Linux System Administration

PM-TRTNB319

Zsolt Schäffer, PTE-MIK

Legal note

These slides form an outline of the curriculum and contain multiple references to the official Red Hat training materials (codes RH124, RH134 and RH254), since students of this subject are also eligible for the named RH learning materials. Some diagrams and pictures originating from the RH courses will show up in this outline as there is an agreement in place which allows for our Faculty to teach this subject based on the Red Hat curriculum.

All other pictures and diagrams sourced from third parties are explicitly marked with a reference to the origin.

Be aware that the mentioned parts and also this series of slides as a whole are property of their respective owners and are subject to copyright law.

Legal note

All students of PTE-MIK who have officially taken the course „Linux System Administration” are eligible to download these slides from the Faculty’s internal network, and are eligible for license keys for Red Hat System Administration online learning materials at rhlearn.gilmore.ca as part of their training program.

Do not share, redistribute, copy or otherwise offer these learning support materials, license keys or account information either for money or free of charge to anyone, as these materials contain **intellectual property.**

Unauthorized distribution is both against the law and university policy and will result in an in-campus inquiry, suing for damages and/or criminal prosecution by the University of Pécs, Red Hat Inc. and other parties.

SSH

Lecture 10

Chapter 1

SSH

- SSH’s (Secure Shell) first version was developed by Tatu Ylönen (Helsinki University) in 1995.
- Current version is v2, an open-source implementation called OpenSSH is available in practically all *nix systems.
- SSH implements public key cryptography for authenticating both server machine and client user.
- SSH applies the Diffie-Hellman key exchange algorithm or one of its variants for securing the channel.
- SSH is considered very secure, it is mathematically proven to be very safe, no one has been able to exploit it so far.
- SSH offers additional features through the established secure channel, like TCP tunneling, file transfer, remote process execution and redirection of standard streams.

Cryptography

- Symmetric cryptography algorithms use the the same key for encrypting and decryption of the message.
- Symmetric algorithms work on blocks of data (that are of the same size as the key).
- One of the most commonly known symmetric algorithms is AES.
- AES is based on the work of Joan Daemen and Vincent Rijmen (Belgium), called the Rijndael encryption.
- Symmetric algorithms work relatively fast, most modern CPUs (even low-powered embedded ones) have built-in hardware instruction(s) for calculating AES for a block with a given key.
- An Intel x86 Core i7-5820K CPU at 3.3 Ghz can calculate AES for more than 4GB of data per each second.

Cryptography

- Asymmetric cryptography incorporates a pair of keys. Data encrypted with one of the keys can only be decrypted with the other and vice versa.
- They are based on irreversible mathematical problems like the discrete logarithm problem or the difficulty to break a number down to prime factors.
- Asymmetric algorithms run slower by magnitudes.
- Usually one part of the key pair is proclaimed public and get distributed through public databases (notary, trust chain), the other one is kept private. When encrypting a message with the private key, it will be readable to all, but this will prove the origin of the message (digital signature). When encrypting data with the public key, everyone can send a message to an addressee, that only the addressed individual can read (encryption).
- Widely known: RSA algorithm (Rivest, Shamir, Adleman)

SSH session encryption

- When establishing an SSH session the following steps are involved.
  - Protocol version, algorithm exchange, agreement.
  - Server public key is sent to the client for comparison to clients records. (**known_hosts**)
  - Client ensures the server has the private key corresponding to the public key received.
  - Diffie-Hellman key exchange (→ See next slide)
  - Symmetric session key is obtained by both parties, the connection is secure from now on.
  - EITHER Authentication with the users password (if enabled), the password can be sent safely by now.
  - OR Authentication with the users private key (if it is available) → Detailed in next slide

SSH session encryption

- Authenticating the user with a key pair involves the following steps.
  - The server has a catalog of all allowed user’s public keys. (**authorized_keys**)
  - Server generates a random number sequence, encrypts it with the users public key, and sends it.
  - The client (if it has the private key) decrypts the number and combines it with the session key, then calculates its MD5 hash.
  - Client sends the result back to server.
  - Server calculates the same hash.
  - If response matches servers calculation, the client user proved that he/she is in possession of the private key, therefore is considered authenticated.

