Patchman platform integration

Introductory notes about platform integration

Patchman needs to interface with your webservers for a couple of reasons:

- Determining which directories need scanning
- Determining which users own which directories, and which of those directories belong to which websites
- Finding out the user hierarchy
- Retrieving contact details for the users

For several popular platform software packages (i.e. cPanel, DirectAdmin and Plesk) the integration ships built-in with the Patchman software and requires no configuration. If you are using one of these software packages, Patchman will automatically detect which of these packages is active and will automatically activate the required integration method.

If you do not use one of these software packages, you can provide Patchman with the required data yourself. This document discusses how to configure the software to use such a custom platform integration method.

Compatibility

This document specifies the custom integration API compatible with Patchman versions 1.7.1 and up.

Document revisions

- 19 October 2021: Add clarification for UTF-8 requirement for string data
- 26 February 2021: Correct erroneous information about expected script exit status
- 17 May 2019: Added explanation for single sign-on functionality through IPC
- 23 April 2019: Clarified user hierarchies
- 23 November 2016: Scrapped `children` field from file/script type 1
- 26 June 2015: Initial version

Supplying user and directory metadata

This chapter discusses the different options for providing the Patchman software with the required metadata for users and directories on your system. Patchman supports two approaches for this:

1. Provide scripts on the webservers that can be called by the agent, returning data in the output
2. Provide pre-generated JSON files on the webservers containing data that can be read by the agent

In both cases, the data needs to be presented in a JSON format specified below.

For all descriptions of files and scripts, examples are provided for reference.

General notes about JSON

Both approaches make use of JSON serialization of data. When building JSON output, take into account that JSON is strongly typed. It distinguishes between the following types:

- Strings (only UTF-8 characters are supported)

```
"data"
""
```

- Integers

```
42
0
-12
```

- Booleans

```
true
false
```

- Objects (similar to dictionaries, maps or associative arrays as used in other serialization methods and programming languages)

```
{}
{"key": "value", "key with differently typed value": 42}
```

- Lists (similar to non-associative arrays or vectors as used in other serialization methods and programming languages)

```
[]
["single entry"]
[1,2,3]
```

- Null values (not to be confused with empty values such as empty strings, empty lists or the integer 0)

```
null
```

None of the object entries in this specification are optional: each key in the main object must always be present. However, occasionally an empty value (e.g. `""`, `[]` or `{}`) or null value (`null`) is allowed. Pay close attention to the documentation to find out which field allows which values.

User hierarchies

Patchman is built to be compatible with hierarchies of users in your control panel. This structure is relevant for your policy configuration regarding notifications, and also decides what data is available to your users through the SSO interface. Consider the following example:

```
admin_user_1 (level 1)
 └ reseller_user_1 (level 2)
 └ reseller_user_2 (level 2)
    └ end_user_1 (level 3)
 └ end_user_2 (level 3)
admin_user_2 (level 1)
 └ reseller_user_3 (level 2)
    └ end_user_3 (level 3)
```

When a notification needs to be sent, depending on the policy configuration, this notification will be sent to different users. The table below describes which user receives the notification, given a notification concerning the user in the row title and the "Notified user level" policy setting in the column header:

	...to the administrator	**...to the reseller**	**...to the affected user**	**...to the descendant of administrator**
admin_user_1	admin_user_1	admin_user_1	admin_user_1	admin_user_1
reseller_user_1	admin_user_1	admin_user_1	reseller_user_1	reseller_user_1
reseller_user_2	admin_user_1	admin_user_1	reseller_user_2	reseller_user_2
end_user_1	admin_user_1	reseller_user_2	end_user_1	reseller_user_2
end_user_2	admin_user_1	admin_user_1	end_user_2	end_user_2
admin_user_2	admin_user_2	admin_user_2	admin_user_2	admin_user_2
reseller_user_3	admin_user_2	admin_user_2	reseller_user_3	reseller_user_3
end_user_3	admin_user_2	reseller_user_3	end_user_3	reseller_user_3

Regarding data access, consider that each reseller user will also have access to data for all users below it in the hierarchy. For example, `reseller_user_2` also has access to data belonging to `end_user_1` and `admin_user_2` can access both `reseller_user_3` and `end_user_3`. Note that despite being a reseller, `reseller_user_1` won't have access to any data owned by another user simply because it isn't the parent of any user.

