Understanding privilege escalation: become

Ansible uses existing privilege escalation systems to execute tasks with root privileges or with another user's permissions. Because this feature allows you to 'become' another user, different from the user that logged into the machine (remote user), we call it `become`. The `become` keyword uses existing privilege escalation tools like *sudo*, *su*, *pfexec*, *doas*, *pbrun*, *dzdo*, *ksu*, *runas*, *machinectl* and others.

- Using become
  - Become directives
  - Become connection variables
  - Become command-line options
- Risks and limitations of become
  - Risks of becoming an unprivileged user
  - Not supported by all connection plugins
  - Only one method may be enabled per host
  - Privilege escalation must be general
  - May not access environment variables populated by pamd_systemd
- Become and network automation
  - Setting enable mode for all tasks
    - Passwords for enable mode
  - authorize and auth_pass
- Become and Windows
  - Administrative rights
  - Local service accounts
  - Become without setting a password
  - Accounts without a password
  - Become flags for Windows
  - Limitations of become on Windows
  - Resolving Temporary File Error Messsages

Using become

You can control the use of `become` with play or task directives, connection variables, or at the command line. If you set privilege escalation properties in multiple ways, review the :ref:`general precedence rules<general_precedence_rules>` to understand which settings will be used.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst`, line 15); *backlink*

Unknown interpreted text role "ref".

A full list of all become plugins that are included in Ansible can be found in the :ref:`become_plugin_list`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst`, line 17); *backlink*

Unknown interpreted text role "ref".

Become directives

You can set the directives that control `become` at the play or task level. You can override these by setting connection variables, which often differ from one host to another. These variables and directives are independent. For example, setting `become_user` does not set `become`.

become

> set to `yes` to activate privilege escalation.

become_user

> set to user with desired privileges â€" the user you *become*, NOT the user you login as. Does NOT imply `become: yes`, to allow it to be set at host level. Default value is `root`.

become_method

> (at play or task level) overrides the default method set in ansible.cfg, set to use any of the :ref:`become_plugins`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst`, line 31); *backlink*

Unknown interpreted text role "ref".

become_flags

> (at play or task level) permit the use of specific flags for the tasks or role. One common use is to change the user to nobody when the shell is set to nologin. Added in Ansible 2.2.

For example, to manage a system service (which requires `root` privileges) when connected as a non-`root` user, you can use the default value of `become_user` (`root`):

```
- name: Ensure the httpd service is running
  service:
    name: httpd
    state: started
  become: yes
```

To run a command as the `apache` user:

```
- name: Run a command as the apache user
  command: somecommand
  become: yes
  become_user: apache
```

To do something as the `nobody` user when the shell is nologin:

```
- name: Run a command as nobody
  command: somecommand
  become: yes
  become_method: su
  become_user: nobody
  become_flags: '-s /bin/sh'
```

To specify a password for sudo, run `ansible-playbook` with `--ask-become-pass` (`-K` for short). If you run a playbook utilizing `become` and the playbook seems to hang, most likely it is stuck at the privilege escalation prompt. Stop it with *CTRL-c*, then execute the playbook with `-K` and the appropriate password.

Become connection variables

You can define different `become` options for each managed node or group. You can define these variables in inventory or use them as normal variables.

ansible_become

> overrides the `become` directive, decides if privilege escalation is used or not.

ansible_become_method

> which privilege escalation method should be used

ansible_become_user

> set the user you become through privilege escalation; does not imply `ansible_become: yes`

ansible_become_password

> set the privilege escalation password. See :ref:`playbooks_vault` for details on how to avoid having secrets in plain text

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst`, line 84); *backlink*

Unknown interpreted text role "ref".

ansible_common_remote_group

> determines if Ansible should try to `chgrp` its temporary files to a group if `setfacl` and `chown` both fail. See Risks of becoming an unprivileged user for more information. Added in version 2.10.

For example, if you want to run all tasks as `root` on a server named `webserver`, but you can only connect as the `manager` user, you could use an inventory entry like this:

```
webserver ansible_user=manager ansible_become=yes
```

Note

The variables defined above are generic for all become plugins but plugin specific ones can also be set instead. Please see the documentation for each plugin for a list of all options the plugin has and how they can be defined. A full list of become plugins in Ansible can be found at :ref:`become_plugins`.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst`, line 96); *backlink*
>
> Unknown interpreted text role "ref".