Diffie-Hellman example

- Bob and Alice agree on a prime modulus (m) and a generator number (g). For e.g. g=3 m=17
  Both Alice (**A**) and Bob (**B**) selects a random **private** number. For e.g. A=15 and B=13.
- Alice calculates her public number $P_A=g^A$ mod m, and sends this to Bob. (e.g. 3^{15} mod 17= 6)
  Bob does the same with his public number $P_B=g^B$ mod m, and send it to Alice. (e.g. 3^{13} mod 17 = 12)
- Alice takes Bob's public number and calculates
  $S = P_B^A$ mod m (e.g. 12^{15} mod 17 = 10)
  Bob does the same with Alice's public number.
  $S= P_A^B$ mod m (e.g. 6^{13} mod 17 = 10).
- S is their shared secret. Note that they did the same calculation: $(3^{15})^{13}$ mod 17 = $(3^{13})^{15}$ mod 17, but an eavesdropper can't follow this calculation without knowing numbers **A** and **B** (13 and 15 in this example).

SSH session encryption

- Note that the DH private and public numbers are unrelated to SSH private and public keys, rather they are randomly generated on the start of each session.
- Note that session encryption is done with a symmetrical algorithm, that provides much faster speeds. Asymmetrical algorithms are only involved on the start of a session, where the identity of the other party is established.
- Note that Diffie-Hellman by itself does not protect against Man-in-the-Middle attacks. That's why a server computer also has a keypair, not just client users.
- SSH clients ask for confirmation when encountering a server for the first time. Later it keeps track of the public keys of servers, and refuses to connect if a server key changes.

SSH session encryption

- MITM attack can only succeed at the first encounter, when client does not yet know the public key of the server. Therefore it is encouraged to propagate the public key of a server via others means (like directly copying from a flash drive instead of over the network.)

ssh first encounter example

```
local-host$ ssh remote-host
The authenticity of host 'remote-host(192.168.1.1)'
can't be established. RSA key fingerprint is
90:9c:46:ab:03:1d:30:2c:5c:87:c5:c7:d9:13:5d:75.
Are you sure you want to continue connecting
(yes/no)? yes
Warning: Permanently added 'somehost' (RSA) to the
list of known hosts.
user@somehost's password: [not shown]
remote-host$
```

SSH host key change

ssh host key change

```
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!@
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING
NASTY! Someone could be eavesdropping on you right
now! (man-in-the-middle attack)
It is also possible that the RSA host key has just
been changed.The fingerprint for the RSA key sent
by the remote host is
90:9c:46:ab:03:1d:30:2c:5c:87:c5:c7:d9:13:5d:75.
Please contact your system administrator.
Add correct host key in /home/user/.ssh/known_hosts
to get rid of this.
Offending key in /home/user/.ssh/known_hosts:1
Password authentication is disabled to avoid man-
in-the-middle attacks.
Agent forwarding is disabled to avoid man-in-the-
middle attacks.
X11 forwarding is disabled to avoid man-in-the-
middle attacks.
```

SSH client

- To start a remote interactive session with ssh simply issue **ssh -p 1822 user@hostname-or-ipaddress**. Where the -p switch defines a non standard port, if omitted, the standard port of 22/tcp will be used. Username can also be omitted, in this case the same username will be tried as the user running the local shell.
- Use **ssh user@host *command*** to execute a command on the remote computer. SSH will print the output of the remote command locally. When the remote execution ends, SSH will terminate the session.
- You can even use pipes and redirects to remote processes this way, ssh will handle the transfer.

shell command line

```
local-host$ dd if=disk1.img | ssh remote-host 'dd
of=/datastore/disk1.img'
```

SSH client

- Basically SSH provides an encrypted channel of standard streams for remote processes (like redirects or pseudo terminals for remote shell sessions). SSH has some additional features, like X11 forwarding, TCP port forwarding, and secure copying files remotely (scp).
- You can invoke remote GUI programs while making them draw to the local screen with the following example: **ssh -X bob@officecomputer word-processor**
- SSH port forwarding works by technically "mirroring" a TCP port on one side of the tunnel to the other side and connecting it to another destination.
- There are 3 types of SSH port forwards: local, remote and dynamic. Dynamic provides a full-featured SOCKS proxy, the other two will be explained in detail later.
- SSH itself can't provide any means for UDP forwarding.