Configuration of user hierarchies depends on two values you can set. It is very important that you set both of these fields correctly for both notifications and SSO access to work properly. For the exact list of supported values, refer to the documentation for your chosen type of integration, later in this document.

- The `parent` field, describing which is the parent user (e.g. `admin_user_1` for `reseller_user_1`, and `null` for `admin_user_1`)
- The `level` field, describing the type of user (e.g. `1` for `admin_user_1`)

Approach 1: Scripts

The scripts will be executed by their owning users through a shell (so shebangs are supported). All scripts should return valid JSON data and an exit status of 0 on success, and must return a nonzero exit status on error. In case the exit status is zero, standard error output is ignored; otherwise, both standard output and standard error output are captured for logging purposes.

User metadata

Script 1 is used for getting user metadata, and is called whenever a user is added or modified. In case audit logging is enabled within Patchman and the logging is sent to to a home directory path, it may be called upon file action execution as well. The result is an object and should only contain elements for matching arguments. All keys for each object entry are required at all times, but some values may be null. If a domain does not have any directories, it should be omitted.

```
# ./get_user_data.sh username1 userthatdoesnotexist
{
  "username1": {
    "homedir": "/home/username1/",
    "email": "address@domain.com",
    "parent": "parent_username",
    "language": "en",
    "suspended": false,
    "level": 3,
    "domains": {
      "domain.com": [
        "/home/username1/domains/domain.com/public_html/",
        "/home/username1/domains/domain.com/private_html/"
      ]
    }
  }
}
```

Notes:

- The homedir field is used when enabling per-customer audit logging in the Portal, which places the logfile in a location relative to the appropriate homedir.
- The email field may contain multiple e-mail addresses, comma-separated. If no e-mail address should be used, supply an empty string.
- The parent field supports null values.
- The language field supports ISO 639-1 language codes. If you have no applicable language, please supply a sensible default.
- The level is used for specifying the user type or 'rank' in an internal hierarchy. A lower level means a higher 'rank'. A common practice is using 1 for admin or root, 2 for resellers and 3 for normal users.
- If a domain does not have any directories, it should be omitted from the list of domains.
- If a user does not have any domains, the domains entry should be an empty object.
- If the domain is internationalized (IDN), it is recommended to use punycode notation.

Path information

Script 2 is used for getting path information. The result is an object and should only contain elements for matching arguments. All keys for each object entry are required at all times. If the script is called without arguments, it should return elements for all existing paths that Patchman should index.

Patchman will only scan a path when it is an actual directory; any symbolic links will be ignored. The number of symbolic links that has been ignored will be displayed in the log files.

```
# ./get_path_info.sh /home/username/domains/domain.com/public_html/subpath \
                     /path/that/does/not/exist
{
  "/home/username/domains/domain.com/public_html/subpath": {
    "username": "username1",
    "domain": "domain.com"
  }
}
```

Notes:

- If the domain is internationalized (IDN), it is recommended to use punycode notation.

Modified users

Script 3 is used for getting the list of users with modified metadata since a given UNIX epoch timestamp (GMT). It is called every five minutes by default, but does have a configurable interval. Its result is a list that contains usernames of the users whose metadata (e-mail address, language, suspended, level or domains) has changed since the given time. In case no users have changed, the result should be an empty list (`[]`).

```
# ./get_changed_users.sh 1234567890
[
  "username1"
]
```

Approach 2: Files

Instead of scripts, you can also opt for generating files with metadata manually from your own control panel in predefined locations, which Patchman will read whenever it requires the data. This is a good option if you don't have an easily accessible API, or if API calls prove relatively expensive. You can generate these files periodically through e.g. cron jobs. Note that file changes should always occur atomically to prevent Patchman from reading intermediate states, so you should never change the file in-place (but e.g. use a temporary file for writing, which is moved over the original file on completion).

