ExtendingPrivilegeSeparation

Introduction

This httpd setup is inspired by DifferentUserIDsUsingReverseProxy. However it takes the idea a couple of steps further, and describes other, everyday aspects one might stumble upon. It's been tested - in production - for many years now, is very stable, scalable and most imporantly: secure.

The first deviation from DifferentUserIDsUsingReverseProxy, is to run all backends on high-ports (as it has only later been added). This has two consequences:

1. All backends can be started and run as unprivileged users, no privilege escalation can happen from malicious scripts executed by httpd. 2. The frontend never executes third party code, it's security concerns are confined within the limits of the Apache HTTPd.

Furthermore we shall demonstrate how to secure PHP applications without safe-mode. Finally we'll peek into automating the entire process with mod_macro

Bare minimum

A great deal of the configurations is shared accross all the instances of httpds, so we'll show it here:

```
ServerRoot "/opt/es"
ServerAdmin webmaster@esotericsystems.at
LoadModule authz_host_module libexec/apache/mod_authz_host.so
LoadModule include_module libexec/apache/mod_include.so
LoadModule charset_lite_module libexec/apache/mod_charset_lite.so
LoadModule env_module libexec/apache/mod_env.so
LoadModule mime_magic_module libexec/apache/mod_mime_magic.so
LoadModule expires_module libexec/apache/mod_expires.so
LoadModule headers_module libexec/apache/mod_headers.so
LoadModule setenvif_module libexec/apache/mod_setenvif.so
LoadModule mime_module libexec/apache/mod_mime.so
LoadModule negotiation_module libexec/apache/mod_negotiation.so
LoadModule dir_module libexec/apache/mod_dir.so
LoadModule alias_module libexec/apache/mod_alias.so

<Directory />
    Options FollowSymLinks
    AllowOverride None
    Order deny,allow
    Deny from all
</Directory>

DirectoryIndex index.html

<FilesMatch "^\.ht">
    Order allow,deny
    Deny from all
    Satisfy All
</FilesMatch>

LogLevel warn

DefaultType text/plain
ExpiresActive on
TypesConfig etc/apache/mime.types
MIMEMagicFile etc/apache/magic

Header append Vary User-Agent env=!dont-vary

Include etc/apache/extra/httpd-mpm.conf
Include etc/apache/extra/httpd-languages.conf
Include etc/apache/extra/httpd-default.conf
```

With this simple config any host can serve static content at the very least. If it's not supposed to do any more than that, it will do it without any kludge. This is all it needs to perform the task.

Frontend

```
# include base config
Include /opt/es/etc/apache/httpd.conf
Include /opt/es/etc/apache/extra/httpd-proxy.conf
Include /opt/es/etc/apache/extra/httpd-deflate.conf
Include /opt/es/etc/apache/extra/httpd-cache.conf
# only the proxy does CustomLogging!
Include /opt/es/etc/apache/extra/httpd-log.conf
# listen to UID
Listen 80
Listen 443
User www
Group www

# supply PID and lock file
PidFile "/var/opt/es/apache/proxy/proxy.pid"
LockFile "/var/opt/es/apache/proxy/proxy.lock"

ErrorLog "/var/opt/es/apache/proxy/error_log"
CustomLog "/var/opt/es/apache/proxy/access_log" vhostcombined env=!dontlog

ServerName borscht

NameVirtualHost *:80

<VirtualHost *:80>
        ServerName esotericsystems.at:80
        ProxyPass / http://127.0.0.1:8001/
        ProxyPassReverse / http://127.0.0.1:8001/
</VirtualHost>
<VirtualHost *:80>
        ServerName omfzd.tld:80
        ProxyPass / http://127.0.0.1:8002/
        ProxyPassReverse / http://127.0.0.1:8002/
</VirtualHost>
# etc...
# SSL VHosts:
Include /opt/es/etc/apache/extra/httpd-ssl.conf
<VirtualHost 1.2.3.4:443>
        ServerName insecure.org:443
        RequestHeader set X_FORWARDED_PROTO 'https'
        SSLEngine On
        SSLCertificateFile "/opt/es/etc/certs/server.insecure.org.cert"
        SSLCertificateKeyFile "/opt/es/etc/certs/private.insecure.org.key"

        ProxyPass / http://127.0.0.1:8003/
        ProxyPassReverse / http://127.0.0.1:8003/
</VirtualHost>
# etc..
```

We'll skip the explanation of the obvious, and come straight to the `CustomLog`. We only log in the front-end. And even here, we only have **one** `CustomLog`, effectively reducing the number of open handles.

You might notice the use `:80` in the `ServerName`, this has proved to be a workaround for some applications, wel'll see more of this in the backends.

In `{ProxyPass} use {{disableruse=on` as a workaround if you're affected by PR#45362

We can also use the frontend as SSL Terminator, leaving the backend to concentrate on it's real business, not on encryption.

Backends

Everything can be a backend. Even though in the above example I've only shown `http://`
for `ProxyPass`, this doesn't keep you from running mongrels, or Tomcats (and thus to use `ajp://`) in the backend.

As we're concentrating on Apache HTTPd, we'll show some examples with that, as well as it's peripherals.