Become command-line options

`--ask-become-pass, -K`
ask for privilege escalation password; does not imply become will be used. Note that this password will be used for all hosts.

`--become, -b`
run operations with become (no password implied)

`--become-method=`*BECOME_METHOD*
privilege escalation method to use (default=sudo), valid choices: [sudo | su | pbrun | pfexec | doas | dzdo | ksu | runas | machinectl]

`--become-user=`*BECOME_USER*
run operations as this user (default=root), does not imply --become/-b

Risks and limitations of become

Although privilege escalation is mostly intuitive, there are a few limitations on how it works. Users should be aware of these to avoid surprises.

Risks of becoming an unprivileged user

Ansible modules are executed on the remote machine by first substituting the parameters into the module file, then copying the file to the remote machine, and finally executing it there.

Everything is fine if the module file is executed without using `become`, when the `become_user` is root, or when the connection to the remote machine is made as root. In these cases Ansible creates the module file with permissions that only allow reading by the user and root, or only allow reading by the unprivileged user being switched to.

However, when both the connection user and the `become_user` are unprivileged, the module file is written as the user that Ansible connects as (the `remote_user`), but the file needs to be readable by the user Ansible is set to `become`. The details of how Ansible solves this can vary based on platform. However, on POSIX systems, Ansible solves this problem in the following way:

First, if :command:`setfacl` is installed and available in the remote `PATH`, and the temporary directory on the remote host is mounted with POSIX.1e filesystem ACL support, Ansible will use POSIX ACLs to share the module file with the second unprivileged user.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst`, line 141); *backlink*
>
> Unknown interpreted text role "command".

Next, if POSIX ACLs are **not** available or :command:`setfacl` could not be run, Ansible will attempt to change ownership of the module file using :command:`chown` for systems which support doing so as an unprivileged user.

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst`, line 146); *backlink*
>
> Unknown interpreted text role "command".

> **System Message: ERROR/3** (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst`, line 146); *backlink*
>
> Unknown interpreted text role "command".

New in Ansible 2.11, at this point, Ansible will try :command:`chmod +a` which is a macOS-specific way of setting ACLs on files.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel) (docs) (docsite) (rst) (user_guide)become.rst`, line 150); *backlink*

Unknown interpreted text role "command".

New in Ansible 2.10, if all of the above fails, Ansible will then check the value of the configuration setting `ansible_common_remote_group`. Many systems will allow a given user to change the group ownership of a file to a group the user is in. As a result, if the second unprivileged user (the `become_user`) has a UNIX group in common with the user Ansible is connected as (the `remote_user`), and if `ansible_common_remote_group` is defined to be that group, Ansible can try to change the group ownership of the module file to that group by using :command:`chgrp`, thereby likely making it readable to the `become_user`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel) (docs) (docsite) (rst) (user_guide)become.rst`, line 153); *backlink*

Unknown interpreted text role "command".

At this point, if `ansible_common_remote_group` was defined and a :command:`chgrp` was attempted and returned successfully, Ansible assumes (but, importantly, does not check) that the new group ownership is enough and does not fall back further. That is, Ansible **does not check** that the `become_user` does in fact share a group with the `remote_user`; so long as the command exits successfully, Ansible considers the result successful and does not proceed to check `allow_world_readable_tmpfiles` per below.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel) (docs) (docsite) (rst) (user_guide)become.rst`, line 163); *backlink*

Unknown interpreted text role "command".

If `ansible_common_remote_group` is **not** set and the chown above it failed, or if `ansible_common_remote_group` *is* set but the :command:`chgrp` (or following group-permissions :command:`chmod`) returned a non-successful exit code, Ansible will lastly check the value of `allow_world_readable_tmpfiles`. If this is set, Ansible will place the module file in a world-readable temporary directory, with world-readable permissions to allow the `become_user` (and incidentally any other user on the system) to read the contents of the file. **If any of the parameters passed to the module are sensitive in nature, and you do not trust the remote machines, then this is a potential security risk.**

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel) (docs) (docsite) (rst) (user_guide)become.rst`, line 171); *backlink*

Unknown interpreted text role "command".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel) (docs) (docsite) (rst) (user_guide)become.rst`, line 171); *backlink*

Unknown interpreted text role "command".

