Linux Security

Security is not inherent in Linux

- Many people think that because they are using a Linux or a UNIX box they are more *secure* than those who use other operating systems. This is not correct
- Security is the product of system administrator’s efforts and security hardware and software
- Reaching a 100% secure system is a myth. However, you can work your way to make it harder and even harder for intruders and intruders to gain access to your system or your sensitive data.
- Because an operating system (UNIX, Linux or others) is made up for a large number of programs and services that have a lot of interdependence, any new features introduced to the system may as well introduce security threats.

Phishing

- It is some sort of "social engineering". It refers to attempting to gain information from unsuspecting users by pretending to have a legitimate authority.
- One famous example is sending an e-mail to a user asking him to reset his Facebook password. The e-mail would appear to be coming from Facebook (it's easy to change the sender's name). Inside the message is a link to a URL that *looks like* one of Facebook links (say http://www.facebook.fcbk.com). This link will lead to a page that has been crafted to look exactly like Facebook (same HTML and CSS) with two fields for username and password. Once the unsuspecting victim enters the credentials, they get instantly transferred to the hacker.
- To combat social engineering an organization may use the following methods:
  - For the previous example, most e-mail servers will recognize *most* phishing e-mail messages and mark them as spam or junk
  - Users must be educated to increase their security awareness
  - Some organizations communicate to their users that no personnel will ever ask them for their passwords, credit card numbers or any personal sensitive information

Software vulnerabilities

- A software vulnerability is a weakness or flaw in the code that was used to build the application.
- Buffer overflow: one of the most famous code vulnerabilities. It refers to placing data in a specific place in memory that is more than the size allocated to that place, which may overwrite data that could be in an adjacent place.
- Buffer overflows have been addressed in many programming languages. Java and .Net languages have their own mechanism to manage data memory allocation. Operating systems also can detect and prevent such an anomaly by preventing illegal memory access. The OS responds with a *segmentation fault* error.

Lenient configuration

- Ignoring important security settings that harden an application or the whole operating system is one of the major vulnerabilities that should be addressed.
- For example, some system administrators choose not to set a password for the BIOS setup. This means that anyone with physical access to the server can change the boot order of the machine to make it boot from a DVD that contains a malicious program.
- Other examples of ignoring important settings are turning off the firewall and SELinux.
- Such a vulnerability can be easily addressed by working with the security team in your organization to create a baseline for any newly installed operating system, where hardening options would be applied

How to secure your system?

- A fresh Linux installation needs some localization and customization in order to achieve the desired level of security baseline
- The first and most important action to take is patching. This includes installing the latest security updates and fixes from the vendor. On a Red Hat system you can use yum. If you are using Centos or Fedora that would be enough. But if you are using Red Hat Enterprise Linux, you will have to pay a subscription fee to able able to download patches from Red Hat. On a Debian system like Ubuntu you can use apt-get to achieve the same result.
- If you are working in a large environment, with many users, you will have to follow a change management document, which documents all the implications and possible effects of the updates you are installing on the running applications as well as the logged in users.

Unneeded Network services

- Even a minimal Linux installation may install some services/daemons that are unnecessary. You should disable or remove those services as they are posing an additional, avoidable risks.
- You can have a quick look at the services that get started on system boot using `chkconfig --list | grep runlevel:on` you can find your current run level using the `runlevel` command
  For example: `runlevel --list | grep 3:on`
- You can also determine which process is using which port using lsof or fuser commands. For example lsof –i:8080 will give you information about the process that is using port 8080 on your system.
- Some services are – by default – insecure and must be replaced by their secure counterparts. For example FTP should be remove in favor of vsFTP, telnet should not be used for login; it should replaced by SSH and so on

Pluggable Authentication Module

- Security programs and binaries by default do not give administrators fine-grained control over their behavior. For example, you cannot make the passwd command require at least 5 characters (including letters, numbers and special characters) without changing the source code or installing a new version.
- PAM solved this problem by placing the authentication logic in a shared library which programs like passwd and login use.
- This has lifted the load off programmers as they no longer have to incorporate authentication code in their programmers. System administrators also enjoyed having more control over the required level of security not only on user logins but also other forms of system access including web site protection

LAB: Tighten login policy using PAM

- Edit the `/etc/pam.d/system-auth` file as follows:
- Add the following as the second auth line

```
auth     required      pam_tally2.so deny=5 unlock_time=1800
```

This makes the system lock the user access after 5 failed attempts. The lock time is 1800 seconds (3 minutes)

- Add the following as the second line in account file:

```
account     required      pam_tally2.so
```

Instructs the system to use the pam_tally2 library

- Add the following as the first and second password lines:

```
password    requisite     pam_cracklib.so try_first_pass retry=5 minlen=8
ucredit=1 lcredit=1 dcredit=1 ocredit=1
```

Maximum password retries is 5, password minimum length is 8, and it must contain at least 1 uppercase letter, 1 lower case letter, 1 number and 1 special character

```
password    sufficient    pam_unix.so sha512 shadow nullok try_first_pass
use_authtok remember=10
```

Use SHA512 encryption method, do not let the user choose a password from the last 10 passwords used.

Scan for open ports (nmap)

- The syntax is simple: `nmap` *`hostname/IP address`*
- It has a large number of useful command line arguments. A simple port scan will need no more than the above example.
- Nmap works by sending a packet that appears to be coming from the middle of a conversation (ESTABLISHED) rather than the packet that starts the connection (NEW). If the packet is accepted, nmap shuts down the connection immediately, otherwise it reports that this port is closed.
- Additional useful information is returned with the list of open and closed ports, like the operating system name and sometimes even the name of the daemon that is owning the open port (for example httpd, or smtpd). The syntax for this is as follows:
  `nmap –sV –O` *`hostname/IP address`*

Scan for weak passwords (John the Ripper)

- It is a free tool that includes a number of password cracking utilities
- You can use it to test for the weak passwords in any file (typically the /etc/shadow file)
- As it works on the file to be cracked, it prints the results (if found) as `username    (password)`
- The syntax is `./john /path/to/file(s)`
- It can be installed as a binary package, or compile from source. It is recommended that you compile the source files to achieve the maximum performance.

LAB: Use John the Ripper to crack a weak password hash

- Download the latest John source packages from http://distro.ibiblio.org/openwall/projects/john/
- Untar the file and navigate to `src` directory under the uncompressed directory.
- Type `make clean linux-x86-64` to compile and install the binaries
- Create a new user with a weak password:
  ```
  useradd test
  passwd test (type test twice)
  ```
- Copy the test user entry from `/etc/shadow` file and place it in a file called `passwords.txt`
- Navigate to ../run
  ```
  cd ../run
  ```
- Run john on the passwords.txt file:
  ```
  ./john passwords.txt
  ```