Linux Professional Institute

LPIC-1

جلسه چهاردهم: مفاهیم اولیه‌ی امنیت و رمزنگاری

By: The Albatross

thealbatross@yandex.com

https://github.com/TheAlbatrossCodes/Linux-In-Persian

فهرست مطالب

مقدمه

جلسه قبل، با مفاهیم اولیه‌ی ایمیل و دیتابیس‌ها آشنا شدیم و کمی با آنها کار کردیم. در این جلسه می‌خواهیم به طور کلی در مورد امنیت شبکه و سیستم صحبت کنیم. ما ابتدا با برخی از ابزارهای مورد استفاده برای بازرسی امنیت سیستم آشنا می‌شویم و سپس در مورد مفاهیم رمزنگاری، چگونگی ارتباط امن با یک سرور و همچنین رمزگذاری فایل‌ها صحبت خواهیم کرد.

امنیت در شبکه

منزلی را در نظر بگیرید که همه‌ی اتاق‌هایش یک درب به بیرون داشته باشند. در چنین منزلی، قبل از خوابیدن یا بیرون رفتن از خانه، باید همه‌ی درب‌ها را بررسی کرده و آنها را قفل کنیم؛ خواه‌ناخواه روزی قفل کردن یکی از این درب‌ها را فراموش می‌کنیم و فاجعه به بار می‌آید. در دنیای شبکه، وجود سرویس‌هایی در سیستم که از آنها استفاده‌ای نمی‌کنیم، شبیه به داشتن منزلی با تعداد زیادی درب به سمت بیرون می‌باشد. در دنیای امنیت سایبری، این مفهوم را با واژه‌ی «سطح حمله» تعریف می‌کنند. سطح حمله، بیانگر کلیه‌ی نقاطی می‌باشد که یک فرد مخرب می‌تواند از آن برای حمله و دسترسی به سیستم ما از آن استفاده کند.

بیایید به مثالِ منزل چند دَربی بازگردیم. فرض کنید در این منزل، چندین درب با قفل‌های قدیمی و زنگ‌زده داشته باشیم که دیگر به درستی کار نمی‌کنند. ما باید در اسرع وقت این قفل‌ها را تعمیر کرده یا آنها را تعویض کنیم. نرم‌افزارهای قدیمی موجود در سیستم، دقیقا مانند این قفل‌های زنگ‌زده می‌باشند و برای جلوگیری از وقوع یک فاجعه، باید آنها را به‌روزرسانی کرده و از اول تنظیم کنیم و یا حتی در برخی از موارد، نرم‌افزار جدیدی را جایگزین آنها کنیم.

در این بخش، در مورد چگونگی بازرسی سیستم خود جهت پیدا کردن سرویس‌هایی که از آنها استفاده‌ای نمی‌شود صحبت خواهیم کرد.

غیرفعال‌کردن سریس‌های بلااستفاده

یکی از روش‌های کاهش سطح حمله‌ی سیستم، غیر فعال کردن سرویس‌هایی که به آنها نیازی نداریم و از آن استفاده نمی‌کنیم می‌باشد. وجود نرم‌افزارهای کمتر در سیستم، اهداف کمتری را در اختیار یک فرد مخرب برای حمله به سیستم قرار می‌دهد.

ممکن است فکر کنید که سیستم زیر دست خود را به خوبی می‌شناسید و از تک‌تک سرویس‌های موجود در آن استفاده می‌کنید؛ در زمینه‌ی امنیت، حتی اگر به سیستم خود و نرم‌افزارهای موجود در آن اطمینان کامل داشته باشید، بهتر است باز هم آن را به صورت دقیق بازرسی کرده تا از امنیت آن اطمینان کامل حاصل کنید.

در این بخش به بررسی ابزارهای متفاوت برای بازرسی سرویس‌های فعال روی سیستم می‌پردازیم.

پیدا کردن پورت‌های باز با *nmap*

ابزار Network Mapper یا nmap که معمولا از آن برای تست نفوذ استفاده می‌شود، در بازرسی سیستم نیز بسیار کاربرد دارد. ابزار nmap به عنوان یک ابزار برای اسکن کردن پورت‌های باز روی سیستم طراحی شده است و قابلیت‌هایی نظیر اسکن کردن چندین سرور ریموت و ارائه‌ی گزارشی در مورد کلیه‌ی پورت‌های و پروتکل‌های پشتیبانی شده توسط آنها را دارد.

زمانی که یک سرویس شبکه‌ای شروع به کار می‌کند، معمولا روی یک پورت شبکه شروع به گوش کردن برای هر کسی که به سرویس‌های او نیاز داشته باشد می‌کند. به این پورت‌ها، اصطلاحا پورت‌های باز می‌گویند. nmap می‌تواند پورت‌هایی که منتظر دریافت ارتباطات TCP و UDP هستند را بررسی کرده و نتیجه‌ی آن را به ما گزارش کند.

nmap به صورت پیش‌فرض روی بسیاری از سیستم‌ها نصب نیست، اما در ریپازیتوری اصلی اکثر توزیع‌ها موجود می‌باشد. برای نصب این برنامه در توزیع CentOS، کافی است دستور زیر را وارد کنیم:

```
[root@localhost ~]# yum install nmap
[...]
Installed:
  nmap.x86_64 2:6.40-19.el7

Complete!
```

بیایید با استفاده از nmap، کلیه‌ی پورت‌های TCP باز موجود در سیستم خودمان را بررسی کنیم. برای این کار، از دستور زیر استفاده می‌کنیم:

```
[root@localhost ~]# nmap -sT 127.0.0.1
Starting Nmap 6.40 ( http://nmap.org ) at 2021-04-13 11:42 +0430
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0017s latency).
Not shown: 998 closed ports
PORT     STATE SERVICE
22/tcp   open  ssh
3306/tcp open  mysql

Nmap done: 1 IP address (1 host up) scanned in 0.13 seconds
```

همانطور که می‌بینید، با وارد کردن دستور nmap با آپشن sT-، اقدام به اسکن پورت‌های TCP سیستم خود کردیم. آپشن s- به nmap می‌گوید که عملیات اسکن را انجام دهد و آپشن T- به nmap می‌گوید که اسکن خود را روی پورت‌های TCP انجام دهد. پس از مشخص کردن آپشن‌ها، ما آدرس آی‌پی 127.0.0.1 را وارد کردیم. این آدرس، که آدرس لوپ‌بک نام دارد، اشاره به سیستم خود ما دارد.

همانطور که می‌بینید پس از اجرای اسکن، nmap گزارش خود را در سه ستون PORT، STATE و SERVICE به ما نشان داد. در اولین ردیف زیر این ستون‌ها، نوشته‌ای نظیر زیر مشاهده می‌کنیم:

```
22/tcp   open  ssh
```

این یعنی پورت ۲۲ در سیستم ما باز بوده و منتظر دریافت ارتباطات TCP به سرویس OpenSSH می‌باشد. ما می‌توانیم با استفاده از nmap، وضعیت پورت‌های باز و سرویس‌های ارائه شده توسط سایر سیستم‌ها را نیز بررسی کنیم. برای این کار، کافی است IP سیستم مورد نظر را به جای آی‌پی 127.0.0.1 قرار دهیم. برای مثال:

```
[root@localhost ~]# nmap -sT 192.168.1.1
Starting Nmap 6.40 ( http://nmap.org ) at 2021-04-14 10:20 +0430
Nmap scan report for 192.168.1.1
Host is up (0.86s latency).
Not shown: 992 closed ports
PORT      STATE SERVICE
21/tcp    open  ftp
23/tcp    open  telnet
80/tcp    open  http
20005/tcp open  btx
49152/tcp open  unknown
MAC Address: 64:70:02:FD:53:BE (Tp-link Technologies CO.)

Nmap done: 1 IP address (1 host up) scanned in 1.86 seconds
```

همانطور که می‌بینید، ما سیستم دارای آدرس آی‌پی 192.168.1.1 را اسکن کردیم و کلیه‌ی پورت‌های TCP باز آن را بررسی کردیم. این امر بسیار کاربردی می‌باشد، چرا که از طریق آن، علاوه بر پیدا کردن پورت‌های باز، به طور غیر مستقیم پورت‌هایی که باز نیستند و توسط فایروال آن سیستم بسته شده‌اند را نیز پیدا می‌کنیم.

برای این که بازرسی ما از یک سیستم کامل شود، لازم است پورت‌های UDP باز در سیستم را نیز اسکن کنیم.

برای این کار به جای استفاده از آپشن sT-، از آپشن sU- استفاده می‌کنیم. برای مثال:

```
[root@localhost ~]# nmap -sU 192.168.1.1
Starting Nmap 6.40 ( http://nmap.org ) at 2021-04-14 11:01 +0430
Nmap scan report for 192.168.1.1
Host is up (0.0014s latency).
Not shown: 993 closed ports
PORT      STATE         SERVICE
53/udp    open          domain
137/udp   open          netbios-ns
MAC Address: 64:70:02:FD:53:BE (Tp-link Technologies CO.)

Nmap done: 1 IP address (1 host up) scanned in 1064.56 seconds
```

نکته: خارج از شبکه‌ی خانگی خود، هیچ گاه ابزار nmap را بدون دریافت اجازه اجرا نکنید، چرا که این کار می‌تواند موجب پیگرد قانونی گردد.

شناسایی پورت‌های باز با استفاده از *netstat*

ابزار دیگری که از آن برای بازرسی پورت‌ها استفاده می‌شود، netstat نام دارد. بر خلاف nmap، این ابزار فقط می‌تواند پورت‌های باز روی سیستم خودمان را شناسایی کند. با این که این ابزار امروزه منسوخ شده است و دیگر در بسیاری از سیستم‌ها نصب نیست، آشنایی با آن برای کار کردن با سیستم‌های قدیمی‌تر، بسیار مهم می‌باشد.

برای نصب این ابزار در توزیع CentOS، از دستور زیر استفاده می‌کنیم:

```
[root@localhost ~]# yum install net-tools
[...]
Complete!
```

ابزار netstat بسیار انعطاف‌پذیر می‌باشد و به همین دلیل، پیچیدگی‌هایی نیز دارد. این ابزار آپشن‌های بسیار زیادی دارد که به ما امکان می‌دهد کاری کنیم که به صورت جزئی و ریز، اطلاعات مورد نظر ما را در اختیارمان قرار دهد.

برای مثال، ما می‌توانیم با استفاده از آپشن tcp-- و listening-- (یا به صورت مخفف، tl-)، کلیه‌ی پورت‌هایی که در حال گوش کردن برای دریافت ارتباطات TCP هستند را مشاهده کنیم:

```
[root@localhost ~]# netstat --tcp --listening
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State
tcp        0      0 0.0.0.0:mysql           0.0.0.0:*               LISTEN
tcp        0      0 0.0.0.0:ssh             0.0.0.0:*               LISTEN
tcp6       0      0 [::]:ssh                [::]:*                  LISTEN
```

همانطور که می‌بینید این دستور به صورت پیش‌فرض، هم اطلاعات مربوط به پورت‌های باز IPv4 و هم اطلاعات مربوط به پورت‌های باز IPv6 را به ما نشان می‌دهد. در ستون Local Address، پس از علامت دونقطه (:)، نام سرویسی که روی آن پورت در حال گوش کردن می‌باشد به ما نشان داده می‌شود. این

نام از فایل /etc/services می‌آید. اگر مقداری برای یک پورت داخل فایل /etc/services وجود نداشته باشد، فقط شماره‌ی پورت نشان داده می‌شود.

برای بازرسی پورت‌های باز UDP، به جای آپشن tcp- - از آپشن udp- - استفاده می‌کنیم.

نکته: سوکت‌ها و پورت‌ها ساختارهای مهمی در سیستم‌های لینوکسی هستند و درک تفاوت بین آنها ما را در فرآیند بازرسی سیستم، یاری می‌دهد.

پورت، شماره‌ای است که توسط پروتکل‌هایی نظیر UDP و TCP جهت شناسایی سرویس یا اپلیکیشنی که در حال ارسال و دریافت اطلاعات می‌باشد استفاده می‌شود. برای مثال، پورت ۲۲ یک شماره پورت شناخته شده می‌باشد که مخصوص سرویس OpenSSH است. بسته‌های UDP و TCP در هدر خود، هم شماره‌ی پورت نرم‌افزار ارسال کننده‌ی اطلاعات و هم شماره پورت نرم‌افزار دریافت کننده‌ی اطلاعات را دارند.

سوکت شبکه، یکی از دو نقطه پایانی موجود در یک ارتباط شبکه‌ای می‌باشد. این نقطه‌ی پایانی، در سیستم کاربر وجود داشته و به یک پورت مخصوص، متصل می‌باشد. بنابراین سوکت شبکه‌ای هم از آدرس آی‌پی (سیستم کاربر) و هم از یک شماره‌ی پورت استفاده می‌کند.

