FreeIPA Community Portal Documentation

Release 0.2

Drew Erny

Jul 13, 2018

Contents

CHAPTER 1

Disclaimer

This project is currently abandoned. There are now other ideas on how this functionality can be accomplished.

If you are interested in this functionality and want to move forward please indicate your interest on the freeipa-users mailing list and we can discuss next steps.

CHAPTER 2

Contents

2.1 FreeIPA Community Portal

The FreeIPA Community Portal is a web frontend for FreeIPA that allows anonymous users to interact with FreeIPA without authenticating. It is designed for users in communities with a public FreeIPA setup, where administrators may be volunteers or geographically dispersed, and where helpdesk-like setups aren't feasible

Currently, the FreeIPA Community Portal has two features: self-service registration, and self-service password reset. Both of these features previously required emailing an administrator.

2.1.1 Self-Service User Registration

The self-service registration workflow is very simple. The user is presented a form into which they can enter basic biographical information, i.e. name, email address, and username. The user also fills out the answer to a captcha. The form is sent to the server and checked for validity.

If the form is not valid, the user is sent back to the form with all the fields filled as the user submitted them (except the captcha) and displayed an error message explaining what has gone wrong. If the form is valid, the user is sent to a completion page informing them that their sign-up will be reviewed.

All other portions of the workflow (user's first sign in, sign up approval, etc.) take place inside the existing IPA WebUI and are outside of the scope of this application.

2.1.2 Self-Service Password Reset

The self-service password reset workflow is marginally more complicated. It consists of two main portions.

In the first part, the user navigates to a page to request a reset. The user is presented with a form prompting a username and a captcha. The user submits the form and it is checked for the presence of text in the username and the correctness of the captcha. If the username field is blank or the captcha is incorrect, the user is sent back to the page with a message explaining the error. Regardless of whether a username exists, the user is sent to a completion page informing them to check their email. At this point, an email with the username and a token is sent to the user.

The user is then directed to page with a form to input the username requesting the password reset and the token that arrived in the email. The user inputs the username and token (or clicks a link provided in the email, which pre-fills the form for the user) and submits the form. If the username and token match, the user's password is changed in the IPA system to a random string of characters. The user is then sent to a new page which displays this temporary password and explains that their password has been changed, that this is the only chance to view the password, and if this temporary password is lost before first login, another password reset will have to be initiated. The user immediately navigates to the WebUI and logs in with this temporary password.

If the username has not requested a password reset, if the token for the username is over three days old, or if the token does not coorespond to the username, the user is sent to an error page. The user is informed that one of these errors has occurred, but not which error in particular, and that if they entered the correct username but an incorrect token, the old token has been expired and they must restart the process. The user is not informed what specific error has occurred, to avoid using the password reset feature to probe for the existance of usernames.

2.1.3 More Information

A more technical guide to the workings of the Community Portal can be found at these pages:

http://www.freeipa.org/page/V4/Community_Portal http://www.freeipa.org/page/V4/Self_Service_Password_Reset

2.2 Deploying the Community Portal

The FreeIPA Community Portal is a stand-alone WSGI web application, built with CherryPy. It is intended to be deployed on its own server, using the provided installation script. However, it can probably be deployed alongside other Apache applications, and possibly even another FreeIPA server, if desired. This behavior is untested and unproven, so your mileage may vary.

2.2.1 Requirement

The community portal has several dependencies which must be installed. Below is a list of commands to install these dependencies, and a rationale for each.

