Git Askpass for Windows

Git Askpass for Windows (Askpass) provides secure Git credential storage for Windows. Askpass provides multi-factor authentication support for Azure DevOps, Team Foundation Server, and GitHub.

Usage

After installation and configuration, Git will use the Git Askpass and you will only need to interact with any authentication dialogs asking for credentials. The Askpass stays invisible as much as possible, so ideally you'll forget that you're depending on Askpass at all.

```
git askpass
```

For Git to use Askpass correctly, the `GIT_ASKPASS` environment variable needs contain the full path to the `git-askpass.exe` executable (example: `setx GIT_ASKPASS "C:\Program Files\Git\mingw64\libexec\git-core\git-askpass.exe"`). SSH can also be configured to use Askpass in the same manner using the `SSH_ASKPASS` environment variable; however, SSH currently will not use Askpass if it detects a TTY console (ala the ability to just ask on the console).

Configuration Options

Git Credential Manager and Git Askpass work out of the box for most users. Configuration options are available to customize or tweak behavior(s).

The Git Credential Manager for Windows [GCM] can be configured using Git's configuration files, and follows all of the same rules Git does when consuming the files. Global configuration settings override system configuration settings, and local configuration settings override global settings; and because the configuration details exist within Git's configuration files you can use Git's `git config` utility to set, unset, and alter the setting values.

The GCM honors several levels of settings, in addition to the standard local > global > system tiering Git uses. Since the GCM is HTTPS based, it'll also honor URL specific settings. Regardless, all of the GCM's configuration settings begin with the term `credential`. Additionally, the GCM respects GCM specific environment variables as well.

Regardless, the GCM will only be used by Git if the GCM is installed and the key/value pair `credential.helper manager` is present in Git's configuration.

For example:

> `credential.microsoft.visualstudio.com.namespace` is more specific than `credential.visualstudio.com.namespace`, which is more specific than `credential.namespace`.

In the examples above, the `credential.namespace` setting would affect any remote repository; the `credential.visualstudio.com.namespace` would affect any remote repository in the domain, and/or any subdomain (including `www.`) of, 'visualstudio.com'; where as the the `credential.microsoft.visualstudio.com.namespace` setting would only be applied to remote repositories hosted at 'microsoft.visualstudio.com'.

For the complete list of settings the GCM understands, see the list below.

Configuration Setting Names

authority

Defines the type of authentication to be used.

Supports `Auto`, `Basic`, `AAD`, `MSA`, `GitHub`, `Bitbucket`, `Integrated`, and `NTLM`.

Use `AAD` or `MSA` if the host is 'visualstudio.com' Azure Domain or Live Account authentication, relatively.

Use `GitHub` if the host is 'github.com'.

Use `BitBucket` or `Atlassian` if the host is 'bitbucket.org'.

Use `Integrated` or `NTLM` if the host is a Team Foundation, or other NTLM authentication based, server.

Defaults to `Auto`.

```
git config --global credential.microsoft.visualstudio.com.authority AAD
```

See GCM_AUTHORITY

httpProxy

Causes the proxy value to be considered when evaluating credential target information. A proxy setting should established if use of a proxy is required to interact with Git remotes.

The value should the URL of the proxy server.

Defaults to not using a proxy server.

```
git config --global credential.github.com.httpProxy https://myproxy:8080
```

See HTTP_PROXY

interactive

Specifies if user can be prompted for credentials or not.

Supports `Auto`, `Always`, or `Never`. Defaults to `Auto`.

```
git config --global credential.microsoft.visualstudio.com.interactive never
```

See GCM_INTERACTIVE

modalPrompt

Forces authentication to use a modal dialog instead of asking for credentials at the command prompt.

Supports `true` or `false`. Defaults to `true`.

```
git config --global credential.modalPrompt true
```

See GCM_MODAL_PROMPT

namespace

Sets the namespace for stored credentials.

By default the GCM uses the ‘git’ namespace for all stored credentials, setting this configuration value allows for control of the namespace used globally, or per host.

Supports any ASCII, alpha-numeric only value. Defaults to `git`.

```
git config --global credential.namespace name
```

See GCM_NAMESPACE

preserve

Prevents the deletion of credentials even when they are reported as invalid by Git. Can lead to lockout situations once credentials expire and until those credentials are manually removed.