به طور کلی، می‌توان گفت که تفاوت پورت و سوکت در این است که سوکت، ترکیب آدرس آی‌پی و شماره‌ی پورت می‌باشد، در حالی که پورت، عددی است که به عنوان شناسه‌ی یک سرویس خاص در شبکه عمل می‌کند. بدین ترتیب، از سوکت برای شناسایی یک سرویس و همچنین سیستمی که آن سرویس روی آن اجرا شده استفاده می‌شود و می‌تواند در هر سیستم متفاوت باشد، در حالی که شماره‌ی پورت، در همه‌ی سیستم‌هایی که از یک سرویس استفاده می‌کنند، یکسان می‌باشد (یا به طور دقیق‌تر، می‌تواند یکسان باشد).

بازرسی سوکت‌های شبکه با ss

ابزار ss، جایگزین نرم‌افزار منسوخ شده‌ی netstat شده است و در اکثر توزیع‌های جدید، به صورت پیش‌فرض نصب می‌باشد. خوبی این نرم‌افزار این است که آپشن‌های آن، شبیه به آپشن‌های netstat می‌باشد و ما مجبور به یادگیری یک سری آپشن جدید نیستیم.

برای مثال، جهت مشاهده‌ی پورت‌های (و سوکت‌های) TCP که در سیستم ما باز هستند، می‌توانیم از آپشن lt-، دقیقا مثل netstat، استفاده کنیم، یا برای مشاهده‌ی پورت‌های (و سوکت‌های) UDP، به سراغ آپشن lu- می‌رویم. بیایید کلیه‌ی پورت‌های باز TCP و UDP موجود در سیستم را با استفاده از ss مشاهده کنیم. برای این کار، کافی است از آپشن ltu- استفاده کنیم. به صورت زیر:

```
[root@localhost ~]# ss -ltu
Netid  State      Recv-Q Send-Q  Local Address:Port                 Peer Address:Port
udp    UNCONN     0      0              *:60875                           *:*
udp    UNCONN     0      0      127.0.0.1:323                             *:*
udp    UNCONN     0      0          [::1]:323                          [::]:*
tcp    LISTEN     0      50             *:mysql                           *:*
tcp    LISTEN     0      128            *:ssh                             *:*
tcp    LISTEN     0      128         [::]:ssh                          [::]:*
```

بازررسی سوکت‌های شبکه با *systemd.socket*

یکی از روش‌های جدیدتر برای بررسی سوکت‌های شبکه‌ای (و همچنین انواع دیگر سوکت‌ها)، استفاده از systemd می‌باشد. systemd از یک روش ویژه برای فعال‌سازی سوکت‌ها استفاده می‌کند که به ما امکان می‌دهد تا بتوانیم سوکت‌ها را به صورت موازی و فارغ از سرویس متصل به آن سوکت استارت بزنیم. این

امر باعث سریع‌تر شدن پروسه‌ی استارت سوکت‌ها می‌شود و همچنین، یک محل جدید برای بازرسی سرویس‌های شبکه‌ای می‌باشد.

ما می‌توانیم از چندین روش برای بازرسی سوکت‌های شبکه‌ای که توسط systemd مدیریت می‌شوند استفاده کنیم. ساده‌ترین روش، استفاده از ابزار `systemctl` می‌باشد. برای مثال، ما می‌توانیم با استفاده از فرمان `list-sockets` و آپشن `--all` - این دستور، لیستی از کلیه‌ی سوکت‌های فعال و غیرفعال مدیریت شده توسط systemd را مشاهده کنیم:

```
[root@localhost ~]# systemctl list-sockets --all
LISTEN                          UNIT                           ACTIVATES
/dev/log                        systemd-journald.socket        systemd-journald.service
/run/dbus/system_bus_socket     dbus.socket                    dbus.service
/run/dmeventd-client            dm-event.socket                dm-event.service
/run/dmeventd-server            dm-event.socket                dm-event.service
/run/lvm/lvmetad.socket         lvm2-lvmetad.socket            lvm2-lvmetad.service
/run/lvm/lvmpolld.socket        lvm2-lvmpolld.socket           lvm2-lvmpolld.service
/run/systemd/initctl/fifo       systemd-initctl.socket         systemd-initctl.service
/run/systemd/journal/socket     systemd-journald.socket        systemd-journald.service
/run/systemd/journal/stdout     systemd-journald.socket        systemd-journald.service
/run/systemd/journal/syslog     syslog.socket                  syslog.service
/run/systemd/shutdownd          systemd-shutdownd.socket       systemd-shutdownd.service
/run/udev/control               systemd-udevd-control.socket   systemd-udevd.service
/var/run/rpcbind.sock           rpcbind.socket                 rpcbind.service
[::]:22                         sshd.socket
kobject-uevent 1                systemd-udevd-kernel.socket    systemd-udevd.service

15 sockets listed.
```

همانطور که می‌بینید در خروجی این دستور، به جز سوکت‌های شبکه‌ای، انواع دیگری از سوکت‌ها را نیز مشاهده می‌کنیم. برای پیدا کردن سوکت‌های شبکه‌ای، باید به نام یک سرویس (یا یونیت)، مثل sshd، و یا شماره‌ی پورت، مثل ۲۲، نگاه کنیم.

برای پیدا کردن فایل‌های تنظیمات هر سوکت شبکه‌ای systemd، باید از فرمان متفاوتی استفاده کنیم. به صورت زیر:

```
[root@localhost ~]# systemctl list-unit-files --type=socket
UNIT FILE                   STATE
dbus.socket                 static
dm-event.socket             enabled
lvm2-lvmetad.socket         enabled
lvm2-lvmpolld.socket        enabled
rpcbind.socket              enabled
rsyncd.socket               disabled
sshd.socket                 disabled
syslog.socket               static
systemd-initctl.socket      static
systemd-journald.socket     static
systemd-shutdownd.socket    static
systemd-udevd-control.socket static
systemd-udevd-kernel.socket  static

13 unit files listed.
```

یکی از نکات بسیار مناسب در مورد خروجی این دستور، مشاهده‌ی فعال یا عدم فعال بودن یک سوکت شبکه‌ای در systemd می‌باشد. غیر فعال بودن یک سوکت (وجود مقدار `disabled` در ستون STATE) به معنای غیر فعال بودن این سرویس نیست، بلکه به این معناست که systemd این سوکت شبکه‌ای را مدیریت نمی‌کند. برای مثال، همانطور که در بالا مشاهده می‌کنید، سوکت `rpcbind` وضعیت `enabled` را دارد،

بدین معنی که توسط systemd مدیریت می‌شود و سوکت sshd، وضعیت disabled را دارد، که بدین معنی است که این سوکت توسط systemd مدیریت نمی‌شود.

برای این که بازرسی ما کامل شود، باید به محتویات یونیت فایل هر سوکت enabled نیز نگاه بیاندازیم. این کار با دستور systemctl cat به علاوه‌ی اسم یونیت فایل سوکت قابل انجام می‌باشد. برای کسب اطلاعات بیشتر در مورد مدیریت سوکت‌ها در systemd، به manpage مربوط به systemd.socket مراجعه کنید.

فایل‌های باز با *lsof* و *fuser*

علاوه بر بازرسی پورت‌ها و سوکت‌ها، ما می‌توانیم با بازرسی فایل‌هایی که در حال حاضر باز هستند، سرویس‌های شبکه‌ای موجود در سیستم را بازرسی کنیم. دستور lsof، لیستی از فایل‌هایی که اکنون در سیستم باز هستند را به ما نشان می‌دهد. از آنجایی که لینوکس ارتباطات شبکه‌ای و سوکت‌ها را به عنوان یک سری فایل می‌بیند، آنها نیز در خروجی lsof نشان داده می‌شوند.

نکته: دستور lsof به صورت پیش‌فرض در توزیع CentOS نصب نیست و ما باید آن را با استفاده از دستور yum install lsof نصب کنیم.

اگر دستور lsof را اجرا کنید، خواهید دید که اطلاعات بسیار زیادی را در خروجی به ما نشان می‌دهد، به طوری که بررسی همه‌ی آنها تقریبا غیرممکن می‌باشد. بهتر است خروجی این دستور را با استفاده از آپشن‌های این دستور، فیلتر کنیم تا فقط اطلاعات مورد نیاز به ما نشان داده شوند. برای مثال، اگر بخواهیم فقط سوکت‌ها و ارتباطات UDP را بازرسی کنیم، از آپشن iUDP- این دستور استفاده می‌کنیم. برای مثال:

```
[root@localhost ~]# lsof -iUDP
COMMAND   PID   USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
chronyd   716 chrony    5u  IPv4  18792      0t0  UDP localhost:323
chronyd   716 chrony    6u  IPv6  18793      0t0  UDP localhost:323
rsyslogd 1296   root   17u  IPv4 200908      0t0  UDP *:50611
rpcbind  9653    rpc    6u  IPv4 205064      0t0  UDP *:sunrpc
rpcbind  9653    rpc    7u  IPv4 205065      0t0  UDP *:924
rpcbind  9653    rpc    9u  IPv6 205067      0t0  UDP *:sunrpc
rpcbind  9653    rpc   10u  IPv6 205068      0t0  UDP *:924
```

همانطور که می‌بینید، استفاده از این آپشن سبب شد که در خروجی فقط اطلاعات مربوط به ارتباطات و سوکت‌های UDP مربوط به IPv4 و IPv6 به ما نشان داده شود.

به همین شکل، ما می‌توانیم با استفاده از آپشن iTCP-، سوکت‌ها و ارتباطات TCP موجود در سیستم را مشاهده کنیم.

ما می‌توانیم خروجی را فراتر از این فیلتر کرده و فقط ارتباطات TCP که در وضعیت Listening و در حال گوش کردن برای دریافت ارتباطات جدید هستند را مشاهده کنیم. برای این کار، به آپشن iTCP-، آپشن sTCP:LISTEN- را اضافه می‌کنیم:

```
[root@localhost ~]# lsof -iTCP -sTCP:LISTEN
COMMAND  PID  USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
sshd    1295  root    3u  IPv4  22902      0t0  TCP *:ssh (LISTEN)
sshd    1295  root    4u  IPv6  22904      0t0  TCP *:ssh (LISTEN)
mysqld  1536 mysql   14u  IPv4  23027      0t0  TCP *:mysql (LISTEN)
rpcbind 9653   rpc    8u  IPv4 205066      0t0  TCP *:sunrpc (LISTEN)
rpcbind 9653   rpc   11u  IPv6 205069      0t0  TCP *:sunrpc (LISTEN)
```

علاوه بر این، می‌توانیم از lsof بخواهیم که فقط اطلاعات مربوط به یک پورت خاص TCP یا UDP را در خروجی به ما نشان دهد. برای این کار از آپشن i- و *PROTOCOL:PORT* استفاده می‌کنیم، به طوری که *PROTOCOL* نشان دهنده‌ی پروتکل مورد نظر (TCP یا UDP) و *PORT* نشان دهنده‌ی شماره‌ی پورت مورد نظر می‌باشد. برای مثال:

```
[root@localhost ~]# lsof -i tcp:22
COMMAND  PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
sshd    1295 root    3u  IPv4  22902      0t0  TCP *:ssh (LISTEN)
sshd    1295 root    4u  IPv6  22904      0t0  TCP *:ssh (LISTEN)
```

بازرسی پراسس‌های استفاده کننده از یک فایل (سوکت یا پروتکل) با *fuser*

ابزار دیگری که به ما قابلیت بازرسی فراتر سیستم را می‌دهد، دستور fuser می‌باشد. این ابزار شماره‌ی پراسس یک برنامه یا کاربری (PID) که از یک پروتکل و پورت خاص استفاده می‌کند را در خروجی به ما نشان می‌دهد. برای مثال، اگر بخواهیم کلیه‌ی پراسس‌هایی که از پروتکل TCP روی پورت ۲۲ استفاده می‌کنند را مشاهده کنیم، از این دستور به صورت زیر استفاده می‌کنیم:

```
[root@localhost ~]# fuser -vn tcp 22
                     USER        PID ACCESS COMMAND
22/tcp:              root       1295 F.... sshd
                     root       9376 F.... sshd
```

آپشن v-، باعث می‌شود که fuser اطلاعات بیشتری را به ما نشان دهد و آپشن n- به ما اجازه می‌دهد که نام پروتکل و پس از آن، شماره‌ی پورت مورد نظر را وارد کنیم.

نکته: در پروسه‌ی بازرسی سرویس‌ها، می‌توانیم از ابزارهای سیستم‌های راه‌انداز نیز استفاده کنیم. در توزیع‌هایی که از سیستم راه‌انداز systemd استفاده می‌کنند، می‌توانیم از دستور زیر برای مشاهده‌ی لیستی از کلیه‌ی سرویس‌هایی که هنگام بوت سیستم اجرا می‌شوند (سرویس‌هایی که دارای وضعیت Enabled هستند) استفاده کنیم:

```
systemctl list-unit-files -t service | grep enabled
```

در توزیع‌هایی که از سیستم راه‌انداز SysV استفاده می‌کنند، می‌توانیم فرآیند بازرسی سرویس‌ها را از دایرکتوری /etc/init.d/ شروع کنیم. هر اسکریپت موجود در این دایرکتوری، نشان دهنده‌ی یکی از سرویس‌هایی می‌باشد که توسط SysV مدیریت می‌شوند. علاوه بر این، می‌توانیم با استفاده از دستور chkconfig --list، سرویس‌های موجود در سیستم و ران‌تایم‌هایی که این سرویس‌ها در آن فعال خواهند بود را مشاهده کنیم. در توزیع‌های خیلی قدیمی‌تر، فایل /etc/inittab نیز ممکن است اطلاعاتی در مورد سرویس‌ها را درون خود داشته باشد.

