Login Management

Crews: User Name Enumeration

The Tractor crews.config file defines user name lists, called "crews", that are used for various purposes throughout the system. Crew names can be used to restrict job access on some farm machines, or to limit job edit permissions, or administrative control over Tractor itself. Login via the Tractor Dashboard or other clients are also affected by crew definitions.

Crews are simply lists of names, although they can be "recursive" in the sense that one crew definition can include another crew name as one of its members, causing all of its users to be added to the new crew.

The crew named "ValidLogins" is especially important. Only the users named in that list are granted login access. This crew must be defined in crews.config, otherwise the engine will declare a configuration error. The ValidLogins definition must enumerate all valid Tractor users in some manner. At small studios this can be a simple list of user names. At larger sites "meta-names" are typically used to defer to existing external username databases.

Special Meta-Names: "@syslogins" and "@externlogins"

The special user name "@syslogins" represents all valid login names visible on the engine host (i.e. those that tractor-engine can resolve using the getpwnam(3) system call). These login names are typically stored in databases such as LDAP, or NIS, or /etc/passwd. Typical crews.config usage would be:

```
"ValidLogins": ["@syslogins"],
```

Alternatively, the special name "@externlogins" causes the engine to skip its internal check for valid user names, and instead defers to the site's own external PAM module or trSiteLoginValidator.py script where a given login name and password will be accepted or denied based on validation checks using their external databases. The use of @externlogins therefore requires password validation to be enabled.

Optional Password Validation

The web-based Tractor Dashboard presents a default login screen that only asks for a login name, the password field is disabled. The name is used to fetch UI preferences such as filtering controls, and to log actions taken by that Dashboard session. This simple level of self-attested login identity is sufficient at many studios where there is an existing high level of trust among users, such as when many users already have administrative permissions, and inbound web access is tightly controlled.

Similarly, password protection may not be required for scripted access to Tractor, with the process owner's name being sufficient for logging and access checks.

Tractor provides administrators with two approaches to adding password validation: a "PAM" backend on the engine, and "hooks" for adding a site-defined challenge/response hashing scheme.

Treat password exposure issues carefully since the passwords used for Tractor access are often the same as those used for access to other important systems. It may be that running Tractor without any password validation at all, in combination with a site policy of good behavior (and retribution for bad behavior), is more realistic than weak password protection that gives users a false sense of security and possibly exposes important system accounts to evildoers.

The "SitePasswordValidator" setting in crews.config defines the validation scheme for passwords sent to the engine from user interfaces and scripts. **Use an empty string "" to disable password checking,** allowing any valid login name to proceed as a client. Any non-empty value will cause passwords to be required, and specifies the type of validation that will be applied.

Pluggable Authentication Module (PAM)

Tractor provides a built-in way to delegate authentication decisions to site-configurable PAM modules, such as those already used to validate user login to the operating system desktop itself. Change the SitePasswordValidator setting in crews.config to name the PAM policy module that you want to use:

```
"SitePasswordValidator": "internal:PAM:tractor",
```

PAM modules are typically defined in /etc/pam.d or /etc/pam.conf, and you must must supply a "tractor" module definition using appropriate policies for your site. Alternatively, you can specify a different PAM module name, such as an existing one that already supports your site's requirements. For example, to get the "su" authentication rules, use **"internal:PAM:su"** instead.

Note: On Linux RHEL 6.x era releases, the pam_fprintd.so module contains a bug causing it to leak file descriptors on every call from tractor-engine. Since PAM modules are loaded into the tractor-engine process, and it performs many authentications over time, the unclosed "pipe" descriptors will accumulate, unknown to the main tractor-engine code and will eventually exhaust the available file descriptor limit for that engine process. While many studios do not depend on fingerprint validation, especially for scripted API access to a system service, the "fprint" module is called indirectly from many common RHEL6 PAM policies, including "login" and "su". It has been removed from the common policies in RHEL 7 era distributions. A workaround for RHEL6 is to create your own "tractor" policy that doesn't include system-auth, or perhaps to specify a less general policy crews.config, such as password-auth.

