Secure Shell (SSH): Public Key Authentication over Hypertext Transfer Protocol (HTTP)

Dorai Ashok S.A. <dash.r@mpipe.net>
0th Root Software Research Private Limited
No.87A, Vembuliamman Koil Street, Periya Nolambur, Chennai-600095, India
Phone: +91 44 2653 2984

June 17, 2015

Abstract

The Secure Shell (SSH) protocol requires all implementations to support public key authentication method (“publickey”) for authentication purposes [2]. Hypertext Transfer Protocol (HTTP) applications which provide a SSH client over the web browser need to support “publickey”. However, restrictions in HTTP, such as Same Origin Policy, make it difficult to perform such authentications. In this document, a system to perform “publickey” authentication over HTTP is provided. It is ensured that no compromise is made that would pose a security risk to SSH protocol.

1 Introduction

The Secure Shell (SSH) protocol is a protocol for secure remote login to servers. It computes a shared key through key exchange mechanisms, such as Diffie-Hellman key exchange [3], and establishes a secure channel for communication to servers. Users are authenticated through the secure channel using one of the accepted authentication methods. Commonly used authentication methods include “password” and “publickey”. As the name suggests, in case of “password” method, the user is simply required to provide his/her password for authentication. However, “publickey” method uses public/private key pairs for authentication.

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for data transfer across the internet. It is generic, stateless and can be used for purposes beyond hypertext.

2 Authentication “publickey”

In “publickey” authentication method, User creates a public/private key pair and adds the public key on the server. For successful authentication, the SSH protocol requires the message [2], shown below, to be signed by the private key of the user. And, the signed message should be verified by the server using the public key added by the user. On successful verification, authentication is successful and the user is permitted to login.

string session identifier
byte SSH_MSG_USERAUTH_REQUEST
string user name
string service name
string "publickey"
boolean TRUE
string public key algorithm name
string public key blob

The session identifier, shown above, is a secret and is specific to that SSH session.

It is common for private keys to be kept in an encrypted format for security reasons. However, this creates the hassle of decrypting the private key for authentication every single time. Authentication agents, such as ssh-agent [4], handle this by managing the decrypted private keys and signing messages for SSH clients on demand.

3 Authentication agents

ssh-agent implements OpenSSH agent protocol [4] to manage private keys and to sign messages. Clients connect to the agent, typically, over an UNIX domain socket to send requests and receive responses. UNIX domain socket is a special file in the file system. To restrict access, the special file has read/write permissions only for the user.

If we could implement an authentication agent, similar to ssh-agent, that would manage private keys but accept signing requests over HTTP. That would be an agent which can enable SSH “publickey” authentications through HTTP clients.
4 SSH Web Agent

Here, we are defining a new authentication agent, ssh-webagent, which responds to authentication requests over HTTP. It accepts requests only from user and only on localhost. And, due to same origin policy, HTTP clients will not accept content from localhost unless Cross Origin Resource Sharing (CORS)\(^1\) is enabled. It is also recommended that ssh-webagent accept HTTP requests over TLS\(^2\) to avoid Mixed Content\(^3\).

4.1 Trusted Servers

On receiving an authentication request, it is important to identify the source of the request. A HTTP client could connect to ssh-webagent out of any webpage, and without verification, a serious security risk exists. To verify the source, a combination of HTTP Referer header and public key of the server should be used. While the Referer header can be used to identify the public key of the server, the actual verification is successful only if the signature provided by the server is valid.

Server’s public key and HTTP Referer changes for each web application. It is therefore recommended that a trusted servers file is used to hold the public key of trusted servers along with the expected values of HTTP Referer header. The trusted servers file should be kept safe to avoid unauthorized modifications. A sample format for trusted servers file has been provided in Appendix A.

4.2 Identifiers

HTTP is stateless and each request is independent of the other. Each HTTP request has a corresponding response but the order of HTTP requests and state, if any, shall be managed by the application.

In case of authentication over a TCP connection, a session is established and can be uniquely identified by 5-tuple\(^4\). However, HTTP provides no session and hence ssh-webagent has to establish a logical session to the trusted server and assign a unique identifier. ssh-webagent can interact with multiple trusted servers at the same time. And, ssh-webagent shall choose the unique identifier.