Supports `true` or `false`. Defaults to `false`.

```
git config --global credential.visualstudio.com.preserve true
```

See GCM_PRESERVE

httpTimeout

Sets the maximum time, in milliseconds, for a network request to wait before timing out. This allows changing the default for slow connections.

Supports an integer value. Defaults to 90,000 milliseconds.

```
git config --global credential.visualstudio.com.httpTimeout 100000
```

See GCM_HTTP_TIMEOUT

tokenDuration

Sets a duration, in hours, limit for the validity of Personal Access Tokens requested from Azure DevOps.

If the value is greater than the maximum duration set for the account, the account value supersedes. The value cannot be less than a one hour (1).

Defaults to the account token duration. Honored when authority is set to `AAD` or `MSA`.

```
git config --global credential.visualstudio.com.tokenDuration 24
```

See GCM_TOKEN_DURATION

useHttpPath

Instructs Git to supply the path portion of the remote URL to credential helpers. When path is supplied, the GCM will use the host-name + path as the key when reading and/or writing credentials.

Note: This option changes the behavior of Git.

Supports `true` or `false`. Defaults to `false`.

```
git config --global credential.bitbucket.com.useHttpPath true
```

username

Instructs Git to provide user-info to credential helpers. When user-info is supplied, the GCM will use the user-info + host-name as the key when reading and/or writing credentials. See RFC: URI Syntax, User Information for more details.

Note: This option changes the behavior of Git.

Supports any URI legal user-info. Defaults to not providing user-info.

```
git config --global credential.microsoft.visualstudio.com.username johndoe
```

validate

Causes validation of credentials before supplying them to Git. Invalid credentials get a refresh attempt before failing. Incurs minor network operation overhead.

Supports `true` or `false`. Defaults to `true`. Ignored when authority is set to `Basic`.

```
git config --global credential.microsoft.visualstudio.com.validate false
```

See GCM_VALIDATE

vstsScope

Overrides GCM default scope request when generating a Personal Access Token from Azure DevOps. The supported format is one or more scope values separated by whitespace, commas, semi-colons, or pipe `'|'` characters.

Defaults to `vso.code_write|vso.packaging`; Honored when host is ‘dev.azure.com’.

```
git config --global credential.microsoft.visualstudio.com.vstsScope vso.code_write|vso.packaging_write|vso.test_write
```

See GCM_VSTS_SCOPE

writeLog

Enables trace logging of all activities. Logs are written to the local `.git/` folder at the root of the repository.

Note: This setting will not override the `GCM_TRACE` environment variable.

Supports `true` or `false`. Defaults to `false`.

```
git config --global credential.writeLog true
```

See GCM_WRITELOG

Sample Configuration

```
[credential "microsoft.visualstudio.com"]
    authority = AAD
    interactive = never
    preserve = true
    tokenDuration = 24
    validate = false
[credential "visualstudio.com"]
    authority = MSA
[credential]
    helper = manager
    writelog = true
```

FAQ

If you have an issue using GCM or Askpass, please review the following FAQ and check our issue history or our Twitter feedbefore opening up an item on our issues page on GitHub.

Q: Why am always prompted for my username and password?

Most likely, your environment is not configured correctly. You can verify that your environment is configured correctly by running `git config --list` and looking for `credential.helper=manager`. If you do not see the line, then you know that Git does not know about the Git Credential Manager. You can configure Git to use the Credential Manager by running `git config credential.helper manager`.

Q: Why does my GUI freeze when I push, pull, or fetch?

Most likely reason is that your GUI “shells out” to git.exe to perform Git operations. When it does so, it cannot respond to the command line prompts for username and password like a real user can. To avoid being asked for your credentials on the command line, and instead be asked via a modal dialog you'll need to configure the Git Credential Manager.

1. Decide if you want this to be a global setting (all of your repositories) or a local setting (just one repository).
2. Start your favorite shell (CMD, PowerShell, bash, etc.).
3. Update your settings, so that Git Credential Manager knows to display a dialog and not prompt at the command line:
   - If you've decided this is a global setting run `git config --global credential.modalprompt true`.
   - If you've decided this a per repository setting, `cd` to your repo and in that repo run `git config credential.modalprompt true`.

