Network Working Group
Request for Comments: 2945
Category: Standards Track

T. Wu
Stanford University

September 2000

The SRP Authentication and Key Exchange System

Status of this Memo

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

Copyright Notice

Copyright (C) The Internet Society (2000). All Rights Reserved.

Abstract

This document describes a cryptographically strong network authentication mechanism known as the Secure Remote Password (SRP) protocol. This mechanism is suitable for negotiating secure connections using a user-supplied password, while eliminating the security problems traditionally associated with reusable passwords. This system also performs a secure key exchange in the process of authentication, allowing security layers (privacy and/or integrity protection) to be enabled during the session. Trusted key servers and certificate infrastructures are not required, and clients are not required to store or manage any long-term keys. SRP offers both security and deployment advantages over existing challenge-response techniques, making it an ideal drop-in replacement where secure password authentication is needed.

1. Introduction

The lack of a secure authentication mechanism that is also easy to use has been a long-standing problem with the vast majority of Internet protocols currently in use. The problem is two-fold: Users like to use passwords that they can remember, but most password-based authentication systems offer little protection against even passive attackers, especially if weak and easily-guessed passwords are used.

Eavesdropping on a TCP/IP network can be carried out very easily and very effectively against protocols that transmit passwords in the clear. Even so-called "challenge-response" techniques like the one described in [RFC 2095] and [RFC 1760], which are designed to defeat simple sniffing attacks, can be compromised by what is known as a "dictionary attack". This occurs when an attacker captures the messages exchanged during a legitimate run of the protocol and uses that information to verify a series of guessed passwords taken from a precompiled "dictionary" of common passwords. This works because users often choose simple, easy-to-remember passwords, which invariably are also easy to guess.

Many existing mechanisms also require the password database on the host to be kept secret because the password P or some private hash h(P) is stored there and would compromise security if revealed. That approach often degenerates into "security through obscurity" and goes against the UNIX convention of keeping a "public" password file whose contents can be revealed without destroying system security.

SRP meets the strictest requirements laid down in [RFC 1704] for a non-disclosing authentication protocol. It offers complete protection against both passive and active attacks, and accomplishes this efficiently using a single Diffie-Hellman-style round of computation, making it feasible to use in both interactive and non-interactive authentication for a wide range of Internet protocols. Since it retains its security when used with low-entropy passwords, it can be seamlessly integrated into existing user applications.

2. Conventions and Terminology

The protocol described by this document is sometimes referred to as "SRP-3" for historical purposes. This particular protocol is described in [SRP] and is believed to have very good logical and cryptographic resistance to both eavesdropping and active attacks.

This document does not attempt to describe SRP in the context of any particular Internet protocol; instead it describes an abstract protocol that can be easily fitted to a particular application. For example, the specific format of messages (including padding) is not specified. Those issues have been left to the protocol implementor to decide.

The one implementation issue worth specifying here is the mapping between strings and integers. Internet protocols are byte-oriented, while SRP performs algebraic operations on its messages, so it is logical to define at least one method by which integers can be converted into a string of bytes and vice versa.

An n-byte string S can be converted to an integer as follows:

$$i = S[n-1] + 256 * S[n-2] + 256^2 * S[n-3] + ... + 256^{(n-1)} * S[0]$$

where i is the integer and S[x] is the value of the x'th byte of S. In human terms, the string of bytes is the integer expressed in base 256, with the most significant digit first. When converting back to a string, S[0] must be non-zero (padding is considered to be a separate, independent process). This conversion method is suitable for file storage, in-memory representation, and network transmission of large integer values. Unless otherwise specified, this mapping will be assumed.

If implementations require padding a string that represents an integer value, it is recommended that they use zero bytes and add them to the beginning of the string. The conversion back to integer automatically discards leading zero bytes, making this padding scheme less prone to error.

The SHA hash function, when used in this document, refers to the SHA-1 message digest algorithm described in [SHA1].

3. The SRP-SHA1 mechanism

This section describes an implementation of the SRP authentication and key-exchange protocol that employs the SHA hash function to generate session keys and authentication proofs.

The host stores user passwords as triplets of the form

```
{ <username>, <password verifier>, <salt> }
```

Password entries are generated as follows:

```
<salt> = random()
x = SHA(<salt> | SHA(<username> | ":" | <raw password>))
<password verifier> = v = g^x % N
```

The | symbol indicates string concatenation, the ^ operator is the exponentiation operation, and the % operator is the integer remainder operation. Most implementations perform the exponentiation and remainder in a single stage to avoid generating unwieldy intermediate results. Note that the 160-bit output of SHA is implicitly converted to an integer before it is operated upon.

Authentication is generally initiated by the client.

```
 Client                              Host
--------                            ------
 U = <username>              -->
                                <--    s = <salt from passwd file>
```

Upon identifying himself to the host, the client will receive the salt stored on the host under his username.

```
   a = random()
   A = g^a % N                 -->
                                      v = <stored password verifier>
                                      b = random()
                               <--    B = (v + g^b) % N

   p = <raw password>
   x = SHA(s | SHA(U | ":" | p))

   S = (B - g^x) ^ (a + u * x) % N    S = (A * v^u) ^ b % N
   K = SHA_Interleave(S)              K = SHA_Interleave(S)
   (this function is described
    in the next section)
```

The client generates a random number, raises g to that power modulo the field prime, and sends the result to the host. The host does the same thing and also adds the public verifier before sending it to the client. Both sides then construct the shared session key based on the respective formulae.

