TCP connection but had not yet registered. If the client had registered, there has been some kind of synchronization error. The client SHOULD generate an alarm.

In all cases, success or error responses, the client MUST consider all subscriptions to this server terminated, and consider all published VServices to this server as unpublished. The client MUST terminate the TCP connection after the response has been received.

8.4. Publishing Services

Publish requests inform the ViPR server of information from the client. There are two types, VService publications and number publications. These differ in the value of the ServiceIdentity attribute.

All publications contain a ServiceContent attribute which contains an XML element that defines the service. The schema for the ServiceContent element depends on whether the publication is a VService or number publication.

The Publish request MUST contain a ServiceVersion attribute. This attribute is a version number that increments by at least one every time a particular service (identified by a unique VService, instance, service ID and sub-service ID value) changes in any way. If the service data different from the previous published value, the

ServiceVersion attribute MUST increase. If the service data is the same as the previous published value, the ServiceVersion SHOULD stay the same, but MAY increase. Consequently, increasing version numbers are not a guarantee that there was a change; only that lack of increasing version number is a guarantee that there was no change.

If a client loses track of the previous version number of the service (due, for example, to a restart), it MUST choose a new instance ID and then it can reset the ServiceVersion.

Finally, the Publish Request MUST contain a ServiceContent attribute. This attribute contains the actual service data. Its actual structure and syntax are a function of the service and sub-service.

If the response was an error response and was of type 472, it means that the client didn't increment the sequence number. More likely, it indicates that the client has inadvertently forgotten the version number of the service and gotten out of sync with the server. The client SHOULD choose a new instance ID for this service, withdraw the old one, and publish the new one.

If the response was an error response and was of type 474, it means that the client sent a Publish request on a TCP connection but had not yet registered. If the client hadn't registered, it MUST now do so. If it had registered, there has been some kind of synchronization error. The client SHOULD generate an alarm. Then, it MUST generate a new register (without the Client-Handle), flushing all subscriptions.

If the response was an error response and was of type 400, it means that the client did not construct the request properly. The client MUST NOT retry unless it changes the content or set of attributes in the request to match the requirements defined here.

If the response was a success, the publication has been accepted.

8.4.1. VService

The VService indicates the critical information for the VService identified by the VService ID. Typically, a call agent will run on many servers, each of which is listening for SIP traffic on a specific IP address and port. Each such IP address and port forms a particular instance of the VService, and represents an alternative SIP destination for receiving incoming calls. The instance ID is a unique identifier, within the scope of the VServiceID, which identifies that call agent server.

The additional information placed into the VService publication will

not vary amongst different instances. That information is:

- The DHT that the client wishes its numbers to be published into for this VService. This must always be the name of the public ViPR DHT, which is "Quetzalcoatl".
- The domain name associated with this VService, e.g., example.com. This domain name is used by the ViPR server at the end of the validation process.
- The set of routes which can be used to reach a SIP server on the call agent instance. Each route contains a SIP URI, in addition to extensions to allow for future advanced routing. This parameter of the VService data is instance specific.
- a black/white list of domains. These are used by the ViPR server during the validation protocol. The white list contains the set of domains that this domain wishes to only federate with. The black list contains the list of domains that this domain does not wish to federate with.
- A count of the number of phone numbers being published for this VService. This is used for quota management on the ViPR server.

Note that the VService does not contain phone numbers. VService information is not stored into the DHT by the ViPR server. It is stored locally on the ViPR server and used to support the validation protocol.

Section 10.1 defines the XML schema for the object included in the Publish request.

The SIP URI is constructed as follows:

1. The scheme MUST be sip.
2. The user part MUST be an identifier which is unique to this agent and is identical for all instances of that call agent. For example, if a call agent consists of two servers for purposes availability, and either can be used, the user part will be identical in the SIP URI published by each server.
3. The domain part MUST be the domain associated with this call agent, and MUST match certificates that the domain can obtain.
4. There MUST be a port and it MUST be the port on which incoming SIP invites can be received.
5. There MUST be an maddr URI parameter, and it MUST contain the IP address or hostname of the instance of the call agent server.
6. The transport URI parameter MUST be present and MUST be TCP.

There will be one or more URI per each instance of the call agent. The IP address in the URI MUST be a publicly reachable one. If the call agent is to be reached through a border element, the IP address and port on the border element MUST be used here.

The use of the IP address in the maddr parameter allows the system to operate without DNS support.

An example document for a VService on the public DHT is:

```
<?xml version="1.0" encoding="UTF-8"?>
<service-description
xmlns="http://www.cisco.com/namespaces/saf-uc" id="has7gg"
xmlns:vt="http://www.cisco.com/namespaces/viprtrunk"
schemaVersion="1.0">
<tns:vservice xmlns:tns="http://www.cisco.com/namespaces/viprtrunk">
 <tns:DHTname>Quetzalcoatl</tns:DHTname>
 <tns:DIDCount>3670</tns:DIDCount>
 <tns:domain>example.com</tns:domain>
 <tns:whitelist>
   <tns:domain>example.com</tns:domain>
   <tns:domain>foo.edu</tns:domain>
 </tns:whitelist>
 <tns:route>
   <tns:SIPURI>
sip:17ahhs7zpaksux6z5==@example.com:2371;maddr=1.2.3.4;transport=tcp
   </tns:SIPURI>
 </tns:route>
 </tns:vservice>
</service-description>
```

Figure 7: Example ServiceContent

The ViPR client SHOULD publish each ViPR trunk service to both its primary and backup ViPR server, for purposes of HA.

8.4.2. ViPR Number Service

The ViPR number service is used to publish the numbers that are associated with the VService. It is published as a separate service due to the differing state requirements associated with the numbers. For the VService, the ViPR server stores the information and does not actually publish it into the DHT. For ViPR number service, the ViPR server immediately writes the data into the DHT and doesn't actually store it locally. The ViPR server does not refresh the data in the DHT on its own, nor does it withdraw the data from the DHT when the client disconnects. The ViPR client is responsible for refreshing the data in the DHT by periodically refreshing each of its numbers in each DHT. The numbers in the DHT have a configurable expiration. Consequently, the ViPR client has to refresh the data prior to the expiration. There is no way in VAP to remove a number from the DHT;

it is merely left to expire.

