+++ title = "Auth Proxy" description = "Grafana Auth Proxy Guide" keywords = ["grafana", "configuration", "documentation", "proxy"] aliases = ["/docs/grafana/latest/tutorials/authproxy/"] weight = 200 +++

Auth Proxy Authentication

You can configure Grafana to let a HTTP reverse proxy handle authentication. Popular web servers have a very extensive list of pluggable authentication modules, and any of them can be used with the AuthProxy feature. Below we detail the configuration options for auth proxy.

```
[auth.proxy]
# Defaults to false, but set to true to enable this feature
enabled = true
# HTTP Header name that will contain the username or email
header_name = X-WEBAUTH-USER
# HTTP Header property, defaults to `username` but can also be `email`
header_property = username
# Set to `true` to enable auto sign up of users who do not exist in Grafana DB. Defaults to
auto_sign_up = true
# Define cache time to live in minutes
# If combined with Grafana LDAP integration it is also the sync interval
sync_ttl = 60
# Limit where auth proxy requests come from by configuring a list of IP addresses.
# This can be used to prevent users spoofing the X-WEBAUTH-USER header.
# Example `whitelist = 192.168.1.1, 192.168.1.0/24, 2001::23, 2001::0/120`
whitelist =
# Optionally define more headers to sync other user attributes
# Example `headers = Name:X-WEBAUTH-NAME Role:X-WEBAUTH-ROLE Email:X-WEBAUTH-EMAIL Groups:X-
headers =
# Non-ASCII strings in header values are encoded using quoted-printable encoding
;headers_encoded = false
# Check out docs on this for more details on the below setting
enable_login_token = false
```

Interacting with Grafana's AuthProxy via curl

```
curl -H "X-WEBAUTH-USER: admin"  http://localhost:3000/api/users
[
    {
        "id":1,
        "name":"",
        "login":"admin",
        "email":"admin@localhost",
        "isAdmin":true
    }
]
```

We can then send a second request to the `/api/user` method which will return the details of the logged in user. We will use this request to show how Grafana automatically adds the new user we specify to the system. Here we create a new user called “anthony”.

```
curl -H "X-WEBAUTH-USER: anthony" http://localhost:3000/api/user
{
    "email":"anthony",
    "name":"",
    "login":"anthony",
    "theme":"",
    "orgId":1,
    "isGrafanaAdmin":false
}
```

Making Apache’s auth work together with Grafana’s AuthProxy

I’ll demonstrate how to use Apache for authenticating users. In this example we use BasicAuth with Apache’s text file based authentication handler, i.e. htpasswd files. However, any available Apache authentication capabilities could be used.

Apache BasicAuth

In this example we use Apache as a reverse proxy in front of Grafana. Apache handles the Authentication of users before forwarding requests to the Grafana backend service.

Apache configuration

```
    <VirtualHost *:80>
        ServerAdmin webmaster@authproxy
        ServerName authproxy
        ErrorLog "logs/authproxy-error_log"
        CustomLog "logs/authproxy-access_log" common

        <Proxy *>
            AuthType Basic
            AuthName GrafanaAuthProxy
            AuthBasicProvider file
            AuthUserFile /etc/apache2/grafana_htpasswd
            Require valid-user

            RewriteEngine On
            RewriteRule .* - [E=PROXY_USER:%{LA-U:REMOTE_USER},NS]
            RequestHeader set X-WEBAUTH-USER "%{PROXY_USER}e"
        </Proxy>

        RequestHeader unset Authorization

        ProxyRequests Off
        ProxyPass / http://localhost:3000/
        ProxyPassReverse / http://localhost:3000/
    </VirtualHost>
```

