Password Manager

Design Document, v1.0

Authored by David Frye, Fryeware LLC

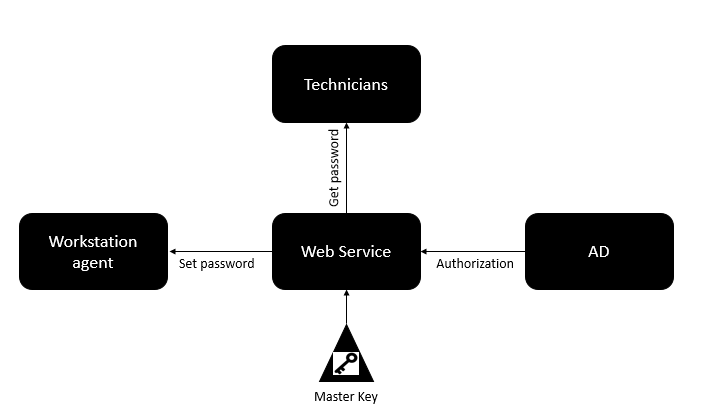
Part 1: Overview

## Purpose:

Password Manager provides an automated and secure solution to the problem of managing local administrator accounts in a Windows domain environment. In a typical Windows Active Directory domain, member computers maintain a local administrative account (“Administrator”) to provide a fallback in cases where domain based credentials are inoperable. The local Administrator account is typically assigned a static, shared password that is never changed for the life of the computer. This creates a potential security vulnerability where if a hacker were to gain knowledge of this local administrator password, the hacker would be able to move laterally within the network at-will. Password Manager solves this problem by implementing strong, unique passwords to all managed systems, and providing a central retrieval and reporting system.

## Description:

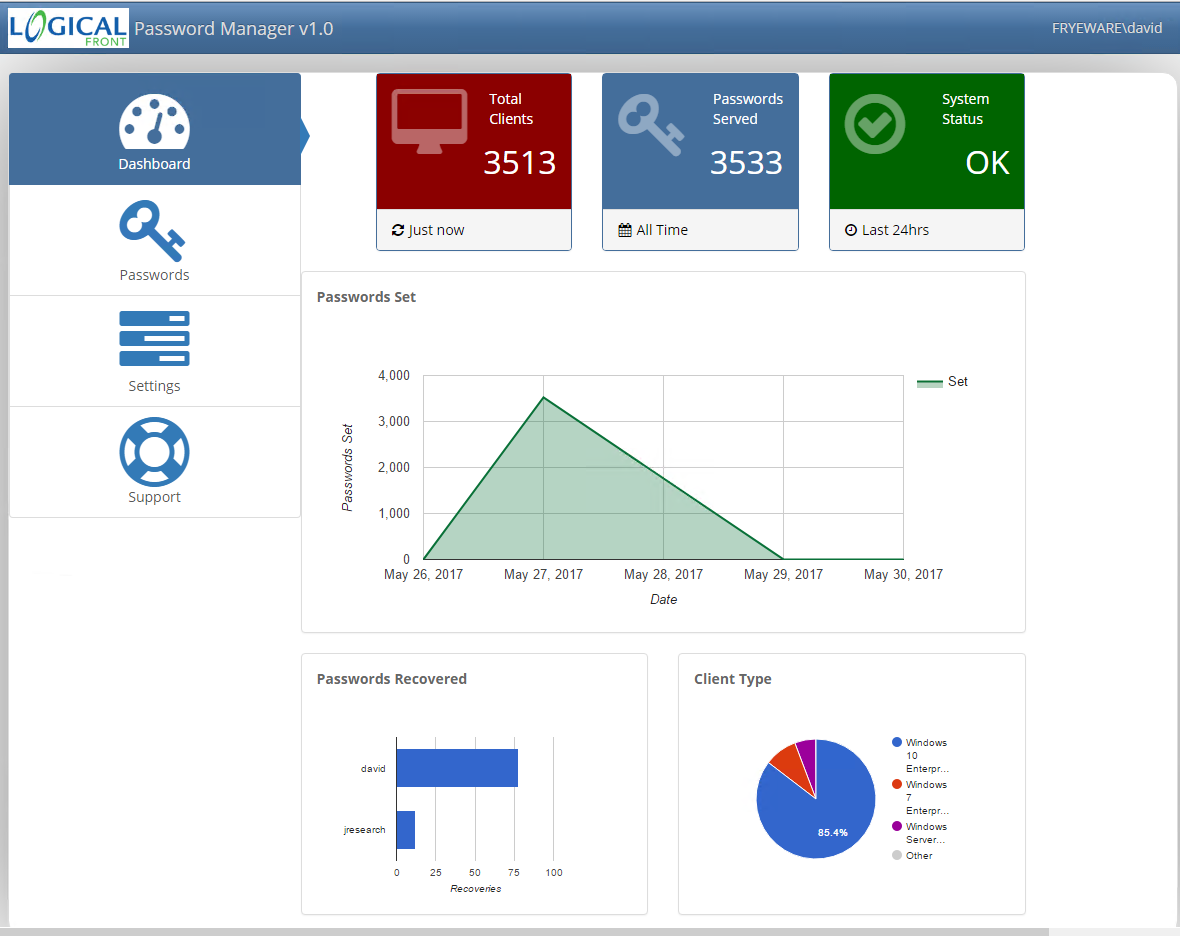
Unlike other commercial solutions, Password Manager does not create randomized passwords. Instead, passwords are calculated on-demand using a password derivation scheme via a master key and hashing algorithms. This approach removes the need to durably store passwords. Clients request new passwords through a web service. Technicians retrieve (recalculate) client passwords via an intranet web site (web Console). Access control for password retrieval is provided by Active Directory OU permissions: any user with WRITE permissions to an OU can recover passwords within that OU.