غیر فعال کردن سرویس‌ها

پس از این که از طریق فرآیند بازرسی، سرویس‌هایی که روی سیستم در حال اجرا هستند را پیدا کردیم، باید لیستی از سرویس‌هایی که به آنها نیاز داریم را ایجاد کرده و سپس کلیه‌ی سرویس‌هایی که در آن لیست قرار ندارند را غیرفعال کنیم. روش مورد استفاده برای غیرفعال کردن سرویس‌ها، بستگی به سیستم راه‌انداز توزیع ما دارد.

در سیستم‌هایی که از سیستم راه‌انداز systemd استفاده می‌کنند، ابتدا باید سرویس را در صورت اجرا بودن، استاپ کنیم. این کار را با اجرای دستور زیر انجام می‌دهیم:

```
systemctl stop SERVICE-NAME
```

برای اطمینان از استاپ‌شدن سرویس، از دستور زیر استفاده می‌کنیم:

```
systemctl status SERVICE-NAME
```

به طوری که *SERVICE-NAME* نشان دهنده‌ی نام سرویسی است که قصد استاپ کردن آن را داریم.

پس از استاپ کردن سرویس، باید حتما آن را غیرفعال کنیم. این کار باعث می‌شود که در صورت ریبوت شدن سیستم، این سرویس دوباره خود را استارت نزند. برای غیر فعال کردن یک سرویس، از دستور زیر استفاده می‌کنیم:

```
systemctl disable SERVICE-NAME
```

در نهایت، برای اطمینان از غیرفعال شدن سرویس، دستور زیر را اجرا می‌کنیم:

```
systemctl is-enabled SERVICE-NAME
```

اگر سرویس به درستی غیرفعال شده باشد، در خروجی این دستور، عبارت `disabled` را مشاهده خواهیم کرد.

در سیستم‌هایی که از سیستم راه‌انداز SysV استفاده می‌کنند، باید ابتدا سرویس مورد نظر را در صورت اجرا بودن، با دستور زیر استاپ کنیم:

```
service SERVICE-NAME stop
```

برای اطمینان از استاپ‌شدن سرویس، از دستور زیر استفاده می‌کنیم:

```
service SERVICE-NAME status
```

پس از استاپ کردن سرویس، باید سرویس را غیرفعال کنیم. در توزیع‌های Red Hat-based، این کار را با دستور زیر انجام می‌پذیرد:

```
chkconfig SERVICE-NAME off
```

برای اطمینان از غیرفعال شدن سرویس، دستور زیر را اجرا می‌کنیم:

```
chkconfig --list SERVICE-NAME
```

اگر سرویس به درستی غیرفعال شده باشد، باید سرویس در همه‌ی ران‌لول‌ها، مقدار `off` را داشته باشد.

نکته: اگر به سرویسی احتیاج ندارید، بهتر است فراتر از غیرفعال کردن آن رفته و آن را به صورت کامل از روی سیستم پاک کنید.

اعمال محدودیت‌ها با استفاده Super Server

زمانی که یک سرویس شبکه‌ای، مثل `chronyd` (در جلسه‌ی ۱۱ در مورد آن صحبت کردیم) استارت می‌شود، یک پورت روی سیستم باز می‌کند. هر پورت یک شماره دارد؛ می‌توان به این شماره، به عنوان شماره‌ی شناسایی یک سرویس شبکه‌ای نگاه کرد. سرویس‌ها در شبکه منتظر بسته‌هایی هستند که شماره‌ی پورت آنها را داشته باشد تا بتوانند بسته را دریافت کرده و آن را پردازش کنند. هنگامی که یک سرویس منتظر دریافت بسته‌های دارای پورت خودش می‌باشد، در وضعیت Listening قرار دارد. زمانی که یک سرویس در این وضعیت باشد، اصطلاحا می‌گوییم که سرویس در حال گوش کردن روی یک شماره پورت می‌باشد.

برخی از سرویس‌ها به جای این که روی یک پوت خاص گوش کنند، از چیزی به اسم Super Server به عنوان محافظ استفاده می‌کنند. در چنین حالتی، به جای این که خود سرویس روی یک پورت خاص گوش کند،

سوپرسرور این کار را انجام می‌دهد. زمانی که یک بسته با شماره‌ی پورت خاص وارد سیستم شود، سوپرسرور پس از یک پردازش اولیه، سرویسی که بسته به آن تعلق دارد را استارت زده و بسته را به او تحویل می‌دهد. استفاده از این روش سبب می‌شود که سیستم خیلی سریع‌تر بوت شود، چرا که سرویس‌های شبکه‌ای فقط موقعی استارت می‌شوند که بسته‌ای برای آنها ارسال شود. علاوه بر این، این روش به ما امکان می‌دهد که برخی از اقدامات کنترلی، نظیر محدود کردن میزان استفاده از یک سرویس را نیز پیاده‌سازی کنیم.

تنظیم *xinetd*

اولین سوپرسرور موجود در لینوکس، `inetd` نام داشت. `xinetd`، جایگزین امن‌تر `inetd` می‌باشد. فایل اصلی تنظیمات `xinetd` در موقعیت `/etc/xinetd.conf` قرار دارد. این فایل، معمولا تنظیمات گلوبالِ `default` را درون خود دارد. بیایید نگاهی به این فایل بیاندازیم:

نکته: `xinetd` به صورت پیش‌فرض در توزیع CentOS نصب نمی‌باشد و باید آن را با استفاده از دستور `yum install xinetd` روی سیستم خود نصب کنید.

```
[root@localhost ~]# cat /etc/xinetd.conf
[..]
defaults
{
# The next two items are intended to be a quick access place to
# temporarily enable or disable services.
#
#       enabled          =
#       disabled =

# Define general logging characteristics.
        log_type = SYSLOG daemon info
        log_on_failure   = HOST
        log_on_success   = PID HOST DURATION EXIT

# Define access restriction defaults
#
#       no_access        =
#       only_from        =
#       max_load = 0
        cps              = 50 10
        instances        = 50
        per_source       = 10

# Address and networking defaults
#
#       bind             =
#       mdns             = yes
        v6only           = no

# setup environmental attributes
#
#       passenv          =
        groups           = yes
        umask            = 002

# Generally, banners are not used. This sets up their global defaults
#
#       banner           =
#       banner_fail      =
#       banner_success   =
}

includedir /etc/xinetd.d
```

در این فایل، هر خطی که با یک علامت # شروع شده باشد، کامنت است و غیرفعال می‌باشد. اگر به آخرین خط موجود در این فایل نگاه کنیم، دستورالعمل `includedir /etc/xinet.d` را مشاهده می‌کنیم. این

دستورالعمل به سوپرسرور xinetd می‌گوید که مدیریت کلیه‌ی سرویس‌هایی که فایل تنظیماتشان در دایرکتوری /etc/xinetd.d قرارگرفته را به عهده گیرد.

برخی از مهم‌ترین دستورالعمل‌های xinetd به شرح زیر می‌باشند:

جدول ۱- مهم‌ترین دستورالعمل‌های قابل استفاده در /etc/xinetd.conf

دستورالعمل	عملکرد
cps	نرخ ماکزیمم ارتباطات وارد شونده به یک سرویس شبکه‌ای را مشخص می‌کند. این دستورالعمل دو عدد از ما می‌گیرد، به طوری که عدد اول نشان دهنده‌ی تعداد ارتباطاتی است که باعث می‌شود سرویس به حالت Pause برود و عدد دوم، نشان دهنده‌ی تعداد ثانیه‌هایی است که سرور باید در حالت Pause باقی بماند.
instances	ماکزیمم تعداد پراسس‌هایی که xinetd می‌تواند برای یک سرویس استارت بزند را مشخص می‌کند. مقدار UNLIMITED برای این دستوراالعمل، هیچ محدودیتی روی تعداد پراسس‌هایی که xinetd می‌تواند استارت بزند قرار نمی‌دهد.
logtype	مشخص می‌کند که لاگ‌ها به کجا ارسال می‌شوند. اگر مقدار این دستورالعمل SYSLOG باشد، لاگ‌ها به برنامه‌ی syslog سیستم ارسال می‌شوند. در صورت انتخاب این مقدار برای این دستورالعمل، باید facility و severity لاگ‌های ارسال شده به برنامه‌ی syslog را نیز مشخص کنیم. اگر مقدار این دستورالعمل برابر با FILE به علاوه‌ی نام یک فایل باشد، کلیه‌ی لاگ‌ها در انتهای فایل مشخص شده نوشته می‌شوند.
log_on_failure	مشخص می‌کند که در صورت عدم موفقیت در استارت پراسس یک سرویس، چه اطلاعاتی (به جز ID یک سرویس)، باید در لاگ‌ها نوشته شود. ما می‌توانیم به این دستورالعمل یک یا ترکیبی از سه مقدار HOST، USERID و ATTEMPT را بدهیم.
log_on_success	مشخص می‌کند که در صورت اجرای موفق یک پراسس از یک سرویس و همچنین خروج موفق از آن، چه اطلاعاتی در لاگ‌ها نوشته شود. ما می‌توانیم به این دستورالعمل یک یا ترکیبی از مقادیر PID، HOST، USERID، EXIT، DURATION و TRAFFIC را بدهیم.
max_load	میانگین یک‌دقیقه‌ای بار مجاز موجود روی یک سرویس را مشخص می‌کند. اگر بار موجود روی سرویس بیش از این مقدار شود، سرویس تا زمان کم شدن میزان بار به هیچ ارتباطی پاسخ نمی‌دهد.
no_access	آدرس هاست‌هایی که نباید به آنها سرویس‌دهی شود را مشخص می‌کند.
only_from	آدرس هاست‌هایی که فقط به آنها باید سرویس‌دهی شود را مشخص می‌کند.

per_source	مشخص می‌کند که برای هر آدرس IP، چه تعداد پراسس برای هر سرویس می‌تواند استارت زده شود.

پیکربندی سرویس‌ها در *xinetd*

همانطور که گفتیم تنظیمات موجود در فایل /etc/xinetd.conf، تنظیمات پیش‌فرض می‌باشند. ما می‌توانیم هر کدام از دستورالعمل‌های موجود در این فایل را در فایل‌های مربوط به پیکربندی هر سرویس، تغییر دهیم. بدین شکل، هر سرویس تنظیمات و دستورالعمل‌های مخصوص به خود را خواهد داشت. فایل تنظیمات مربوط به هر سرویس، در دایرکتوری /etc/xinetd.d قرار می‌گیرد.

فایل تنظیمات یک سرویس در دایرکتوری /etc/xinetd.d معمولا نمایی شبیه زیر دارد:

```
# This is the configuration for the tcp/stream echo service.
service echo
{
# This is for quick on or off of the service
       disable = yes
# The next attributes are mandatory for all services
       id = echo-stream
       type = INTERNAL
       wait = no
       socket_type = stream
# protocol = socket type is usually enough
[…]
# Logging options
#      log_type =
#      log_on_success =
#      log_on_failure =
[…]
# Access restrictions
#      only_from =
#      no_access =
#      access_times =
#      cps = 50 10
#      instances = UNLIMITED
#      per_source = UNLIMITED
#      max_load = 0
[…]
}
```

یکی از مهم‌ترین دستورالعمل‌ها در فایل تنظیمات مربوط به هر سرویس، دستورالعمل disable می‌باشد. اگر این دستورالعمل مقدار yes را داشته باشد، سرویس غیرفعال خواهد بود، اما اگر این دستورالعمل مقدار no را داشته باشد، سرویس فعال خواهد بود. پیشنهاد می‌شود که در صورت استفاده از xinetd، مقدار دستورالعمل disabled موجود در فایل‌های تنظیمات مربوط به همه‌ی سرویس‌ها بررسی کرده تا سرویس‌هایی که باید غیرفعال باشند، به اشتباه فعال نشده باشند.

توجه داشته باشید که تعدادی دستورالعمل اجباری وجود دارد که باید در فایل تنظیمات مربوط به هر سرویس وجود داشته باشد. این دستورالعمل‌ها id، type، wait و socket type می‌باشند. همانطور که می‌بینید، در این فایل بسیاری از دستورالعمل‌هایی که در فایل xinetd.conf موجود بودند و ما در جدول ۱ در مورد آنها صحبت کردیم نیز وجود دارند. در صورتی که به این دستورالعمل‌ها مقدار دهیم و علامت # را از ابتدای آنها برداریم، این دستورالعمل‌ها، جایگزین دستورالعمل‌های پیش‌فرض برای این سرویس خاص می‌شوند.

پس از انجام تغییرات، اضافه یا حذف کردن تنظیمات به فایل xinetd.conf یا فایل‌های موجود در /etc/xinetd.d، باید سرویس xinetd را restart کنیم تا تغییرات جدید اعمال شوند.

