Exploring PAM

An established pluggable authentication framework

Matthew Heller[1]
[1]Vanderbilt University, Nashville, TN 37203

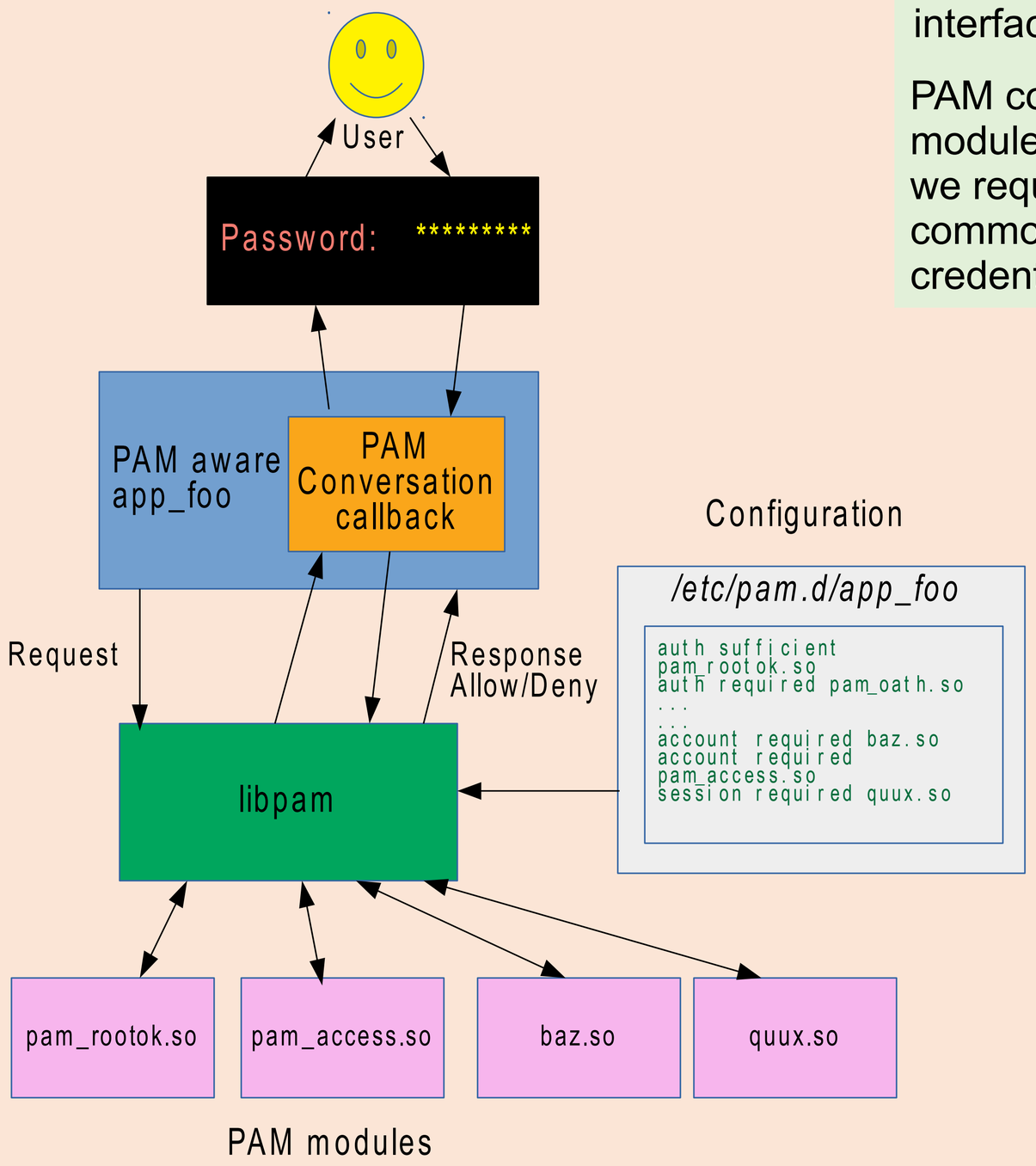
ACCRE Advanced Computing Center for Research & Education

VANDERBILT UNIVERSITY

Introduction

The Pluggable Authentication Module (PAM) security framework gives developers of applications and services the ability to add authentication without re-inventing the wheel by providing access to well tested authentication implementations. More importantly it does so in a way that allows the authentication mechanism to be changed without any change to the authentication enabled application or service. Not only can the authentication mechanism be changed but new modules implementing new security mechanisms can be added to extend the authentication implementations offered by PAM. This decouples software from authentication and gives system administrators power to configure authentication for new applications and services to meet the needs of their site.