The state of the authentication process shall be maintained in association with the identifier. However, it should be noted that, a failure in the authentication process may not always be known to ssh-webagent. So, it is essential to maintain a reasonably short timeout for the authentication process, beyond which the identifier and its associated state is destroyed.

4.3 Local User and Local Host

Since ssh-webagent accepts authentication request on localhost, it shall be ensured that the request is from the same user who runs ssh-webagent. Support for this feature varies across operating systems and Appendix C shows how the owner of a TCP connection can be identified on systems running Linux kernel.

When ssh-webagent binds to a specific port on localhost, no other user will be able to use the same port. So, applications which intend to support multiple local users, can choose multiple ports and send authentication request to all of them. A standard has been proposed to use a specific local IP address and port for ssh-webagent in Appendix B.

4.4 Session

ssh-webagent shall establish a secure session with the trusted server before processing authentication requests. The session is initiated by a signed request from the trusted server with Diffie-Hellman (DH)\(^5\) parameters. On successful verification, ssh-webagent shall send a Diffie-Hellman response with an encrypted identifier. The secret established through Diffie-Hellman key exchange shall be used in the encryption of the identifier. And, once the trusted server successfully decrypts this identifier, the session is established.

The authentication process following session creation, will use the identifier in both clear and encrypted data. ssh-webagent shall identify the session using the clear text identifier. However, it shall verify that the identifier in encrypted data matches the one in clear text.

5 Proposed Protocol

In this section, we define a protocol over HTTP that can be used for “publickey” authentication. It involves both session creation and authentication process. We start with the format of HTTP requests and responses before discussing each of them in detail.

5.1 HTTP Request/Response

The protocol uses base64 encoded messages in both HTTP request and response. HTTP request shall use method POST and content type application/x-www-form-urlencoded, while HTTP response shall use content type text/plain. HTTP request in this format will classify as simple cross-site request under CORS specification\(^6\).

\(^1\)source IP address, source port, destination IP address, destination port, transport protocol

\(^2\)Diffie-Hellman Key Exchange algorithm\(^7\)
As a POST request, the key P shall be used to pass messages and, optionally, the keys U and S shall be used to pass user and service names respectively.

[Message]

The response body simply contains the message in response to the request and nothing more.

5.2 Message

Message is the basic unit of the protocol. A single message is received in the HTTP request and a single message is sent in the HTTP response. Message uses a binary format and data types used, in representation of the format, are as defined in RFC 4251 [1], Section 5 “Data Type Representations Used in the SSH Protocols”.

string “SSHWebAgent”
byte version
byte type
string data

version shall indicate the version of the protocol and shall be used to check compatibility of ssh-webagent and trusted server.

VERSION_1_1 0x11
type shall indicate the type of data the message holds.

KEX_DH_REQUEST 0x02
KEX_DH_RESPONSE 0x03
PRIVATE 0x04
data holds request parameters and response values in binary format. The format varies based on type and, hence, are described in their respective sections.

5.3 Session

Session is initiated by a message from trusted server containing key exchange parameters and its signature. The message will contain data of type KEX_DH_REQUEST in the format described below.

mpint p
mpint g
mpint e
string d
string k
string sign

p, g are Diffie-Hellman key exchange parameters prime number and generator, respectively.

e is the computed value of trusted server in key exchange

d is session data, if any, added by trusted server. It shall be used by ssh-webagent without any interpretation.

k, sign are the public key and signature, of the trusted server, represented in their respective formats as defined in RFC 4253 [3], Section 6.6 “Public Key Algorithms”.

5.3.1 Verification

On receiving the message, ssh-webagent uses the trusted servers file and checks if the public key and the HTTP Referer are trusted. On a successful match, the message shall be checked for authenticity through signature verification. The following values are used in the specified order for signing and verification,

mpint p
mpint g
mpint e
string method
string referer
string k
string d

method shall be the method used in HTTP request. As mentioned in Section 5.1, this shall be POST.

referer shall be the value of Referer field in HTTP request header.

5.3.2 Response

On successful verification of signature, ssh-webagent shall send a message containing data of type KEX_DH_RESPONSE in the format described below.

mpint f
string E

f is the computed value of ssh-webagent in key exchange.

