INTERNET-DRAFT
draft-petke-ext-intro-00.txt
Expires: 15-May-97

G Brown
CompuServe

15 November 1996

Remote Passphrase Authentication
Part One: Extended Introduction

Status of this Memo

Abstract

Remote Passphrase Authentication provides a way to authenticate a user to a service by using a pass phrase over an insecure network, without revealing the pass phrase to eavesdroppers. In addition, the service need not know and does not learn the user's pass phrase, making this scheme useful in distributed environments where it would be difficult or inappropriate to trust a service with a pass phrase database or to allow the server to learn enough to masquerade as the user in a future authentication attempt.

This draft is part one of a four part series and contains an extended introduction to the problem and potential solutions to the problem. It is optional reading for those already familiar with the general issues of authentication over insecure networks. Part two (draft-petke-mech-00.txt) explains the RPA mechanism. Part three (draft-petke-http-auth-scheme-00.txt) explains how to incorporate the mechanism into HTTP. Part four (draft-petke-serv-deity-protocol-00.txt) explains the protocol between the service and deity.

This scheme was inspired by Dave Raggett's Mediated Digest Authentication paper.

Table of Contents

1. Introduction

In this introduction we'll explain the problem--fundamentally, how to authenticate a user to a service without revealing a pass phrase, and without requiring the service to know the user's pass phrase--and consider several alternatives and their flaws, leading to the reasons for developing this authentication mechanism. If you're already familiar with the concept of authentication and the surrounding issues, you might prefer to skip to Part two of the specification, (draft-petke-mech-00.txt), returning to this part only if you want more information about the motivation for the mechanism.

We'll speak of an environment in which a user communicates with a service that wishes to learn and authenticate the user's identity and vice versa. You may, of course, think in terms of client and server, but those terms generally refer to an implementation. We're speaking at a higher level where there's no direct correspondence between server and service nor user and client.

We'll use CompuServe and America Online as concrete examples of services, but the same concepts apply even to a single Web server or BBS that wants to authenticate users. There are three aspects of this environment of interest:

Identification--the way in which we refer to a user.

Authentication--the way in which a user may prove his or her identity.

Authorization--the way in which we determine what a given user may do.

The same aspects apply to services as well as users.

1.1 Identification

A user's identity consists of a user name and a realm name. A realm is a universe of identities; CompuServe Information Service user IDs and America Online screen names are two examples of realms. The combination of username and realm--typically shown as name@realm--identifies a user. Any given service will recognize some particular set of identities. A realm doesn't have to be large, though, either in number of users or size of service. For example, a single Web server might have its own realm of users.

Often, a service recognizes only one realm: CIS recognizes only identities within the CIS realm, and AOL recognizes only identities within the AOL realm. But one can imagine a service that has agreements with both CIS and AOL. The service gives the user a choice of realms--"Please supply a CIS or AOL identity, and prove it"--and the user chooses a realm in which he has an identity.

1.2 Authentication

Identification provides the ability to identify, or refer to, a user. Authentication provides the ability to prove identity. When you ask to do something for which your identity matters, we ask for your identity--your username and realm--and we make you prove that you are who you say you are.

To accomplish this, we'll use a secret that we call a pass phrase, although it's not necessarily derived from text. Such a secret is sometimes called a secret key, but we won't be using it for encryption.

The fundamental problem to be solved is, How can you prove to me that you know your pass phrase without revealing the pass phrase in the process? We'll explore this problem in more detail momentarily.

1.3 Authorization

Authorization refers to the process of determining whether a given user is allowed to do something. For example, may he post a message? May he use a surcharged service? We won't say much about this topic, but it's important to realize that authentication and authorization are distinct processes, one related to proving an identity, and the other related to the properties of an identity.

Our mechanism has nothing to do with authorization, but it is designed to co-exist with authorization mechanisms.

2. The problem and how not to solve it

Imagine that I'm a service who wishes to authenticate you, a user. You must identify yourself and prove to me that you know your pass phrase. That's easy: I'll prompt you for your pass phrase.