The ViPR client SHOULD publish each service to both its primary and backup ViPR server, for purposes of HA. Next, the client constructs a ViPR number service advertisement. Unlike VService advertisements, which utilize an XML object in the ServiceContent attribute, number services utilize only VAP attributes. The Publish message will contain a ServiceIdentity attribute and a CalledNum attribute. The VServiceID of the ServiceIdentity attribute indicates the VService for this number, and is used by the ViPR server to determine which DHT to publish into. The CalledNum attribute contains the number to be published into the DHT. The ServiceVersion attribute is not present.

8.5. Updating the VService

A client can change the VService information at any time. Typically, changes in the black or white list will require an updated VService publication, as will changes in the set of servers listening for incoming SIP traffic.

To update a VService, the client modifies its service description, and creates a new Publish request. This request is first formed as described in Section 4. This request MUST contain the ServiceIdentity attribute, identifying the service to be modified. The request MUST also contain the ServiceContent attributes, containing the relevant information for the service.

The request MUST contain a ServiceVersion attribute. That version number MUST be at least one higher than the version number in the previous publication for the same service (as identified by service ID, subservice ID and instance).

If the response was an error response and was of type 472, it means that the client didn’t increment the sequence number. More likely, it indicates that the client has inadvertently forgotten the version number of the service and gotten out of sync with the server. The client SHOULD choose a new instance ID for this service, unregister, reconnect, re-register, and republish.

If the response was an error response and was of type 474, it means that the client sent a Publish request on a TCP connection but had not yet registered. If the client hadn’t registered, it MUST now do so. If it had registered, there has been some kind of synchronization error. The client SHOULD generate an alarm. Then, it MUST generate a new register (without the Client-Handle).

If the response was an error response and was of type 400, it means

that the client did not construct the request properly. The client MUST NOT retry unless it changes the content or set of attributes in the request to match the requirements defined here.

If a client is no longer capable of receiving SIP requests at the URI it previously published, it should remove its VService by sending an Unpublish request.

8.6. Uploading VCRs

When the call agent initiates or receives a call that goes towards the PSTN, whether it be through a PSTN gateway or through a SIP trunk to a service provider, the call agent MUST send an UploadVCR request to its primary server ViPR server. It SHOULD send its terminating UploadVCRs to its secondary ViPR server, and SHOULD NOT send its originating UploadVCRs to its secondary. The UploadVCR request is first constructed like any other VAP request. This means it will contain the USERNAME, REALM, and MESSAGE-INTEGRITY attributes.

In addition, it MUST contain a CallingNum, CalledNum, StartTime and StopTime attribute. The CallDirection attribute is set as described in Section 10.3.14.

The UploadVCR request MUST contain a ServiceIdentity attribute. The serviceID is 100, the subservice ID is 3 (ViPR number service) and the VService ID must identify the VService for which this UploadVCR is associated. The instance is arbitrary and are ignored by the ViPR server.

If the response was an error response and was of type 474, it means that the client sent a UploadVCR request on a TCP connection but had not yet registered and had not yet sent a VService publication with a VServiceID matching that of the UploadVCR. If the client hadn’t registered and published a matching VService, it MUST now do so. If it had, there has been some kind of synchronization error. The client SHOULD generate an alarm. Then, it MUST disconnect, generate a new register (without the Client-Handle) and a new VService publication.

If the response was an error response and was of type 400, it means that the client did not construct the request properly. The client MUST NOT retry unless it changes the content or set of attributes in the request to match the requirements defined here.

8.7. Subscribing to Number Service

In order to learn about validated numbers, a ViPR client MUST subscribe for the ViPR Number Service. The client should subscribe

to just its primary ViPR server.

To create a subscription, the client creates a Subscribe request. The request is formed as described in Section 4. It MUST NOT be sent if the client has not previously generated a successful Register request on this connection.

Each initial Subscribe request MUST omit the SubscriptionID attribute; that attribute is only used when withdrawing a subscription. The client MUST include a ServiceIdentity attribute in the request. The service ID MUST be 101, the subserviceID MUST be 3, the VServiceID MUST be the VServiceID for the VService from which learned numbers are desired, and the instance value MUST be all ones. This will cause the client to receive notifications upon validated numbers learned as a consequence of an UploadVCR for that VService.

8.8. Unsubscribing to Services

A client MAY terminate a subscription at any time. To do that, it sends an Unsubscribe request. This request MUST contain the SubscriptionID attribute identifying the subscription to be terminated. Note that this unsubscription will affect only the subscription identified by the subscription ID. Other subscriptions will continue to be in effect.

The client MAY generate additional Unsubscribe requests while the transactions for previous Subscribe, Publish or Unpublish requests are in progress. By definition a client can only Unsubscribe a subscription for which it had already received a successful response to a Subscribe request that created the subscription.

If the response was an error response and was of type 474, it means that the client sent a Subscribe request on a TCP connection but had not yet registered. If the client hadn’t registered, it MUST now do so. If it had registered, there has been some kind of synchronization error. The client SHOULD generate an alarm. Then, it MUST generate a new register (without the Client-Handle).

If the response was an error response and was of type 476, it means that the client sent an Unsubscribe request for a subscription which does not exist. The client SHOULD generate an alarm, since a synchronization error has occurred. It should however proceed as if the withdrawal was successful.

If the response was an error response and was of type 400, it means that the client did not construct the request properly. The client MUST NOT retry unless it changes the content or set of attributes in the request to match the requirements defined here.

8.9. Receiving Notify

The ViPR server will generate a Notify request when a new number and route are learned. It will send this Notify request to all clients which have subscribed to the corresponding VService.

Once the client has received a successful response to its Subscribe request, the client MUST be prepared to receive Notify requests on the TCP connection to its ViPR server. When the client receives a Notify request, it searches for the SubscriptionID attribute in the request. This informs the client of the subscription that this notification is associated with. If this subscriptionID is known to the client, it proceeds. Otherwise, it MUST generate a Notify error response with a 476 response code in an ERROR-RESPONSE attribute. When this occurs, there has been a synchronization error between the client and server in the set of valid subscriptions. This event SHOULD be alarmed, and the contents of the Notify not used.

