Task 4 - Access Control Lists Walkthrough

What is a Access Control List

Access Control Lists (ACL) are structures, that we use to define as set of rules. We use these rules to determine if the connection we are receiving is going to be accepted or rejected. This is of course based off of the conditions that we specify in the ACL. We have just described (in a simple way!) what a Network ACL is, there are additional catagories and applications of ACLs; One such example is their use in File Systems - in that case they determine if actions can be taken on certain files or directories based on the rules we define.

Why do we want to utilize Access Control List

This is hopefully a simple question. We use ACLs to limit and control access to some resource based on the conditions and requirements that we define. They are **not** firewalls, but they can act in a similar way.

Common Use cases of Access Control List

Access Control Lists are not only found in proxies, we can also find them implemented in routers, both physical and virtual (in the case of AWS). Their implementation in the routers, are used in a similar manner to their use in proxies. We can use them to limit or grant access based on a set of conditions that we define. There is also the use of ACLs in a filesystem as was mentioned earlier.

Solutions

As with the previous solution, we have provided multiple solutions with the same outcome - but they may be used in different scenarios with varying levels of difficulty. Students are only required to produce the Nginx solution.

We wish to point out theses solutions do not utilize TLS, the addition of TLS and other options will not affect the use of ACLs as they are described here - the only additions to the configurations are those necessary for listening on port 443, and enabling SSL (SSL certs, options, ect).

Nginx Implementation

1. Open (or create if it does not exist) a configuration file **/etc/nginx/nginx.conf** or add a server to the **/etc/nginx/sites-available/**. Below is an example where modifications are made to the **/etc/nginx/nginx.conf**.
    - Note there may be issues if you are using nginx and the default site still exists, as this may have priority over the configuration file in **/etc/nginx/conf.d/proxy.conf**. You can overwrite the site in **/etc/nginx/sites-available/default** with the proxy configuration, you can remove the default site or add another site in the **/etc/nginx/sites-available/** directory.

This is The configuration that should forward all traffic to the backend

```
# Set the user and group for nginx to run under
user nginx;
worker_processes auto;
```

```
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

events {
    worker_connections 1024;
}

# Configure the HTTP module
http {
  # Define the main server block
  server {
    # Listen on port 8080 for incoming requests
    listen 80;
    server_name  <insert_url_here>; # Example www.team6.umlcyber.club
    location / {
        proxy_pass http://<hostname/IP:PORT> # Example  http://127.0.0.1:8080 or
http://web-container:8080

    }
  }
}
```

1. Add the following block to the frontend, this will limit access to the admin page. This will be inserted **below** the "listen 80" line but **above** the "location / { ...}" lines.

```
# The following will be inserted into the frontend
location /admin {
    allow <IP>; # the more specific rules and conditions must come before the more
general ones
    allow <LOCALHOST>;
    deny all; # If this was the first condition all others would be blocked
    proxy_pass http:// http://<hostname/IP:PORT>/admin # If this is a container
change out 127.0.0.1 for the container name.
}
```

1. Add similar sections as the one described above for each page you would like to block
2. **Optional** We can modify the traffic the proxy sends so it contains some more useful information. The lines are provided below.

```
    proxy_set_header        Host           $host;
    # Provide info on the sending host
    proxy_set_header        X-Real-IP      $remote_addr;
    proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
```

- We can include these optional configurations as a separate file that is included as described in a Full Config Example

Once the above steps are done we will have the following configuration file which is a **Minimum Viable Proxy** we can of course add additional configurations that we do not describe.

```
# Set the user and group for nginx to run under
user nginx;
worker_processes auto;
error_log /var/log/nginx/error.log;
pid /run/nginx.pid;

events {
    worker_connections 1024;
}

# Configure the HTTP module
http {
  # Define the main server block
  server {
    # Listen on port 80 for incoming requests
    listen 80;
    server_name  <insert_url_here>; # Example www.team6.umlcyber.club
    # Modify the information contained in the packet so the backend server will
log the correct information
    proxy_set_header        Host            $host;
    proxy_set_header        X-Real-IP       $remote_addr;
    proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;

    # Change actions if the user is attempting to access the admin page
    location /admin {
        allow 10.0.0.10; # Change IP!
        allow 127.0.0.1;
        deny all;
        proxy_pass http://<hostname/IP:PORT>/admin;
    }
    # All other traffic
    location / {
        proxy_pass http://<hostname/IP:PORT>;
    }
  }
}
```

> We can add "listen 443" and the necessary SSL configurations to make the proxy a SSL proxy, they are not included here for simplicity

We must now restart the nginx service

```
$ sudo systemctl restart nginx
```

Docker

1. We can use the following command to make nginx re-read the configuration
   - docker kill --signal=HUP <nginx_container>
   - ***STOP AFTER THIS ONLY CONTINUE IF NO CHANGES OCCUR***
2. Remove the existing Nginx container
   - docker ps
   - docker rm -f <container-Name/ID>
3. Start the new container, with the new configuration
   - docker run -d -p 80:80 -p 443:443 --name <name> --network <Proxy-Server-Network> -v **/etc/nginx/nginx.conf**:**/etc/nginx/nginx.conf** nginx
     - Note that the path in the **-v** flag on the host may need to change ex:**-v -/home/manager/nginx.cong:/etc/nginx/nginx.conf**

Non-Docker

1. Restart Nginx
   - sudo systemctl restart nginx

Testing

We can test whether this is working in 3 steps.

1. Access the website as you have done before - it should continue to work
2. Attempt to access the admin page - you should get a 403 forbidden
3. On the **Virtual Machine** run the command below. You should het a index.html page

```
$ wget localhost:80/admin
```

Resources

1. https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/
2. https://docs.nginx.com/nginx/admin-guide/security-controls/controlling-access-proxied-tcp/
3. https://docs.nginx.com/nginx/admin-guide/security-controls/controlling-access-proxied-http/
4. https://www.nginx.com/resources/wiki/start/topics/examples/full/

HaProxy Implementation

1. Open the configuration file. Below is an example file with a single front end, and a single backend

```
This will be an example file
```

1. Add the following line to the frontend. This will handle allowing access to a page if the user is from a specific IP

```
    acl network_allowed src \<ip1\> \<ip2\>
```

1. Add the following line to the frontend. This is the one that would be repeated for all paths we would like to block.

```
    acl restricted_page path_beg /admin
```

1. Add the following line to the frontend, this is the instruction that will actually block the connection if the conditions listed are true.

```
    block if restricted_page !network_allowed
```

We will have the following configuration

```
frontend fe
    bind *:80
    bind *:443
    acl network_allowed src <ip1> <ip2>
    acl restricted_page path_beg, url_dec -i /admin
    block if restricted_page !network_allowed
    use_backend be

backend be
    mode http
    server s1 <IP-S1>:8080
```

Resources:

- https://raymii.org/s/snippets/haproxy_restrict_specific_urls_to_specific_ip_addresses.html -- Full Example