اعمال محدودیت با استفاده از TCP Wrapperها

TCP Wrapperها یک روش قدیمی برای کنترل دسترسی به سرویس‌های شبکه‌ای می‌باشند. سرویس‌هایی که می‌توانند از TCP Wrapperها استفاده کنند، لایبرری libwrap را همراه خود دارند. ما می‌توانیم وجود این لایبرری در یک سرویس را با استفاده از دستور ldd بررسی کنیم. برای مثال:

```
[root@localhost ~]# which sshd
/usr/sbin/sshd
[root@localhost ~]# ldd /usr/sbin/sshd | grep libwrap
      libwrap.so.0 => /lib64/libwrap.so.0 (0x00007f0b6a2be000)
```

همانطور که می‌بینید، سرویس sshd لایبرری libwrap را همراه خود دارد و در نتیجه می‌تواند از TCP Wrapperها استفاده کند.

TCP Wrapperها از دو فایل برای مشخص کردن این که چه کسی می‌تواند به یک سرویس دسترسی داشته باشد استفاده می‌کنند. این دو فایل، /etc/hosts.allow و /etc/hosts.deny نام دارند. همانطور که از نام این فایل‌ها پیداست، فایل /etc/hosts.allow شامل آدرس هاست‌هایی می‌باشد که می‌توانند به سرویس دسترسی داشته باشند و /etc/hosts.deny شامل آدرس هاست‌هایی می‌باشد که اجازه‌ی دسترسی به سرویس را ندارند.

اضافه کردن موارد داخل این دو فایل، از فرمت زیر پیروی می‌کند:

SERVICE: IPADDRESS...

برای مثال:

```
sshd: 192.168.1.1, 10.2.3.8
```

البته ما مجبور نیستیم آدرس‌های آی‌پی را به صورت تکی وارد کنیم و می‌توانیم یک ساب‌نت را به صورت کامل مشخص کنیم. برای مثال، ما کل ساب‌نت 172.243.26.0/24 را به صورت زیر مشخص می‌کنیم (به نقطه‌ی موجود در آخر آدرس توجه کنید):

```
sshd: 172.243.26.
```

ترتیب جستجوی دو فایلِ hosts.allow و hosts.deny توسط سیستم بسیار مهم می‌باشد. زمانی که یک هاست یک درخواست را به یک سرویس ارسال می‌کند، سیستم به صورت زیر عمل می‌کند:

۱- سیستم فایل hosts.allow را به منظور پیدا کردن هاست‌نیم یا آدرس آی‌پی هاست جستجو می‌کند.
   - اگر آدرس هاست در این فایل پیدا شد، سیستم به آن هاست دسترسی می‌دهد و به سراغ بررسی فایل hosts.deny نمی‌رود.

۲- سیستم فایل hosts.deny را به منظور پیدا کردن هاست‌نیم یا آدرس آی‌پی هاست جستجو می‌کند.
   - اگر آدرس هاست در این فایل پیدا شد، سیستم به آن هاست دسترسی نمی‌دهد.
   - اگر آدرس هاست در این فایل پیدا نشد، سیستم به آن هاست دسترسی می‌دهد.

با این حساب، اگر آدرس یک هاست در هیچکدام از این دو فایل قرار نداشته باشد، به آن هاست دسترسی داده می‌شود. به همین دلیل، بهتر است در فایل /etc/hosts.deny، به صورت زیر از وایلدکارد ALL استفاده کنیم:

```
ALL: ALL
```

بدین شکل، به کلیه‌ی هاست‌هایی که آدرسشان در فایل hosts.allow نباشد، دسترسی داده نمی‌شود. لازم به ذکر است که برخی از توزیع‌ها به جای وایلدکارد ALL، از عبارت PARANOID استفاده می‌کنند.

نکته: TCP Wrapperها بسیار قدیمی می‌باشند و زمانی که هنوز استفاده از فایروال مرسوم نبود، استفاده می‌شدند. امروزه استفاده از TCP Wrapperها پیشنهاد نمی‌شود و بهتر است از فایروال استفاده شود.

مدیریت امنیت محلی

یکی از مهم‌ترین بخش‌ها در امنیت سیستم، مدیریت امنیت محلی می‌باشد. مدیریت امنیت محلی شامل موارد ساده‌ای نظیر اجبار وجود رمزهای امن روی همه‌ی سیستم‌ها، قرار دادن یک رمز امن برای کاربر روت و مواردی از این قبیل می‌باشد. علاوه بر این، داشتن اطلاعات در مورد چگونگی پیدا کردن فایل‌های خطرناک نیز بسیار مهم می‌باشد. ما در این بخش به بررسی این مسائل می‌پردازیم.

امنیت پسوردها

ما در مورد پسوردها در جلسه‌ی دهم به صورت کامل صحبت کردیم، اما برخی از مسائل هستند که دانستن آنها، ما را در حفظ امنیت یاری می‌دهد. در این قسمت به برخی از قواعد پایه‌ای که ما را در حفظ امنیت پسوردها یاری می‌دهند می‌پردازیم.

نگاهی به چگونگی ذخیره‌ی پسوردها در لینوکس

اوایل، توزیع‌های لینوکسی پسوردها را در فایل /etc/passwd ذخیره می‌کردند. البته این پسوردها ابتدا Hash شده و سپس در این فایل قرار می‌گرفتند. Hash یک تابع ریاضی یک طرفه می‌باشد که به ازای دریافت یک رشته‌ی معمولی (قابل خواندن و درک توسط انسان)، یک رشته‌ی رمزی (غیرقابل درک توسط انسان) به ما ارائه می‌دهد. همانطور که گفتیم Hash یک تابع یک طرفه می‌باشد، بدین معنی که ما نمی‌توانیم رشته‌ی رمزی شده را به این تابع پس داده تا رشته‌ی اصلی را دریافت کنیم. لینوکس چیزی به اسم Salt نیز به فرآیند Hashing اضافه می‌کند که باعث می‌شود رمز Hash شده امنیت بالاتری داشته باشد.

با این حال، برخی از کاربران مخرب چیزی به اسم Rainbow Tableها ایجاد کرده‌اند که یک دیکشنری پر از رمزهای معمول و مقدار Hash شده‌ی آنها می‌باشد. Rainbow Tableها به ما اجازه می‌دهند که مقدار Hashشده‌ی یک رمز را وارد کرده و در ازای آن، رشته‌ی اصلی را دریافت کنیم.

وجود ابزارهایی نظیر Rainbow Tableها، پسوردهای ذخیره شده در فایل /etc/passwd را تحت خطر قرار می‌دهد. دلیل این امر، مجوزهای فایل /etc/passwd می‌باشد:

```
[root@localhost ~]# ls -l /etc/passwd
-rw-r--r--. 1 root root 1318 Apr 19 11:31 /etc/passwd
```

همانطور که می‌بینید، همه‌ی کاربران اجازه‌ی خواندن این فایل را دارند و در نتیجه، همه‌ی کاربران می‌توانند رمزهای Hash شده‌ی موجود در این فایل را بخوانند و اقدام به پیدا کردن مقدار اصلی پسوورد با استفاده از ابزارهایی نظیر Rainbow Table و... کنند. ما نمی‌توانیم مجوز این فایل را تغییر دهیم، چرا که ابزارهایی نظیر شِل و... از طریق خواندن این فایل، موقعیت Home Directory کاربر و... را پیدا می‌کنند.

به همین دلیل، در توزیع‌های مدرن لینوکس، برای محافظت از رمزهای Hash شده‌ی کاربران، رمزها از فایل /etc/passwd به فایل /etc/shadow منتقل شده‌اند. بیایید نگاهی به مجوزهای این فایل بیاندازیم:

```
[root@localhost ~]# ls -l /etc/shadow
----------. 1 root root 997 Apr 19 11:31 /etc/shadow
```

همانطور که می‌بینید، این فایل مجوز ویژه‌ای دارد که به هیچ کس اجازه‌ی مشاهده‌ی محتویات این فایل را نمی‌دهد (البته کاربر روت می‌تواند این فایل را مشاهده کند). با این که دیگر هیچ توزیع مدرنی پسوردها را در فایل /etc/passwd قرار نمیدهد، اگر روی یک توزیع خیلی قدیمی هستید که هنوز پسوردها را در /etc/passwd ذخیره می‌کند، می‌توانید با استفاده از دستور pwconv، پسووردها را به فایل /etc/shadow منتقل کنید.

نکته: هنگام لاگین کردن در سیستم و وارد کردن رمز خود، سیستم رمز وارد شده توسط ما را گرفته، الگوریتم Hash را اجرا کرده و در طول این فرآیند، Salt را نیز به آن اضافه می‌کند. سپس نتیجه‌ی به دست آمده که رمز Hash شده‌ی ما می‌باشد را با رمز ذخیره شده در /etc/shadow مقایسه می‌کند. اگر این دو مقدار با هم برابر باشند، سیستم به ما اجازه‌ی ورود می‌دهد.

حل کردن مشکلات پسورد

در همه‌ی سیستم‌ها، خواه‌ناخواه برخی از کاربران دچار مشکلاتی در دسترسی به سیستم می‌شوند. خیلی از اوقات ممکن است کاربر با این که به صورت فیزیکی یا ریموت به سیستم دسترسی دارد، نتواند با وارد کردن یوزرنیم و پسورد خود، وارد سیستم شود. در چنین حالتی، ما چندین آپشن برای حل این مشکل خواهیم داشت.

اگر حساب‌کاربری که مشکل ورود به سیستم دارد را به تازگی ایجاد کرده باشیم، احتمال دارد که فراموش کرده باشیم پسوردی برای آن حساب کاربری انتخاب کنیم. اکثر ادمین‌ها با استفاده از دستور useradd اقدام به ایجاد یک حساب‌های کاربری جدید می‌کنند، اما فراموش می‌کنند که با اجرای دستور passwd رمزی روی حساب کاربری ایجاد شده قرار دهند. ما می‌توانیم وجود یا عدم وجود رمز روی یک حساب کاربری را با استفاده از دستور grep یا getent بررسی کنیم:

```
[root@localhost ~]# getent passwd ninja
ninja:x:1235:1235::/home/ninja:/bin/bash
[root@localhost ~]# getent shadow ninja
ninja:!!:18742:0:99999:7:::
```

همانطور که می‌بینید در فیلد دوم مربوط به اطلاعات حساب کاربری ninja در فایل shadow، علامت !! مشاهده می‌شود که به معنای عدم وجود پسورد روی این حساب کاربری می‌باشد.

نکته: خیلی از اوقات، کاربران سایر سیستم‌عامل‌ها نظیر ویندوز، ممکن است ندانند که در سیستم‌های لینوکسی، یورزنیم‌ها حساس به حروف بزرگ و کوچک می‌باشند. پس هنگام بررسی و حل مشکلات کاربر در لاگین کردن، به این امر نیز توجه داشته باشید.

برخی از مشکلات ورود به سیستم، ممکن است به دلیل قفل بودن یک اکانت باشد. ما می‌توانیم قفل بودن یک حساب کاربری را با استفاده از دستور passwd -S یا getent بررسی کنیم. برای مثال:

```
[root@localhost ~]# passwd -S jethro
jethro LK 2021-04-25 0 99999 7 -1 (Password locked.)
```

وجود حروف LK پس از jethro، به معنای قفل بودن حساب کاربری jethro می‌باشد. مشکل در این است که اگر روی یک حساب کاربری رمزی قرار نداده باشیم نیز اجرای دستور passwd -S به ما در خروجی خود دقیقا همین گزارش را می‌دهد. پس بهتر است به سراغ استفاده از getent و فایل shadow برویم:

```
[root@localhost ~]# getent shadow jethro
jethro:!![...]:18742:0:99999:7:::
```

وجود علامت تعجب در ابتدای فیلد دوم، نشان دهنده‌ی این می‌باشد که اکانت jethro قفل می‌باشد. برای این که یک اکانت از حالت قفل درآید، می‌توانیم از دستور passwd -u یا usermod -U به علاوه‌ی نام حساب کاربری مورد نظر، استفاده کنیم.

نکته: یکی دیگر از موارد بسیار معمول در مشکلات ورود به سیستم، مشکلات خارجی می‌باشد. این مشکلات خارجی می‌توانند سخت‌افزاری یا نرم‌افزاری باشند. مثلا ممکن است کلید خاصی روی کیبورد کاربر خراب باشد و در نتیجه، کاراکترهای مورد نظر کاربر را به سیستم ارسال نکند یا ممکن است کاربر به زبان کنونی سیستم هنگام وارد کردن پسورد خود، توجهی نداشته باشد و در نتیجه از کاراکترهای اشتباه حین وارد کردن رمز خود استفاده کند.

یکی دیگر از دلایل وجود مشکل در ورود به سیستم، منقضی شدن حساب کاربری می‌باشد. ما می‌توانیم اطلاعات مربوط به تاریخ انقضای یک حساب کاربری را با استفاده از chage به دست‌آوریم:

```
[root@localhost ~]# date
Sun Apr 25 11:58:38 +0430 2021
[root@localhost ~]# chage -l hatred
[...]
Account expires                 : Apr 20, 2021
[...]
```

همانطور که می‌بینید، این حساب کاربری ۵ روز است که منقضی شده و به همین دلیل، کاربر نمی‌تواند با استفاده از آن وارد سیستم شود. ما می‌توانیم با استفاده از دستور chage -E یک تاریخ انقضای جدید برای این حساب کاربری انتخاب کنیم.