The Notify request will contain a ServiceIdentity attribute and a ServiceContent attribute, in addition to the standard authentication attributes and the SubscriptionID attribute. The ViPR client must verify that the ServiceIdentity has service 100, subservice 3. It looks at the instance value, and checks that the topmost 64 bits of the instance contain a VServiceID that matches one for which the ViPR client is currently interested in learning about. The ViPR client then extracts the contents of the ServiceContent attribute. This will be an XML object, formatted as described below.

The client SHOULD store the phone number, SIP URI and Ticket. When receiving a future call to that phone number, it SHOULD send a SIP INVITE request to the SIP URI and include the ticket in an X-Cisco-ViPR-Ticket header field.

8.10. Receiving PublishRevoke

The PublishRevoke method is defined only for the VService, not for the Number Service. The ViPR server will send a PublishRevoke for a VService if the corresponding DHT is no longer available. The request will contain the ServiceIdentity attribute, which indicates the specific VService and instance that are being withdrawn. If these correspond to a known VService, the client should consider that service deactivated, and periodically try to republish it.

9. ViPR Server Procedures

9.1. Connection Establishment

The ViPR server MUST be prepared to receive incoming TCP or TLS connections on a configure port. Whether or not TCP or TLS is used, is a configured property of that port.

9.2. Registration

The purpose of registrations is to create VAP client objects, which represent a VAP connection and contain the state described in Section 6, and then link those with a TCP connection. Each VAP connection can be considered a client object, linked to one and only one TCP connection at a time.

The first request that the server will receive over the TCP connection will be a Register request. This request is first processed as described in Section 5.3. Assuming those procedures succeed, the server checks for the Client-Handle attribute in the Register request. If present, the server checks if it currently has a client state object with that handle. If the client object was already bound to another TCP connection, that other TCP connection MUST be closed by the server, and then the new TCP connection MUST be bound to the client object.

If the Register request had a Client-Handle attribute, but there were no client objects with that handle, the server MUST generate an error response and MUST include an ERROR-CODE attribute with a response code of 471. This is due to a state synchronization error between the client and server. The server SHOULD generate an alarm.

If the Register did not have a Client-Handle attribute, it is a request to create a client object. The server examines the Protocol-Version attribute from the request. If the major version indicated in the attribute is higher than the version supported by the server, the server MUST reject the Register request with an error response and include an ERROR-CODE attribute with a response code of 478. That error response MUST include a Protocol-Version attribute that contains the major and minor protocol versions supported by the server.

Next, the server MUST create a new client object, and allocate a new Client-Handle for it. The Client-Handle MUST be unique amongst all other Client-Handles known to this server, across all clients that are connected to it.

If the registration succeeds, the server sends a success response. This response MUST include the Client-Handle attribute containing the handle created by the server. The response MUST include a Keepalive

attribute, indicating the time in milliseconds that the server will need to see traffic from the client in order to continue to maintain the client object.

9.3. Unregistration

The client can gracefully disconnect by using an Unregister request.

If the server receives an Unregister request on a TCP connection, it first looks for the client object bound to that connection. If there is no client object bound to it, it means that the client has sent an Unregister request prior to registering, or there has been some kind of synchronization error. The server MUST respond with an error response, and MUST include an ERROR-CODE attribute with a response code of 474.

Otherwise, if the client object is known to the server, it MUST generate a success response. Once it does, the server MUST destroy the client, its associated subscriptions, and published VService instances. It then sets a timer equal to thirty seconds. If the client has not closed the TCP connection bound to this client object, the server MUST close the TCP connection.

If, as a consequence of the deletion of those VService instances, there are no longer any instances left for a VService, that VService and its associated data (BlackWhite, DHT, numberCount) are removed.

Note that unregistration does not ever remove VCRs.

9.4. Publication

Behavior depends on whether the publication is for the VService or the ViPR number service.

The ViPR server extracts the ServiceIdentity attribute. If the value is not one of the following:

1. ServiceID is 101 and SubserviceID is 3.
2. ServiceID is 101 and SubserviceID is 4

the ViPR server sends a 400 response.

9.4.1. VService

If the Publish request is for service 100, sub-service 4, it indicates that this was for the VService. The ViPR server first looks for the client object bound to that connection. If there is no client object bound to it, it means that the client has sent a

Publish request prior to registering, or there has been some kind of synchronization error. The server MUST respond with an error response, and MUST include an ERROR-CODE attribute with a response code of 474.

The ViPR server extracts the contents of the ServiceContent attribute. This will be an XML object structured as defined in Section 10.1. It also extracts the VServiceID and Instance values from the ServiceIdentity attribute.

First, the ViPR server checks if it has any VService objects with the VServiceID from the publish.

- If it does, it replaces the BlackWhite, numberCount, domain, and DHTName parameters of that VService with the ones from the publish. Next, it checks to see if the instance is currently an instance associated with that VService:
-
  - If it is, the route elements for that instance are replaced with the route values from the publish.
  - If it is not, a new instance object is created, associated with the client and the VService, and is linked with the route values from the publish.
- If it does not, it creates a new VService object, and associates it with the values of the BlackWhite, numberCount, domain, and DHTName parameters of the VService. Next, it creates a new instance, associates it with the VService, The route values from the publish are associated with that instance.

ViPR server sends a Publish success response. The ViPR server looks for all other ViPR services in the same DHT as the one from this Publish, it sums up their numberCounts, and includes that value in the "current" field of the Quota attribute in the Publish response. Since there is a limit on the count of the numbers that can be published into the DHT, this mechanism allows the ViPR server to inform the clients about the total usage across all clients of this ViPR server. Note further, that since the ViPR server itself does not have local memory of the numbers it stored into the DHT, the ViPR server cannot determine how many numbers have been placed into the DHT for a particular VService. That information is known only to the client. That is why the client informs the ViPR server of how many numbers it has published as part of the VService publication.

The ViPR server places its configured per-DHT limit for that DHT into the "limit" field in the Quota attribute in the Publish response. This tells the clients the maximum count of phone numbers which can be published.