Once the module is done executing, Ansible deletes the temporary file.

Several ways exist to avoid the above logic flow entirely:

- Use *pipelining*. When pipelining is enabled, Ansible does not save the module to a temporary file on the client. Instead it pipes the module to the remote python interpreter's stdin. Pipelining does not work for python modules involving file transfer (for example: :ref:`copy <copy_module>`, :ref:`fetch <fetch_module>`, :ref:`template <template_module>`), or for non-python modules.

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel) (docs) (docsite) (rst) (user_guide)become.rst`, line 186); *backlink*

  Unknown interpreted text role "ref".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst`, line 186); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst`, line 186); *backlink*

Unknown interpreted text role "ref".

- Avoid becoming an unprivileged user. Temporary files are protected by UNIX file permissions when you `become` root or do not use `become`. In Ansible 2.1 and above, UNIX file permissions are also secure if you make the connection to the managed machine as root and then use `become` to access an unprivileged account.

Warning

Although the Solaris ZFS filesystem has filesystem ACLs, the ACLs are not POSIX.1e filesystem acls (they are NFSv4 ACLs instead). Ansible cannot use these ACLs to manage its temp file permissions so you may have to resort to `allow_world_readable_tmpfiles` if the remote machines use ZFS.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst`, line 203)

Unknown directive type "versionchanged".

```
.. versionchanged:: 2.1
```

Ansible makes it hard to unknowingly use `become` insecurely. Starting in Ansible 2.1, Ansible defaults to issuing an error if it cannot execute securely with `become`. If you cannot use pipelining or POSIX ACLs, must connect as an unprivileged user, must use `become` to execute as a different unprivileged user, and decide that your managed nodes are secure enough for the modules you want to run there to be world readable, you can turn on `allow_world_readable_tmpfiles` in the :file:`ansible.cfg` file. Setting `allow_world_readable_tmpfiles` will change this from an error into a warning and allow the task to run as it did prior to 2.1.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst`, line 205); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst`, line 215)

Unknown directive type "versionchanged".

```
.. versionchanged:: 2.10
```

Ansible 2.10 introduces the above-mentioned `ansible_common_remote_group` fallback. As mentioned above, if enabled, it is used when `remote_user` and `become_user` are both unprivileged users. Refer to the text above for details on when this fallback happens.

Warning

As mentioned above, if `ansible_common_remote_group` and `allow_world_readable_tmpfiles` are both enabled, it is unlikely that the world-readable fallback will ever trigger, and yet Ansible might still be unable to access the module file. This is because after the group ownership change is successful, Ansible does not fall back any further, and also does not do any check to ensure that the `become_user` is actually a member of the "common group". This is a design decision made by the fact that doing such a check would require another round-trip connection to the remote machine, which is a time-expensive operation. Ansible does, however, emit a warning in this case.

Not supported by all connection plugins

Privilege escalation methods must also be supported by the connection plugin used. Most connection plugins will warn if they do not support become. Some will just ignore it as they always run as root (jail, chroot, and so on).

Only one method may be enabled per host

Methods cannot be chained. You cannot use `sudo /bin/su -` to become a user, you need to have privileges to run the command as that user in sudo or be able to su directly to it (the same for pbrun, pfexec or other supported methods).

Privilege escalation must be general

You cannot limit privilege escalation permissions to certain commands. Ansible does not always use a specific command to do something but runs modules (code) from a temporary file name which changes every time. If you have '/sbin/service' or '/bin/chmod' as the allowed commands this will fail with ansible as those paths won't match with the temporary file that Ansible creates to run the module. If you have security rules that constrain your sudo/pbrun/doas environment to running specific command paths only, use Ansible from a special account that does not have this constraint, or use AWX or the :ref:`ansible_platform` to manage indirect access to SSH credentials.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst`, line 250); ***backlink***

Unknown interpreted text role "ref".

May not access environment variables populated by pamd_systemd

For most Linux distributions using `systemd` as their init, the default methods used by `become` do not open a new "session", in the sense of systemd. Because the `pam_systemd` module will not fully initialize a new session, you might have surprises compared to a normal session opened through ssh: some environment variables set by `pam_systemd`, most notably `XDG_RUNTIME_DIR`, are not populated for the new user and instead inherited or just emptied.