But that doesn't work. We learned long ago that plaintext pass phrases cannot be transmitted through a network. X.25 networks have been compromised, and LANs, modem pools, and "The Internet" likewise are not suitable for plaintext pass phrases. Prompting for the pass phrase is not the answer.

2.1 Encrypt the pass phrase?

How about encrypting the pass phrase? Sounds good. You encrypt your pass phrase, send me the result, and I'll decrypt it. Techniques like Diffie-Hellman can create a one-time key that prevents an eavesdropper from decrypting your pass phrase.

But that doesn't work, either. What if somebody else--a spoofer--pretends to be the service? He'll decrypt the result, learning your pass phrase and gaining the ability to masquerade as you. Perhaps that sounds unlikely, but it's not; even in dial-up modem days, people have spoofed services--"Here's a new telephone number they left out of their directory. It's much faster than the listed numbers!"

We need a mechanism that won't reveal your pass phrase to anyone, even if you're not talking to whom you think you're talking.

2.2 A challenge-response mechanism?

How about a challenge-response mechanism? Now we're on the right track. I send you a challenge, which is a random number, and you use a one-way function to calculate a result that depends on the challenge and your pass phrase. You send me the result, and I perform the same calculation and see if my result matches yours. Done correctly, this reveals no information to eavesdroppers, nor does it allow a spoofer to acquire your pass phrase--if someone pretends to be me, they learn only your result for a particular challenge, which is of no value.

Although such a mechanism works, it doesn't quite solve our problem. If I'm the service, I must know your pass phrase in order to reproduce your calculation and verify your response to my challenge. But what if I don't know your pass phrase?

2.3 What if I don't know your pass phrase?

Why might I, the service, not know your pass phrase? Consider a set of services that share a set of users' identities. For example, imagine a collection of Web servers, scattered throughout the world, all of which are a part of Gary's Information Service; you may use your GIS name and pass phrase to identify yourself to any GIS service.

The obvious implementation--each physical server has a copy of all pass phrases or access to a master database--is awkward at best, especially if some are third-party servers, not directly under the control of our imaginary GIS.

Or consider a service that accepts identities in multiple realms. Imagine a service that has agreements with both CIS and AOL. The service gives the user a choice of realms--"Please supply a CIS or AOL identity, and prove it"--and the user chooses a realm in which he has an identity. It's unlikely that CIS and AOL will entrust a copy of their pass phrase databases to a third-party service--or to each other.

So, if I don't know your pass phrase, how can you prove to me that you do know it? And that's the fundamental question addressed by this mechanism. We'll begin by pointing out a couple of solutions that don't work.

2.4 Two more ways not to solve the problem

Wrong answer #1--I'll prompt you for your pass phase. Let's make this example more concrete: I'll display an HTML form with a box that asks for your name and a box for your pass phrase. We'll use SSL or SHTTP so an eavesdropper can't see it. When I get your reply, I can use a challenge-response mechanism to verify your pass phrase with a server that knows the pass phrases.

But that won't work. It's important to teach users not to type their pass phrases just because somebody asks for it--that's a standard technique for cracking others' accounts. Teaching users to provide their pass phrases in an HTML form is a bad idea.

And I'll see your pass phrase, which is precisely what we want to avoid, especially if I'm a spoofer.

Wrong answer #2--We'll create a pass-phrase database server. I'll ask it for a copy of your pass phrase. Now that I know it, we can use an ordinary challenge-response mechanism.

That won't work. We'd need a way to get the pass phrase from that database to me, safely. And if I can look up your pass phrase, what's to stop somebody else from doing the same? (Don't say "a firewall." Services that need to verify your identity exist outside firewalls, too.)

If anything, this is even worse--I could dump the entire pass-phrase database--and, again, I should never see your pass phrase.

But there is a solution, which we'll cover in Part two of this specification (draft-petke-mech-00.txt).

3. Security Considerations

This entire document is about security.

4. Author's Address

Gary S. Brown
CompuServe Incorporated
5000 Britton Rd
P.O. Box 5000
Hilliard OH 43026-5000
USA
+1 614 723 1127
<gsb@csi.compuserve.com>

This Internet-Draft expires on May 15, 1997.