- The first four lines of the virtualhost configuration are standard, so we won't go into detail on what they do.
- We use a **<proxy>** configuration block for applying our authentication rules to every proxied request. These rules include requiring basic authentication where user:password credentials are stored in the **/etc/apache2/grafana_htpasswd** file. This file can be created with the `htpasswd` command.
  - The next part of the configuration is the tricky part. We use Apache's rewrite engine to create our **X-WEBAUTH-USER header**, populated with the authenticated user.
    - **RewriteRule .* - [E=PROXY_USER:%{LA-U:REMOTE_USER}, NS]**: This line is a little bit of magic. What it does, is for every request use the rewriteEngines look-ahead (LA-U) feature to determine what the REMOTE_USER variable would be set to after processing the request. Then assign the result to the variable PROXY_USER. This is necessary as the REMOTE_USER variable is not available to the RequestHeader function.
    - **RequestHeader set X-WEBAUTH-USER "%{PROXY_USER}e"**: With the authenticated username now stored in the PROXY_USER variable, we create a new HTTP request header that will be sent to our backend Grafana containing the username.
- The **RequestHeader unset Authorization** removes the Authorization header from the HTTP request before it is forwarded to Grafana. This ensures that Grafana does not try to authenticate the user using these credentials (BasicAuth is a supported authentication handler in Grafana).
- The last 3 lines are then just standard reverse proxy configuration to direct all authenticated requests to our Grafana server running on port 3000.

Full walkthrough using Docker.

For this example, we use the official Grafana Docker image available at Docker Hub.

- Create a file `grafana.ini` with the following contents

```
[users]
allow_sign_up = false
auto_assign_org = true
auto_assign_org_role = Editor

[auth.proxy]
enabled = true
header_name = X-WEBAUTH-USER
header_property = username
auto_sign_up = true
```

Launch the Grafana container, using our custom grafana.ini to replace `/etc/grafana/grafana.ini`. We don't expose any ports for this container as it will only be connected to by our Apache container.

```
docker run -i -v $(pwd)/grafana.ini:/etc/grafana/grafana.ini --name grafana grafana/grafana
```

Apache Container

For this example we use the official Apache docker image available at Docker Hub

- Create a file `httpd.conf` with the following contents

```
ServerRoot "/usr/local/apache2"
Listen 80
LoadModule mpm_event_module modules/mod_mpm_event.so
LoadModule authn_file_module modules/mod_authn_file.so
LoadModule authn_core_module modules/mod_authn_core.so
LoadModule authz_host_module modules/mod_authz_host.so
LoadModule authz_user_module modules/mod_authz_user.so
LoadModule authz_core_module modules/mod_authz_core.so
LoadModule auth_basic_module modules/mod_auth_basic.so
LoadModule log_config_module modules/mod_log_config.so
LoadModule env_module modules/mod_env.so
LoadModule headers_module modules/mod_headers.so
LoadModule unixd_module modules/mod_unixd.so
LoadModule rewrite_module modules/mod_rewrite.so
LoadModule proxy_module modules/mod_proxy.so
LoadModule proxy_http_module modules/mod_proxy_http.so
<IfModule unixd_module>
User daemon
```

```
Group daemon
</IfModule>
ServerAdmin you@example.com
<Directory />
    AllowOverride none
    Require all denied
</Directory>
DocumentRoot "/usr/local/apache2/htdocs"
ErrorLog /proc/self/fd/2
LogLevel error
<IfModule log_config_module>
    LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\"" combined
    LogFormat "%h %l %u %t \"%r\" %>s %b" common
    <IfModule logio_module>
    LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-Agent}i\" %I %O" combinedi
    </IfModule>
    CustomLog /proc/self/fd/1 common
</IfModule>
<Proxy *>
    AuthType Basic
    AuthName GrafanaAuthProxy
    AuthBasicProvider file
    AuthUserFile /tmp/htpasswd
    Require valid-user
    RewriteEngine On
    RewriteRule .* - [E=PROXY_USER:%{LA-U:REMOTE_USER},NS]
    RequestHeader set X-WEBAUTH-USER "%{PROXY_USER}e"
</Proxy>
RequestHeader unset Authorization
ProxyRequests Off
ProxyPass / http://grafana:3000/
ProxyPassReverse / http://grafana:3000/
```