SSH local forward

- This can be used to bypass local firewalls, or to provide access to server side computers. Take the following example: **ssh -L 8080:redhat.com:80 bob@my_server** In this example, (after connecting the SSH client to the daemon on remote my_server machine), pointing a browser on the client computer to localhost:8080 will download a page from redhat.com's port 80. The redhat.com server will perceive as the page was downloaded from my_server and not you local machine.
- The last part of the -L argument is interpreted from the SSH servers perspective. For e.g. **12345:localhost:5432** would connect local 12345 port to the servers 5432 (postgresql) port.
- Likewise you can access restricted computers on the server side like this: **-L 17380:192.168.1.173:80**

SSH remote forward

- Remote forwards work similar but the opposite direction. An example of **ssh -R 9000:localhost:80 bob@serverXY** would do the following: Anyone opening port 9000 on the server would actually connect to your local computers web server (as long as your client is connected to the SSH session).
- Both local and remote forward requires server side agreement. Local forwarding is allowed by default, but remote forwarding is disabled in the default sshd config. Add **GatewayPorts yes** to your server configuration to allow it.
- In case of -R the second part of the argument is interpreted from the ssh client's perspective, localhost in this example means the ssh client that originated the SSH session.

scp, sshfs

- One additional element of the OpenSSH suite is the scp utility, which allows for copying files to/from remote computers via an SSH encrypted channel. Either source or destination can be a remote file (or dir), but not both. Use this command like you would use the regular cp command. E.g.: **scp thesis.txt bob@univ:/home/bob**
- By executing scp and various filesystem operation commands remotely via ssh, you can implement a file server. For convenience there is wrapper for this functionality (openssh-sftp-server), the involved protocol is called SFTP (not FTPS!).
- You can even mount a remote directory and access it locally in the regular way. Behind the scenes the work is done by the fuser kernel module, the remote sftp-server and the local ssh client. Use sshfs like you would use mount. E.g. **sshfs bob@server:/work ~/bobs-work-dir**

Client side ssh settings

- The settings for the SSH client are stored in the **/etc/ssh/ssh_config** file. The directives can be overridden be each users **~/.ssh/config** file.
- The key pairs of users are stored under **~/.ssh/id_*xxx*** and **~/.ssh/id_*xxx*.pub** for each user, where xxx refers to the algorithm, e.g. id_rsa, id_dsa, id_ecdsa, …
- The list of already encountered servers and their public key's fingerprints are stored in **~/.ssh/known_hosts**.

/etc/ssh/ssh_config

```
Host dev
    HostName dev.example.com
    Port 22000
    User fooey
    IdentityFile ~/.ssh/dev-example.key
Host tunnel
    HostName database.example.com
    LocalForward 9906 127.0.0.1:3306
```

User keypairs

- You can use **ssh-keygen -t dsa -b 2048 -C "home comp"** to create a key pair. The location to save the files to will be prompted for. You can accept the defaults (see previous slide). You can use the -t parameter to specify the algorithm, e.g rsa, dsa, ecdsa. Use -b for setting the key size (default for rsa is 1024 bits), the optional -C argument gives a comment to the key.
- The private part of the key should always be kept secret and should assume a mode of **600**.
- You will also be prompted for a passphrase. It can protect your private key, without the passphrase it will be useless to anyone who steals the key file.
- In case you use the key for password-less login, you still have to enter the key passphrase (can be inconvenient). Use an ssh-agent to keep keys in memory, this way you only have to type the pass once, until turning your PC off .