Q: Why am I not seeing my SSH keys being saved?

The Git Credential Manager supports caching of SSH key password through git-askpass. Unfortunately, OpenSSH will only interact with an askpass helper if there no TTY detected (no console available). This mean, in general and for the vast majority of users, the GCM does not help with SSH passwords or certificates.

Fortunately, the Posh-Git support automatic startup of 'ssh-agent.exe' which does assist with SSH certificate password caching.

Q: Is there a NuGet Package available?

Yes there is: https://www.nuget.org/packages/Microsoft.Alm.Authentication. It supports the core authentication library and the VSTS specific components. If you're looking to extend the GCM, or need a way to authenticate with VSTS but cannot leverage the GCM directly, then it is likely what you're after.

Q: Why doesn't Git Credential Manager work on Windows XP, Mac OS, or Linux?

The Git Credential Manager does not work on Windows XP, Max OS, or Linux because we had to scope our work and we decided to support the same operating systems that Visual Studio support. Why Visual Studio? Well, because it is our favorite IDE and in order to support Azure DevOps we had to use the Azure Directory Authentication Libraries which only have multi-factor interactive logon support in their .NET libraries. Using .NET means using Visual Studio (which we love anyways) and using Visual Studio means Windows 7 or newer.

Q: Will there ever be support for Windows XP, Mac OS, or Linux?

We can safely say that we have no interest in supporting Windows XP. Even Microsoft has ended support for Windows XP. Support for Mac OS and Linux are handled by Microsoft Git Credential Manager for Mac and Linux.

Q: Why is my distribution/version of Git not supported? Why is Git for Windows favored?

The Credential Manager deployment helpers (`install.cmd` and `GCMW-{version}.exe`) are focused on support for Git for Windows because Git for Windows conforms to the expected/normal behavior of software on Windows. It is easy to detect, has predictable installation location, etc. This makes supporting it easier and more reliable.

That said, so long as your favorite version of Git supports Git's git-credential flow, it is supported by the Git Credential Manager for Windows. Setup will have to be manual, and if you find a way to script it we would love to have you contribute that to our project.

1. Copy the contents of the `gcm-<version>.zip` to your Git's /bin folder. This varies per distribution, but it is likely next to other git tools like `git-status.exe`.
2. Update your Git configuration by running `git config --global credential.helper manager`.

Q: I thought Microsoft was maintaining this, why does the GCM not work as expected with TFS?

Team Foundation Server, when deployed on a corporate Active Directory, uses the Microsoft Kerberos protocol for authentication. Git hasn’t traditionally be able to “speak” the Kerberos protocol. However, starting with v1.14.0 Git for Windows supports Microsoft Secure Channel. To enable Secure Channel support run `git config --global http.sslBackend=schannel` or selecting the ‘Native Windows Secure Channel library’ option during installation of Git for Windows. Secure Channel provides better integration with Windows’ networking and certificate management; enabling easier use of proxies and alternate authentication mechanisms previously unavailable to Git for Windows users.

Git needs to be convinced to “forward” credentials by supplying a blank credential set (username and password). The GCM will attempt to detect the Team Foundation Server via the HTTP headers returned when an unauthenticated request is handled by the server. If the server is configured to allow NTLM as a supported authentication protocol, the GCM will detect the setting and instruct Git to use NTLM instead of basic authentication.

Alternatively, you can configure the GCM to assume a host supports NTLM without checking. To do so, update your Git configuration by running `git config --global credential.{my-tfs}.authority NTLM`, where `{my-tfs}` can be replaced by the host name of your TFS server; the port number is not required for GCM configuration but you will want it for the Git remote.

Note: Previous versions of the GCM suggested using `git credential.{url}.integrated true`; while this configuration option continues to work, it has been deprecated in favor of specifying the correct authority.

Once updated, the new configuration tells the GCM to only negotiate via NTLM with the host and forward Windows credentials. Most likely, this is **not** what you want. Therefore, it strongly suggested that you restrict the configuration setting to the URL of your TFS Git host.

Q: Why doesn’t SourceTree use the credentials in the GCM?

You need to configure SourceTree to use the version of Git installed for the entire system. By default, SourceTree uses a local copy of portable Git.