First, we install the web server. Obviously:

```
dnf install httpd mod_wsgi
```

The web server also needs to be an IPA client. In addition, having the admin tools makes the installation easier. If you're trying to minimize the the number of installeded programs on your server, you can run the ipa commands from a different computer and skip installing freeipa-admintools. Also, due to a deficiency in FreeIPA, the client currently depends on python-memcached:

```
dnf install freeipa-client freeipa-admintools python-memcached
```

This guide installs a couple of python packages from git, so we need this tool, if you don't already have it:

```
dnf install git
```

The CAPTCHA functionality relies on the Pillow library:

```
dnf install python-pillow
```

These components are the core application. CherryPy as the web framework, Jinja2 provides templating, and SQLAlchemy is used for the database:

```
dnf install python-cherrypy python-jinja2 python-sqlalchemy
```

Here, we switch to using pip. We install captcha from PyPI. We need at least version 0.2 of the captcha package:

```
pip install captcha >= 0.2
```

Finally, the portal itself:

```
pip install git+https://github.com/freeipa/freeipa-community-portal.git
```

This will automatically unpack a couple of things to the places that we need them. Of note is that it unpacks freeipa_community_portal.wsgi, which unpacks to <python_path>/libexec/, and which is an executable, WSGI-compatible script.

Before continuing into the installation, the server should be enrolled as a FreeIPA client of the FreeIPA domain it belongs to. Running:

```
freeipa-client-install
```

with your favorite options will do.

2.2.2 Installation

The recommended installation method is to use the freeipa-portal-install command, which will perform most installation actions automatically. If you're using this script, you can skip this section and jump to the next thing, which outlines some post-install necessities

First, if it is not already present, the installer copies share/freeipa_community_portal/conf/freeipa_community_portal.ini to /etc/freeipa_community_portal.ini. The latter location is where the portal searches for the config, which is mostly used for email settings. If it is not present or formatted correctly, the portal will crash on start. Even if you're not using the install, I recommend copying this file over, instead of typing it from scratch, to avoid errors.

You must edit this configuration file in order for the application to work properly. If the email settings are misconfigured, the application will crash.

Next, the installer copies the apache config from the conf directory to /etc/httpd/conf.d/freeipa_community_portal.conf. If you're doing a custom installation of the portal, you probably will not need this file, because you probably know what you're doing.

Then, the installer creates the directory where the portal keeps its database:

```
mkdir -p -m 750 /var/lib/freeipa_community_portal
chown apache:apache /var/lib/freeipa_community_portal/
```

If Apache doesn't own this folder, it will vomit when attempting to put database in it. Next, the installer generates a random key and stores it in a file called "captcha.key" the above directory. The portal uses this key to secure the captcha. It would be mostly harmless if this key gets compromised, so there's no need to take any special precautions to secure it.

After this, the installer does:

```
setsebool -P httpd_can_sendmail on
```

which loosens SELinux security so that the portal can send mail. Without this, the portal will crash when it attempts to send mail.

Finally, the portal creates a directory, /var/www/wsgi, and symlinks the wsgi executable into this directory, so:

```
mkdir -P /var/www/wsgi
ln -s /usr/libexec/freeipa_community_portal.wsgi \
      /var/www/wsgi/freeipa_community_portal.wsgi
```

This is the expected location of the WSGI file according to the provided httpd conf file. Is this best practice? I have no idea. Probably not. I'm not very good at Apache. If you choose to install it somewhere different, just make sure that change is reflected in your Apache configuration file.

2.2.3 Post-installation

After installation, the application still needs a few things set up in order to run. The first is a user account on the FreeIPA to run commands as. The portal relies on a few permissions:

System: Add Stage User to create a new stage user

System: Read Stage User to query the newly created stage user

System: Change User Password to set a temporary password for the password reset feature

System: Read User Standard Attributes to query user by uid for password reset (usually available to anyone)

System: Read User Addressbook Attributes to read the mail attribute to send the password reset mail (usually available to all authenticated users)

You can create an account manually with these permissions, or you can use the included "create-portal-user" script, which contains all of the commands to add a user called "portal" with the requisite permissions.