E is the encrypted message body of type NEW viewable only by trusted server. message body is described, in detail, in Section 5.4.

5.3.3 Secret

The secret S is computed by ssh-webagent through Diffie-Hellman key exchange and it is used to compute a shared secret, a secret key and an initialization vector.
1. **Shared Secret**

*shared secret* is the computed hash of the following values using 256-bit Secure Hash Algorithm (SHA-256) [8].

- string method
- string referer
- mpint e
- mpint f
- mpint S

2. **Secret Key**

*secret key* shall be used as the key for symmetric encryption. A SHA-256 hash of the following values is the *secret key*.

- mpint S
- string shared secret
- byte 'A'
- string referer

3. **Initialization Vector**

Depending on the encryption algorithm, if needed, the *initialization vector* can be obtained by computing a SHA-256 hash of the following values,

- mpint S
- string shared secret
- byte 'B'
- string referer

Once the trusted server receives the response message, it computes its own values for the above and decrypts the *message body*.

### 5.4 Message Body

The *message body* holds the cipher text for communication between *ssh-webagent* and trusted server. It also includes the session identifier and the encryption algorithm for decryption purposes. The type of message body and its related contents are encrypted and stored in cipher text.

#### 5.4.1 format

The binary format of message body is shown below,

- byte[4] random
- byte type
- string identifier
- ... payload ...
- byte[n] padding

*random* shall be a 32-bit (4 bytes) cryptographically secure random number.

*type* indicates the message body type and it shall be used to interpret the *payload*. The supported types are listed below,

- NEW 0x02
- AUTH_REQUEST 0x03
- AUTH_RESPONSE 0x04

*identifier* is the session identifier and it shall be the same as *identifier* defined in Section 5.4.1. *payload* shall be zero or more bytes of *message* content. The format of *payload* depends on message body type and they are described in detail in their respective sections.

*padding* shall be zero or more bytes of random padding to meet the block size requirements of encryption algorithms.

#### 5.4.3 NEW

First message with a message body, in a *session*, is always of type NEW. This message is sent by *ssh-webagent* to trusted server. The format of *plaintext* in message body of type NEW is shown below,

- byte[4] random
- byte type
- string identifier
- byte[n] padding

It should be noted that *payload* is empty and, hence, is not shown in *plaintext*.

### 5.5 Authentication

The authentication process shall be initiated by trusted server after establishing a *session*. The messages used in the process are as defined in Section 5.2 and shall contain *message body* as *data* and shall use *type* PRIVATE.

#### 5.5.1 Request

A request for “publickey” authentication is sent by the trusted server in the form of a *message* containing *data* of type PRIVATE and *message body* of type AUTH_REQUEST.
This request provides the secret SSH session identifier required to generate SSH message for signing and authentication. The format of the request message is shown below,

```
string "SSHWebAgent"
byte VERSION_1_1
byte PRIVATE
string message body
```

The format of message body is as follows,

```
byte algorithm
string identifier
string ciphertext
```

And, the format of plaintext that is encrypted to form the ciphertext is,

```
byte[4] random
byte AUTH_REQUEST
string identifier
string SSH session identifier
byte[n] padding
```

SSH session identifier is as defined in RFC 4253 [3], Section 7.2, “Output from Key Exchange”

### 5.5.2 Response

In response to the request, ssh-webagent shall use SSH session identifier, construct the SSH message and sign it with one or more private keys of the user. The resulting signatures are sent securely in a response message to the trusted server for successful authentication. The format of plaintext in response message is as follows,

```
byte[4] random
byte AUTH_RESPONSE
string identifier
boolean status
string signatures
string options
byte[n] padding
```

**status** shall indicate whether the signing process was successful or not. It shall be used to communicate failures to trusted server.

**signatures** contain one or more signatures of the SSH message, along with the corresponding public key and comments, if any. They are formatted as follows,

```
uint32 n
string[n] item
```

And, each item shall use the format shown below,

```
string publickey
string signature
string comment
```

string[n]/ shall represent multiple strings, with the number of strings indicated by n. When n is zero, string[n]/ shall be empty and, hence, shall not be present.