To fix this go to ‘Tools > Options > Git’ and click the ‘Use System Git’ button. This works in v1.8.3.0 of SourceTree.

Q: Why is Git not using the GCM in some of my repositories (but instead using SSH authentication)?

Check that you are using the HTTP(S) URL instead of the SSH URL for your repository. You can do this by running `git remote show origin`. The Fetch URL and Push URL should start with `https://` or `http://`. If this is not the case, look for the HTTP(S) URL in the web interface of Azure DevOps, TFS, GitHub or Bitbucket, and then run `git remote set-url origin <url>`, where `<url>` is the HTTP(S) URL.

Q: Why is git.exe failing to authenticate after linking/unlinking your Azure DevOps account from Azure Active Directory?

When the tenant backing the Azure DevOps account changes like when you Connect VSTS account to Azure Active Directory (Azure AD), the tenant cache needs to be cleared if you’re using a GCM version prior to v1.15.0. Clearing the tenant cache is as easy as deleting the *%LocalAppData%.cache* file on each machine returning a login error like below. The GCM will automatically recreate and populate the cache file as needed on subsequent login attempts.

Example:

```
Error: cannot spawn askpass: No such file or directory
Error encountered while pushing to the remote repository: Git
failed with a fatal error.
could not read Username for ‘https://******.********.***’: ter
minal prompts disabled
```

Q: Why doesn’t the GCM work with ServicePointManager?

When you have `git config http.proxy` or `HTTPS_PROXY` configured to use [`ServicePointManager`]() proxy, and the URL doesn’t begin with `http://` the GCM will be unable to negotiate with the proxy. This is due the way the .NET Framework [`WebProxy`]() and `ServicePointManager` interact. You can read “[ServicePointManager does not support proxies of https scheme]()” for additional information about this issue.

The work around is to use a proxy URL which starts with `http://`, or discontinue use of `ServicePointManager`.

Build Agents and Automation

Build agents, and other automation, often require specialized setup and configuration. While there is detailed documentation on [GCM configuration options](), below are common recommendations for settings agents often require to operate.

Note: SSH is often a better choice for automated system because requiring interactivity is a non-default option, and SSH is known to be secure and reliable.

Recommendations for Azure DevOps Build Services

The majority of build definitions will work with a single repository, or at least a set of repositories which all have the same authentication requirements. In this case, it is generally better to rely on [Azure DevOps Build Variables](); specifically the `$(System.AccessToken)` build process OAuth token. To enable scripts to use the build process OAuth token, go to the `Options` tab of the build definition and select `Allow Scripts to Access OAuth Token`. For more information, read Azure DevOps: Use the OAuth token to access the REST API.

Recommendations for Other Build Services

Build agents cannot manage modal dialogs, therefore we recommended the following configuration.

```
git config --global credential.interactive never
```

Build agents often need to minimize the amount of network traffic they generate.

To avoid Microsoft Account vs. Azure Active Directory look-up against an Azure DevOps account use…

… for Azure Directory backed authentication:

```
git config --global credential.authority Azure
```

… for Microsoft Account backed authentication:

```
git config --global credential.authority Microsoft
```

… to restrict the lifetime of VSTS personal access tokens:

```
git config --global credential.tokenDuration 1
```

If your agents rely on an on premise instance of Team Foundation Server and Windows Domain Authentication, use:

```
git config --global credential.authority NTLM
```

To avoid unnecessary service account credential validation, when relying on Microsoft Account or Azure Active Directory use:

```
git config --global credential.validate false
```

Development and Debugging

Developing for GCM and/or Askpass requires Visual Studio 2017 or newer, any version (including the free Community Edition).

Getting Started

The easiest way to get started is to:

1. Install Visual Studio.
2. Clone the repository.
3. Open the solution file (GitCredentialManager.sln) using Visual Studio.
4. Right-click the solution node in Solution Explorer and choose ‘Restore NuGet Packages’. This will download and setup all of the dependencies.
5. Right-click the ‘Cli-CredentialHelper’ project in Solution Explorer and select ‘Properties’.
6. In the ‘Properties’ window, select the ‘Debug’ tab from the left side.
7. In the ‘Properties: Debug’ window, add the word “get” to the ‘Command line arguments:’ text box.
8. Close the “Properties” window.
9. Hit <F5>, or ‘Debug’ >> ‘Start Debugging’ from the top menu of Visual Studio.