مورد دیگری که بررسی آن خالی از لطف نیست، تاریخ انقضای رمز کاربر می‌باشد. این امر نیز با استفاده از دستور chage -l قابل انجام می‌باشد، اما ما به توضیح فراتر آن نمی‌پردازیم.

محدود کردن دسترسی *root*

ما تا به اینجا به خاطر سادگی کار، همیشه از اکانت root استفاده کرده‌ایم، اما این امر کاری بسیار اشتباه و خطرناک می‌باشد. حساب کاربری root، اجازه‌ی دسترسی به همه‌جای سیستم را دارد و هر کاری را می‌تواند انجام دهد، به همین دلیل باید از این حساب کاربری به صورت ویژه محافظت بکنیم. در سیستم‌های واقعی و غیرآموزشی، باید موارد زیر را برای حفاظت از اکانت root در نظر بگیریم:

- اجازه‌ی لاگین مستقیم به اکانت root را ندهیم.
- به هر حساب کاربری، اجازه‌ی سوئیچ به یک حساب کاربری دیگر را ندهیم.
- روی حساب‌های کاربری موقت، تاریخ انقضا قرار دهیم.
- حساب‌های کاربری که از آنها استفاده‌ای نمی‌شود را حذف کنیم.

تمرکز ما در این قسمت روی حساب کاربری root می‌باشند. بار دیگر تکرار می‌کنیم که استفاده از root، حتی اگر فقط خود شما به root دسترسی دارید، پیشنهاد نمی‌شود. راهکارهای امن‌تری برای دسترسی به بخش‌هایی از سیستم به عنوان root وجود دارد که در این قسمت به آنها می‌پردازیم.

تغییر حساب کاربری با *su*

با استفاده از ابزار su، می‌توانیم به سادگی وارد یک حساب کاربری دیگر شویم (به شرط این که رمز آن حساب را داشته باشیم). برای ورود به حساب کاربری root، از دستور su - استفاده می‌کنیم. پس از این کار، باید رمز root را وارد کرده تا بتوانیم وارد این حساب کاربری شویم.

```
[behnam@localhost]$ su -
Password:
[root@localhost ~]# whoami
root
```

نکته: قرار دادن علامت - پس از دستور su، به su می‌گوید که یک شِل جدید ایجاد کرده و با آن شِل وارد حساب کاربری مشخص شده شود. این امر باعث می‌شود که هنگام تغییر حساب کاربری، محیطی نظیر محیطی که آن کاربر با آن کار می‌کرده را داشته باشیم.

برای ورود به حساب کاربری شخصی به جز root، کافی است پس از علامت -، نام آن حساب کاربری را وارد کنیم. برای مثال:

```
[ninja@localhost ~]$ su - behnam
Password:
[behnam@localhost ~]$ whoami
behnam
```

پس از وارد شدن به یک حساب کاربری دیگر، می‌توانیم هر دستور و عملکرد دیگری را با مجوزهای آن حساب کاربری انجام دهیم. برای خروج از حساب کاربری و بازگشت به حساب کاربری خود، کافی است از دستور exit یا logout استفاده کنیم.

اگر بخواهیم فقط یک دستور را با مجوزهای root اجرا کنیم، می‌توانیم از دستور su -c استفاده کنیم. برای مثال، اگر بخواهیم پسوورد یک حساب کاربری دیگر را عوض کنیم، نیاز به مجوزهای روت داریم. به شرط این که رمز حساب کاربری روت را داشته باشیم، می‌توانیم بدین شکل یک دستور را با مجوزهای روت اجرا کنیم:

```
[ninja@localhost ~]$ whoami
ninja
[ninja@localhost ~]$ su -c "passwd puppy"
Password:
Changing password for user puppy.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
[ninja@localhost ~]$ whoami
ninja
```

همانطور که می‌بینید، پس از وارد کردن دستور su -c “passwd puppy”، سیستم از ما درخواست رمز کرد. در این قسمت، باید رمز اکانت root را وارد کنیم تا دستور passwd puppy با مجوزهای روت اجرا شود. از آنجایی که ما پسورد روت را به درستی وارد کردیم، سیستم دستور passwd puppy را اجرا کرد

و از ما خواست که پسورد جدید حساب کاربری puppy را مشخص و ایجاد کنیم.
نکته‌ای که باید به آن توجه کنیم این است که اگر در دستوری که می‌خواهیم با مجوزهای روت اجرا شود فاصله‌ی خالی وجود داشته باشد، باید آن دستور را بین دو علامت " قرار دهیم ("passwd puppy").

اجرای دستورها با مجوزهای روت با استفاده از *sudo*

با این که استفاده از su -c برای اجرای یک دستور به عنوان روت بسیار کاربردی می‌باشد، اما از نظر امنیتی زیاد مناسب نیست، چرا که هر کس که بخواهد یک دستور را با مجوزهای روت اجرا کند، باید پسوورد روت را داشته باشد. دستور sudo، اقدام به حل این مشکل می‌کند.
دستور sudo، به کاربران اجازه می‌دهد که یک دستور را با مجوزهای روت اجرا کنند. کلیه‌ی دستورهایی که با sudo اجرا می‌شوند لاگ شده و درون یک فایل لاگ ذخیره می‌شوند. این لاگ‌ها دقیقا به ما می‌گویند که چه کاربری در چه ساعتی و مکانی، چه دستوری را با مجوز روت اجرا کرده است.

نکته: ورود به سیستم با استفاده از اکانت روت، یک محیط انکاری ایجاد می‌کند. محیط انکاری، محیطی است که یک فرد می‌تواند کلیه‌ی کارهای انجام شده در آن را انکار کند. بنابراین اگر یک شخص با اکانت روت وارد سیستم شود و سیستم را بهم ریخته یا اطلاعاتی را بدزدد، آن فرد می‌تواند به صورت قانونی مسئولیت انجام این کار را برعهده نگیرد. در سیستم‌هایی که هیچ کس اجازه‌ی ورود به حساب کاربری روت را ندارد، ما یک محیط غیر انکاری داریم. این یعنی که کلیه‌ی عملکردهای کاربر لاگ شده و کاربر نمی‌تواند مسئولیت انجام آن کارها را برعهده نگیرد. sudo نیز یک محیط غیر انکاری ایجاد می‌کند.

فایل تنظیمات sudo در موقعیت /etc/sudoers قرار دارد. بیایید نگاهی به این فایل بیاندازیم:

```
[root@localhost ~]# cat /etc/sudoers
[...]
## This file must be edited with the 'visudo' command.
[..]
## Next comes the main part: which users can run what software on
## which machines (the sudoers file can be shared between multiple
## systems).
## Syntax:
##
##    user   MACHINE=COMMANDS
##
## The COMMANDS section may have other options added to it.
##
## Allow root to run any commands anywhere
root   ALL=(ALL)    ALL

## Allows members of the 'sys' group to run networking, software,
## service management apps and more.
# %sys ALL = NETWORKING, SOFTWARE, SERVICES, STORAGE, DELEGATING, PROCESSES, LOCATE,
DRIVERS

## Allows people in group wheel to run all commands
%wheel ALL=(ALL)    ALL

## Same thing without a password
# %wheel     ALL=(ALL)    NOPASSWD: ALL

## Allows members of the users group to mount and unmount the
## cdrom as root
# %users  ALL=/sbin/mount /mnt/cdrom, /sbin/umount /mnt/cdrom
```

```
## Allows members of the users group to shutdown this system
# %users  localhost=/sbin/shutdown -h now

## Read drop-in files from /etc/sudoers.d (the # here does not mean a comment)
#includedir /etc/sudoers.d
```

بیایید به خطی که با نام root شروع می‌شود دقیق‌تر نگاه کنیم:

```
root    ALL=(ALL)     ALL
```

خط‌های دارای این فرمت، خط‌هایی هستند که مجوزهای دسترسی را مشخص می‌کنند. به طور کلی، برای ارائه‌ی دسترسی به کاربران جهت اجرای دستورات با مجوز سوپریوزر (یا روت)، باید از فرمت زیر استفاده کنیم:

```
USERNAME      HOSTNAME-OF-SYSTEM=(USER:GROUP)      COMMANDS
```

با توجه به این فرمت، می‌توان دید که کاربر root می‌تواند فارغ از هاست‌نیم سیستم (ALL=)، دستورات را به عنوان هر کاربر و هر گروه (ALL:ALL) اجرا کرده و به همه‌ی دستورها نیز دسترسی داشته باشد (ALL).

نکته: شدیدا پیشنهاد می‌شود که هرگز فایل /etc/sudoers را با یک ادیتور استاندارد باز نکنید؛ چرا که اگر چند کاربر به صورت همزمان در حال اعمال تغییرات در این فایل باشند، فایل دچار خرابی شده و سیستم دچار مشکل خواهد شد. دستور visudo، فایل /etc/sudoers را به صورت امن درون یک ادیتور باز می‌کند و به ما اجازه می‌دهد که بدون خطر وقوع خرابی در فایل، تغییرات خود را اعمال کنیم. visudo، مانند ادیتور vi که قبلا با آن آشنا شده‌ایم عمل می‌کند.

برای ساده‌تر کردن ارائه‌ی دسترسی به کاربران، بسیاری از توزیع‌ها یک گروه، مانند sudo (اوبونتو) یا wheel (ردهت) در فایل /etc/sudoers اضافه می‌کنند. موارد مربوط به گروه‌ها در این فایل، با یک علامت درصد (%) شروع می‌شوند. در این حالت، هنگامی که بخواهیم به یک کاربر دسترسی sudo بدهیم، دیگر نیاز نیست که فایل sudoers را تغییر دهیم، بلکه کافی است آن کاربر را به یکی از گروه‌های تعریف شده در این فایل اضافه کنیم. برای مثال:

```
[root@localhost ~]# whoami
root
[root@localhost ~]# usermod -aG wheel behnam
[root@localhost ~]# su - behnam
[behnam@localhost ~]$ whoami
behnam
[behnam@localhost ~]$ groups
behnam wheel
```

پیشنهاد می‌شود که کاربران و گروه‌های جانبی را به جای اضافه کردن در فایل /etc/sudoers، در یک فایل تنظیمات داخل دایرکتوری /etc/sudoers.d قرار دهیم. اگر به انتهای فایل /etc/sudoers توجه کرده باشید، خطی با محتوای #includedir /etc/sudoers.d مشاهده می‌کنید. این خط باعث می‌شود که سیستم کلیه‌ی فایل‌های موجود در /etc/sudoers.d را به عنوان بخشی از تنظیمات sudo بخواند.

پس از انجام صحیح تنظیمات، هر کاربری که دارای مجوز sudo باشد می‌تواند دستور sudo را قبل از کلیه‌ی دستورهایی که نیاز به اجرا شدن با مجوزهای روت دارند، قرار داده و آن دستور را اجرا کند. البته در این حالت، کاربر باید رمز حساب کاربری خود را قبل از اجرای دستور وارد کند. برای مثال:

```
[behnam@localhost ~]$ sudo getent shadow behnam
[sudo] password for behnam:
behnam:80r[...]:18710:0:99999:7:::
```

کلیه‌ی دستورهایی که با sudo اجرا می‌کنیم، توسط سیستم لاگ می‌شوند. بیایید صحت این امر را تایید کنیم:

```
[behnam@localhost ~]$ sudo journalctl -r -n  | grep sudo
Apr 26 13:29:38 localhost.localdomain sudo[45567]:    behnam : TTY=pts/0 ;
PWD=/home/behnam ; USER=root ; COMMAND=/bin/getent shadow Behnam
```

همانطور که می‌بینید، استفاده‌ی ما از sudo و دستوری که از طریق آن اجرا کردیم به شکل کامل در ژورنال لاگ شده است.

بازرسی کاربران دسترسی یافته به سیستم

برخی از ابزارها به ما امکان می‌دهند که کاربرانی که اکنون در حال استفاده از سیستم هستند یا قبلا در حال استفاده از سیستم بوده‌اند را بازرسی کنیم. درا ین بخش، با این ابزارها آشنا می‌شویم.

آشنایی با دستورهای *who* و *w*

با استفاده از دستور who، می‌توانیم اطلاعاتی در مورد حساب کاربری خود و همچنین کلیه‌ی کاربرانی که اکنون در حال کار با سیستم هستند مشاهده کنیم. برای مثال:

```
[behnam@localhost ~]$ who
puppy    tty1         2021-04-27 11:42
root     pts/0        2021-04-27 11:35 (192.168.1.102)
behnam   pts/1        2021-04-27 11:39 (192.168.1.106)
```

همانطور که می‌بینید، وارد کردن این دستور، به ما گفت که چه کاربرانی در حال استفاده از سیستم هستند، از چه ترمینالی استفاده می‌کنند و اگر کاربران ریموت هستند، آی‌پی آنها چیست.