SSH server

- The server side configuration is stored in **/etc/ssh/sshd_config**. The server's host key is stored in **/etc/ssh/ssh_host_*xxx*_key** and **ssh_host_*xxx*_key.pub**, where xxx denotes the algorithm (e.g. rsa,dsa, ecdsa, …) Use **ssh-keygen** (the same way as with user keys) to create a host key. The only difference is the file's location. You can also use the -f switch to ssh-keygen to specify the output filename directly from command line.
- Public keys of users are stored in **~/.ssh/authorized_keys** on the server side for each user. For e.g. by adding Alice's public key to both /root/.ssh/authorized_keys and /home/bob/.ssh/authorized_keys file, Alice will be able to log in to this server both as the root user and as Bob. The file mode has to be **600**.
- Run **ssh-copy-id user@server** on the client side to add a key to a users authorized_keys on the server. (password needed)

SSH server

/etc/ssh/sshd_config

```
ListenAddress                0.0.0.0
Port                         22
HostKey                      /etc/ssh/ssh_host_rsa_key
HostKey                      /etc/ssh/ssh_host_dsa_key
SyslogFacility               AUTHPRIV
AuthorizedKeysFile           .ssh/authorized_keys
StrictModes                  yes

RhostsAuthentication         no
RSAAuthentication            no
PasswordAuthentication       yes
PubkeyAuthentication         yes

PermitEmptyPasswords         no
PermitRootLogin              without-password

AllowTcpForwarding           yes
GatewayPorts                 no
X11Forwarding                yes
Subsystem sftp  /usr/libexec/openssh/sftp-server
```

SSH server

- Complete practice **RH124 CH 9.2.** (ssh,w)
- Complete practice **RH124 CH 9.4.** (ssh-copy-id)
- Complete practice **RH124 CH 9.6.** (PermitRootLogin)
- Complete practice **RH124 CH 9.7.** (ssh-copy-id, PermitRootLogin)

DHCP server

Lecture 10

Chapter 2

dhcpd

- To add DHCP provider functionality to your server, install the package with **yum install dhcp-server**. This will set you up with a functioning example configuration file under **/etc/dhcp/dhcpd.conf**.
- The service can be controlled with **systemctl**, the unit name is **dhcpd.service**.
- The configuration file has options for different scopes. For. e.g there are global options (see next slide), subnet or host specific options.
- It is also possible to specify directives with conditions or matches. (see next slide: if and match constructs)
- Remember that systemd-networkd can also provide basic dhcp functions. Do not enable more than one dhcp service, usually it will lead to confusion.

dhcpd.conf

/etc/dhcp/dhcpd.conf

```
#server options
authoritative;
allow bootp;
#you can define custom dhcp variables
option space gpxe;
option gpxe.keep-san code 8 = unsigned integer 8;

#global options, can be overriden in sub-sections
subnet mask 255.255.255.0
option domain-name-servers 8.8.8.8

#subnet specific options, dhcpd will find the
#relevant interface from the IP address and mask.
subnet 192.168.5.0 netmask 255.255.255.0 {
   range 192.168.5.100 192.168.5.200;
   option broadcast-address 192.168.5.255;
   option routers 192.168.5.1;
   next-server 192.168.5.10;
   filename /pxelinux.0;
}
```

dhcpd.conf

/etc/dhcp/dhcpd.conf continued

```
host DBServer {
 hardware ethernet c0:c4:c0:45:19:ac;
 fixed-address 192.168.5.20;}
#matching rules and classes
class "denied" {
   match if substring (hardware,1,3) = 00:54:36;}
pool {
   deny members of "denied";
   range 192.168.100.100 192.168.100.200;}
#conditional options, e.g. answer depends on what
#dhcp client asks for the value: pxe gets different
#response than actual operating system.
host test_1 {
   hardware ethernet 08:00:27:1c:b1:e6;
   if exists user-class and option user-class =
                                            "gPXE" {
      option root-path "aoe:e0.1";
   } else {
      filename "gpxe.kpxe";}
}
```