Debugging Code

Once the GCM starts you’ll be presented with an *unsatisfying* console window. This is because the GCM expects to be launched by Git, not directly. However, it is easy to play the role of Git. The GCM expects Git to supply at least two pieces of information: the protocol being use and the host name for which the current operation is happening.

An example of faking Git request for GitHub credentials:

```
protocol=https
host=github.com
```

Notice the **blank last line**. That empty line is how Git and the GCM notify the other side that they're done sending data. Until an empty line is sent to the GCM, it'll keep attempting to read from standard input.

Once the blank line is fed to the GCM it'll "do its thing". Ideally, you can watch it work, so that you can learn how it works and then improve it. To do so, place a break point in the `Main` method of the 'Program.cs' file. Doing so will allow you to "break in" when the execution pointer reaches your break point. You'll notice that the GCM doesn't read from standard input immediately; instead it does some setup work to determine what it expected of it and then only reads from standard input if it is expected to.

In the case of "`git credential-manager get`", the `Main` method will call the `Program.Get` method, which in turn will allocate a new `OperationArguments` object and initialize it with the process' standard input pipe. This is when standard input will be consumed by the GCM.

Notable Code

- `Program.LoadOperationArguments` method is where the GCM scans, parses, and consumes environmental and configuration setting values.
- `Program.QueryCredentials` method is where the "action" happens.
- `OperationArguments` class is where the GCM consumes standard input and keeps internal state.

Installer

Changes to the installer (`GCMW-{version}.exe`) requires Inno Setup Compiler 5.5.9 or later to compile. Additionally, the IDP plugin for Inno Setup is also required.

The setup compiler pulls content from the "Deploy/" folder, therefore a completed Debug or Release build needs to have been completed prior to running the setup compiler. Content in the "Deploy/" folder will be used in the setup compilation.

Microsoft.Alm.Authentication NuGet Package

The Microsoft.Alm.Authentication NuGet package is automatically created when the Microsoft.Alm.Authentication project is built. The generated ".nupkg" files can be found in the "Debug/" or "Release/" (depending on your build target) under "Microsoft.Alm.Authentication/bin/". Both the binary and symbol packages are automatically created.

Updates to the NuGet package stream are reserved for officially built binaries.

Signing

Only Microsoft can sign binaries with the Microsoft certificate. Therefore, while anyone can build and use their own binaries from the GCM source code, only officially releases binaries will be signed by Microsoft.

Documents

The documentation is formatted using markdown and generated using Pandoc.

Logging

To enable logging, use the following:

```
git config --global credential.writelog true
```

Log files will be written to the repository's local `.git/` folder.

Additionally, the GCM outputs *GIT_TRACE* compatible logging. To use the *GIT_TRACE* mechanism you can either:

1. Open a Command Prompt.
2. Run `SET GCM_TRACE=1`; this will cause the GCM to log directly to the console.
3. If you'd like Git to interleave its traces as well run `SET GIT_TRACE=1`; now Git will log directly to the console.

Or

1. Open a Command Prompt.
2. Run `setx GCM_TRACE %UserProfile%\git.log`; this will cause the GCM to log to a file located at "%UserProfile%.log".

3. Close and re-open any console windows, and restart any application which need to make use of the new environment variable.
4. If you'd like Git to interleave its traces as well run `setx GIT_TRACE %UserProfile%\git.log`; now Git will log to the file located at "%UserProfile%.log".
5. To remove the environment variable, run `setx GCM_TRACE ""` or `GIT_TRACE ""`.

Note: The path for logging is arbitrary and both GCM and Git will create/append the file, however neither Git nor the GCM will create any folders; therefore it is up to the user to specify a folder that exists. Additionally, if the specified path contains spaces, be sure to wrap the path in double-quote characters (*example:* "%UserProfile%git.log"). Finally, inaccessible paths could cause either Git or the GCM to fail unexpectedly, therefor avoid specifying paths which the user's account does not have access to; for example paths like "%ProgramFiles%.log" are not recommended because only elevated processes can write to "%ProgramFiles%".

Debug build of the GCM will perform extended logging to the console, which is convenient for debugging purposes bug too noisy for day-to-day usage.