CS594 Trent Wilson & Alec Greenaway
Internet Draft Portland State University
Intended status: Class Project Specification March 15, 2020
Expires: September 2021

Secure Server-based Password Manager Class Project
draft-irc-pdx-cs594-00.pdf

Status of this Memo

Copyright Notice

Abstract

This memo describes the communication protocol for a client/server password manager system for the Internetworking Protocols class at Portland State University.

Table of Contents

1. Introduction

This specification describes a simple Server-based Password Manager Protocol by which a client communicates with the server performing create, read, update, and delete operations. This system uses a TLS connection, IP spam checking, and login information as security measures.

Errors are handled for incorrect login, spammed login attempts, and every case of interacting with the user’s password table that does not make sense e.g. deleting a nonexistent account password.

2. Conventions used in this document

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 RFC 2119 [RFC2119].

In this document, these words will appear with that interpretation only when in ALL CAPS. Lower case uses of these words are not to be interpreted as carrying significance described in RFC 2119.

In this document, the characters ">>" preceding an indented line(s) indicates a statement using the key words listed above. This convention aids reviewers in quickly identifying or finding the portions of this RFC covered by these keywords.

3. Basic Information

All communication described in this protocol takes place over TLS layered on top of TCP/IP, with the server listening for connections on any port specified by the server system administrator when launching the program. Once launched, the server is actively listening and will continue to do so until the server system administrator terminates the program. Clients connect to the specified port and exchanges information passed as arguments to the server. Many clients may connect over time to the one persistent server process.

Both the server and client may terminate the connection at any time for any reason. They MAY choose to send an error message to the other party informing them of the reason for connection termination.

The server MAY choose to allow only a finite number of active client connections at a given time, depending on the implementation and resources of the host system. Error codes and messages are available to notify connecting clients of one of many potential issues that may arise.

4. Client Messages

4.1. Login Credentials

```
(args.username + ' ' + args.password).encode()
```

where encode() translates the string into a sequence of unicode bytes.

4.1.1. Usage

Before subsequent messages can be sent, a connecting client MUST provide a user name and password.

The server MUST associate the client's user name and password with an existing account. This message MUST be sent only once; if the server receives the message more than once, the server MUST send an error message and terminate the client's connection.

4.1.2 Field Definitions

o args.username is used to identify the user
o args.password is used to identify the password associated with the given user.

4.1.3 Response

Server MUST return a bytestream of unicode characters beginning with a 0 or a 1, a space, followed by any log messages or errors. See [5.2] for more details. A 1 signifies success. See [6] for explanation of error codes following a 0.

4.2. Action Request

Either:

```
(args.action + ' ' + args.account[0] + ' ' +
args.account[1]).encode()
```

Where:
args.action = CREATE or UPDATE
args.account[0] = account name for the password
args.account[1] = new password for the action query
And encode() translates the string into a sequence of unicode bytes.

Or:

```
(args.action + ' ' + args.account[0]).encode()
```

args.action = READ or DELETE
args.account[0] = account name for the password
And encode() translates the string into a sequence of unicode bytes.

(Note the difference is only that READ and DELETE actions do not require a password for the action to take place)

4.2.1. Usage

Sent by the client to request one of the available program actions: either create a new account/password pair, read a password for a specified account, update the password for an existing account, or delete an existing account.

4.2.2. Response

Server MUST return a bytestream of unicode characters beginning with a string, a space, followed by any errors. A non-zero first string signifies a success. See [5.3] for more details. See [6] for explanation of error codes following a 0.

5. Server Messages

5.1. Client Connection Response

```
str(pw).encode() + log.encode()
```

5.1.1. Usage

Before any messages from the client are received, their IP is checked against a server-side database to check how many failed attempts have occurred in the last administrator-defined amount of time (default is 5 minutes) if the threshold is reached, the server MUST send a fail format message (containing a leading 0) with an error message and MUST terminate the connection.

5.1.2. Field definitions

pw: either a 0 or 1 signifying pass or fail of the log-in-spam check.
log: containing the error messages. See [6.FIX] for details on this specific error message.

5.1.3. Response

See [4.1] for the next client response.

5.2. Login Response

Before the user can access any passwords in the server-database, valid credentials must be confirmed. Having received the clients username and password combination, it will check the login database table and respond with success or failure based on whether it has found a match or not. On fail, the server MUST send a fail format message (containing a leading 0) with an error message and MUST terminate the connection.

5.2.1. Usage

```
str(pw).encode() + log.encode()
```

5.2.2. Field definitions

pw: either a 0 or 1 signifying pass or fail of the log-in-spam check.
log: containing the error messages. See [6.FIX] for details on this specific error message.

5.2.3. Response

See [4.2] for the next client response.

5.3. Action Request Response

The final server response is similar in format to the previous messages (which are nearly identical) but instead of a 1 or 0 indicating success, it is either a non-zero string or a 0. For CREATE, READ, and UPDATE actions, the non-zero success string represents the relevant password. For DELETE this will be a 1 like previous requests. A password for DELETE does not make sense in this context. On fail, the server MUST send a fail format message (containing a leading 0) with an error message. Regardless of whether it was a success or failure, the client program has received all it is expecting and so the server MUST terminate the connection.

5.3.1. Usage

```
(str(pw) + ' ' +  log).encode()
```

5.3.2. Field definitions

pw: either a non-zero string containing the password for CREATE, READ, and UPDATE action requests or 1 for DELETE signifying pass. If the action failed, 0.
log: containing the error messages. See [6.FIX] for details on this specific error message.

5.3.3. Response

The connection is terminated and so there is no client response.

6. Error Handling

Both server and client MUST detect when the socket connection linking them is terminated, the other MUST consider itself disconnected.
As stated previously, it is optional for one party to notify the other in the event of an error.

6.1 Error Messages

When client tries to connect to the server but the server is not listening:

```
[ERROR]: [Errno 111] Connection refused
```

When client tries to connect with invalid credentials:

```
[ERROR]: [FAIL]: Login credentials invalid.
```

When client tries to brute-force connect:

```
[FAIL]: 5 spam attempts detected from this IP.
```

Reading a record that doesn’t exist:

```
[FAIL]: That account doesn't exist for your account, so no pw could
be retrieved.
```

Updating a record that doesn’t exist:

```
[FAIL]: that account doesn't exist for this user.
```

Deleting a record that doesn’t exist:

```
[FAIL]: No account with that name, so can't delete any pw for it.
```

Creating/Updating an account without giving a password:

```
[FAIL]: Two args needed for --account if --action == create or
update. [newacct newpw] two strings were not detected.
```

Requesting an action other than CREATE/READ/UPDATE/DELETE:

```
[FAIL]: Two args needed for --account if --action == create or
update. [newacct newpw] two strings were not detected.
```

7. Conclusion & Future Work

This specification provides a framework for multiple clients to communicate with a central server in order to retrieve those account passwords associated with their account and no passwords belonging to other accounts.

Future work is incorporating the FIDO2 protocol for Two Factor Authentication (2FA) for additional assurance the client computer is the correct person accessing their passwords. The Python3 library to leverage to add this functionality is:
https://github.com/Yubico/python-fido2

8. Security Considerations

The features mentioned at length previously were implemented with the intent to increase security. However, care was not given to protect against SQL injection attacks. This is another candidate for “Future Work” however, too much database work would be out of scope for an internetworking protocols course.

9. IANA Considerations

None

9.1. Normative References

[1]
Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

10. Acknowledgments

This document was prepared with guidance from the sample RFC provided on Piazza.