Network Working Group
Internet-Draft
Intended status: Informational
Expires: 18 February 2026

P. Gutmann
University of Auckland

17 August 2025

A Pre-Authentication Mechanism for SSH
draft-gutmann-ssh-preauth-04

Abstract

Devices running SSH are frequently exposed on the Internet, either because of operational considerations or through misconfiguration, making them vulnerable to the constant 3-degree background radiation of scanning and probing attacks that pervade the Internet. This document describes a simple pre-authentication mechanism that limits these attacks with minimal changes to SSH implementations and no changes to the SSH protocol itself.

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 https://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 18 February 2026.

Copyright Notice

Copyright (c) 2025 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction

Devices running SSH are frequently exposed on the Internet, either because of operational considerations or through misconfiguration, making them vulnerable to the constant 3-degree background radiation of scanning and probing attacks that pervade the Internet. This document describes a simple pre-authentication mechanism that limits these attacks with minimal changes to SSH implementations and no changes to the SSH protocol itself.

1.1. Conventions Used in This Document

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

2. Background

This section covers the background and threat model for the SSH pre-authentication process and mechanism.

2.1. Requirements

The mechanism to limit scanning and probing attacks needs to meet the following requirements:

* It should stop attackers at the gate, preventing probing past the first message exchanged. This both limits information leakage and mitigates against exploitation of pre-auth vulnerabilities in implementations.

* It should require no changes to the SSH protocol, for example the addition of new handshake messages or changes to existing handshake messages.

In addition to these requirements there are also additional desirable properties:

* Ideally it would require no user-visible changes to the operation of an SSH client or server, in other words no need to supply additional or auxiliary keying material or perform other configuration changes. Unfortunately this goal can’t easily be met, see the comments in Section 7, with one configuration change on the client and server being required to enable pre-authentication.

* In order to encourage adoption by implementers of embedded SSH, it should require minimal effort to retrofit to existing SSH implementations, both because embedded systems using SSH are frequent targets and because these systems often have minimal effort applied to keep current with new mechanisms.

Note that although this mechanism can be applied to any SSH implementation, its primary intended target is embedded SSH where few if any mitigations such as privilege separation or frequent patches to address vulnerabilities are possible.

2.2. Threat Model

This document considers two different attacker types:

1. The generic three-degree background radiation of non-targeted Internet scanning and probing from off-path attackers. Any pre-authentication measure, for example including a static non-public value at the start of the handshake, will stop this type of attack.

2. More targeted attacks from on-path attackers, which require something like a challenge/response mechanism to stop.

The pre-authentication mechanism described here targets both off-path and on-path attackers.

2.3. Usage Scenarios

This document considers three different SSH usage scenarios:

1. A conventional server, possibly behind a firewall. Firewall rules and security/access-control proxies, if available, would typically be used to handle any required SSH access control.

2. An embedded device that, for operational reasons or possibly just through misconfiguration, is exposed to the Internet.

3. As above, but on a private network that's been penetrated by attackers who are probing it for targets. In other words the call is coming from inside the building.

The pre-authentication mechanism described in this document is primarily targeted at the latter two scenarios.

3. Description

The pre-authentication mechanism for SSH takes the existing exchange of client and server ID strings and adds a simple challenge/response to them, preventing the exchange of any SSH handshake messages, in other words any actual SSH protocol messages, unless the pre-authentication succeeds. It does this by adding a random challenge in the Comment field of the server's SSH ID, with the client responding with the response in the comment field of its SSH ID. The server challenge in the comment field is denoted with 'C=<challenge>' and the client response with 'R=<response>'. When the pre-authentication mechanism used in this document is used, these MUST be the first values in the Comment field, with any further entries that follow separated by a comma.

The challenge is a 64-bit server-generated nonce which is then base64-encoded to create a text string suitable for use in the Comment field. This encoded form, and the base64-encoded response from the client, are sent without any base64 padding characters '=' at the end, so that the encoded values consist of 11 alphanumeric characters.

The response to the challenge is an HMAC-SHA256 of the challenge, with the MAC value truncated to 64 bits and base64-encoded to an 11-character form in the same manner as the server's challenge. The server challenge is MAC'd in 11-character base64 form as sent, without decoding back to binary form.

Computing the response to the challenge requires a shared secret 'preAuthSecret' between the client and server. This SHOULD NOT be the same as any user password or other authentication value that might be used for authentication but should be created or generated independently and only used for pre-authentication. In the situation where more than a single user or account exists on the device, the preAuthSecret functions in a manner similar to a WiFi password, with the preAuthSecret granting access to the SSH server and subsequent user authentication granting access to whatever sits behind the server.