برای این که اطلاعات مربوط به حساب کاربری خودمان را مشاهده کنیم، به صورت زیر دستور who را وارد می‌کنیم:

```
[behnam@localhost ~]$ who am i
behnam   pts/1        2021-04-27 11:39 (192.168.1.106)
```

علاوه بر این، دستور who is great نیز به اطلاعات عمومی شما اضافه می‌کند:

```
[behnam@localhost ~]$ who is great
behnam   pts/1        2021-04-27 11:39 (192.168.1.106)
```

دستور w، بر خلاف اسم کوتاهش، اطلاعات بسیار مناسبی را در اختیار ما قرار می‌دهد. برای مثال:

```
[behnam@localhost ~]$ w
 11:50:42 up 26 min,  3 users,  load average: 0.02, 0.04, 0.05
USER     TTY      FROM             LOGIN@   IDLE   JCPU   PCPU  WHAT
puppy    tty1                      11:42   10.00s  1.96s  1.55s glances
root     pts/0    192.168.1.102    11:35   26.00s  0.66s  0.38s vim
behnam   pts/1    192.168.1.106    11:39    2.00s  0.13s  0.03s w
```

همانطور که می‌بینید، خروجی w جزئیات بسیار بیشتری را در اختیار ما قرار می‌دهد. خط اول موجود در خروجی این دستور، اطلاعات زیر را به ما رائه می‌دهد:

- ساعت
- مدت زمان روشن بودن سیستم

- تعداد کاربرانی که اکنون در حال استفاده از سیستم هستند.
- میانگین بار CPU در ۱ دقیقه، ۵ دقیقه و ۱۵ دقیقه‌ی گذشته

خطوط بعدی، اطلاعاتی در مورد همه‌ی کاربرانی که اکنون در حال استفاده از سیستم هستند به ما نشان می‌دهد. این اطلاعات، در قالب چندین ستون به ما نشان داده می‌شوند. معنای هر ستون به شرح زیر می‌باشد:

- USER: یوزرنیم حساب کاربری
- TTY: ترمینال مورد استفاده توسط کاربر
- FROM: آدرس IP کاربر، در صورت اتصال به سیستم از راه دور
- LOGIN@: زمان ورود کاربر به حساب کاربری خود
- IDLE: زمان سپری شده از آخرین استفاده‌ی کاربر از سیستم
- JCPU: میزان مصرف CPU Time توسط کاربر
- PCPU: میزان CPU Time مصرفی توسط آخرین دستور اجرا شده توسط کاربر
- WHAT: دستوری که کاربر در حال اجرای آن می‌باشد.

ممکن است از خود بپرسید که دستور w این اطلاعات را از کجا می‌آورد. این دستور، اطلاعات نشان داده شده را از فایل `/var/run/utmp` و همچنین دایرکتوری‌های موجود در `/proc` به دست می‌آورد.

نکته: در صورت مشاهده‌ی یک کاربر مشکوک در سیستم، بهتر است ابتدا اکانت آن کاربر را قفل کرده و سپس PID آن کاربر را `kill` کنید تا کاربر مشکوک از سیستم بیرون انداخته شود. وجود کاربر مشکوک در سیستم، نشان دهنده‌ی وجود یک مشکل پایه‌ای در سیستم می‌باشد، پس بهتر است پس از بیرون انداختن کاربر، به هیچکس اجازه‌ی ورود به سیستم را ندهیم تا بتوانیم سیستم را به درستی بازرسی کرده و مشکلات امنیتی خود را پیدا کنیم. برای بستن اجازه‌ی ورود به سیستم، می‌توانیم از فایل `nologin` استفاده کنیم. برای این کار باید از مجوزهای روت استفاده کرده (با استفاده از `sudo`) و دستور `touch /etc/nologin` را وارد کنیم. تا زمانی که این فایل در سیستم وجود داشته باشد، کاربران جدید نمی‌توانند وارد سیستم شوند. ما می‌توانیم داخل این فایل پیامی نیز قرار داده تا کاربرانی که می‌خواهند وارد سیستم شوند، از دلیل عدم امکان ورود به سیستم مطلع شوند.

مشاهده‌ی سوابق دسترسی به سیستم با استفاده از *last*

دستور `last`، با استفاده از اطلاعات موجود در فایل `/var/log/wtmp`، لیستی از کلیه‌ی کاربرانی که قبلا به سیستم دسترسی پیدا کرده‌اند یا اکنون در حال دسترسی به سیستم هستند را به همراه تاریخ و ساعت دقیق، به ما ارائه می‌دهد. بیایید عملکرد این دستور را با هم بررسی کنیم:

```
[behnam@localhost ~]$ last
ninja    pts/2        192.168.1.102    Wed Apr 28 10:14   still logged in
behnam   pts/1        192.168.1.106    Wed Apr 28 10:13   still logged in
jethro   pts/0        192.168.1.102    Wed Apr 28 10:11   still logged in
puppy    tty1                          Tue Apr 27 11:42   still logged in
behnam   pts/1        192.168.1.106    Tue Apr 27 11:39 - 16:42  (05:03)
jethro   pts/0        192.168.1.102    Tue Apr 27 11:35 - 16:42  (05:07)
[...]
wtmp begins Fri Mar 20 10:57:22 2020
```

توجه داشته باشید که فایل /var/log/wtmp به صورت اتوماتیک توسط کرون، Rotate می‌شود (ما در مورد Log Rotation در جلسات قبلی صحبت کردیم) و در نتیجه اطلاعات قدیمی‌تر موجود در خروجی last پس از مدتی دیگر به ما نشان داده نمی‌شوند و فقط اطلاعات جدید به ما نشان داده می‌شود.

دستور last و سایر ابزارهایی که در این بخش با آنها آشنا شدیم، ابزارهایی بسیار کاربردی و مناسب برای بازرسی کاربران، زمان ورود آنها به سیستم، عملکرد آنها در سیستم و.... می‌باشند. این ابزارها حتی زمانی که در حال انجام بازرسی امنیتی نیستیم نیز به کار می‌آیند، چرا که ما را در حل کردن مشکلات متفاوت یاری می‌دهند.

محدود کردن استفاده از RAM و پراسس‌ها

دستور ulimit به ما امکان می‌دهد که بتوانیم دسترسی هر حساب کاربری به منابع سیستم را محدود کنیم. این امر برای جلوگیری از مصرف کامل منابع توسط یک برنامه بسیار مناسب می‌باشد.

برای مشاهده‌ی منابع در اختیار کاربر کنونی، کافی است آپشن a- را به دستور ulimit بدهیم. به صورت زیر:

```
[behnam@localhost ~]$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 3795
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 3795
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
```

لازم به ذکر است که برای مشاهده‌ی محدودیت‌های سایر کاربران، کافی است یوزرنیم کاربر مورد نظر را پس از آپشن a- قرار دهیم.

هر چه کاربر بیشتر از سیستم کار بکشد، روی سیستم بار بیشتری قرار می‌دهد و RAM و CPU Time مصرف می‌کند. در محیط‌های لینوکسی چندکاربره، ممکن است مجبور باشیم محدودیتی‌هایی روی مقدار منابع مصرفی توسط هر کاربر قرار دهیم. این محدودیت شامل تعداد لاگین‌های همزمان، تعداد پراسس‌هایی که می‌توان استارت زد و همچنین میزان RAM قابل استفاده می‌باشد.

دستور ulimit می‌تواند این محدودیت‌ها را اعمال کند. در جدول ۲ برخی از مهم‌ترین آپشن‌های این دستور را مشاهده می‌کنیم:

جدول ۲- برخی از آپشن‌های کاربردی دستور ulimit

آپشن	عملکرد
-a	محدودیت‌های اعمال شده روی حساب کاربری کنونی را نشان می‌دهد.
-c	ماکزیمم سایز فایل‌های Core را مشخص می‌کند. فایل‌های Core، فایل‌هایی هستند که توسط سیستم در صورت استاپ شدن غیر نرمال یک پراسس ایجاد می‌شوند. این فایل‌ها شامل کلیه‌ی

	اطلاعاتی که آن برنامه درون RAM داشته می‌باشند. این فایل‌ها می‌توانند بسیار بزرگ شده و حجم زیادی از سیستم بگیرند، پس قرار دادن محدودیت روی آنها کاری منطقی می‌باشد.
`-f`	ماکزیمم سایز فایلی که کاربر می‌تواند ایجاد کند را مشخص می‌کند.
`-t`	ماکزیمم میزان CPU Time که کاربر می‌تواند استفاده کند را مشخص می‌کند.
`-u`	ماکزیمم تعداد پراسس‌هایی که کاربر می‌تواند به صورت همزمان اجرا کند را مشخص می‌کند.
`-v`	ماکزیمم میزان حافظه‌ی مجازی (Virutal Memory) قابل دسترسی توسط کاربر را مشخص می‌کند.

البته این همه‌ی محدودیت‌های قابل اعمال توسط این دستور نیست؛ اگر به manpage این دستور مراجعه کنید، خواهید دید که این دستور می‌تواند در زمینه‌های متفاوت، به صورت جزئی اعمال محدودیت کند.

اعمال محدودیت‌ها با استفاده از `ulimit` به صورت همیشگی نمی‌باشد و پس از ریبوت سیستم یا حتی ایجاد شِل جدید، محدودیت‌ها به حالت قبلی خود باز می‌گردند. برای این که محدودیت‌ها همیشگی باشند، باید محدودیت‌ها را داخل فایل `/etc/security/limits.conf` تعریف کنیم. ما به توضیح این فایل نمی‌پردازیم، اما می‌توانید با استفاده از دستور `man limits.conf`، اطلاعاتی در مورد این فایل به دست آورید.

پیداکردن فایل‌های دارای مجوز SUID/SGID

بسیاری از ابزارها در لینوکس برای عملکرد صحیح، نیاز به مجوزهای ویژه‌ی SUID و SGID دارند. اما برخی از کاربران مخرب با استفاده از این مجوزها می‌توانند مشکلاتی را در سیستم ایجاد کنند. اگر به خاطر داشته باشید، SUID مجوزی بود که به کاربر اجرا کننده‌ی اسکریپت، مجوزهای مالک فایل را میداد. در نتیجه اگر مجوز SUID روی یک فایل که مالک آن روت می‌باشد قرار گیرد، کاربر اجرا کننده حین اجرای اسکریپت، مجوزهای روت را خواهد داشت.

بررسی فایل‌های دارای SUID و SGID در سیستم به صورت مداوم، می‌تواند ما را در جلوگیری از به وجود آمدن مشکلات احتمالی در آینده یاری دهد. ما می‌توانیم فایل‌های دارای این مجوز را با استفاده از دستور `find` پیدا کنیم. دستور دقیقی که برای انجام این کار وارد می‌کنیم به شرح زیر می‌باشد:

```
find / -perm /6000 -type f
```

بیایید این دستور را به طور کلی آنالیز کنیم:

- علامت / پس از دستور `find`، به این دستور می‌گوید که ما می‌خواهیم در کل سیستم این جستجو را انجام دهیم. / نشان دهنده‌ی دایرکتوری روت می‌باشد.
- آپشن `-perm` به `find` می‌گوید که ما فقط دنبال فایل‌هایی هستیم که دارای یک مجوز خاص می‌باشند.
- ما به آپشن `-perm`، آرگمان `/6000` را داده‌ایم. این آرگمان به `find` می‌گوید که فقط دنبال فایل‌هایی بگردد که هم دارای مجوز SUID (دارای مقدار اکتال ٤) و هم دارای مجوز SGID (دارای مقدار اکتال ۲) می‌باشند. جمع این دو، به ما ٦۰۰۰ را می‌دهد.
- علامت / قبل از `6000/`، به `find` میگوید که سایر مجوز ها (مجوزهای به جز ۶) را در نظر نگیرد. یعنی در این حالت، `find` کاری به مقادیر مجوزهای مالک، گروه و سایرین، ندارد (`000`).

- آپشن `-type f` از `find` می‌خواهد که فقط به دنبال فایل‌ها باشد و کاری به دایرکتوری‌ها نداشته باشد.

بنابراین، اجرای این دستور باعث می‌شود که `find` همه‌ی فایل‌هایی که دارای مجوز SUID یا SGID هستند را به ما نشان دهد. از آنجایی که تعداد زیای فایل در سیستم هستند که دارای این مجوزها می‌باشند، بهتر است خروجی دستور `find` را داخل یک فایل ری‌دایرکت کنیم. علاوه بر این، `find` همیشه تعدادی ارور به ما در مورد فایل‌هایی که اجازه‌ی دسترسی به آن را ندارد نیز می‌دهد. از آنجایی که این ارورها روی STDERR نوشته می‌شوند، ردیدایرکت کردن STDOUT داخل یک فایل، عملیات بازرسی را ساده‌تر می‌کند. پس ما دستور را به صورت زیر اجرا می‌کنیم:

```
[behnam@localhost ~]$ find / -perm /6000 -type f > S-UIDGID_audit.txt
```

حال ما می‌توانیم فایل `S-UIDGID_audit.txt` را مطالعه کرده و از عدم وجود فایل‌های مخرب دارای این مجوزها در سیستم، اطمینان حاصل کنیم. پیشنهاد می‌شود که مجوزهای این فایل را محدود کرده تا بتوانید در آینده از این فایل، برای مقایسه‌ی نتایج بازرسی استفاده کنید.