The ViPR server includes a DHTLifetime attribute in the response. This attribute indicates the amount of time that data will remain in the DHT prior to be expunged. This is a configured property of the DHT.

9.4.2. ViPR Number Service

If the server receives a Publish request for service 100, sub-service 3, it indicates that this was for the ViPR Number Service. The ViPR server first looks for the client object bound to that connection. If there is no client object bound to it, it means that the client has sent a Publish request prior to registering, or there has been some kind of synchronization error. The ViPR server MUST respond with an error response, and MUST include an ERROR-CODE attribute with a response code of 474. The ViPR server extracts the VServiceID from the ServiceIdentity attribute. It checks that, for that VServiceID, there is a VService object currently being stored. If not, the ViPR server MUST respond with an error response, and MUST include an ERROR-CODE attribute with a response code of 474.

Next, the ViPR server extracts the number from the CalledNum attribute. The ViPR server extracts the DHT from the VService object associated with the VServiceID from the Publish. For the number, the ViPR server takes the number and treats it as an ASCII string, called the suffix seed.

Next, the ViPR server generates two additional strings. The first is formed by taking the suffix seed, and prepending the string "COPY1". The second is formed by taking the suffix seed and prepending the string "COPY2".

Each of the three values is passed through the SHA-1 hash function, producing 160 bits. The least significant 128 bits of this are taken. Those 128 bits, for each of the three values, form the Resource-ID against which a STORE is to be performed. Three separate stores are performed in order to provide security in the DHT. Each store operation writes an object into the DHT whose value is a dictionary (or map) entry.

Conceptually:

Store(Resource-ID, object)

Where Resource-ID is the 128 bit Resource-ID computed above. The stored object is a dictionary entry which has a key and a value:

Object = {key,value}

Here, the key is formed by taking the peerID of the storing node in hex format, without the "0x", appending a "+", followed by the VServiceID in hex format, without the "0x". For example, if a peer with peerID 0x8e60f5fab753037f64ab6c53947fd532 receives a Publish with a VServiceID of 0x7eeb6a7036478351, the resulting key is:

8e60f5fab753037f64ab6c53947fd532+7eeb6a7036478351

Both parts of this key are important. Using the peerID of the node performing the store basically segments the keyspace of the dictionary so that no two peers ever store using the same key. Indeed, the responsible node will verify the signature over the stored data and check the peerID against the value of the key, to make sure that a conflict does not take place. The usage of the VService allows for a single ViPR server to service multiple call agents, and to ensure that numbers published by one call agent (using one VServiceID) do not clobber or step on numbers published by another call agent (using a different VServiceID). The responsible node does not verify or check the VServiceID.

In this version of the protocol, only one of the three stored objects is read. Three are stored to allow an enhancement in the future, which will read all three and use a simple voting algorithm to handle inconsistencies in the results. In this way, if a malicious node returns no result or fakes the result, as long as the remaining two results are retrieved, the validation process can continue. This means that the compromise of a single node has, with only extremely low probability (order Log(N)/N where N is the number of nodes in the ring) of being able to disrupt validation against a number.

The value of the dictionary entry is a sequence of TLV attributes, with the same format used by VAP. In this case, it is a single attribute, the peerID attribute. This attribute is populated with the peerID of the ViPR server in the DHT into which the STORE is being performed. The reason for using the TLV construct is to provide extensibility in the contents of the DHT. In the future, if needed, new ViPR nodes can add additional data, each with a specific attribute type. Older nodes will ignore any unknown attributes and go right for the peerID attribute, while newer ones can process the new and old attributes.

The Store operations are paced into the DHT at a fixed rate. The ViPR server maintains a queue. This queue is filled with store requests. The ViPR server services that queue at a fixed, provisioned rate, the Store Rate Limit. When serviced, the next Store operation in the queue is serviced. Because transactions from clients are pipelined, there can only be as many Store operations in the queue as there are simultaneously connected clients, times three

(three Stores per Publish, one Publish at a timer per client). The Publish is then responded to with a success response. Note that, a success response is not sent until all three Store operations have been performed. If there is a failure due to inability to store into the DHT, the server returns a 481 error response. Note that a ViPR server cannot disambiguate the first Publish for a service and an updated Publish. It performs identical processing for each.

Note further that, the DHT itself will replicate each of the three stored values, producing a total of nine copies of each number into the DHT.

9.5. Unpublish

The ViPR client can only Unpublish for the VService.

The ViPR server extracts the VServiceID and instance from the ServiceIdentity in the Unpublish. It checks to see if there is an instance with that ID associated with the VService with that VServiceID. If there is, it removes the instance object and its associated SIPURI. If, as a consequence, there are no longer any instances associated with the VService, it deletes the VService object and its associated attributes.

9.6. Subscribe

If the server receives a Subscribe request on a connection, it first looks for the client object bound to that connection. If there is no client object bound to it, it means that the client has sent a Subscribe request prior to registering, or there has been some kind of synchronization error. The server MUST respond with an error response, and MUST include an ERROR-CODE attribute with a response code of 474.

The ViPR server checks that the ServiceIdentity from the request. If verifies that the ServiceID is 101 and the SubServiceID is 3. Any other combination causes the server to return a 400 response. The subscription is to the VServiceID identified in the ServiceIdentity attribute.

If the ServiceIdentity is valid, the server MUST create a new subscription object. It MUST allocate a SubscriptionID for this subscription. This ID MUST be unique across all SubscriptionIDs associated with this client. The subscription MUST be linked with the client object. It is not permitted for there to be multiple subscriptions from a client with identical VServices since each subscription is for a unique service/subservice/VServiceID/instance, the ViPR server can hash these to get a 32 bit SubscriptionID, or

assign them sequentially and store the associations.

The ViPR server then checks the VServiceID from the ServiceIdentity attribute. The ViPR server adds a subscription object to the client object, and associates it with a SubscriptionID and the VServiceID which is being watched.

The server then generates a success response to the Subscribe request. It MUST include the SubscriptionID attribute in the response, identifying this subscription.