Part 2: Concept of Operations

Password Manager is a client-server solution. An agent is deployed to all managed clients, and a web site/service is deployed to at least one server on the network. This agent checks in with the central web service periodically and sets/resets the password on a local administrator account. The managed local administrator account is named “LocalAdmin”. Since the built-in Administrator account is a well-known target, it is considered a security best practice to disable the built-in Administrator account. Once the agent is deployed and successful password retrieval is sufficiently demonstrated, Active Directory Group Policy should be configured to disable the built-in “Administrator” account. From that point on administrators and technicians should use the managed LocalAdmin account for all local account activities.

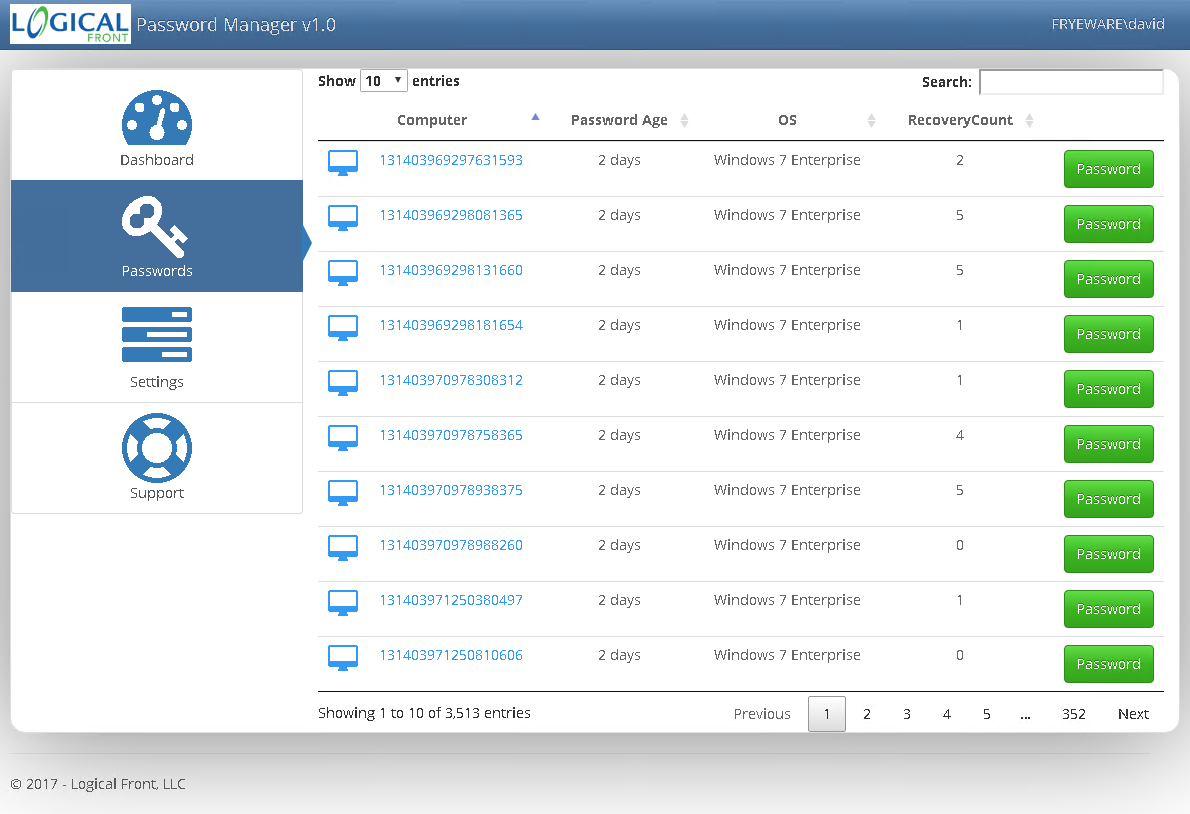
Passwords for LocalAdmin are retrieved via the Password Manager Web Console:



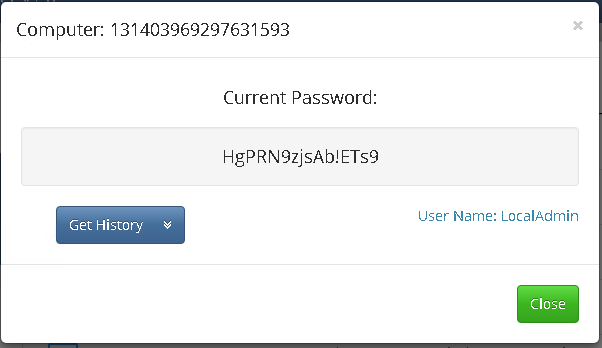
The Password Manager web console is broken down into four vertical tabs. The home tab is the dashboard view. The dashboard displays general health and usage statistics for your installation. The System Status widget will display in green with the text “OK” if no system errors are detected. “WARN” or “ERROR” indicators may display under System Status if significant errors are detected. Logs can be viewed to diagnose any reported issues and will be discussed later in this document.

## Password Retrieval

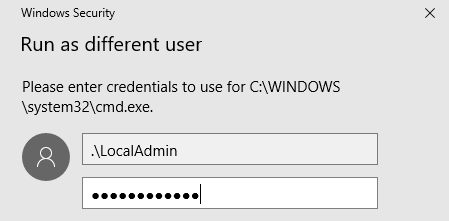
To retrieve the password for a managed client, click the Passwords tab and then find the computer record in the Passwords table:



You can find the target computer by either typing the computer name into the Search box at the top of the screen, or paging through the lists of results until you find the computer of interest. Then, click the green Password button on the right. You will be presented with a password recovery dialog like this:

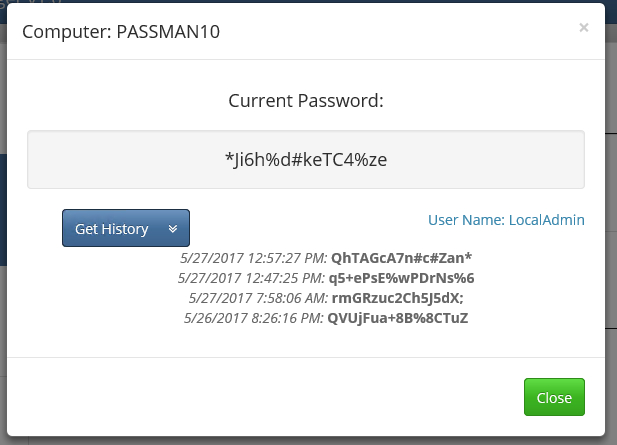


In most scenarios, you will want the last password registered with the server. To use it, you can simply copy the text string displayed under “Current Password” and paste it into a logon window to logon as LocalAdmin.