Apache

Lecture 10

Chapter 3

HTTP

- Both CERN (in 1992) and NCSA (in 1993) defined the HTTP specification. The original goal was to access text easily files over network. Later on the protocol was extended with non-text transfers.(Mosaic - the first browser)
- After 1994 NCSA abandoned the project, enthusiasts continued to develop it. Enhancements where shared in the form of patches→ A Patchy → Apache
- This later became a formal organization (Apache Server Project), with an own open-source licensing scheme (Apache license).
- Look up https://httpd.apache.org/ for a comprehensive documentation.
- HTTP Methods: GET, HEAD, POST, PUT, DELETE, OPTIONS, TRACE, CONNECT

URL

- The client requests a remote resource from the server. If access is granted, the resource will be sent over network. Otherwise an error message is sent instead.
- The remote resource is described with a URL (Universal resource location) Based on RFC 1738, a URL has the following components: protocol://user:password@host:port/path-inside-server
- Any component of the URL can be omitted, except for host name. All others parts have sane defaults. For e.g. pointing a web browser to redhat.com will assume the following defaults: protocoll is http since we are using a web browser, user and password may not be required for public content, port is 80 by defualt for http protocol, and path will have a server defined default, like for e.g. /index.html or /index.php.

Example transfer

- The client sends the following to the server's port 80/tcp:

HTTP GET request

```
GET /hello.html HTTP/1.1
Host: webapp0.example.com
User-Agent: Mozilla/5.0 (X11; Linux x86_64; [...]
Accept: text/html,application/xhtml+xml;appli [...]
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
Cache-Control: max-age 0
```

- The server responds with a header and contents:

HTTP GET reply

```
HTTP/1.1 200 OK
Date: Wed, 19 Sep 2017 08:34:34 GMT
Server: Apache 2.4.20 (Red Hat) OpenSSL/1.0.1e[...]
Content-Length: 12
Keep-Alive: timeout=5, max=82
Content-Type: text/plain; charset=UTF-8
Hello World!
```

HTTP

- Reply status codes are categorized into series:
  - series 100: informational messages
  - series 200: client request successful
  - series 300: redirection happened
  - series 400: client error
  - series 500: server side error
- There are multiple HTTP servers available for different OSs. The biggest market share belongs to Apache, which is available on all common OSs including non-Unix ones, like Microsoft Windows.
- A competing web server is nginx, which is also full-featured and excels in serving static content more efficiently than Apache.