9.7. Unsubscribe

If the server receives an Unsubscribe request on a connection, it first looks for the client object bound to that connection. If there is no client object bound to it, it means that the client has sent an Unsubscribe request prior to registering, or there has been some kind of synchronization error. The server MUST respond with an error response, and MUST include an ERROR-CODE attribute with a response code of 474.

Next, the server extracts the SubscriptionID attribute from the request. If it contains a SubscriptionID not known to the server, there has been a synchronization error. The server MUST reject the Unsubscribe request with an error response and MUST include an ERROR-CODE attribute with a value of 476.

Assuming the SubscriptionID is known, the server MUST remove the subscription object from the client object, and destroy it. The server will therefore no longer send notifications associated with this subscription. The server MUST respond to the Unsubscribe request with a success response.

9.8. UploadVCR

The ViPR server first processes the request like any other VAP request, specifically it will perform the message integrity check and follow associated procedures.

If the UploadVCR was received on a TCP connection but the client had not yet registered over that connection, it is an error and the ViPR server returns a 474. If the client had registered, but the VServiceID from the ServiceIdentity doesn’t match a known VService, the UploadVCR is rejected with a 474.

Otherwise, the ViPR server extracts the CallDirection, StartTime, StopTime, CallingNum and CalledNum attributes, and stores them. Further processing depends on whether it was an originating or

terminating UploadVCR.

9.8.1. Originating

Once stored, the ViPR server starts timer Tv. Tv is selected as a random number, in seconds, starting from 30 and ending at the maximum validation time, which is a configured parameter of the ViPR Server for the DHT associated with the VService. The validation request - which includes the VCR - is stored until that timer fires. The validation request includes the details from the UploadVCR (calling, called numbers, start and stop time), along with the VService associated with the UploadVCR.

When the timer fires, the ViPR server examines the called party number. This number will be a plus followed by N digits. Using this number, it forms a lookup key K. K is equal to the least significant 128 bits of the SHA1 hash of the called party number in string form, including the + sign. Next, the ViPR server extracts VService associated with the VCR. It checks to see if this VService is currently being published. If so, it performs a lookup into the DHT using key K. Each DHT node has a queue on read transactions. These lookups are queued because the node has, per-DHT, a limit on the rate at which it will perform read requests.

Once the lookup request comes to the top of the queue and it can be serviced, the resulting fetch will be a result, a no-match, or a timeout. If there is a no-match or timeout, ViPR server processing is complete.

If there is a result, the ViPR server will now have all of the dictionary entries associated with the Resource-ID. Each dictionary entry is a key and a value. The key is the concatenation of a peerID and VServiceID, and the value is a set of TLV attributes. The ViPR server parses each dictionary entry as a sequence of TLV attributes, and extracts the first TLV value whose type is peerID (type 0x2008). From this, the ViPR server obtains a set of {peerID, VServiceID}s.

The ViPR server SHOULD perform validation, using the validation protocol [VIPR-PVP]. A ViPR server MAY use any algorithm of its choosing to determine whether a number should be validated once, many times, or not at all. When the ViPR server is satisfied that a number has been sufficiently validated, it SHOULD send a Notify. Furthermore, during validation, the ViPR server SHOULD compare the domain of the learned number with the blacklist for the VService associated with the matching VCR. If the domain is on the blacklist or not on the whitelist, a Notify SHOULD NOT be sent.

If a Notify is to be sent as a consequence of a validation success,

the ViPR server looks to see if there is currently a subscription from a client whose VServiceID matches the one from the VCR that triggered the validation that is causing the notification. For each matching one, it sends a Notify message. The ServiceContent in the Notify contains a ValInfo XML containing the SIPURI and ticket learned from the validation. It also contains the full E.164 number of the called number which validated.

9.8.2. Terminating

When the ViPR server receives a terminating UploadVCR, it stores the information, awaiting the receipt of a validation query. This information MUST be stored for a minimum whose value is a configured property of the DHT.

9.9. Sending Notify

The ViPR server MUST NOT send a Notify until it had already sent a response to the Subscribe message that created the subscription, for which the Notify is being sent.

When a Notify is to be sent, it must contain the SubscriptionID attribute associated with the subscription on which the notification is being sent. This will differ for each client that is subscribed.

The Notify MUST contain the ServiceIdentity attribute, containing service 100, subservice 3, a VServiceID for the VService on which the number was learned, and an instance ID whose instance is all ones. The content of the ServiceContent attribute is an XML document, which is the scrubbed document from the ValExchange response. An example document is:

```
<?xml version="1.0" encoding="utf-8"?>
<valinfo xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:noNamespaceSchemaLocation="valinfo.xsd">
 <number>+17325552496</number>
 <ticket>7hasd88a7sd6a6d7989xkk8g7a6sdq78ekaz</ticket>
 <route>
   <SIPURI>
     sip:17ahhs$7zpaksux6z5==@example.com:2371;maddr=1.2.3.4
   </SIPURI>
 </route>
 <route>
   <SIPURI>
     sip:17ahhs$7zpaksux6z5==@example.com:2371;maddr=1.2.3.5
   </SIPURI>
 </route>
</valinfo>
```

Figure 8: Example Notify XML

9.10. Sending PublishRevoke

The ViPR server is only permitted to PublishRevoke the VService; it cannot withdraw Number Service publications. It should PublishRevoke published VServices when the corresponding DHT is no longer available. If this should happen, the ViPR server sends a PublishRevoke for each VService that was published which utilized the DHT which is no longer available. That PublishRevoke MUST include a ServiceIdentity attribute indicating the VServiceID and instanceID of the PublishRevoke service. Furthermore, it SHOULD include a ServiceContent attribute with the corresponding service description; this is used strictly for diagnostic purposes and is not needed by the client. Once sent, the ViPR server removes that instance of that VServiceID from its internal state.