آشنایی با مفاهیم اولیه‌ی رمزنگاری

هدف اصلی رمزنگاری، کدگذاری داده جهت مخفی کردن مفهوم اصلی آن می‌باشد. در رمز نگاری، متن ساده یا PlainText (متنی که قابل خواندن توسط انسان‌ها یا ماشین‌ها می‌باشد) تبدیل به رمزمتن یا CipherText (متنی که توسط انسان‌ها یا ماشین‌ها قابل خواندن نمی‌باشد) می‌شود. این عمل با استفاده از الگوریتم‌های رمزنگاری صورت می‌پذیرد. تبدیل متن ساده به رمزمتن، Encryption یا رمزگذاری نام دارد و تبدیل رمز متن به متن ساده، Decrytion یا رمزگشایی نام دارد.

الگوریتم‌های رمزنگاری، از یک سری کلید برای رمزگذاری و رمزگشایی اطلاعات استفاده می‌کنند. به این کلیدها، Cipher Key می‌گویند. بسته به نوع رمزنگاری مورد استفاده، ممکن است برخی از این کلیدها با سایرین به اشتراک گذاشته شوند. در بخش بعد، با مفهوم کلیدها آشنا می‌شویم.

درک مفهوم کلیدهای رمزنگاری

آشنایی مفهوم و عملکرد کلیدهای رمزنگاری در درک چگونگی انجام رمزنگاری بسیار مهم می‌باشد. به طور کلی، کلیدهای رمزنگاری به دو دسته‌ی کلیدهای خصوصی (متقارن) و کلیدهای عمومی/خصوصی(نامتقارن) تقسیم می‌شوند. بیایید با این دو نوع کلید بیشتر آشنا شویم:

- **کلیدهای خصوصی**

کلیدهای خصوصی، متقارن یا مخفی، اطلاعات را با استفاده از یک الگوریتم رمزنگاری و یک تک‌کلید رمزگذاری می‌کنند. در این روش، متن ساده با استفاده از همان تک‌کلید، هم رمزگذاری و هم رمزگشایی می‌شود. این کلید معمولا با استفاده از یک پسوورد که به آن Passphrase می‌گویند محافظت می‌شود. رمزنگاری متقارن بسیار سریع می‌باشد، اما بزرگترین ضعف آن، نیاز به اشتراک‌گذاری این کلید با سایرین می‌باشد. یعنی اگر ما یک متن ساده را با استفاده از این روش رمزگذاری کنیم و بخواهیم فرد دیگری بتوان آن را رمزگشایی کند، باید کلید را با آنها نیز به اشتراک بگذاریم.

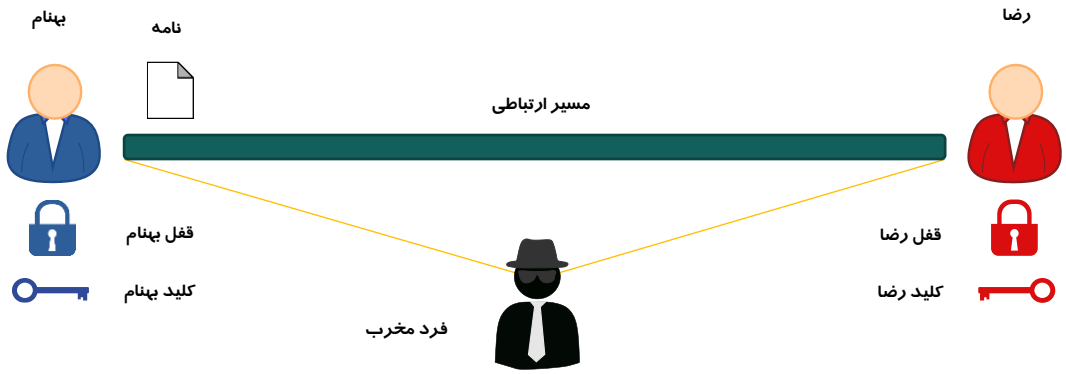
- **کلیدهای عمومی/خصوصی**

کلیدهای نامتقارن، یا جفت‌کلیدهای عمومی/خصوصی، داده‌ها را با استفاده از یک الگوریتم و دو کلید، رمزنگاری می‌کنند. بهتر است با یک معما، این نوع رمزنگاری را توضیح دهیم.

فرض کنید بهنام می‌خواهد یک نامه‌ی سرّی به رضا ارسال کند. بهنام و رضا هر کدام یک کلید و یک قفل منحصر به فرد دارند؛ به طوری که کلید بهنام، قفل بهنام را باز کرده و کلید رضا، قفل رضا را باز می‌کند. اما اینجا مشکلی وجود دارد.

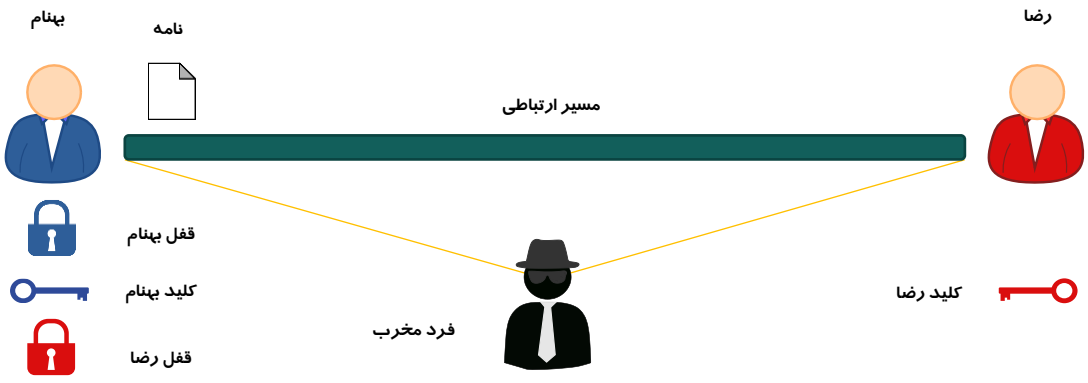
یک فرد مخرب، در حال رصد مسیر ارتباطی بهنام و رضا می‌باشد و او قصد دارد که محتویات نامه‌ی بهنام به رضا را بدزدد. با توجه به این که بهنام و رضا هر کدام یک قفل منحصر به فرد دارند، بهنام چگونه می‌تواند نامه‌ی خود را بدون این که در خطر خوانده شدن توسط فرد مخرب قرار گیرد به رضا ارسال کند؟

در تصویر ۱، این معما را به صورت مصورسازی شده مشاهده می‌کنیم.



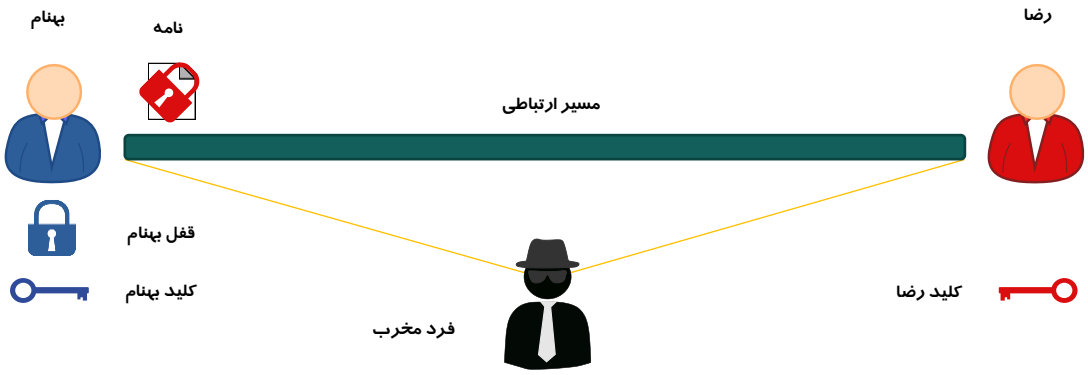
تصویر ۱- مصورسازی یک معما

بهترین راه برای ارسال سرّی نامه‌ی بهنام به رضا، این است که بهنام از رضا بخواهد که قفل خود را برای او بفرستد:



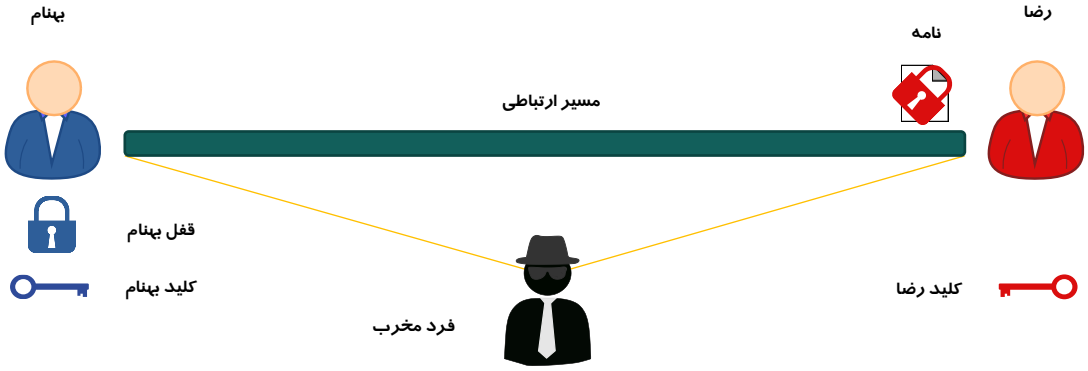
تصویر ۲- رضا قفل خود را برای بهنام ارسال می‌کند.

حال، کافی است بهنام با اسفاده از قفل رضا، نامه را قفل کند:



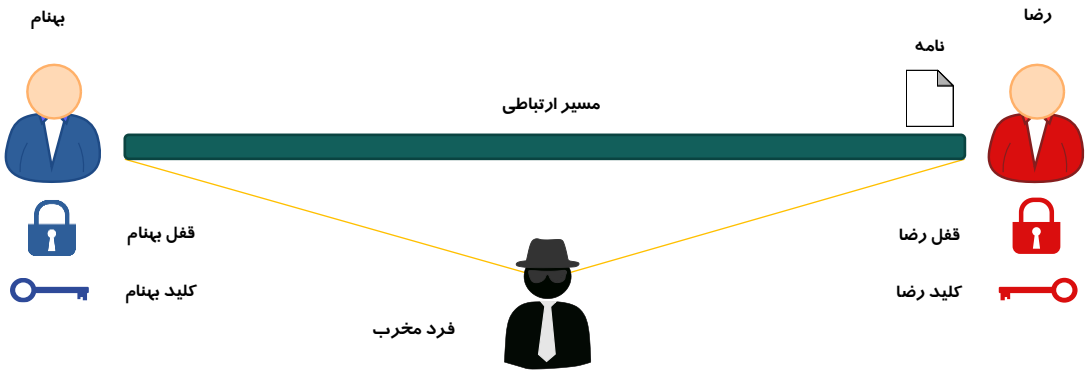
تصویر ۳- قفل کردن نامه توسط بهنام با استفاده از قفل رضا

حال بهنام می‌تواند با خیال راحت نامه را از مسیر ارتباطی برای رضا ارسال کند:



تصویر ٤- ارسال نامه‌ی قفل شده برای رضا

از آنجا که رضا کلید قفل خود را دارد، می‌تواند قفل نامه را باز کرده و آن را بخواند:



تصویر ۵- رضا قفل نامه را با استفاده از کلید خود باز می‌کند.

هنگام استفاده از این روش، فرد مخرب حتی اگر نامه را بدزدد، نمی‌تواند نامه را بخواند، چرا که نامه با قفل رضا قفل شده و برای باز کردن قفل رضا، به کلید رضا نیاز دارد و از آنجایی که کلید رضا نزد خود او محفوظ است، فرد مخرب در ماموریت خود شکست خورده است. نقطه‌ی قوت این روش در این است که کاربران برای ارسال اطلاعات به یکدیگر، نیازی به رد و بدل کردن کلید خود ندارند و کلید قفل هر کاربر، پیش خود او محفوظ میماند.

حال اگر رضا نیز بخواهد جواب نامه‌ی بهنام را به او بدهد، کافی است از بهنام بخواهد که قفل خود را به او ارسال کند تا رضا بتواند پاسخ نامه را با قفل بهنام قفل کرده و با خیال راحت، آن را برای بهنام ارسال کند.

در رمزنگاری کلید عمومی/خصوصی نیز ساختاری شبیه همین را داریم؛ به طوری که کلید عمومی نقش قفل را بازی کرده و کلید خصوصی نقش کلید را بازی می‌کند. اگر کاربر A بخواهد یک پیام مخفی به کاربر B بفرستند، با استفاده از کلید عمومی کاربر B آن پیام را رمزگذاری کرده و آن را به B ارسال می‌کند. کاربر B به محض دریافت پیام رمزگذاری شده، آن را با استفاده از کلید خصوصی خود، رمزگشایی می‌کند.

بر خلاف رمزگذاری کلید عمومی یا متقارن، در این روش رمزگذاری و رمزگشایی با یک تک‌کلید انجام نمی‌پذیرد، بلکه رمزگذاری با یک کلید و رمزگشایی با یک کلید دیگر صورت می‌پذیرد. این امر امنیت را بسیار بالا می‌برد، چرا که کلید مخصوص رمزگشایی همیشه پیش خود کاربر محفوظ میماند و نیازی به اشتراک آن وجود نخواهد داشت.