User metadata

File type 1 is used for getting user metadata. There should be one file per existing user, with one JSON object per file.

```
# cat username1.json
{
  "username1": {
    "homedir": "/home/username1/",
    "email": "address@domain.com",
    "parent": "parent_username",
    "language": "en",
    "suspended": false,
    "level": 3,
    "domains": {
      "domain.com": [
        "/home/username1/domains/domain.com/public_html/",
        "/home/username1/domains/domain.com/private_html/"
      ]
    }
  }
}
```

Notes:

- The homedir field is used when enabling per-customer audit logging in the Portal, which places the logfile in a location relative to the appropriate homedir.
- The email field may contain multiple e-mail addresses, comma-separated. If no e-mail address should be used, supply an empty string.
- The parent field supports null values.
- The language field supports ISO 639-1 language codes. If you have no applicable language, please supply a sensible default.
- The level is used for specifying the user type or 'rank' in an internal hierarchy. A lower level means a higher 'rank'. A common practice is using 1 for admin or root, 2 for resellers and 3 for normal users.
- If a domain does not have any directories, it should be omitted from the list of domains.
- If a user does not have any domains, the domains entry should be an empty object.
- If the domain is internationalized (IDN), it is recommended to use punycode notation.

Path information

File type 2 is used for getting path information. There should be one file with one JSON object, containing a mapping of paths to username and domain. All keys for each object entry are required at all times.

Patchman will only scan a path when it is an actual directory; any symbolic links will be ignored. The number of symbolic links that has been ignored will be displayed in the log files.

```
# cat domains.json
{
  "/home/username/domains/domain.com/public_html/": {
    "username": "username1",
    "domain": "domain.com"
  },
  "/home/username/domains/domain.com/private_html/": {
    "username": "username1",
    "domain": "domain.com"
  }
}
```

Notes:

- If the domain is internationalized (IDN), it is recommended to use punycode notation.

Modified users

File type 3 is used for getting the list of users with modified metadata since a given UNIX epoch timestamp (GMT). It should contain one JSON object with a mapping of username to UNIX timestamp that marks the last moment that user's metadata (e-mail address, language, suspended, level or domains) was changed. In case no users are present yet, the result should be an empty object (`{}`).

```
# cat userchanges.json
{
  "username1": 1234567890
}
```

Enabling custom integration

Within the Patchman Portal, custom integration is enabled per server group. Make sure the files or scripts are locally available on each server within the server group you enable this for.

After the use of custom integration has been enabled for your account, you can follow these steps for configuration:

1. Go to Servers -> Server Groups
2. Open the page for the appropriate group
3. Under Platform, select the applicable option (script-based or file-based integration)
4. Fill out the three input fields asking for the fully qualified paths to the scripts or files
5. Click Update to save the changes

After the configuration has been updated, it may take several minutes before the first synchronization of the directory tracking index on each server is started. Keep an eye on the logfiles in /var/log/patchman/ for information on whether the task has started yet, and to see if the integration could be used without error.

Creating Patchman Portal single sign-on buttons for end users

You can create buttons for your customers to log on to a subsection of the Portal, giving them access to detections for their account only. Think of this as the user-level detections view in the Portal, without the left sidebar. You have some control over what end users can do in this view through the policy configuration. The policy also defines which users have access to this end-user view (through the settings "Enable login for end users").

Which users to show the button for

On the webhosting servers where Patchman is installed, a list of users that currently have access to the Portal can be found in the file /var/lib/patchman/plugin. Each line in this file is an entry specifying a username and user level for which the Portal is available to them. Consider the following example file:

```
john,user
peter,user
peter,reseller
```

In this case, both the users john and peter have access to the Portal dashboard on a user level. However, since peter is also a reseller, he has access to the reseller level as well. This level in the Portal gives an overview of the users controlled by his reseller user in your panel, and thus allows that reseller to access detection data for those underlying users. Depending on the layout of your panel, you can opt to show each button in the appropriate location (e.g. if a reseller has a separate user-level view as well as the reseller-level view), or to only show the topmost level (i.e. the user button for john and the reseller button for peter).

Currently only the user and reseller levels are supported.