10. Syntax Details

10.1. XML Schema for VService

This document is included in publications for the ViPR service. Note its target namespace.

```
<?xml version="1.0" encoding="utf-8"?>

<xs:schema xmlns="http://www.cisco.com/namespaces/saf-uc"
 attributeFormDefault="unqualified" elementFormDefault="qualified"
 targetNamespace="http://www.cisco.com/namespaces/saf-uc"
 xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

```
      <xs:element name="service-description">
        <xs:complexType>
          <xs:choice>
     <xs:element name="vservice">
       <xs:complexType>
         <xs:sequence>
           <xs:element name="DHTname" type="xs:string" />
           <xs:element name="DIDCount" type="xs:integer" />
           <xs:element minOccurs="1" maxOccurs="1" name="domain"
             type="xs:string" />
           <xs:choice minOccurs="0" maxOccurs="1">
             <xs:element
               xmlns:q1="http://www.cisco.com/namespaces/viprtrunk"
               name="blacklist" type="q1:whiteOrBlackList" />
             <xs:element
               xmlns:q2="http://www.cisco.com/namespaces/viprtrunk"
               name="whitelist" type="q2:whiteOrBlackList" />
           </xs:choice>
           <xs:sequence minOccurs="1" maxOccurs="unbounded">
             <xs:element
               xmlns:q1="http://www.cisco.com/namespaces/viprtrunk"
               name="route" type="q1:routeType" />
           </xs:sequence>
           <xs:any minOccurs="0" maxOccurs="unbounded"
             namespace="##other"
             processContents="lax" />
         </xs:sequence>
       </xs:complexType>
     </xs:element>
          </xs:choice>
          <xs:attribute name="schemaVersion" type="xs:string"
            use="required" />
          <xs:attribute name="id" type="xs:string" use="required" />
        </xs:complexType>
      </xs:element>
     <xs:complexType name="whiteOrBlackList">
        <xs:sequence minOccurs="1" maxOccurs="unbounded">
          <xs:element name="domain" type="xs:string" />
        </xs:sequence>
      </xs:complexType>
      <xs:complexType name="routeType">
        <xs:sequence>
          <xs:element minOccurs="1" maxOccurs="unbounded" name="SIPURI"
            type="xs:string" />
          <xs:any minOccurs="0" maxOccurs="unbounded"
            namespace="##other" />
        </xs:sequence>
      </xs:complexType>
```

```
</xs:schema>
```

Figure 9: VService XML Schema

10.2. XML Schema for ValInfo

This document is passed from the terminating ViPR server to the originating, containing the ticket, routes and number which was validated. The originating ViPR server verifies this and passes it to the client in VService notifications.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema elementFormDefault="qualified"
   xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="valinfo">
   <xs:complexType>
     <xs:sequence minOccurs="0" maxOccurs="unbounded">
       <xs:element minOccurs="1" maxOccurs="1" name="number"
         type="xs:string" />
       <xs:element minOccurs="1" maxOccurs="1" name="ticket"
         type="xs:string" />
       <xs:element minOccurs="1" maxOccurs="unbounded" name="route"
         type="routeType" />
       <xs:any minOccurs="0" />
     </xs:sequence>
   </xs:complexType>
 </xs:element>
 <xs:complexType name="routeType">
   <xs:sequence>
     <xs:element minOccurs="1" maxOccurs="unbounded" name="SIPURI"
       type="xs:string" />
     <xs:any minOccurs="0" maxOccurs="unbounded"
       namespace="##other" />
   </xs:sequence>
 </xs:complexType>
</xs:schema>
```

Figure 10: ValInfo XML Schema

10.3. VAP Attributes

This section enumerates the attributes used by VAP. The attribute names and corresponding types are:

Attribute Name	Type
USERNAME	0x0006
MESSAGE-INTEGRITY	0x0008
REALM	0x0014
ERROR-CODE	0x0009
Client-Name	0x1001
Client-Handle	0x1002
Protocol-Version	0x1003
Client-Label	0x1005
Keepalive	0x1006
ServiceIdentity	0x1007
ServiceVersion	0x100b
ServiceContent	0x100c
SubscriptionID	0x100e
CallDirection	0x2001
StartTime	0x2002
StopTime	0x2003
CallingNum	0x2004
CalledNum	0x2005
peerID	0x2008
Quota	0x200a
DHTLifetime	0x200b

Figure 11: VAP Attributes

10.3.1. USERNAME

The USERNAME attribute is used for authentication. It identifies the shared secret used in the message integrity check. Consequently, the USERNAME MUST be included in any request that contains the MESSAGE-INTEGRITY attribute.

The value of USERNAME is a variable length opaque value of UTF-8 characters. Note that, as described above, if the USERNAME is not a multiple of four bytes it is padded for encoding into the VAP message, in which case the attribute length represents the length of the USERNAME prior to padding.

10.3.2. REALM

The REALM attribute is present in requests and responses. It contains text which meets the grammar for "realm" as described in RFC 3261 [RFC3261], and will thus contain a quoted string (including the quotes).

The value of this attribute MUST always be "ViPR".

10.3.3. MESSAGE-INTEGRITY

The MESSAGE-INTEGRITY attribute contains an HMAC-SHA1 [RFC2104] of the message. The MESSAGE-INTEGRITY attribute can be present in any message type. Since it uses the SHA1 hash, the HMAC will be 20 bytes. The text used as input to HMAC is the message, including the header, up to and including the attribute preceding the MESSAGE-INTEGRITY attribute. That text is then padded with zeroes so as to be a multiple of 64 bytes. The MESSAGE-INTEGRITY attribute MUST be the last attribute in the message.

The 16-byte key for MESSAGE-INTEGRITY HMAC 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, (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. Note that the password itself never appears in the message.

Since the hash is computed over the entire message, it includes the length field from the message header. This length indicates the length of the entire message, including the MESSAGE-INTEGRITY attribute itself. Consequently, the MESSAGE-INTEGRITY attribute MUST be inserted into the message as the last attribute (with dummy content) prior to the computation of the integrity check. Once the computation is performed, the value of the attribute can be filled in. This ensures the length has the correct value when the hash is performed.

10.3.4. ERROR-CODE

The ERROR-CODE attribute is present in error responses. It is a numeric value in the range of 100 to 699 plus a textual reason phrase encoded in UTF-8, and is consistent in its code assignments and semantics with [RFC3261] and [RFC2616]. The reason phrase is meant for user consumption (typically freeform fields in alarms and logs), and can be anything appropriate for the response code. Recommended reason phrases for the defined response codes are presented below.