- Create a htpasswd file. We create a new user **anthony** with the password **password**

  ```
  htpasswd -bc htpasswd anthony password
  ```

- Launch the httpd container using our custom httpd.conf and our htpasswd file. The container will listen on port 80, and we create a link to the **grafana** container so that this container can resolve the hostname **grafana** to the Grafana container's IP address.

  ```
  docker run -i -p 80:80 --link grafana:grafana -v $(pwd)/httpd.conf:/usr/local/apache2/c
  ```

Use grafana.

With our Grafana and Apache containers running, you can now connect to http://localhost/ and log in using the username/password we created in the htpasswd file.

Team Sync (Enterprise only)

> Only available in Grafana Enterprise v6.3+

With Team Sync, it's possible to set up synchronization between teams in your authentication provider and Grafana. You can send Grafana values as part of an HTTP header and have Grafana map them to your team structure. This allows you to put users into specific teams automatically.

To support the feature, auth proxy allows optional headers to map additional user attributes. The specific attribute to support team sync is `Groups`.

```
# Optionally define more headers to sync other user attributes
headers = "Groups:X-WEBAUTH-GROUPS"
```

You use the `X-WEBAUTH-GROUPS` header to send the team information for each user. Specifically, the set of Grafana's group IDs that the user belongs to.

First, we need to set up the mapping between your authentication provider and Grafana. Follow [these instructions]({{< relref "team-sync.md#enable-synchronization-for-a-team" >}}) to add groups to a team within Grafana.

Once that's done. You can verify your mappings by querying the API.

```
# First, inspect your teams and obtain the corresponding ID of the team we want to inspect t
curl -H "X-WEBAUTH-USER: admin" http://localhost:3000/api/teams/search
{
  "totalCount": 2,
  "teams": [
    {
      "id": 1,
      "orgId": 1,
      "name": "Core",
      "email": "core@grafana.com",
      "avatarUrl": "/avatar/327a5353552d2dc3966e2e646908f540",
      "memberCount": 1,
      "permission": 0
    },
    {
      "id": 2,
      "orgId": 1,
      "name": "Loki",
      "email": "loki@grafana.com",
      "avatarUrl": "/avatar/102f937d5344d33fdb37b65d430f36ef",
      "memberCount": 0,
      "permission": 0
    }
  ],
  "page": 1,
  "perPage": 1000
}

# Then, query the groups for that particular team. In our case, the Loki team which has an
curl -H "X-WEBAUTH-USER: admin" http://localhost:3000/api/teams/2/groups
[
  {
    "orgId": 1,
    "teamId": 2,
    "groupId": "lokiTeamOnExternalSystem"
  }
]
```

Finally, whenever Grafana receives a request with a header of `X-WEBAUTH-GROUPS: lokiTeamOnExternalSystem`, the user under authentication will be placed into the specified team. Placement in multiple teams is supported by using comma-separated values e.g. `lokiTeamOnExternalSystem,CoreTeamOnExternalSystem`.

```
curl -H "X-WEBAUTH-USER: leonard" -H "X-WEBAUTH-GROUPS: lokiteamOnExternalSystem" http://loc
{
  "meta": {
    "isHome": true,
    "canSave": false,
    ...
}
```

With this, the user `leonard` will be automatically placed into the Loki team as part of Grafana authentication.

[Learn more about Team Sync]({{< relref "team-sync.md" >}})

Login token and session cookie

With `enable_login_token` set to `true` Grafana will, after successful auth proxy header validation, assign the user a login token and cookie. You only have to configure your auth proxy to provide headers for the /login route. Requests via other routes will be authenticated using the cookie.

Use settings `login_maximum_inactive_lifetime_days` and `login_maximum_lifetime_days` under `[auth]` to control session lifetime. [Read more about login tokens]({{< relref "overview/#login-and-short-lived-tokens" >}})