Apache

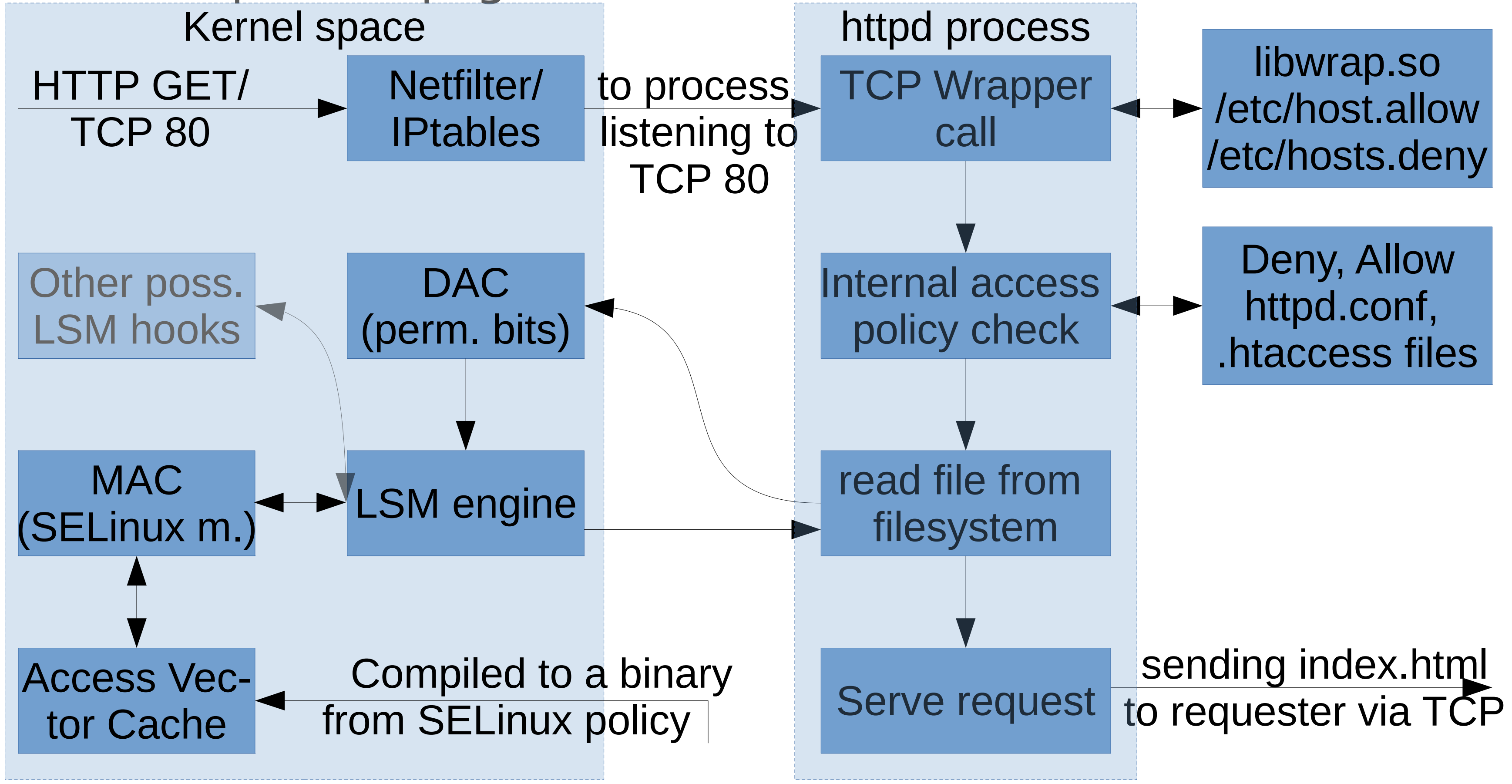
- Apache accesses several directories on the file system:
  - The apache binary, apache modules and shared libraries in /usr/bin, /usr/lib
  - Configuration files in /etc/httpd/*
  - Web content in /var/www/html (by default)
  - Server logs in /var/log/httpd/*
  - Executable scripts (cgi-bin, python, php, ...) in the web application directories for e.g. /usr/share/owncloud/*
  - Documentation in /usr/share/doc/apache (The documentation is available in html format served as a web site)
  - Possible more custom files/directories

Apache

- Most important security considerations:
  - The apache process should run with a dedicated technical account (with no shell to log in). Not the 'root' user, neither 'nobody' user.
  - The apache user and relevant group (e.g. webmasters or apache) should only own the files, that are involved in web serving.
  - Set an ACL or regular Unix permissions for the files involved, e.g.:
    setfacl -R -m d:g:webmasters:rwX /var/www/html
  - Set SELinux labels properly: e.g: **semanage fcontext -a- t httpd_sys_content_t '/location/of/web/contetnt(/.*)?'**
  - Open up firewall port(s) for the web server: e.g.:
    firewall-cmd --permanent --add-service=http

General flow of security measures for network resources

- In Linux several checks are performed before a resource (e.g. file) is served to a requester. Consider this diagram depicting the flow of events when a user wants to access a simple web page over the network.


Kernel space
HTTP GET/ TCP 80
Netfilter/ IPtables
to process listening to TCP 80
httpd process
TCP Wrapper call
libwrap.so /etc/host.allow /etc/hosts.deny
Other poss. LSM hooks
DAC (perm. bits)
Internal access policy check
Deny, Allow httpd.conf, .htaccess files
MAC (SELinux m.)
LSM engine
read file from filesystem
Access Vector Cache
Compiled to a binary from SELinux policy
Serve request
sending index.html to requester via TCP

Apache modules and multi-processing

- Apache has a modular build. Each set of functionalities is organized into binary modules, that can be loaded separately. Plugging more modules in Apache will add new features and new possible configuration directives to apache's knowledge.
- The more modules you use, the greater the memory impact.
- Apache has to load a few modules even for basic operation. E.g. you have to have a multi- processing module loaded even to start a basic server, like either one of the **prefork, worker** or **event** modules.
- Prefork starts a number of processes, each listening and serving one RQ at a time. Worker launches several threads inside processes: one listener and multiple work threads. Event is similar to worker, it tries to fix the 'keep alive' problem of open and stale connections from clients.