To facilitate processing, the class of the error code (the hundreds digit) is encoded separately from the rest of the code.

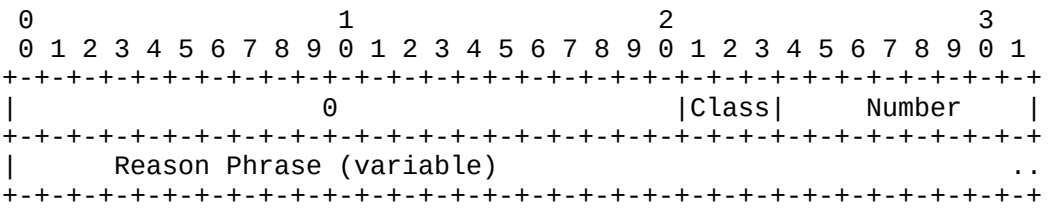


Figure 12: ERROR-CODE Syntax

The class represents the hundreds digit of the response code. The value MUST be between 1 and 6. The number represents the response code modulo 100, and its value MUST be between 0 and 99.

If the reason phrase has a length that is not a multiple of four bytes, it is padded for encoding into the message, in which case the attribute length represents the length of the entire ERROR-CODE attribute (including the reason phrase) prior to padding.

The following response codes, along with their recommended reason phrases (in brackets) are defined at this time:

400 (Bad Request): The request was malformed. The requestor should not retry the request without modification from the previous attempt.
431 (Integrity Check Failure): The request contained a MESSAGE-INTEGRITY attribute, but the HMAC failed verification. This could be a sign of a potential attack, or misconfiguration of the password .
436 (Unknown Username): The username was not known. This was likely due to a misconfiguration.
471 (Bad Client Handle): The client handle provided in the Register request is not known.
472 (Version Number Too Low): The client published a service whose version was lower than the currently held one by the server.
474 (Unregistered): The client tried an operation, such as publish or subscribe, but it has not yet registered.
476 (Unknown Subscription): The referenced subscription does not exist.
477 (Already Registered): The client tried an initial Register request, but it is already registered.
478 (Unsupported Protocol Version): The server does not support the protocol version requested by the client.
481 (Publication Failed): The publication was attempted but could not be performed due to an error reaching the DHT. The client should try again.

10.3.5. Client-Name

The Client-Name attribute is included the Register request. It contains a textual description, in UTF-8, of the software being used by the client, including 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 Client-Name is variable length. If the value of Client-Name is not a multiple of four bytes, it is padded for encoding into the VAP message, in which case the attribute length represents the length of the attribute prior to padding. However, it MUST be less than 255 characters and MUST be at least one character long.

It is RECOMMENDED that it be constructed as:

<vendor>/<product>/<version>/<hostname or IP>

Where version includes major, minor and build.

10.3.6. Client-Handle

This attribute has a 32 bit value, representing an unsigned integer to be used as the client handle.

10.3.7. Protocol-Version

This attribute is 32 bits, consisting of two 16-bit unsigned integers, representing the major and minor version numbers of the protocol:

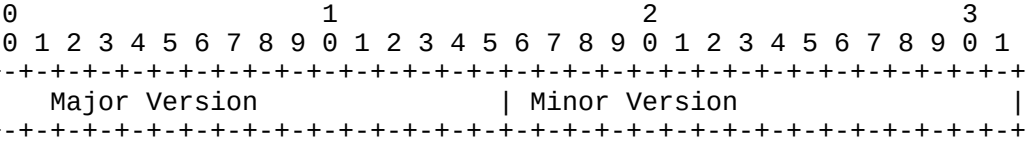


Figure 13: Protocol-Version Syntax

10.3.8. Client-Label

This attribute is a UTF-8 string, which MUST be between 1 and 255 characters. It is not used by this specification.

10.3.9. Keepalive

This attribute is a 32 bit unsigned integer, representing the number of milliseconds that the server will retain client state after the last message from the client has been received.

10.3.10. ServiceIdentity

The format of the ServiceIdentity attribute is:

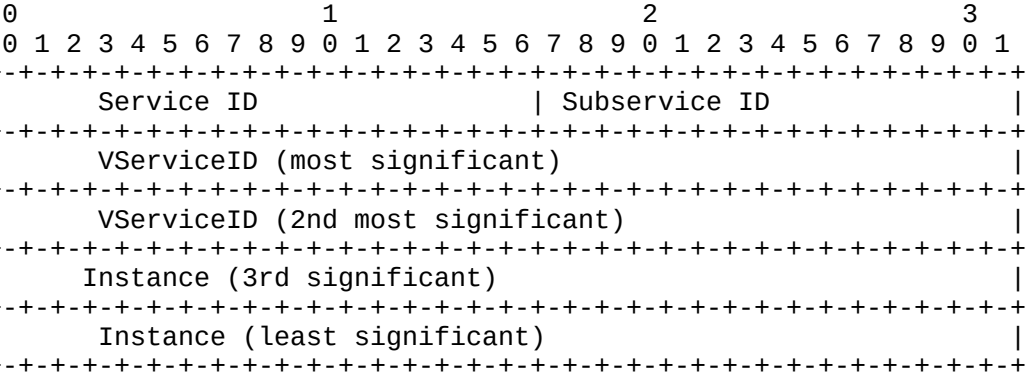


Figure 14: ServiceIdentity Attribute

The value of serviceID must always be 101. A Subservice value of 4 indicates VService publications. A subservice value of 3 indicates number publications.

10.3.11. ServiceVersion

The ServiceVersion field is a 32 bit unsigned integer. It contains the version number for the service advertised in the Publish request. It always increments by at least one for each change in the service.

10.3.12. ServiceContent

The ServiceContent is the actual content of the service definition. It is an arbitrary number of bytes. If the number of bytes of content are not a multiple of four, the content is padded with arbitrary data so that it is a multiple of four. The value of the length field of the attribute is the length prior to padding.

The ServiceContent MUST be less than 32k, despite the fact that the length field of the attribute itself would allow content up to 64k.

10.3.13. SubscriptionID

The SubscriptionID is present in successful responses to Subscribe and in Unsubscribe messages. It contains an identifier for the subscription. It is a unique handle, unique within all subscriptions between the client and this server. It is an unsigned 32 bit integer. It is also present in Notify and Withdraw requests.