This might cause trouble when trying to invoke systemd commands that depend on `XDG_RUNTIME_DIR` to access the bus:

```
$ echo $XDG_RUNTIME_DIR

$ systemctl --user status
Failed to connect to bus: Permission denied
```

To force `become` to open a new systemd session that goes through `pam_systemd`, you can use `become_method: machinectl`.

For more information, see this systemd issue.

Become and network automation

As of version 2.6, Ansible supports `become` for privilege escalation (entering `enable` mode or privileged EXEC mode) on all Ansible-maintained network platforms that support `enable` mode. Using `become` replaces the `authorize` and `auth_pass` options in a `provider` dictionary.

You must set the connection type to either `connection: ansible.netcommon.network_cli` or `connection: ansible.netcommon.httpapi` to use `become` for privilege escalation on network devices. Check the :ref:`platform_options` documentation for details.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst`, line 294); ***backlink***

Unknown interpreted text role "ref".

You can use escalated privileges on only the specific tasks that need them, on an entire play, or on all plays. Adding `become: yes` and `become_method: enable` instructs Ansible to enter `enable` mode before executing the task, play, or playbook where those parameters are set.

If you see this error message, the task that generated it requires `enable` mode to succeed:

```
Invalid input (privileged mode required)
```

To set `enable` mode for a specific task, add `become` at the task level:

```
- name: Gather facts (eos)
  arista.eos.eos_facts:
    gather_subset:
      - "!hardware"
  become: yes
  become_method: enable
```

To set enable mode for all tasks in a single play, add `become` at the play level:

```
- hosts: eos-switches
  become: yes
  become_method: enable
  tasks:
    - name: Gather facts (eos)
      arista.eos.eos_facts:
        gather_subset:
          - "!hardware"
```

Setting enable mode for all tasks

Often you wish for all tasks in all plays to run using privilege mode, that is best achieved by using `group_vars`:

group_vars/eos.yml

```
ansible_connection: ansible.netcommon.network_cli
ansible_network_os: arista.eos.eos
ansible_user: myuser
ansible_become: yes
ansible_become_method: enable
```

Passwords for enable mode

If you need a password to enter `enable` mode, you can specify it in one of two ways:

- providing the :option:`--ask-become-pass <ansible-playbook --ask-become-pass>` command line option

  System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst`, line 348); *backlink*

  Unknown interpreted text role "option".

- setting the `ansible_become_password` connection variable

Warning

As a reminder passwords should never be stored in plain text. For information on encrypting your passwords and other secrets with Ansible Vault, see :ref:`vault`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst`, line 353); *backlink*

Unknown interpreted text role "ref".

authorize and auth_pass

Ansible still supports `enable` mode with `connection: local` for legacy network playbooks. To enter `enable` mode with `connection: local`, use the module options `authorize` and `auth_pass`:

```
- hosts: eos-switches
  ansible_connection: local
  tasks:
    - name: Gather facts (eos)
      eos_facts:
        gather_subset:
          - "!hardware"
      provider:
        authorize: yes
        auth_pass: " {{ secret_auth_pass }}"
```

We recommend updating your playbooks to use `become` for network-device `enable` mode consistently. The use of `authorize` and of `provider` dictionaries will be deprecated in future. Check the :ref:`platform_options` and :ref:`network_modules` documentation for details.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst, line 373); *backlink*

Unknown interpreted text role "ref".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst, line 373); *backlink*

Unknown interpreted text role "ref".

Become and Windows

Since Ansible 2.3, `become` can be used on Windows hosts through the `runas` method. Become on Windows uses the same inventory setup and invocation arguments as `become` on a non-Windows host, so the setup and variable names are the same as what is defined in this document.

While `become` can be used to assume the identity of another user, there are other uses for it with Windows hosts. One important use is to bypass some of the limitations that are imposed when running on WinRM, such as constrained network delegation or accessing forbidden system calls like the WUA API. You can use `become` with the same user as `ansible_user` to bypass these limitations and run commands that are not normally accessible in a WinRM session.