Apache configuration

- Apache has a default configuration file, in RH systems it’s located in **/etc/httpd/conf/httpd.conf**. The default config is quite complex and includes several other .conf files with a C-style include directive. This kind of configuration, when modified, requires the server to be restarted or reloaded.
- Apache also has a way of reading location sensitive configuration directives directly from the served content folder with .htaccess files. In this case security settings can be moved together with the web content folder. Also note, that changing these doesn’t require the server to be reloaded. The downside is a performance impact, since access files have to be re-read and evaluated on every relevant request.
- The AllowOverride directive in the regular config decides whether the .htaccess files should be considered or not.

Apache configuration

- The configuration file(s) contain some global directives, and some location or context specific (container) directives, like **<Directory /var/www> … </Directory>**
- Directives inside a container only apply to the relevant subset of files, contexts. The more specific a container is, the higher priority it has, meaning that in case of conflicting settings, directives for a sub-directory override the directives for it's parent directory.
- The **Directory** container section applies to a whole filesystem directory, the **Files** container applies to a single file. The **Location** container applies to a match in the URL. e.g.: depending on symlinks, rewrites, aliases the two following examples are not necessarily pointing to the same object: server.com/webapp/store_index and the directory /webapp/store_index under the Document-Root directory of server.com are not always the same.

Apache minimalist configuration

/etc/httpd/conf/httpd.conf

```
Listen 80
User apache
Group apache
ServerName example123.com
DocumentRoot "/var/www/html"
ServerRoot /etc/httpd
ErrorLog /var/log/httpd/error.log
PidFile   run/httpd.pid
LoadModule mpm_event_module
                      modules/mod_mpm_event.so
LoadModule authz_core_module
                      modules/mod_authz_core.so
LoadModule unixd_module modules/mod_unixd.so
LoadModule systemd_module modules/mod_systemd.so
<Directory />
   Require all denied
</Directory>
<Directory /var/www/html>
   Require all granted
</Directory>
```

Apache basic configuration examples

- Some additional important basic configuration directives:

/etc/httpd/conf/httpd.conf [excerpt]

```
LoadModule authz_host_module
                   modules/mod_authz_host.so
LoadModule dir_module
                   modules/mod_dir.so
LoadModule access_compat_module
                   modules/mod_access_compat.so
#access_compat allows for old style apache 2.2
#directives like: Allow from all

<Directory /var/www/html>
   FollowSymLinks
   AllowOverride none
   Require ip 192.168.56.0/24 #mod_authz_host rqd.
   Require not ip 192.168.56.98 #mod_authz_host
</Directory>
AddDefaultCharset UTF-8
IncludeOptional conf.d/*.conf
DirectoryIndex index.html #dir_module is needed
```

Apache configuration: password auth

- Several modules are required for this feature, one for the auth capability, one for reading users from file, groups from a file, and so on. Format examples below: first line for apache groups, second line for apache users file
- groupname: space separated list of user names
- username:$apr1$6O6rAQFK$g2M/z214SKdN7KQ.

/etc/httpd/conf/httpd.conf [excerpt]

```
LoadModule authz_user_module
                  modules/mod_authz_user.so
LoadModule authz_groupfile_module
                  modules/mod_authz_groupfile.so
LoadModule auth_basic_module
                  modules/mod_auth_basic.so
LoadModule authn_file_module
                  modules/mod_authn_file.so
LoadModule authn_core_module
                  modules/mod_authn_core.so
```

Apache configuration: password auth

- Use **htpasswd** to create entries in the users file. E.g. **htpasswd -c /etc/httpd/apachusers jane**, use -c only the first time, it stands for creating the file, otherwise it will just append new users. Also note **require valid-user**.
- You can define several requirements for an access, use **RequireAny** for setting up an OR logical operation, use **RequireAll** to create an AND logical operation.