Generating the single sign-on link

You will need to generate and retrieve a single sign-on link through the Agent IPC interface. In order to do so, send a JSON-formatted request to the UNIX datagram socket located at `/var/run/patchman-sso.sock`. The request should contain:

- A "username" field (required) with username as reported by your integration.
- A "next" field (optional) with the location the end user will be redirected to after login. For example, you can set this to `/detections/reseller/` for reseller users to forward them to the appropriate part of the Portal interface right away.

The following is an example of a valid request:

```
{
  "username": "peter",
  "next": "/detections/reseller/"
}
```

The agent will reply with a JSON response containing a single sign-on URL in the 'redirect_to' field:

```
{
  "username": "peter",
  "redirect_to": "https://portal.patchman.co/t/login/token12345/"
}
```

Redirect the user to the URL to sign them in to the Portal. If an error occurred, the response will not include a `redirect_to` field but may instead contain a `detail`, `username` or `next` field describing the error.

Reference implementations are included below in PHP and Python.

PHP

```php
#!/usr/bin/env php
<?php
$fields = array('username' => $username);

// In case user is a reseller:
// $fields = array('username' => $username, 'next' => '/detections/reseller/');

$request_json = json_encode($fields);
$response_json = '';

// Default location of the SSO socket. Can be changed.
$their_path = '/var/run/patchman-sso.sock';

// Creates a random name for our socket.
$our_path = tempnam('/tmp', 'patchman');
unlink($our_path);

$socket = socket_create(AF_UNIX, SOCK_DGRAM, 0);

socket_bind($socket, $our_path);
socket_sendto($socket, $request_json, strlen($request_json), 0, $their_path);
socket_recvfrom($socket, $response_json, 8096, 0, $their_path);

$data = json_decode($response_json);

if (isset($data->redirect_to)) {
    // Success - redirect user to the retrieved single sign-on link.
    header('Location: ' . $data->redirect_to);
} else if ($data !== NULL) {
    // An error occurred. Perhaps a username was supplied that the Portal has no record of, or the user does
    // not have access to the Portal, in which case it is likely that this user also doesn't appear in the
    // /var/lib/patchman/plugin file, and the button should not be available to them. You may want to log or
    // inspect $data for more details.
} else {
    // Unknown error, inspect $response_json for more information.
}
```

Python

```python
#!/usr/bin/env python
import json
import os
import socket
import tempfile

with tempfile.NamedTemporaryFile() as our_path:
    os.remove(our_path.name)

    sock = socket.socket(socket.AF_UNIX, socket.SOCK_DGRAM, 0)
    sock.bind(our_path.name)

    request = {'username': ..., 'next': ...}
    request_json = json.dumps(request).encode('utf-8')
    sock.sendto(request_json, '/var/run/patchman-sso.sock')

    reply_json = sock.recv(8096)

    try:
        reply = json.loads(reply_json.decode('utf-8'))

        if 'redirect_to' in reply:
            # Success - redirect user to the retrieved single sign-on link.
            print('HTTP/1.1 302 Found\r')
            print('Location: {}\r\n\r'.format(reply['redirect_to']))
        else:
            # An error occurred. Perhaps a username was supplied that the Portal has no record of, or
            # the user does not have access to the Portal, in which case it is likely that this user
            # also doesn't appear in the /var/lib/patchman/plugin file, and the button should not be
            # available to them. You may want to log or inspect the reply for more details.
            ...
    except json.JSONDecodeError:
        # Unknown error, inspect reply_json for more information.
        ...

    sock.close()
```

Legacy

An alternative approach, which communicates with the Portal API directly using Patchman license certificate files, is included below as a reference for existing implementations. This approach is legacy and deprecated, and using it for new implementations is highly discouraged. The API documentation for this particular call can be found at https://portal.patchman.co/api/#token.