Administrative rights

Many tasks in Windows require administrative privileges to complete. When using the `runas` become method, Ansible will attempt to run the module with the full privileges that are available to the remote user. If it fails to elevate the user token, it will continue to use the limited token during execution.

A user must have the `SeDebugPrivilege` to run a become process with elevated privileges. This privilege is assigned to Administrators by default. If the debug privilege is not available, the become process will run with a limited set of privileges and groups.

To determine the type of token that Ansible was able to get, run the following task:

```
- Check my user name
  ansible.windows.win_whoami:
  become: yes
```

The output will look something similar to the below:

System Message: WARNING/2 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst, line 416)

Cannot analyze code. No Pygments lexer found for "ansible-output".

```
.. code-block:: ansible-output

    ok: [windows] => {
        "account": {
            "account_name": "vagrant-domain",
            "domain_name": "DOMAIN",
            "sid": "S-1-5-21-3088887838-4058132883-1884671576-1105",
            "type": "User"
        },
        "authentication_package": "Kerberos",
        "changed": false,
        "dns_domain_name": "DOMAIN.LOCAL",
        "groups": [
            {
                "account_name": "Administrators",
                "attributes": [
                    "Mandatory",
                    "Enabled by default",
                    "Enabled",
                    "Owner"
                ],
                "domain_name": "BUILTIN",
                "sid": "S-1-5-32-544",
                "type": "Alias"
            },
            {
                "account_name": "INTERACTIVE",
                "attributes": [
                    "Mandatory",
                    "Enabled by default",
                    "Enabled"
                ],
                "domain_name": "NT AUTHORITY",
                "sid": "S-1-5-4",
                "type": "WellKnownGroup"
            },
        ],
        "impersonation_level": "SecurityAnonymous",
        "label": {
            "account_name": "High Mandatory Level",
            "domain_name": "Mandatory Label",
            "sid": "S-1-16-12288",
            "type": "Label"
        },
        "login_domain": "DOMAIN",
        "login_time": "2018-11-18T20:35:01.9696884+00:00",
        "logon_id": 114196830,
        "logon_server": "DC01",
        "logon_type": "Interactive",
        "privileges": {
            "SeBackupPrivilege": "disabled",
            "SeChangeNotifyPrivilege": "enabled-by-default",
            "SeCreateGlobalPrivilege": "enabled-by-default",
            "SeCreatePagefilePrivilege": "disabled",
            "SeCreateSymbolicLinkPrivilege": "disabled",
            "SeDebugPrivilege": "enabled",
            "SeDelegateSessionUserImpersonatePrivilege": "disabled",
            "SeImpersonatePrivilege": "enabled-by-default",
            "SeIncreaseBasePriorityPrivilege": "disabled",
            "SeIncreaseQuotaPrivilege": "disabled",
            "SeIncreaseWorkingSetPrivilege": "disabled",
            "SeLoadDriverPrivilege": "disabled",
            "SeManageVolumePrivilege": "disabled",
            "SeProfileSingleProcessPrivilege": "disabled",
            "SeRemoteShutdownPrivilege": "disabled",
            "SeRestorePrivilege": "disabled",
            "SeSecurityPrivilege": "disabled",
            "SeShutdownPrivilege": "disabled",
            "SeSystemEnvironmentPrivilege": "disabled",
            "SeSystemProfilePrivilege": "disabled",
            "SeSystemtimePrivilege": "disabled",
            "SeTakeOwnershipPrivilege": "disabled",
            "SeTimeZonePrivilege": "disabled",
            "SeUndockPrivilege": "disabled"
        },
        "rights": [
            "SeNetworkLogonRight",
            "SeBatchLogonRight",
            "SeInteractiveLogonRight",
            "SeRemoteInteractiveLogonRight"
        ],
        "token_type": "TokenPrimary",
        "upn": "vagrant-domain@DOMAIN.LOCAL",
        "user_flags": []
    }
```

Under the `label` key, the `account_name` entry determines whether the user has Administrative rights. Here are the labels that can be returned and what they represent:

- `Medium`: Ansible failed to get an elevated token and ran under a limited token. Only a subset of the privileges assigned to user are available during the module execution and the user does not have administrative rights.
- `High`: An elevated token was used and all the privileges assigned to the user are available during the module execution.
- `System`: The `NT AUTHORITY\System` account is used and has the highest level of privileges available.