10.3.14. CallDirection

This attribute is a 32 bit unsigned integer. A value of 0 indicates a received call. A value of 1 indicates a sent call. Other values are reserved and not valid in this version of the protocol.

10.3.15. StartTime

The start and is a 64 bit NTP time value. The start time is measured in the following way:

1. For calls sent to the PSTN (i.e., originated by this call agent), the start time is measured from the instant of the receipt of the call acceptance message indicating that the called party answered the call. For SIP, this would correspond to receipt of the 200 OK to the original SIP INVITE.
2. For calls received from the PSTN, (i.e., received by this call agent), the start time is measured from the instant of transmission of the call acceptance message towards the PSTN, indicating that the called party answered the call. For SIP, this would correspond to transmission of the 200 OK to the original SIP INVITE.

10.3.16. StopTime Attribute

The stop time is a 64 bit NTP value and is measured in the following way:

1. For the call agent which terminates the call, it corresponds to the transmission of the call termination message towards the PSTN. For SIP, this corresponds to the transmission of a SIP BYE request.
2. For the call agent which receives the termination, it corresponds to the receipt of the call termination message from the PSTN. For SIP this corresponds to the receipt of a SIP BYE request.

10.3.17. CallingNum Attribute

The calling party number MUST be expressed in fully qualified E.164 format, and the attribute is a string with variable length.

The calling party number is complicated. This is because this value is often munged and modified by the PSTN as it traverses the network. Fortunately, ViPR does not depend on it being delivered or being correct, but when it is delivered it improves security. Its presence is also needed for validating numbers which connect to multiple users, such that multiple calls to that number are often in progress at the same time. For example, 800 numbers.

For the originating call agent, the value is the E.164 number of calling party number delivered to the PSTN. For the terminating call agent, the value is E.164 normalized value of the caller ID received from the PSTN. This will require that national numbers delivered over a PRI are normalized to include their country code.

10.3.18. CalledNum Attribute

The called party number MUST be expressed in fully qualified E.164 format, and it is represented in the attribute as a string with variable length. The following rules apply for computation of the called party number:

For the call agent which initiates the call, the called party number is the E.164 number, including the leading plus, of the target of the call. Of course, this may not (and is probably not) the same as the digit sequence dialed by the calling party. The originating call agent MUST normalize this number to E.164 format based on its local dialing rules.

For the call agent which receives the call, the called party number is the E.164 number, including the leading plus, of the target of the call. Of course, this may not (and is probably not), the same as the called party number as delivered by the PSTN. It is likely that country codes, for example, are omitted from the message delivered by the PSTN. It is the responsibility of the terminating call agent to reconstruct the E.164 number of the called party.

10.3.19. Quota Attribute

This attribute consists of two 32 bit values. The first is the quota limit, which is the total number of numbers that can be published by this and other call agents attached to this ViPR server into this DHT. The second is the current total number of numbers being published by this and other call agents attached to this ViPR server into this DHT. If the current value is less than the quota value, everything is fine. Once it exceeds it, the DHT is likely to begin dropping entries and the admin needs to reduce the number of numbers being published.

10.3.20. DHTLifetime Attribute

This attribute is a 32 bit unsigned integer. It indicates the number of seconds that data written into the DHT will remain in the DHT prior to being expunged.

11. Security Considerations

11.1. Outsider Attacks

VAP prevents against traditional outsider attacks by means of TLS along with password-based digest authentication. That mechanism MUST be implemented by clients and servers and SHOULD be used.

11.2. Insider Attacks

Of much more concern are attacks whereby the client is authenticated, but it misuses the VAP connection to attack the overall system.

The principal attack to be considered is where an attacker injects false numbers by sending Publish requests for the number service containing numbers that the client doesn't own. This attack is the fundamental security problem that ViPR overall addresses with the validation mechanism, and so that attack is handled outside of VAP.

Another potential attack is a flooding attack where a client sends a large amount of numbers into the DHT. This attack is prevented by the distributed quota mechanism within the ViPR RELOAD usage, and thus prevented outside of VAP. Similarly, an attacker might try to DOS the ViPR network by sending a large volume of reads or writes into the DHT. This is prevented by means of the rate control mechanisms enforced by the ViPR server.

12. IANA Considerations

There are no IANA considerations associated with this specification.

13. References

13.1. Normative References

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

[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.

[RFC5389] Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for NAT (STUN)", RFC 5389, October 2008.

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

[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.

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

[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.

[VIPR-PVP]
Rosenberg, J., Jennings, C., and M. Petit-Huguenin, "The Public Switched Telephone Network (PSTN) Validation Protocol (PVP)", draft-rosenberg-dispatch-vipr-pvp-03 (work in progress), October 2010.

13.2. Informative References

[VIPR-OVERVIEW]
Rosenberg, J., Jennings, C., and M. Petit-Huguenin, "Verification Involving PSTN Reachability: Requirements and Architecture Overview", draft-rosenberg-dispatch-vipr-overview-04 (work in progress), October 2010.

Appendix A. Release notes

This section must be removed before publication as an RFC.

A.1. Modifications between rosenberg-03 and rosenberg-02

- Nits.
- Shorter I-Ds references.
- Added terminology section.
- Changed figures to fit in the page width.
- Change reference from RFC 2401 to RFC 2104
- Removed cut & paste error from STUN.
- Fixed some invalid lists.
- Section 9.1: Removed mutual authentication to be consistent with 5.1.
- Fixed the text for the creation of the resource name in 9.4.2, to be consistent with -reload-usage.
- Fixed example to really contain hexadecimal.

Authors' Addresses

Jonathan Rosenberg
jdrosen.net
Monmouth, NJ
US

Email: jdrosen@jdrosen.net
URI: http://www.jdrosen.net

Cullen Jennings
Cisco
170 West Tasman Drive
MS: SJC-21/2
San Jose, CA 95134
USA

Phone: +1 408 421-9990
Email: fluffy@cisco.com

Marc Petit-Huguenin
Stonyfish

Email: marc@stonyfish.com