/etc/httpd/conf/httpd.conf [excerpt]

```
<Directory /some/path>
   AuthType basic
   AuthName "Give password for finance charts"
   AuthGroupFile /etc/apache2/apache_groups
   AuthUserFile /etc/apache2/apache_users
   <RequireAny>
      require group finance
      require user boss
   </RequireAny>
</Directory>
```

Apache configuration virtual hosts

/etc/httpd/conf/httpd.conf [excerpt]

```
</VirtualHost *:80>
   ServerName www.butterflies.com
   DocumentRoot /var/www/html/butter
   ErrorLog /var/www/logs/butter_error_log
   ServerAdmin webmaster@butterflies.com
</VirtualHost>
</VirtualHost *:80>
   ServerName www.examples.com
   DocumentRoot /var/www/html/examples
   ErrorLog /var/www/logs/examples_error_log
</VirtualHost>
```

- You can run several isolated web sites/web applications on the same server. You can define virtual hosts based on IP address and/or port number. If you only have one IP address, and want to run several pages on default port 80, use name based virtual hosts. In this case the URL typed by the client will be evaluated (remember the GET header) and matched against the ServerName directive.

Apache configuration SSL

/etc/httpd/conf/httpd.conf [excerpt]

```
<VirtualHost *:443>
   ServerName www.example.com
   SSLEngine on
   SSLProtocoll all
   SSLCertificateFile
          "/path/to/www.example.com.cert"
   SSLCertificateKeyFile
          "/path/to/www.example.com.key"
   SSLCertificateChainFile
          "/etc/pki/tls/certs/example-ca.crt"
   SSLCipherSuite HIGH:!aNULL:!MD5
   SSLHonorCipherOrder on
   Documentroot /srv/example.com/html
</VirtualHost>
<VirtualHost *:80>
   ServerName www.example.com
   RewriteEngine on
   Rewriterule ^(/.*)$ https://%{HTTP_HOST}$1
                                  [redirect=301]
</VirtualHost>
```

Apache configuration dynamic web content, scripts

- To allow the execution of cgi scripts: first label them as **httpd_sys_script_exec_t**, grant access to the cgi-bin directory with a **<Directory>** block and add the following to your apache configuration:

```
ScriptAlias /cgi-bin/ "/var/www-cgi-bin"
```

- To serve dynamic PHP content: first install **mod_php**, and add the following to your config:

```
<FilesMatch \.php$>
        SetHandler application/x-httpd-php
</FilesMatch>
DirectoryIndex index.php
```

- To serve dynamic python content: install **mod_wsgi** first, label your files with **httpd_sys_content_t** and add the following to Apache's configuration:

```
WSGIScriptAlias /mywebapp/ /srv/mywebapp/some.py
```

Apache getting help

- Install the httpd-manual package with **yum install httpd-manual** and point your browser to http://localhost/manual to view the documentation.
- This is also available at **httpd.apache.org/docs/current**
- The documentation details each configuration directive with examples, explaining what module has to be loaded for the feature/directive to become available.
- Complete practice **RH254 CH 10.2.** (httpd-manual)
- Complete practice **RH254 CH 10.4.** (virtual hosts)
- Complete practice **RH254 CH 10.6.** (TLS/SSL)
- Complete practice **RH254 CH 10.8.** (php web appl.)
- Complete practice **RH254 CH 10.9.** (python with TLS)

Recommended reading

Lecture 10

Chapter 4

Relevant chapters in RH:

- RH 124 CH9 (SSH)
- RH254 CH10 (Apache/httpd)

Read by next lecture:

- RH134 CH12 (Samba)
- RH254 CH8 (File storage servers/Samba)

Use your rhlearn.gilmore.ca login to access the learning materials.

Thank you for your attention!

Lecture 10, PM-TRTNB319, Linux System Administration

Zsolt Schäffer, PTE-MIK