The output will also show the list of privileges that have been granted to the user. When the privilege value is `disabled`, the privilege is assigned to the logon token but has not been enabled. In most scenarios these privileges are automatically enabled when required.

If running on a version of Ansible that is older than 2.5 or the normal `runas` escalation process fails, an elevated token can be retrieved by:

- Set the `become_user` to `System` which has full control over the operating system.
- Grant `SeTcbPrivilege` to the user Ansible connects with on WinRM. `SeTcbPrivilege` is a high-level privilege that grants full control over the operating system. No user is given this privilege by default, and care should be taken if you grant this privilege to a user or group. For more information on this privilege, please see Act as part of the operating system. You can use the below task to set this privilege on a Windows host:

```
- name: grant the ansible user the SeTcbPrivilege right
```

```
  ansible.windows.win_user_right:
    name: SeTcbPrivilege
    users: '{{ansible_user}}'
    action: add
```

- Turn UAC off on the host and reboot before trying to become the user. UAC is a security protocol that is designed to run accounts with the `least privilege` principle. You can turn UAC off by running the following tasks:

```
- name: turn UAC off
  win_regedit:
    path: HKLM:\SOFTWARE\Microsoft\Windows\CurrentVersion\policies\system
    name: EnableLUA
    data: 0
    type: dword
    state: present
  register: uac_result

- name: reboot after disabling UAC
  win_reboot:
  when: uac_result is changed
```

> **Note**
>
> Granting the `SeTcbPrivilege` or turning UAC off can cause Windows security vulnerabilities and care should be given if these steps are taken.

Local service accounts

Prior to Ansible version 2.5, `become` only worked on Windows with a local or domain user account. Local service accounts like `System` or `NetworkService` could not be used as `become_user` in these older versions. This restriction has been lifted since the 2.5 release of Ansible. The three service accounts that can be set under `become_user` are:

- System
- NetworkService
- LocalService

Because local service accounts do not have passwords, the `ansible_become_password` parameter is not required and is ignored if specified.

Become without setting a password

As of Ansible 2.8, `become` can be used to become a Windows local or domain account without requiring a password for that account. For this method to work, the following requirements must be met:

- The connection user has the `SeDebugPrivilege` privilege assigned
- The connection user is part of the `BUILTIN\Administrators` group
- The `become_user` has either the `SeBatchLogonRight` or `SeNetworkLogonRight` user right

Using become without a password is achieved in one of two different methods:

- Duplicating an existing logon session's token if the account is already logged on
- Using S4U to generate a logon token that is valid on the remote host only

In the first scenario, the become process is spawned from another logon of that user account. This could be an existing RDP logon, console logon, but this is not guaranteed to occur all the time. This is similar to the `Run only when user is logged on` option for a Scheduled Task.

In the case where another logon of the become account does not exist, S4U is used to create a new logon and run the module through that. This is similar to the `Run whether user is logged on or not` with the `Do not store password` option for a Scheduled Task. In this scenario, the become process will not be able to access any network resources like a normal WinRM process.

To make a distinction between using become with no password and becoming an account that has no password make sure to keep `ansible_become_password` as undefined or set `ansible_become_password:`.

> **Note**
>
> Because there are no guarantees an existing token will exist for a user when Ansible runs, there's a high change the become process will only have access to local resources. Use become with a password if the task needs to access network resources

Accounts without a password

> **Warning**

As a general security best practice, you should avoid allowing accounts without passwords.

Ansible can be used to become a Windows account that does not have a password (like the `Guest` account). To become an account without a password, set up the variables like normal but set `ansible_become_password: ''`.

Before become can work on an account like this, the local policy Accounts: Limit local account use of blank passwords to console logon only must be disabled. This can either be done through a Group Policy Object (GPO) or with this Ansible task:

```
- name: allow blank password on become
  ansible.windows.win_regedit:
    path: HKLM:\SYSTEM\CurrentControlSet\Control\Lsa
    name: LimitBlankPasswordUse
    data: 0
    type: dword
    state: present
```

Note

This is only for accounts that do not have a password. You still need to set the account's password under `ansible_become_password` if the become_user has a password.