The HMAC key is calculated as:

```
key = SHA256( string    challenge
              string    preAuthSecret )
```

The response is then computed as a truncated HMAC:

```
rawResponse = HMAC-SHA256( key, challenge )
response = base64( rawRespone[ 0...7 ]
```

In other words the response is the base64 encoding (without adding base64 padding) of the first 64 bits of the HMAC value to give the required 11-byte response value.

4. Test Vectors

Test vectors to be added when the format is finalised.

5. Contributors/Acknowledgements

6. IANA Considerations

7. Security Considerations

As the introduction points out, using this pre-authentication mechanism for SSH is not intended to be all things to all people but to address a specific problem, stopping scanning and probing attacks of SSH-enabled devices at the gates. Conceptually it works like a WiFi password, granting initial access to the SSH server while subsequent user authentication grants access to whatever sits behind the server. Its primary usage scenario is embedded devices with few users, most commonly only one. Use with public SSH systems with large numbers of users is optionally possible, although unlike embedded devices these are expected to have up-to-date software and proper mitigations in place.

However even in the latter case it can provide some protection against protocol and implementation vulnerabilities by reducing the possibility of compromise of the vulnerable implementation or system until the software is updated. Legitimate users could continue to access the system while attackers who couldn't pass the pre-authentication check would be prevented from attacking the vulnerable system behind it.

(When fuzzing an implementation that provides pre-authentication protection, remember to disable the authentication check for the fuzzing process otherwise the fuzzer will be prevented from progressing through to fuzzing the SSH protocol implementation itself).

The pre-authentication can also be used to provide risk-based security in the same way that CAPTCHAs are used on some web sites in which suspicious accesses requiring a CAPTCHA in addition to the normal logon process. For example accesses from the local subnet or only one or two hops away could be permitted without pre-authentication while ones that don't fall into these requirements could only be permitted with pre-authentication, complicating the task for remote attackers while leaving local users unaffected.

The use of a separate preAuthSecret is the lesser of two evils, the other option being to reuse existing authentication information like a password, after due cryptographic processing, for the pre-authentication. This makes the pre-authentication process transparent without requiring the management of additional keying material, since more than two decades of SSH compromise history have taught us that many users are not good at managing such keying material. However with a simple gatekeeper pre-authentication mechanism of the kind described here it appears to be impossible to implement it in a manner that doesn't open it up to an offline dictionary attack by an on-path attacker who, even in the presence of

countermeasures like a severely truncated MAC that leads to false positives, can intercept many challenge/response pairs over time and use those to get to true positives.

An additional benefit of using a distinct preAuthSecret is that it makes enabling and use of pre-authentication explicit. A downside of using a distinct preAuthSecret is that it requires explicit configuration actions to enable and use pre-authentication.

It is recommended that implementations perform rate-limiting on pre-authentication attempts, throttling back responses if too many pre-authentication failures occur in a given time interval. To further confound attackers, servers may in addition opt to continue with an emulated handshake if the pre-authentication fails, eventually failing anyway or dropping the attacker into a tarpit.

Since the authentication is unidirectional, a pass-the-hash attack is possible in the presence of an active attacker who convinces a victim to connect to a fake server and then MITMs the challenge and response from/to the genuine server. However this is merely a pre-authentication mechanism whose main design goal is to protect the SSH implementation behind it from scanning and probing attacks rather than a full-blown (and complex) client/server authentication protocol. The SSH protocol behind the protective pre-authentication step provides the required full client/server authentication, assuming the client verifies the server fingerprint [fingerprints].

The server nonce generation process is described explicitly rather than in general terms like "a random string" to avoid implementations using things like the server name as a nonce value, or more generally just hardcoding something into the otherwise static server ID.

Following Grigg’s Law, "There is only one mode and that is secure", the pre-authentication mechanism hardcodes use of SHA256, the de facto universal standard hash in SSH implementations. Since the security property required of the hash function is preimage resistance rather than collision resistance, and even beyond that the ability to find one specific preimage rather than any valid preimage, almost any hash function would suffice; SHA256 is chosen because of its universal acceptance and use.

8. References

8.1. Normative References

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", RFC 8174, May 2017, <http://www.ietf.org/rfc/rfc8174.txt>.

8.2. Informative References

[fingerprints] Gutmann, P., "Do Users Verify SSH Keys?", ;login Volume 36, Number 4, August 2011, <https://github.com/jscep/jscep>.

Author’s Address

Peter Gutmann
University of Auckland
Department of Computer Science
Auckland
New Zealand
Email: pgut001@cs.auckland.ac.nz