PHP

```php
#!/usr/bin/env php
<?php
$c = curl_init();

// Supply the username
$username = 'root';

// Initialize the curl call
curl_setopt($c, CURLOPT_URL, "https://client-portal.patchman.co/api/v1/token/");
curl_setopt($c, CURLOPT_SSLCERT, "/etc/patchman/license/patchman.crt");
curl_setopt($c, CURLOPT_SSLKEY, "/etc/patchman/license/patchman.key");
curl_setopt($c, CURLOPT_RETURNTRANSFER, 1);

// This code tells the single sign-on link to redirect to the reseller view. If you want
// to stick to the user-level view, simply remove the "next" field from the call below.
curl_setopt($c, CURLOPT_POSTFIELDS, array("username" => $username,
                                          "next" => "/detections/reseller/"));

// This code ensures the curl call uses the external interface that Patchman's license
// was generated for. If you don't do this, on multi-interface systems, this may result
// in an IP mismatch and connection refusal on the Portal side.
if (function_exists('openssl_x509_parse')) {
    // False warnings from https://bugs.php.net/bug.php?id=66636 will interfere with
    // headers, so silence the output
    $cert = file_get_contents('/etc/patchman/license/patchman.crt');
    $certificate_info = @openssl_x509_parse($cert);

    if ($certificate_info) {
        curl_setopt($c, CURLOPT_INTERFACE, $certificate_info['subject']['CN']);
    }
}

$version = curl_version();

// Workaround for an NSS bug: https://bugzilla.redhat.com/show_bug.cgi?id=733794
if (strpos($version['ssl_version'],"NSS/") === 0) {
    curl_setopt($c, CURLOPT_CAINFO, "/etc/patchman/license/patchman.crt");

    if (file_exists("/etc/pki/tls/")) {
        curl_setopt($c, CURLOPT_CAPATH, "/etc/pki/tls/");
    } else {
        curl_setopt($c, CURLOPT_CAPATH, "/etc/ssl/certs/");
    }
}

// Perform the call and decode the response as JSON
$data = json_decode(curl_exec($c));

curl_close($c);

if ($data->redirect_to) {
    // Success - redirect user to the retrieved single sign-on link.
    header("Location: ".$data->redirect_to);
} else if ($data->username == "This user does not exist.") {
    // You supplied a username that the Portal has no record of.
} else if ($data->username == "This user can't access the portal.") {
    // This user does not have access to the Portal. Most likely this user also doesn't
    // appear in the /var/lib/patchman/plugin file, and the button should not be available
    // to them.
} else {
    // Unknown error, inspect $data for more information.
}
```

Python

```python
#!/usr/bin/env python
import socket
from cryptography import x509
from cryptography.hazmat.backends import default_backend
from cryptography.x509.oid import ObjectIdentifier

# Supply the username
username = "root"

# This code ensures the curl call uses the external interface that Patchman's license
# was generated for. If you don't do this, on multi-interface systems, this may result
# in an IP mismatch and connection refusal on the Portal side.

# Retrieve the license certificate IP
with open('/etc/patchman/license/patchman.crt', 'r') as fd:
    cert = x509.load_pem_x509_certificate(fd.read(), default_backend())

ip_address = cert.subject.get_attributes_for_oid(ObjectIdentifier('2.5.4.3'))[0].value

# Monkey-patch socket.socket to support binding source IP
unbound_socket = socket.socket

class bound_socket(unbound_socket):
    def connect(self, *args, **kwargs):
        if self.family == socket.AF_INET:
            self.bind((ip_address, 0))
        unbound_socket.connect(self, *args, **kwargs)

socket.socket = bound_socket

# Import requests after monkey-patching to ensure this patched socket is used
import requests

# This code tells the single sign-on link to redirect to the reseller view. If you want
# to stick to the user-level view, simply remove the "next" field from the call below.
r = requests.post("https://client-portal.patchman.co/api/v1/token/",
                  data={"username": username,
                        "next": "/detections/reseller/"},
                  cert=("/etc/patchman/license/patchman.crt",
                        "/etc/patchman/license/patchman.key"))

# Parse the response body as JSON
result = r.json()

if 'redirect_to' in result:
    # Success - redirect user to the retrieved single sign-on link.
    print("HTTP/1.1 302 Found\n")
    print("Location: {url}\n\n".format(url=result['redirect_to']))
elif 'username' in result:
    if result['username'] == "This user does not exist.":
        # You supplied a username that the Portal has no record of.
        pass
    elif result['username'] == "This user can't access the portal.":
        # This user does not have access to the Portal. Most likely this user also doesn't
        # appear in the /var/lib/patchman/plugin file, and the button should not be available
        # to them.
        pass
else:
    # Unknown error, inspect result for more information.
    pass
```