Architecture



The four primary components of the PAM framework are the PAM aware application, the PAM library, the PAM configuration for the application, and the PAM modules. There are actually four types of modules, authentication (“auth”) modules verify identity, “account” modules check that the account associated with the identity is allowed to use the application or service under the present conditions, “password” modules are for updating credentials, and “session” modules define behavior at the start and end of a authentication session.

Demo: 2FA for Hello World!

Here is the classic ‘Hello World!’ program adapted to utilize PAM. To the right is the result when the associated PAM configuration requires two factor authentication (2FA), a time-based on-time passcode plus local user account password. However an edit to the configuration file can cause completely different type of credentials to be requested with no change to the program.

In minimal form adding PAM to an application comprises of simply a call to start a transaction with a reference to a conversation function to be used for user interaction, a call to trigger authentication, and a call to end the transaction. The conversation function here is *misc_conv* for text based interaction which is included with Linux-PAM. Custom conversation functions can be created to fit other uses and user interfaces.

```
$ ./hello_pam
login:mfh
One-time password (OATH) for `mfh':
Password:
Hello mfh!
$
$ ./hello_pam
login:badguy
One-time password (OATH) for `badguy':
Password:
I don't trust you, bye!
  PAM error 7 - Authentication failure
$
```

hello_pam.c main() function

```c
int main(int argc, char **argv)
{
    int retval;
    const char *user = NULL;
    pam_handle_t *pamh;
    const struct pam_conv pamc = {misc_conv, NULL};

    retval = pam_start("hello_pam", NULL, &pamc, &pamh);
    if(PAM_SUCCESS == retval)
        retval = pam_authenticate(pamh, 0);
    if(PAM_SUCCESS == retval)
        retval = pam_get_item(pamh, PAM_USER, (const void **)&user);
    if(PAM_SUCCESS == retval){
        printf("Hello %s!\n",user);
    }else{
        printf("I don't trust you, bye!\n  PAM error %d - %s\n",
                retval, pam_strerror(pamh, retval));
    }
    retval = pam_end(pamh, retval);
    return retval;
}
```

/etc/pam.d/hello_pam

```
auth required  pam_oath.so usersfile=/etc/demo/users.oath window=10
@include common-auth
@include common-account
@include common-session
```

See presentation for full source code, compilation details, and setup of shared secret for one-time-password

PAM configuration is rule based and it allows multiple security modules to be combined to express a wide variety of policies. Here we require that both the one-time password module and the included common user authentication of the local system both approve the credentials provided for a successful login.

Evaluation

PAM originated from Sun Microsystems in 1995 and is now available on Linux via the Linux-PAM project and on macOS, *BSD, and other Unix flavors via the OpenPAM project. The two implementations are not 100% compatible but software can support both with minimal effort.

The conversation mechanism is customizable yet it is based on queries and answers in unstructured text which makes machine to machine usage difficult and less reliable. A conversation mechanism can be made to send queries to remote clients but the security of the transport of that sensitive info is outside the scope of PAM and the burden falls on application to, for example, establish a secure TLS channel. The PAM framework also is inflexible in some ways that limit the types of Kerberos authentication transactions that are possible. For these reasons other technologies like GSSAPI, SPNEGO, SASL are often favored for uses involving Kerberos authentication, web apps, and/or machine-to-machine communication.

Additional Info

http://www.linux-pam.org/

https://www.openpam.org/

https://www.vanderbilt.edu/accre/sc18/pam