Linux DFIR - bash login sequence

Taz Wake | Halkyn Consulting Ltd | Originally published 2025-03-11

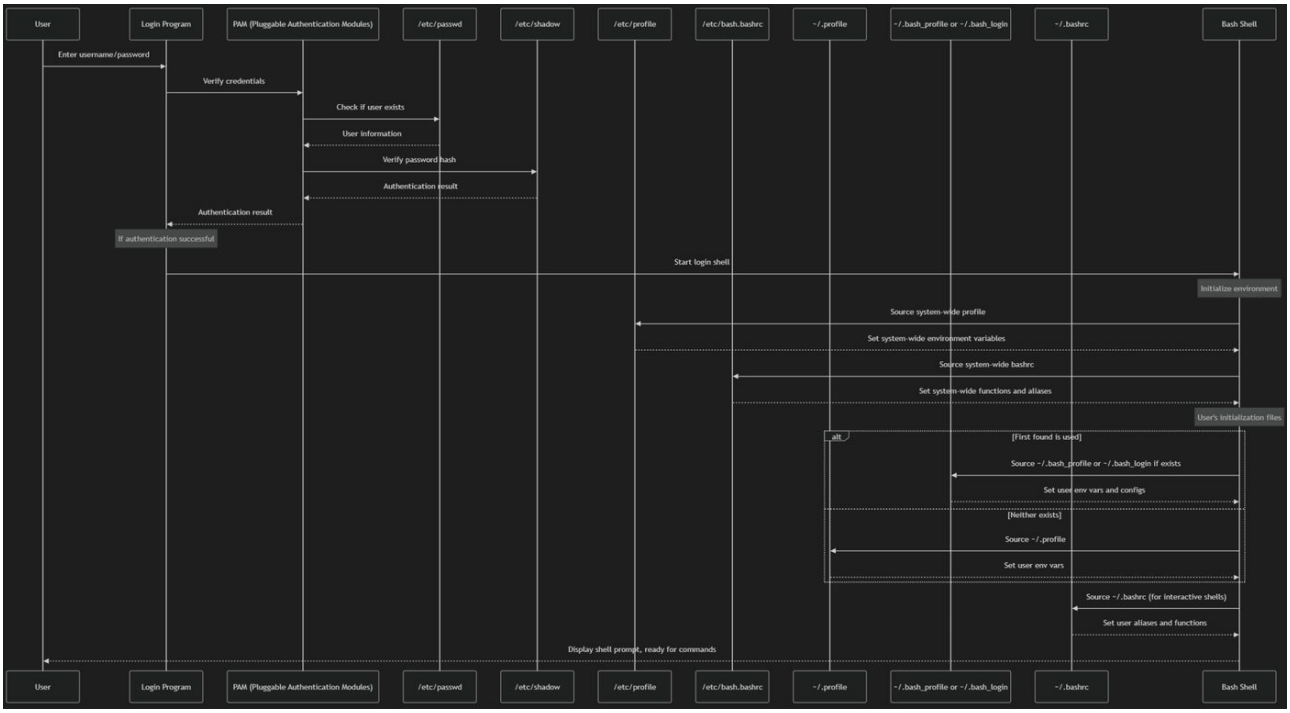
As an incident responder, it is really useful to understand what happens between a user typing in their password and getting a functional shell. This is the mysterious land where attackers can hang out because, to most users, it is something they never really pay any attention to.

Users, generally, just press enter, wait, and use their system. They *might* notice if the "wait" takes longer than expected but, really, most Operating Systems are erratic enough here that most people won't notice an extra second or even two. This means it *can* be a goldmine for attackers looking to establish persistence.

This means we, as incident responders, often have to search through these areas to find where the attacker is hiding. To try and help, this article provides an overview of the login sequence and identifies the key files to examine.

Note: As with most things Linux-related, there is a lot of variation. This article focuses on bash, other shells may behave differently, and users can make significant changes to how their system works. This is an overview, not a software architect's plan.

Bash Login Overview



Bash Login Overview

That graphic isn't the easiest thing to read on LinkedIn, so we can break it down into a more reader-friendly flow. If you want to download this workflow as a PNG, it is at: https://for577.com/bashlogin

Broadly, the sequence is in three phases and the key steps of each phase are covered below.

Authentication Phase

The first phase is the authentication process. This is where the system verifies the user's identity through a series of security checks, ensuring only authorized users gain access to the system.

The sequence here is:

- The user enters a username and password at the login prompt
- The login program passes credentials to PAM (Pluggable Authentication Modules)
- PAM checks **/etc/passwd** to verify the user exists and get user information
- PAM verifies the password against the hash stored in /etc/shadow
- The authentication result is returned to the login program

Shell Initialisation Phase

Once authenticated, the shell initialisation phase sets up the user's environment by processing configuration files in a specific hierarchical order, applying both system-wide and user-specific settings.

- After successful authentication, bash is started as a login shell
- System-wide initialization files are processed first:

/etc/profile (sets system-wide environment variables, PATH, etc.)

/etc/bash.bashrc (Debian family) (sets system-wide functions and aliases)

/etc/bashrc (RHEL family) (sets system-wide functions and aliases)

- User-specific initialization files are processed next:

First, bash looks for **~/.bash_profile** or **~/.bash_login** If neither exists, it reads **~/.profile** instead

For interactive shells, **~/.bashrc** is also read (contains user aliases and functions)

Session Ready for User

The final phase completes the login process by presenting an interactive shell prompt, giving the user full access to the system's resources and functionality.