The second thing needed is a way to authenticate via Kerberos as the user created in the previous step. Specifically, we need to authenticate as a user principal, and not a service principal. There's no canonical solution for this yet. A keytab for the portal user is an easy way to automatically authenticate the portal user. A client keytab for the portal can be acquired with `ipa-getkeytab`. You must properly secure the keytab, so it can only be read by the webserver:

```
ipa-getkeytab -s IPA_SERVER_HOSTNAME -p portal@YOUR.REALM -k /etc/ipa/portal.keytab
chown apache:apache /etc/ipa/portal.keytab
chmod 640 /etc/ipa/portal.keytab
```

If you don't remember the values for IPA server and realm, have a look at `/etc/ipa/default.conf`. You can set the path to keytab in `/etc/freeipa_community_portal.ini`. The app sets the environment variable `KRB5_CLIENT_KTNAME`, when the value is not empty. ipalib picks the keytab up automatically.

After all this, you should probably set up and configure mod_ssl and put the app behind HTTPS, but that is outside of the scope of this guide.

2.3 The CAPTCHA

2.3.1 Overview

Several forms on the Community Portal site have the potential for abuse. A warped-characters captcha forms the first line of defense against automated spam. While probably not sufficient to ward off a determined and sophisticated adversary, the captcha will deter untargeted, drive-by spam and casual targetted attacks.

2.3.2 Use Cases

A captcha is present on each form with the potential for abuse, namely the self-service user registration form and the password reset initiation form. The password reset completion form is probably not susceptible to abuse and does not need a captcha.

2.3.3 Design

1.) The user requests a form protected with a captcha.

2.) A random string of 4 characters is generated. The string is fed to the python captcha library, which returns a BytesIO object representing the captcha image, which is then turned straightaway into a bytes object. The string is also mixed with a secret key to form an HMAC string. The HMAC and a timestamp are stored in a database.

3.) The image, as a data-uri, and the HMAC string, as a hexidecimal text string, are sent to the client as part of the web form being protected. The HMAC is placed into a hidden form field to be submitted with the form. The user solves the captcha and enters the solution into a field in the protected form. The form is submitted by the user.

4.) Before any processing is done on form data, the captcha is verified by mixing the user's solution with the secret key from step 2 and compared securely to the HMAC. If the two match, a correct solution has been found. The HMAC is then looked up in the database and, if found, is deleted, regardless of a correct solution or not, to prevent multiple attempts at the same captcha.

5.) If the solution does not match the HMAC or the HMAC is not found in the database, the user is returned to the form and informed that their captcha solution is incorrect. If the captcha is correct, processing of form data begins and the rest of the form's workflow is carried out.

2.3.4 Implementation

The captcha system depends on a python library called captcha, which does not current appear to be available in the Fedora package repos, but is available through pip.

Old captcha data will be periodically expired and deleted from the database with a script run regularly by cron.

2.3.5 Feature Management

The secret key for the HMAC is currently hard-coded into the program, but will be read from a configuration file in the future.

2.3.6 How to Test

Automated unit tests will be written.

2.4 Setting up a development environment

The application should run in a development environment without trouble. Install the dependencies described in deploy.md except the application itself, and then do:

```
pip install -e .
```

in the root of the tree. This should install a local, editable copy of the app, and put all of the configuration files and assets where they are expected.

You can configure exactly where the application spews its files by editing the freeipa_community_portal_dev.ini file in freeipa_community_portal/conf and plugging in values that make you happy. By default the development server uses var/ in your current working directory to store its database and captcha key file. The directory, sqlite database and key files are created automatically.

Before you run the app, even in tree, you should kinit as a user with sufficient permissions as outlined in the deployment doc. You can also drop a client keytab in your var/ directory.

To run the application in-tree, do:

```
python -m freeipa_community_portal
```

On an IPA server Dogtag PKI is already occupying port 8080. For that reason the development server listens on port 10080 on localhost. You can change the port in freeipa_community_port/__main__.py if the port is already used on your machine.

CHAPTER 3

Indices and tables

- genindex
- modindex
- search