The PAM scheme allows Tractor administrators to quickly enable password checking, but it is only appropriate at sites where the LAN or VPN already provides adequate security for credential transfer from the client to the engine. In the PAM scheme, credentials are collected from the user by the client app, but they are not validated on the client host (consider a tablet running Tractor Dashboard in a web browser). The encoded credentials are sent to the engine where the validation is performed by passing them through the PAM API. The transfer of the encoded credentials from the client to the engine is susceptible to certain kinds of attacks, hence the cautions raised in the section above.

Authentication for Tractor Utility Scripts

Tractor engine (since 1.5) supports a challenge token URL that makes it straightforward to support external tractor scripts at sites requiring authentication.

For python scripts, the tractor.base.TrHttpRPC module found in the distribution contains login support. Use of the TrHttpRPC module for engine connections is handy, but not required; details are provided below for sites using other scripting languages or who wish to write their own authenticated scripts.

The TrHttpRPC module provides two methods related to authentication. The PasswordRequired() method returns a boolean indicating whether a password will be required to login to the monitor. The Login(user, passwd) method takes two arguments, the username and password used to login.

Similar to the dashboard UI, the Login() method downloads a file called trSiteFunctions.py which contains the trSitePasswordHash() function. This file resides in the engine configuration directory. The trSitePasswordHash() function defines the function to be used to hash the user password before sending to the monitor to be processed by the login validator. A python None is returned to indicate that no password checking is required.

Python scripts intending to support password authentication should call the PasswordRequired() function first, to determine whether or not a password is required. There is no built-in function for requesting the password by a script, although the tractor utility scripts provide examples of ways to handle this. See nimby.py and jobMonitor.py for examples.

Implementing Authentication in External Scripts

In addition to hashing the users password, external scripts which login to the tractor monitor need to properly format the challenge/authentication string for the monitor login URL. Scripts needing to support authentication should follow the details below in preparing the challenge/authentication URL.

- The script should first connect to the tractor monitor and request a challenge token. This is done by sending the /Tractor/monitor?q=gentoken URL to the monitor.
- The URL will return a JSON dictionary of the format:

  { "challenge": "2376751924132450113012563356O" }
- The script must either "know" the password hashing function, or it should request it from the monitor. One way to do this is to save the hash function defined in a file located in the engine configuration directory, and request this file using the /Tractor/config?file=configurationfile URL. For example, the provided example utility scripts request a python file using the URL /Tractor/config?file=trSiteFunctions.py. This URL will deliver the contents of the named file as text to the script. Depending upon the scripting language used, this data will likely need to be evaled or sourced into the scripting interpretter.
- When the hashing function is defined, it should be called with a test password to determine if a password is actually required. If no password is required the hashing function should return a NULL or None type of reponse.
- If authentication is required, then the next step is to format the challenge token and the password hash for the login URL.
- First the challenge token needs to be joined with a vertical bar | and the result of running the users password through the hashing function. This would look roughly like challengestring|hashedpassword.
- This string then needs to be converted into a hex representation, where every byte in this string needs to be converted to a 02xformat representation. For example, the string above *challengestring|hashedpassword* would encode to6368616c6c656e6765737472696e677c686173686564706173737776f7264.
- Finally this hex coded passwordhash string needs to be URL encoded and sent to the monitor using the login URL. That URL format would be:

  /Tractor/monitor?q=login&user=user&c=urlencodedchallengestring
- If the login validates correctly, the monitor will return a JSON dictionary which looks like:

```
{
  "rc": 0, "login": "ok",
  "host": "192.168.1.1",
  "user": "rdavis",
  "tsid": "4f04f1c9-4ef2488a-0021",
  "crews": ["ValidLogins", "Wranglers"]
}
```

- The "tsid" entry in the dictionary should be retained, and added to all URL's sent to the monitor. From the example above every URL should now end in *....&tsid=4f04f1c9-4ef2488a-0021*.

Alternative Authentication Methods

For alternative authentication methods that do not use PAM, see Alternative Authentication.