- Bash displays the shell prompt, indicating that the session is ready
- The user can now enter commands

Filesystem artifacts

Now we have identified the login sequence, we can start to identify the files on a typical filesystem that would be involved with this activity.

The problem is that there are a **lot** of them. This is an example of the files you might find involved in a single user-login event.

- **/bin/login** or **/usr/bin/login**: The login program that handles initial username/password input
- **/etc/passwd**: Contains basic user account information (username, UID, GID, home directory, default shell)
- **/etc/shadow**: Stores encrypted user passwords and password expiry information
- **/etc/pam.d/***: Configuration files for PAM modules for different services
- **/lib/security/*** or **/lib64/security/***: PAM module libraries
- **/etc/nsswitch.conf**: Name Service Switch configuration that determines sources for user information
- **/etc/securetty**: List of terminals where root is allowed to log in
- **/etc/profile**: System-wide environment variables and startup programs for login shells
- **/etc/profile.d/*.sh**: Additional system-wide profile scripts sourced by /etc/profile
- **/etc/bash.bashrc** or **/etc/bashrc**: System-wide functions and aliases for bash
- **/etc/motd**: "Message of the Day" displayed after successful login
- **/etc/issue**: Pre-login message displayed at the login prompt
- **/etc/environment**: Global environment variables set by PAM
- **~/.profile**: User-specific environment variables and startup programs
- **~/.bash_profile**: User-specific initialization for bash login shells (preferred over .profile)
- **~/.bash_login**: Alternative to .bash_profile, used if .bash_profile doesn't exist
- **~/.bashrc**: User-specific aliases and functions for interactive bash shells
- **~/.inputrc**: Customizes keyboard mappings for bash
- **~/.pam_environment**: User-specific environment variables set by PAM
- **/var/log/wtmp**: Login/logout history log
- **/var/log/btmp**: Failed login attempts log
- **/var/log/lastlog**: Last login time for each user
- **/var/log/auth.log** or **/var/log/secure**: Authentication related logs
- **/var/log/journal/***: The more modern approach to logging kernel messages and other OS data.
- **/etc/login.defs**: Shadow password suite configuration
- **/etc/shells**: List of valid login shells
- **/etc/security/limits.conf**: Resource limits for users

Note: There is a huge caveat here in that distros can vary wildly in what they use and even within distros, versions can vary wildly.

Investigative notes

There is no escaping that in some investigations you will need to capture all of the files listed above, and you might even need to read all of them to understand what has happened on a system. However, the good news is that this isn't very common! Most of the time you can have a more targeted investigation to find the evidence you need, quickly.

My recommendation here is that you approach this in different ways, depending on your investigative goals. Some examples of this are:

The attacker appears to have persistence and is regaining access when a specific user logs in.

This is one of the most common forms we see in intrusions. The areas to check here are anything related to that user. In practice this tends to be:

- ~/.profile
- ~/.bash_profile
- ~/.bash_login
- ~/.bashrc

The attacker is regaining access when any user logs in.

This is often a subtle difference and is often overlooked during the rush of IR. However, the main difference is that the attacker has privileged persistence and is likely regaining access across multiple user sessions.

For this, the areas to check are the system-wide locations, most often:

- /etc/profile
- /etc/profile.d/*.sh
- /etc/bash.bashrc or /etc/bashrc

On some rare occasions, you might see attacker traces in some of the other system-wide files.

The attacker is able to simply log in...

This can be down to a lot of attack techniques. The most common is a weak password choice, but also we see adversaries manipulate the whole process to keep account access.

Areas to check include:

- /bin/login or /usr/bin/login
- /etc/pam.d/*
- /lib/security/* or /lib64/security/*

- /etc/securetty
- /etc/environment

Also, remember the best places to see account manipulation are the account/credential stores themselves:

- /etc/passwd
- /etc/shadow

What else?

The potential scope for attacker behaviour is pretty broad, certainly broader than can be covered in this article, so it really comes down to your own analytical expertise here. Look at what the evidence tells you the attacker is doing and investigate the ways it might be possible.

However, and this is important, be realistic. Avoid focusing on attacks that happen 1 time in 100,000,000 events and start with the most likely things. Remember, logging (and evidence in general) in Linux is often pretty sparse so it is more likely you simply dont have the record of what the attackers have done rather than the threat actor used some esoteric exploitation technique.

An example of this is something I have seen once in an incident, but only once. The attackers had rewritten **/etc/motd** to contain a heavily obfuscated malicious script and then placed **cat /etc/motd | bash** in the **/etc/profile**, making it look fairly innocuous. Although this is kind of an attack via the Message of the Day file, it was (theoretically) detectable in the system-wide profile file. So rather than focus on finding evil in the **/etc/motd**, keeping the focus narrow is still effective, even with esoteric attacks.

About Halkyn Consulting

Halkyn Consulting Ltd is a UK-based security and risk management consultancy providing 24/7 incident response, digital forensics, threat hunting, virtual security leadership (vCISO) and security compliance support to organisations worldwide. We deliver clear, practical, vendor-independent advice based on real-world threats — no hype, no upsell.

More free security resources are available at www.halkynconsulting.co.uk/security-resources.

Web: www.halkynconsulting.co.uk | **Email:** info@halkynconsulting.co.uk | **Tel:** +44 1244 940858