The parameter u is a 32-bit unsigned integer which takes its value from the first 32 bits of the SHA1 hash of B, MSB first.

The client MUST abort authentication if B % N is zero.

The host MUST abort the authentication attempt if A % N is zero. The host MUST send B after receiving A from the client, never before.

At this point, the client and server should have a common session key that is secure (i.e. not known to an outside party). To finish authentication, they must prove to each other that their keys are identical.

```
     M = H(H(N) XOR H(g) | H(U) | s | A | B | K)
                                 -->
                                 <--    H(A | M | K)
```

The server will calculate M using its own K and compare it against the client's response. If they do not match, the server MUST abort and signal an error before it attempts to answer the client's challenge. Not doing so could compromise the security of the user's password.

If the server receives a correct response, it issues its own proof to the client. The client will compute the expected response using its own K to verify the authenticity of the server. If the client responded correctly, the server MUST respond with its hash value.

The transactions in this protocol description do not necessarily have a one-to-one correspondence with actual protocol messages. This description is only intended to illustrate the relationships between the different parameters and how they are computed. It is possible, for example, for an implementation of the SRP-SHA1 mechanism to consolidate some of the flows as follows:

```
 Client                                Host
--------                              ------
 U, A                          -->
                               <--     s, B
 H(H(N) XOR H(g) | H(U) | s | A | B | K)
                               -->
                               <--     H(A | M | K)
```

The values of N and g used in this protocol must be agreed upon by the two parties in question. They can be set in advance, or the host can supply them to the client. In the latter case, the host should send the parameters in the first message along with the salt. For maximum security, N should be a safe prime (i.e. a number of the form N = 2q + 1, where q is also prime). Also, g should be a generator modulo N (see [SRP] for details), which means that for any X where 0 < X < N, there exists a value x for which g^x % N == X.

3.1. Interleaved SHA

The SHA_Interleave function used in SRP-SHA1 is used to generate a session key that is twice as long as the 160-bit output of SHA1. To compute this function, remove all leading zero bytes from the input. If the length of the resulting string is odd, also remove the first byte. Call the resulting string T. Extract the even-numbered bytes into a string E and the odd-numbered bytes into a string F, i.e.

```
E = T[0] | T[2] | T[4] | ...
F = T[1] | T[3] | T[5] | ...
```

Both E and F should be exactly half the length of T. Hash each one with regular SHA1, i.e.

```
G = SHA(E)
H = SHA(F)
```

Interleave the two hashes back together to form the output, i.e.

```
  result = G[0] | H[0] | G[1] | H[1] | ... | G[19] | H[19]
```

The result will be 40 bytes (320 bits) long.

3.2. Other Hash Algorithms

SRP can be used with hash functions other than SHA. If the hash function produces an output of a different length than SHA (20 bytes), it may change the length of some of the messages in the protocol, but the fundamental operation will be unaffected.

Earlier versions of the SRP mechanism used the MD5 hash function, described in [RFC 1321]. Keyed hash transforms are also recommended for use with SRP; one possible construction uses HMAC [RFC 2104], using K to key the hash in each direction instead of concatenating it with the other parameters.

Any hash function used with SRP should produce an output of at least 16 bytes and have the property that small changes in the input cause significant nonlinear changes in the output. [SRP] covers these issues in more depth.

4. Security Considerations

This entire memo discusses an authentication and key-exchange system that protects passwords and exchanges keys across an untrusted network. This system improves security by eliminating the need to send cleartext passwords over the network and by enabling encryption through its secure key-exchange mechanism.

The private values for a and b correspond roughly to the private values in a Diffie-Hellman exchange and have similar constraints of length and entropy. Implementations may choose to increase the length of the parameter u, as long as both client and server agree, but it is not recommended that it be shorter than 32 bits.

SRP has been designed not only to counter the threat of casual password-sniffing, but also to prevent a determined attacker equipped with a dictionary of passwords from guessing at passwords using captured network traffic. The SRP protocol itself also resists active network attacks, and implementations can use the securely exchanged keys to protect the session against hijacking and provide confidentiality.

SRP also has the added advantage of permitting the host to store passwords in a form that is not directly useful to an attacker. Even if the host's password database were publicly revealed, the attacker would still need an expensive dictionary search to obtain any passwords. The exponential computation required to validate a guess in this case is much more time-consuming than the hash currently used by most UNIX systems. Hosts are still advised, though, to try their best to keep their password files secure.

5. References

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

[RFC 1704] Haller, N. and R. Atkinson, "On Internet Authentication", RFC 1704, October 1994.

[RFC 1760] Haller, N., "The S/Key One-Time Password System", RFC 1760, Feburary 1995.

[RFC 2095] Klensin, J., Catoe, R. and P. Krumviede, "IMAP/POP AUTHorize Extension for Simple Challenge/Response", RFC 2095, January 1997.

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

[SHA1] National Institute of Standards and Technology (NIST), "Announcing the Secure Hash Standard", FIPS 180-1, U.S. Department of Commerce, April 1995.

[SRP] T. Wu, "The Secure Remote Password Protocol", In Proceedings of the 1998 Internet Society Symposium on Network and Distributed Systems Security, San Diego, CA, pp. 97-111.

6. Author's Address

Thomas Wu
Stanford University
Stanford, CA 94305

EMail: tjw@cs.Stanford.EDU

7. Full Copyright Statement

Copyright (C) The Internet Society (2000). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.