**options** shall be used to pass information to the trusted server in the form of key-value pairs. It shall use the format as specified below,

```
uint32 n
byte es
string[n] option
```

And, each option shall hold a key and a value as shown below,

```
string key
string value
```

The value shall be encrypted using an encryption scheme es. The following encryption schemes shall be supported,

```
PKCS1_RSAES_OAEP 0x02
```

PKCS1_RSAES_OAEP encryption scheme shall use the public key of trusted server to encrypt the value.

On receiving the response message, the trusted server shall have all that it needs to perform “publickey” authentication in Secure Shell (SSH) protocol.

### 6 Performance

The authentication method ensures that the remote login to server is secure. But, for the task at hand, it is most likely a hindrance and sooner it is done, the better. The primary contributing factors to longer authentication time would be network latency to trusted server. Data encryption, signing and verification are reasonably quick on modern processors and are negligible in comparison. The latency to web agent will also be negligible since it resides on localhost.

The protocol uses HTTP requests to receive and send messages. Hence, session establishment and authentication process will each require three HTTP requests, when done synchronously. First request shall be used to get the message (request) from trusted server, second request shall be used to pass the message (request) to web agent and receive a message (response) and the final request shall be used to pass the message (response) to trusted server. With two requests to trusted server per session establishment and authentication process. A
total of four requests to trusted server remain the primary contributing factors to performance.

HTTP request to trusted server will be over TLS. Hence, apart from the usual TCP connection negotiation, a TLS session negotiation will also happen when a new connection is established. So, the first request will take a significantly longer time, in comparison to subsequent requests to trusted server. The total time taken to complete four requests can be significant and can be a hinderance. To achieve better performance, it is recommended that the network latency to trusted server remains below 80ms.

7 Conclusion

Secure Shell protocol has been commonly used for remote login to servers. It is the first step to managing servers remotely. As the number of servers increase, the problem of logging to multiple servers, a time consuming task, become apparent. These problems can be solved to an extent through applications which abstract out and automatically login to servers. However, these applications may require the private key of user for automatic login.

In a world with increasing use of virtualization, more and more servers (or virtual servers) are purchased from providers and managed remotely over public internet using a web browser. In the absence of a method for secure remote login to these servers, users are left to use traditional tools for login or at worst, take a security risk and, give away the private key to third-party applications.

Hence, solving the problem of secure remote login (“publickey”) over the web, ie., HTTP, will open doors for easier access to servers. And, bring back private keys to the hands of the user. When combined with a powerful server management application, this can bring servers to general public, who, without any technical knowledge, can avail internet services through their servers in a decentralized manner.

References

[1] Ylonen, T. and C. Lonvick, Ed., “The Secure Shell (SSH) Protocol Architecture”, RFC 4251, January 2006.
[2] Ylonen, T. and C. Lonvick, Ed., “The Secure Shell (SSH) Authentication Protocol”, RFC 4252, January 2006.
[3] Ylonen, T. and C. Lonvick, Ed., “The Secure Shell (SSH) Transport Layer Protocol”, RFC 4253, January 2006.
[4] OpenSSH, “PROTOCOL.agent”, ssh-agent
[5] World Wide Web Consortium, “Cross-Origin Resource Sharing”, CORS, January 2014
[6] Mozilla Developer Network, “Mixed Content”, 2015
[7] Rescorla, Eric, “HTTP Over TLS”, RFC 2818, May 2000
[8] US National Institute of Standards and Technology, “Secure Hash Standard (SHS)”, Federal Information Processing Standards Publication 180-2, August 2002.
A Trusted Servers File

The public key\(^5\) of trusted server and its acceptable HTTP Referer prefixes are the contents of this file. Each trusted server has an entry starting with its public key on the first line, followed by one or more acceptable prefixes of HTTP Referer values per line. The entry for trusted server ends with a line containing a single dot (.), followed by an entry for next trusted server.