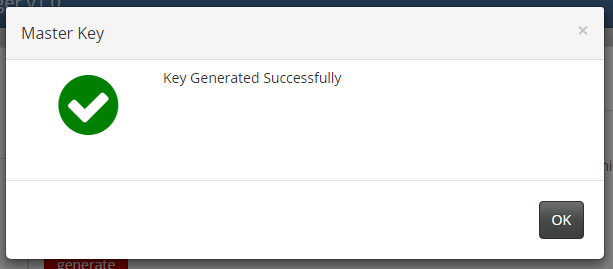
By default, managed passwords expire every 30 days. When expiry occurs, the server will generate a new password during the next client connect process (the client connect process is described in detail in the Technical Design section of this document). However, once a password is retrieved, that password is immediately expired regardless of age. Password Manager will maintain metadata to compute the most recent 5 passwords for each managed client (the current password, plus 4 historical passwords).

In some situations, you don’t want the most current version of the password. This is especially true in VM scenarios where you revert a VM to an old snapshot of the OS. When retrieving a password, you will see button just below the current password called “Get History”. Clicking this button will reveal the previous 4 passwords along with the date set:



## Server Settings

In version 1.0 there are two configurable items in the Settings tab of the web console: Master Keys and DB Auto-Purge. Both of these are set at install time automatically and should not need to be adjusted very often. Master Keys is used to generate a new system wide key. In order to compute a password, an HMAC256 keyed hashing algorithm is used to combine unique metadata about the client with the master key. A single key is all that is needed to compute all unique client passwords and you may never need to generate new keys. However, there are some scenarios where rekeying will be warranted. For instance, if the network on which Password Manager was installed fell victim to a cyber intrusion then generating a new key should be considered an important part of the overall response and clean up procedure. Rekeying your Password Manager installation is simply a matter of clicking the red Generate button. Doing so will create a key and provide a success response:



DB Auto-purge controls how long the database should retain records for Clients that are no longer connecting to the server. The default value is 60 which means that the server will automatically delete records for clients with a last connect older than 60 days. Computers get retired and re-imaged as part of normal IT operations. Having the system automatically clean up dead client records will help keep the UI clean and the database small. A value of 60 days should be a reasonable value for most environments.

Permissions are required to adjust any of the settings in the Settings tab. Only members of the local Administrators group on the Password Manager server can adjust the settings. It is recommended that you keep the team of people with this privilege down to a relatively small set of individuals, as changes to these settings do have site wide effects. Due to limitations within the ASP.NET authorization engine, Administrator membership cannot be determined when using the web Console directly on the server (e.g. using Internet Explorer on the Password Manager server to connect to the web console). Settings changes need to be made from a remote browser. Attempts to modify the settings from a web browser running on the server will not be saved. This limitation may be addressed in future updates.

## Maintenance and Diagnostics

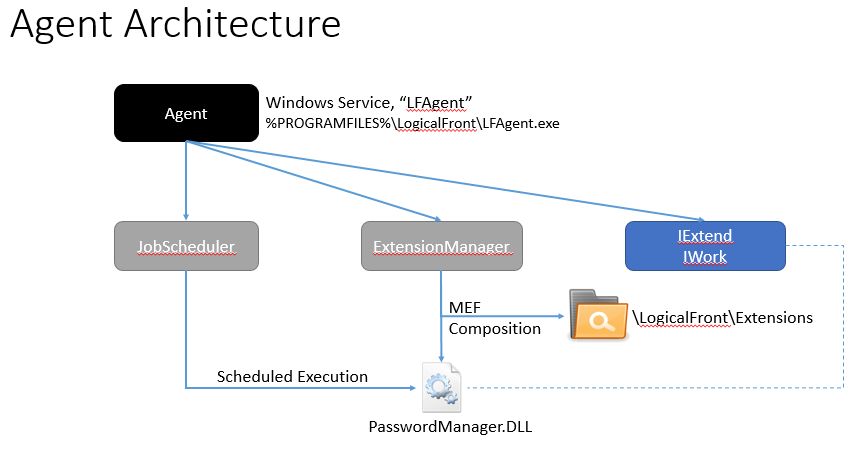
Password Manager is designed to run without much maintenance or administration. However, there are a few administration items that should be tended to with any Password Manager installation:

1. **SSL**. As suggested in the Installation Guide, configuring SSL for the Password Manager web console is very important. Without it, all passwords (even though the passwords are complex, unique, and rotating) will be sent in the clear over the wire allowing potential hackers to gain access.
2. **Backups**. While one of the strengths of Password Manager is that it does not store passwords centrally, there is the Master Key that must be protected. Master Keys are stored in the Keys table of the LogicalFront database. This database should be backed up as part of normal IT operations. Care should be taken to prevent unauthorized access to the SQL backup files.

An error log is maintained within the LogicalFront database to track significant events with the application. This log can be viewed by clicking on the System Status widget on the Dashboard tab (which will automatically change color when significant errors are detected), or from the Support tab. There is a text log for the server containing more verbose information which can be found at C:\Windows\Temp\PasswordManager.Server.Console.log. The data in this log is transient and will be overwritten when the application (or server) restarts.

On clients, a log can be found at: C:\Windows\Temp\LFAgent.log. This log, like the server log, is transient and will wrap at agent (or computer) restart. For critical errors (e.g. an error that the Agent encounters when setting the LocalAdmin password), a POST message is sent to the API of the server and the client error is logged centrally. You can then view these errors from the Error log of the web Console.

Part 3: Technical Design



The client runs inside of a Windows Service called LFAgent. The files for the client are located within a subdirectory of Program Files, or Program Files (x86) on 64-bit Windows, called Logical Front. Within this directory is the LFAgent executable, support libraries and a special folder called “Extensions”. The Extensions folder is used in a late assembly binding strategy called MEF Composition provided by the Microsoft Extensibility Framework (MEF). MEF Composition works by establishing a loosely coupled relationship between the agent service (LFAgent) and the DLL files that provide desired functionality (PasswordManager.dll). The loose coupling is provided by the IExtend and IWork interfaces. New functionality can be added to the agent by simply dropping a DLL that implements one of these interfaces into the Extensions folder.

After MEF Composition, the functionality in the PasswordManager DLL is executed via the JobScheduler. The JobScheduler reads the execution interval defined within the PasswordManager DLL file and handles all tasks associated with running the extensions on the defined interval. The JobScheduler checks to see if any IWork extensions are scheduled to run every second. When an extension is due for execution, JobScheduler creates a new thread and executes the Start method of the extension. The code contained within the Start method is granted 2 minutes to complete its processing. JobScheduler tracks elapsed time and will raise a HUNG WORKER THREAD exception if processing by the extension exceeds this two minute limit. In version 1.0 of the Agent, no automated action is taken in response to HUNG WORKER THREAD (other than logging), the extension will either complete or the machine will eventually. Upon restart, the extension will be reloaded, the execution interval will be awaited, and the process repeats. In the future, should additional extensions be developed and integrated into the Agent, the Agent will implement strict enforcement of the 2 minute limit.

For diagnostic and troubleshooting purposes, a log file is maintained by the Agent for its own operations as well as for the activity of Extensions. The file is called LFAgent.log and is found in the system Temp directory, usually located at C:\Windows\Temp. In addition, certain runtime metrics are written to the Windows registry at: HKLM\Software\Logical Front.