Become flags for Windows

Ansible 2.5 added the `become_flags` parameter to the `runas` become method. This parameter can be set using the `become_flags` task directive or set in Ansible's configuration using `ansible_become_flags`. The two valid values that are initially supported for this parameter are `logon_type` and `logon_flags`.

Note

These flags should only be set when becoming a normal user account, not a local service account like LocalSystem.

The key `logon_type` sets the type of logon operation to perform. The value can be set to one of the following:

- `interactive`: The default logon type. The process will be run under a context that is the same as when running a process locally. This bypasses all WinRM restrictions and is the recommended method to use.
- `batch`: Runs the process under a batch context that is similar to a scheduled task with a password set. This should bypass most WinRM restrictions and is useful if the `become_user` is not allowed to log on interactively.
- `new_credentials`: Runs under the same credentials as the calling user, but outbound connections are run under the context of the `become_user` and `become_password`, similar to `runas.exe /netonly`. The `logon_flags` flag should also be set to `netcredentials_only`. Use this flag if the process needs to access a network resource (like an SMB share) using a different set of credentials.
- `network`: Runs the process under a network context without any cached credentials. This results in the same type of logon session as running a normal WinRM process without credential delegation, and operates under the same restrictions.
- `network_cleartext`: Like the `network` logon type, but instead caches the credentials so it can access network resources. This is the same type of logon session as running a normal WinRM process with credential delegation.

For more information, see dwLogonType.

The `logon_flags` key specifies how Windows will log the user on when creating the new process. The value can be set to none or multiple of the following:

- `with_profile`: The default logon flag set. The process will load the user's profile in the `HKEY_USERS` registry key to `HKEY_CURRENT_USER`.
- `netcredentials_only`: The process will use the same token as the caller but will use the `become_user` and `become_password` when accessing a remote resource. This is useful in inter-domain scenarios where there is no trust relationship, and should be used with the `new_credentials` `logon_type`.

By default `logon_flags=with_profile` is set, if the profile should not be loaded set `logon_flags=` or if the profile should be loaded with `netcredentials_only`, set `logon_flags=with_profile,netcredentials_only`.

For more information, see dwLogonFlags.

Here are some examples of how to use `become_flags` with Windows tasks:

```
- name: copy a file from a fileshare with custom credentials
  ansible.windows.win_copy:
    src: \\server\share\data\file.txt
    dest: C:\temp\file.txt
    remote_src: yes
  vars:
    ansible_become: yes
    ansible_become_method: runas
    ansible_become_user: DOMAIN\user
    ansible_become_password: Password01
    ansible_become_flags: logon_type=new_credentials logon_flags=netcredentials_only

- name: run a command under a batch logon
  ansible.windows.win_whoami:
  become: yes
  become_flags: logon_type=batch

- name: run a command and not load the user profile
  ansible.windows.win_whomai:
  become: yes
  become_flags: logon_flags=
```

Limitations of become on Windows

- Running a task with `async` and `become` on Windows Server 2008, 2008 R2 and Windows 7 only works when using Ansible 2.7 or newer.
- By default, the become user logs on with an interactive session, so it must have the right to do so on the Windows host. If it does not inherit the `SeAllowLogOnLocally` privilege or inherits the `SeDenyLogOnLocally` privilege, the become process will fail. Either add the privilege or set the `logon_type` flag to change the logon type used.
- Prior to Ansible version 2.3, become only worked when `ansible_winrm_transport` was either `basic` or `credssp`. This restriction has been lifted since the 2.4 release of Ansible for all hosts except Windows Server 2008 (non R2 version).
- The Secondary Logon service `seclogon` must be running to use `ansible_become_method: runas`

Resolving Temporary File Error Messsages

"Failed to set permissions on the temporary files Ansible needs to create when becoming an unprivileged user" * This error can be resolved by installing the package that provides the `setfacl` command. (This is frequently the `acl` package but check your OS documentation.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)become.rst`, line 759)

Unknown directive type "seealso".

```
.. seealso::

   `Mailing List <https://groups.google.com/forum/#!forum/ansible-project>`_
       Questions? Help? Ideas?  Stop by the list on Google Groups
   :ref:`communication_irc`
       How to join Ansible chat channels
```