A sample trusted servers file in this format has been shown below,

```
AAAAB3NzaC1yc2EAAAADAQABAAABA
QCrLjgFEA7tLyydh5eS2DCglbS5/767i5MaCoXoxZApH1/JqT062nYJ6P
G0Hyu8sEpE2kcTtNH10mcsygFEaa8v1FjxYL7dW/HufafyQ+eH2Az/xPtTu
oSW6y19gKj1fnaxF9IItdZV0kcSwuEmlJfkgf6fbyrn8M+bIK86y5K4
s110aN48 tGEYe9Xe8C3ZJdhCyv97fhpELIXCyq0qjQpekSTTe4lpiQKv
12n677/E5V61nv419WkJJAU7q71x+fW5bfqgXnGnmOSawRGzf0EtsbHJ
Wn41c/0HIiYJ0MRKd7/+V4qBEF+0CZIAs4QyQt6eLCkmu5f7yf
https://webssh.example.com/ssh/
https://webssh.example.com:444/ssh/
https://webssh.example.com:445/ssh/.
AAAAB3NzaC1yc2EAAAADAQABAAABA
QDPyjl9euMQ4Crj/0VyP69+ltELAM4Wt0GyG8Y3ENEtpa/QvOXJ1I283
1R5w5 Cy2e2LJKQJWnK1Xo3uQJ+JdCA10Xh91a8w0WEMs62HibBOJNe78
g1vPy169qLvp48Xyxtz28Bah2ZP5d7k5Xn1X4HIUuab3t1HvsO2L0qj9p
QQPDeh+GdmMgEvo2QMYB6ukJql+Rd1dGwHDRU+pfRi2xeMPQgPcqOo
7ykKM18G/L13Q7R6dIrW9Ah0/BrdF1NEA31Fyf90aPBP351k8Bl1L17V
PgoVP24VJKzSojEKpn4KkHGLTfg+5Tq67x36b1lHZpx3g8tcxQt
https://sshclient.example.com/ssh
```

B Local IP, Port and Domain

SSH Web Agent binds to local loopback address and accepts request on localhost. The IPv4 address range assigned for local loopback is 127.0.0.0/8\(^6\).

Any address in this range can be used by web agent for accepting requests. However, for easier inter-operability, the IPv4 address, in this range, 127.82.11.29 and port number 8211 is suggested for use.

The domain `localhost-ssh-webagent.in` has been assigned for this purpose and it resolves to the suggested IPv4 address, 127.82.11.29. The whois information for localhost-ssh-webagent.in provides the current contact to obtain TLS certificates.

C Connection Owner

A web agent listening on loopback interface can be connected to by any user. This results in a privilege escalation and unauthorized access to private key. This security issue can be mitigated by ensuring that the remote end of the local connection belongs to the owner of the web agent process itself. The premise here is that the owner of the web agent process is given access to the private key and hence can allow access to the same user.

On linux kernels, `/proc/net/tcp` provides a formatted output of active connections. In this output, the columns `local_address`, `rem_address` and `uid` are of relevance to us. The IPv4 address and port in the output are shown in hexadecimal.

The sample output below shows active local connections,

```
local_address rem_address uid
1D0B527F:2013 00000000:0000 1000
2 00000000:0016 00000000:0000 0
3 0100007F:0019 00000000:0000 0
```

HTTP/TLS protocol requires web agent to use a TLS certificate signed by a trusted certificate authority. TLS certificates are, typically, issued for a domain instead of an IP address.

TLS certificate and its private key will need to be distributed along with web agent. Hence, they should be considered public. And, on the same note, for security reasons, it is recommended that the entire domain is considered public and subdomains are not used for any other purpose.

The domain `localhost-ssh-webagent.in` has been assigned for this purpose and it resolves to the suggested IPv4 address, 127.82.11.29. The whois information for localhost-ssh-webagent.in provides the current contact to obtain TLS certificates.

HTTP/TLS protocol requires web agent to use a TLS certificate signed by a trusted certificate authority. TLS certificates are, typically, issued for a domain instead of an IP address.

TLS certificate and its private key will need to be distributed along with web agent. Hence, they should be considered public. And, on the same note, for security reasons, it is recommended that the entire domain is considered public and subdomains are not used for any other purpose.

The domain `localhost-ssh-webagent.in` has been assigned for this purpose and it resolves to the suggested IPv4 address, 127.82.11.29. The whois information for localhost-ssh-webagent.in provides the current contact to obtain TLS certificates.

---

\(^5\)Public key should be in a format as defined in RFC 4253

\(^6\)RFC 6890, “Special-Purpose IP Address Registries”

---