## Password Manager Extension

The PasswordManager extension provides the explicit functionality to manage the password and privileges of a local user account. PasswordManager implements the IWork extensibility interface described in the Agent section. The execution interval is set to 10 minutes, and so the Agent will attempt to execute PasswordManager once every 10 minutes. Client side randomization of the connect interval is not implemented. This is implemented in many client-server systems management tools but is deemed unnecessary for this solution. Even if all clients initially start at or around the same time, natural randomization is expected as clients drift out of phase with one another due to normal operations such as machines going to sleep, machines going off the network or shutdown. Additionally, the work of PasswordManager is extremely inexpensive from an I/O and computation standpoint thus diluting the value of complex load balancing strategies.

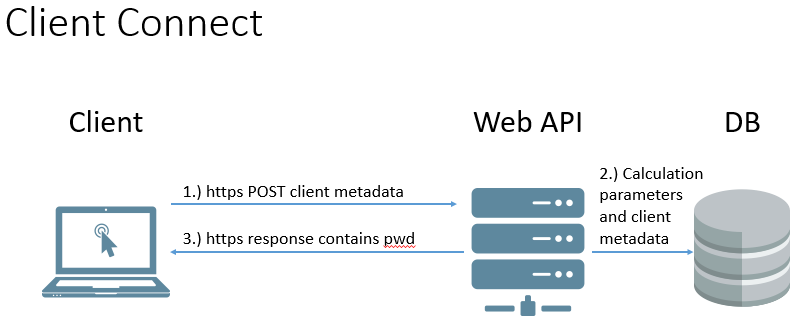
PasswordManager is hard coded to manage a local user account named “LocalAdmin”. Future versions could make this a configurable option, but keeping this standard simplifies things. If LocalAdmin does not exist, PasswordManager creates it. If LocalAdmin is not a member of the Administrators group, PasswordManager will add it. For the password, a REST call to the Logical Front API is made. The URL for the REST service is read from the Windows Registry at:

HKLM\SOFTWARE\Logical Front\PasswordManager\ServiceUrl

This value is set at installation time by the Agent installer, but can be overridden by modifying the Registry value. Windows protects this area of the Registry such that local overrides of the ServiceUrl value cannot be made by non-administrators. The password retrieved by the REST call is not cached or written to disk in any way other than through the Windows API call to set the password on LocalAdmin.

If errors are encountered by the PasswordManager extension when attempting to set the password, an exception is raised. The exception detail is written to the local LFAgent log, also the Agent will attempt to post the exception detail to the server for central reporting purposes.

## Client Connect Process:



The client attempts to contact the web api once every 10 minutes. The client connect is a lightweight operation sending only a small payload to the web API server. At time of this writing, only the computer OS name and device type (server, desktop, mobile) are sent to the server. The Web API POST method first gets the connecting computer’s name from the server’s Thread object. Specifically, a call to Thread.CurrentPrincipal.Identity.Name is used. This is important because the web API must authoritatively know who the caller is so that the appropriate password can be calculated. Thread.CurrentPrincipal is part of the .NET Security namespace and is intended specifically for Role Based security scenarios such as this. In addition to the caller’s name (samAccountName), the Web API must also know the Active Directory Organizational Unit (OU) the computer belongs to. A search query is performed using the caller’s samAccountName to obtain the caller’s OU and AD Guid. Once the caller identity is securely determined, the Web API then pulls from the database the age of the caller’s current password. Several branches then become possible:

1. A database record for the caller exists and the age of the password is within the allowable range. At this point, the Web API updates the LastSeen and OU fields of the caller’s database record along with the client’s metadata. The Web API then recalculates the existing password and returns it to the client. Even though a new password is not generated, the Web API returns the existing password to resynchronize the caller’s state with the server and to overwrite any out-of-band changes that occurred on the client since the previous Client Connect.
2. A database record for the caller exists and the age of the password is beyond the allowable range. The Web API calculates a new password, updates the caller’s LastSeen, OU and metadata fields, and returns the password to the client.
3. A database record does not yet exist for the caller. In this scenario, the Web API must do extra work to create a Computer table record for the caller. The Web API uses the caller’s name (retrieved from the connection’s Thread object discussed previously) to query Active Directory for the required Computer data, namely: the computer object’s ObjectGUID attribute, and OU. The caller’s Computer table record is created, and a password is calculated and returned to the caller.