Base Config

All backends have a certain config style in common, and we'll first show that (from a template) to outline the basic idea:

```
# include base config
Include /opt/es/etc/apache/httpd.conf
# listen to UID
Listen 127.0.0.1:UID
User template.tld
Group template.tld
ServerName template.tld

# include other useful stuff:
Include /opt/es/etc/apache/extra/httpd-multilang-errordoc.conf
Include /opt/es/etc/apache/extra/httpd-php.conf

# supply PID and lock file
PidFile "/var/opt/es/apache/template.tld/pid"
LockFile "/var/opt/es/apache/template.tld/lock"
ErrorLog "/var/opt/es/apache/template.tld/error_log"

<Directory /srv/web/template.tld>
        Options +MultiViews
        Allow from All
        AllowOverride None
</Directory>

NameVirtualHost 127.0.0.1:UID

Include /opt/es/etc/apache/vhosts/template.tld/www-httpd.conf
# Maybe Include some more (sub domains...)
```

The baseconfig defines a `User` and a `Group`, our convention is to name it same as the `ServerName`. In the `Listen` directive we see that this convention is further translated to listening to this user's UID.

We have one ErrorLog per domain, but if you like to log per vhost, you can of course change it.

We then define some sane settings for `<Directory>` where our vhosts will be located, start off name-based vhosting and start including vhosts.

Before looking into the vhosts, I'd like to dwell on the subject of structuring websites. We've chosen a rather simple setup:

```
/srv/web/omfzd.tld
|-- www
|   |-- htdocs
|   |-- session
|   `-- tmp
`-- intra
    |-- htdocs
    |-- session
    `-- tmp
```

Discussing whether or not it's a good idea to have the default vhost be *www.* is moot. It's just a convention, you can name it whatever you like.

Putting each domain in one folder, and each of it's subdomains in a sub-folder thereof. This organization eases the structuring of configurations, the separation of privileges and also enables you to interface with other daemons such as an `OpenSSHd`.

We also see here a `session` and a `tmp` directory. More on this soon!

The VHosts

We'll be using the same vhosts as in the front-end example to gradually increase complexity and show different aspects of the configurations.

Simple

The most simple of vhosts serves static content and looks like this:

```
<VirtualHost 127.0.0.1:8001>
    ServerName http://esotericsystems.at:80
    ServerAlias www.esotericsystems.at
    DocumentRoot "/srv/web/esotericsystems.at/www/htdocs"
</VirtualHost>
```

Note that again we're using `http://esotericsystems.at:80` as `ServerName`, this is very important for Redirects!

Also some applications take this as a hint where they're really running on, because not many applications bother to check *X-Forwarded-For*...

PHP

PHP is not to be trusted. However running it in safe-mode is just a pain. As we've already taken care of privilege separation, we'll now go a step further and cut it off from the rest of the world using `open_basedir`.

But instead of sharing a common `/tmp/` for sessions and uploads, we separate those as well, as already hinted by the folder-structure:

```
<VirtualHost 127.0.0.1:8002>
    ServerName http://omfzd.tld:80
    ServerAlias www.omfzd.tld
    DocumentRoot "/srv/web/omfzd.tld/www/htdocs"
    php_admin_value open_basedir /srv/web/omfzd.tld/www/:/opt/es/share/pear/
    php_admin_value session.save_path /srv/web/omfzd.tld/www/session/
    php_admin_value upload_tmp_dir /srv/web/omfzd.tld/www/tmp/
</VirtualHost>
```

In `open_basedir` we have to include all the paths that our PHP application needs access. If for instance, you're serving a MediaWiki, your `open_basedir`

line would look something like this:

```
php_admin_value open_basedir /srv/web/omfzd.tld/www/:/opt/es/share/pear/:/usr/bin/diff:/usr/bin/convert
```

This would allow PHP access to `/usr/bin/diff`, but also to `/usr/bin/diff3`
and other variations thereof! Please refer open_basedir documentation for more information, or to the php.ini documentation in general.

Another directive we could use here, is PHPIniDir. It would enable us to have an unique per-domain (! Not per-vhost!) `php.ini`.

Complex

This example shows our SSL VHosts, it includes a sample for configuring mod_passenger as well as authentication:

```
LoadModule passenger_module /var/lib/gems/1.8/gems/passenger-2.2.2/ext/apache2/mod_passenger.so
PassengerRoot /var/lib/gems/1.8/gems/passenger-2.2.2
# Disable Passenger in Server Context, we'll only enable it where needed
PassengerEnabled off

<VirtualHost 127.0.0.1:8003>
    ServerName http://insecure.org:443
    ServerAlias www.insecure.org
    DocumentRoot "/srv/web/insecure.org/www/htdocs"

    RailsEnv production
    RailsBaseURI /projects
    <Location /projects>
           PassengerEnabled on
    </Location>

    <Location />
        Require valid-user
        Authuserfile /srv/web/insecure.org/www/.insec_user
        Authname "Authorized Access Only."
        Authtype basic
    </Location>
</VirtualHost>
```

Again the `ServerName` is `https://insecure.org:443`. Because even that doesn't help much with some applications (Redmine in this case), we set:

```
        RequestHeader set X_FORWARDED_PROTO 'https'
```

in the frontend (because I thought it's a more appropriate place) as suggested by their FAQ.

As the comments suggest, we disable mod_passenger for the Server Context. We only want it where we need it, in this case in `<Location /projects>`.

And finally we can see that authentication requests can be required from the backend. The frontend will transparently put it through to the clients browsing your website.

On the otherhand, if you have a backend which doesn't know how to deal with authentication, but needs protection, you can do the authentication in the frontend.

Automation with mod_macro