بار دیگر ذکر می‌کنیم که در این روش، کاربر باید همیشه کلید خصوصی خود را نزد خود محفوظ نگه دارد. معمولا از کلید خصوصی با استفاده از یک پسوورد، محافظت می‌کنند، در حالی که کلید عمومی می‌تواند در اختیار همه قرار گیرد. نکته‌ی قابل توجه دیگر این است که همانطور که یک کلید خاص فقط می‌تواند یک قفل خاص را باز کند، یک کلید خصوصی نیز فقط می‌تواند موارد رمزگذاری شده توسط یک کلید عمومی خاص را رمزگشایی کند. به عبارت دیگر، اگر کلید خصوصی کلید عمومی خود را از دست دهیم، دیگر نمی‌توانیم پیام‌های رمزنگاری شده با استفاده از آن کلید عمومی را رمزگشایی کنیم.

به دلیل امنیت بالا، رمزگذاری نامتقارن توسط بسیاری از برنامه‌های سیستم، نظیر SSH، به کار برده می‌شود. با این حال، روش رمزگذاری نامتقارن نیز بدون مشکلات خود نیست، برای مثال کسب اطمینان از این که کلید عمومی یک فرد، واقعا کلید عمومی آن فرد می‌باشد کار دشواری است؛ برای حل این مشکل از روش‌هایی مثل امضای دیجیتال استفاده می‌کنیم که بعدا به توضیح آن می‌پردازیم.

تصدیق اطلاعات

یکی از مفاهیم مهم در رمزنگاری، مفهوم Hash می‌باشد. Hashing که قبلا نیز به آن اشاره کردیم، یک تابع یک طرفه‌ی ریاضی می‌باشد که متن ساده را تبدیل به یک متن رمزی دارای طول ثابت می‌کند. از آنجایی که این تابع یک طرفه می‌باشد، ما نمی‌توانیم متن Hash شده را تبدیل به متن ساده‌ی اولیه کنیم. متن رمزی ایجاد شده توسط Hashing را Message Digest، Hash، Hash Value، Fingerprint یا Signature می‌گویند.

خوبی استفاده از Digestها در این است که ما را در مقایسه و تصدیق اطلاعات یاری می‌دهند؛ یعنی اگر دایجست فایل A برابر با دایجست فایل B باشد، این دو فایل دقیقا مانند هم می‌باشد و حاوی اطلاعات یکسان هستند. برای مثال وقتی یک فایل بزرگ را از اینترنت دانلود می‌کنیم، می‌توانیم با استفاده از دایجست آن، یکپارچگی و صحت آن فایل را بررسی کنیم. البته الگوریتم‌های Hashing نباید Collision داشته باشند. Collision به زمانی اشاره دارد که دو فایل کاملا متفاوت، یک دایجست را تولید می‌کنند. برخی از الگوریتم‌های قدیمی‌تر Hashing، مثل MD5، دارای Collision هستند و نباید از آنها استفاده شود.

امضای دیجیتال

یکی دیگر از کاربردهای Hashing در امضای دیجیتال می‌باشد. امضای دیجیتال یک ژتون رمزنگاری می‌باشد که به ما امکان اعتبارسنجی و تصدیق اطلاعات را می‌دهد. امضای دیجیتال در واقع دایجست پیام ارسالی می‌باشد که توسط کلید خصوصی فرستنده رمزگذاری می‌شود و به همراه پیام رمزگذاری شده به گیرنده ارسال می‌شود.

گیرنده امضای دیجیتال، امضا را با استفاده از کلید عمومی فرستنده رمزگشایی می‌کند و به دایجست پیام دریافتی دسترسی پیدا می‌کند. سپس کاربر پیام رمزگذاری شده‌ی دریافتی را رمزگشایی کرده و دایجست آن را به دست می‌آورد. گیرنده با مقاسیه‌ی دایجست موجود در امضای دیجیتال و دایجست پیام دریافتی، می‌تواند صحت پیام دریافتی را تصدیق کند.

بررسی SSH

عدم استفاده از یک ارتباط رمزگذاری شده هنگام اتصال به یک سرور ریموت از طریق شبکه، به کاربران مخرب این امکان را می‌دهد که ارتباط ما با سرور را رصد کنند و بتوانند کلیه‌ی اطلاعات ارسالی و دریافتی را مشاهده کنند. SSH یا Secure Shell، با رمزگذاری ارتباط ما با سرور، اقدام به حل این مشکل می‌کند. امروزه SSH تبدیل به پراستفاده‌ترین روش برای ارسال و دریافت اطلاعات از سرورهای ریموت شده است.

SSH از روش رمزنگاری نامتقارن (کلید عمومی/خصوصی) برای رمزنگاری ارتباطات خود استفاده می‌کند. هنگام ایجاد یک ارتباط SSH بین دو سیستم، هر سیستم کلید عمومی خود را به دیگر ارسال می‌کند.

اتصال به یک سرور ریموت با دستور *ssh*

نرم‌افزار OpenSSH معمولا به صورت پیش‌فرض روی بسیاری از توزیع‌های لینوکسی موجود می‌باشد و در صورت عدم وجود نیز، نصب آن زیاد کار دشواری نمی‌باشد.

برای ایجاد ارتباط SSH بین دو سیستم، به صورت زیر از دستور ssh استفاده می‌کنیم:

```
ssh [OPTIONS] USERNAME@HOSTNAME
```

به طوری که *HOSTNAME* نشان دهنده‌ی آدرس سروری است که می‌خواهیم به آن متصل شویم و *USERNAME* نشان دهنده‌ی یوزرنیم کاربری است که میخواهیم به عنوان آن در سرور ریموت لاگین کنیم.

برای این که برقراری ارتباط SSH بین دو سیستم ممکن باشد، نرم‌افزار OpenSSH باید روی هر دو سیستم نصب شده باشد. بیایید به یک سیستم ارتباط SSH بزنیم:

```
[root@localhost ~]# ssh behnam@192.168.1.50
The authenticity of host '192.168.1.50 (192.168.1.50)' can't be established.
ECDSA key fingerprint is SHA256:svw8tfoc04LIHvvVe301fr58nBFjW66xGg5k9bp9do8.
ECDSA key fingerprint is MD5:8a:54:6d:87:d3:d0:9a:59:33:1b:fb:ac:05:e8:0b:88.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.1.50' (ECDSA) to the list of known hosts.
behnam@192.168.1.50's password:
Last login: Wed Apr 28 10:13:54 2021 from 192.168.1.102
```

همانطور که می‌بینید، ما در اینجا بدون هیچ آپشنی دستور ssh را نوشتیم و پس از آن، یوزرنیمی که در سرور ریموت می‌خواهیم به عنوان آن لاگین شویم را وارد کرده و سپس آدرس سرور ریموت را وارد کردیم.

OpenSSH لیستی از کلیه‌ی سیستم‌هایی که قبلا به آن متصل شده‌ایم را در فایل `~/.ssh/known_hosts` نگهداری می‌کند. این فایل شامل کلید عمومی کلیه‌ی سرورهایی که به آن متصل شده‌ایم می‌باشد.

نکته: اگر هنگام استفاده از دستور `ssh` با خطای No route to host مواجه شدید، اول از اجرا بودن `ssh` در سیستم اطمینان حاصل کنید. در توزیع‌هایی که از راه‌انداز `systemd` استفاده می‌کنند، این کار با دستور `systemctl status sshd` صورت می‌پذیرد. اگر سرویس OpenSSH در حال اجرا بود، تنظیمات فایروال را چک کنید.

اگر تا به حال با `ssh` به یک سیستم ریموت خاص متصل نشده باشید، با پیامی طولانی، نظیر پیامی که در مثال صفحه قبل می‌بینید مواجه می‌شوید. این پیام در واقع به شما می‌گوید که سروری که قصد اتصال به آن را دارید در فایل `known_hosts` شما وجود ندارد. به محض تایپ `yes` هنگام پرسش `ssh` مبنی بر تمایل به اتصال به این سیستم، آدرس سیستم داخل `known_hosts` نوشته شده و سری بعد با این پیام مواجه نخواهید شد.

ارسال فایل‌ها به یک سرور ریموت با استفاده از *scp*

اگر به خاطر داشته باشید، ما قبلا با ابزار `cp` آشنا شدیم و گفتیم که این ابزار برای کپی یک فایل از یک موقعیت به موقعیت دیگر به کار می‌رود. `scp` ابزاری کاملا شبیه به `cp` می‌باشد، با این تفاوت که می‌تواند فایل‌ها را از سیستم ما داخل یک سرور ریموت کپی کند. `scp` برای این کار از SSH استفاده کرده و فایل‌ها را از یک مسیر رمزگذاری شده به سرور ارسال می‌کند. برای ارسال یک فایل به یک سرور ریموت با استفاده از `scp`، به صورت زیر عمل می‌کنیم:

`scp` *`source`* `username@hostname:`*`dest`*

به طوری که *`source`* نشان دهنده‌ی فایلی است که می‌خواهیم به سرور ریموت ارسال کنیم و *`dest`* نشان دهنده‌ی موقعیتی در سیستم ریموت است که می‌خواهیم فایل درون آن قرار گیرد.

برای مثال:

```
[root@localhost ~]# ls
secret_script.sh
[root@localhost ~]# scp secret.sh behnam@192.168.1.50:~/secret.sh
behnam@192.168.1.50's password:
secret.sh                     100%    0     0.0KB/s   00:00
```

همانطور که می‌بینید، ما با استفاده از دستور `scp`، فایل `secret.sh` را به یک سیستم ریموت ارسال کردیم. برای این کار، ابتدا دستور `scp` را وارد کرده، سپس موقعیت فایلی که می‌خواهیم به سرور ریموت ارسال کنیم را مشخص کردیم. در اینجا فایل در دایرکتوری کنونی بود، پس فقط نام فایل را وارد کردیم. پس از آن، یوزرنیم و آی‌پی سروری که می‌خواهیم فایل درون آن قرار گیرد را وارد کردیم. بلافاصله پس از آدرس آی‌پی سروری که می‌خواهیم به آن متصل شویم علامت دونقطه (:) را قرار دادیم و آدرس مکانی که می‌خواهیم فایل ارسالی درون آن قرار گیرد را وارد کردیم.

ارسال یک دستور به یک سرور ریموت با استفاده از *ssh*

می‌توانیم از ssh برای ارسال یک دستور به یک سرور ریموت استفاده کنیم. برای این کار، دقیقا مثل قبل از دستور ssh استفاده می‌کنیم و دستوری که می‌خواهیم در سرور ریموت اجرا شود را بلافاصله پس از مشخص کردن آدرس سرور ریموت، داخل دو علامت " قرار می‌دهیم. به صورت زیر:

```
[root@localhost ~]# ssh behnam@192.168.1.50 "ls -l secret.sh"
behnam@192.168.1.50's password:
-rw-r--r--. 1 behnam behnam 0 May  4 12:55 secret.sh
```

همانطور که می‌بینید با استفاده از ssh دستور `ls -l secret.sh` را در سیستم ریموت اجرا کردیم. با این کار، در واقع از کپی شدن فایلمان درون این سرور، اطمینان حاصل کردیم.

نکته: اگر قبلا از طریق SSH به یک سرور متصل شده باشید و پیام هشداری با عنوان WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED دریافت کردید، احتمال دارد که کلید عمومی سروری که به آن متصل شده‌اید، دچار تغییر شده باشد. امکان دیگر این است که یک فرد مخرب، در حال تقلید سروری که میخواهید به آن متصل شوید باشد، پس بهتر است حواستان را جمع کنید.

پیکربندی SSH

برای اطمینان از وجود تنظیمات درست هنگام اتصال به یک سرور ریموت، بهتر است با تنظیمات SSH و دستورالعمل‌های متفاوت آن آشنا شویم. به طور کلی، OpenSSH سه فایل تنظیمات اصلی دارد که به شرح زیر می‌باشند:

جدول ۳- فایل‌های تنظیمات OpenSSH و عملکرد آنها

موقعیت فایل	عملکرد
~/.ssh/config	تنظیمات مربوط به کلاینت OpenSSH را درون خود نگهداری می‌کند. آپشن‌هایی که به دستور ssh می‌دهیم، سبب باطل شدن تنظیمات داخل این فایل می‌شود.
/etc/ssh/ssh_config	تنظیمات مربوط به کلاینت OpenSSH را درون خود نگهداری می‌کند. آپشن‌هایی که به دستور ssh داده یا در فایل ~/.ssh/config قرار می‌دهیم، سبب باطل شدن تنظیمات داخل این فایل می‌شود.
/etc/sshd_config	تنظیمات Daemon مربوط به OpenSSH را درون خود دارد.

برای تنظیم صحیح OpenSSH باید بدانیم که در هر وضعیت، کدام فایل را تغییر دهیم. پیدا کردن فایلی که باید تغییر دهیم، کاملا بستگی به هدفی که داریم خواهد داشت:

- هر کاربری که می‌خواهد به یک سرور ریموت متصل شود، می‌تواند تنظیمات مورد نیاز و منحصر به خود را در ~/.ssh/config قرار دهد.
- اگر بخواهیم کلیه‌ی کاربرانی که می‌خواهند به یک سرور ریموت متصل شوند از تنظیمات خاصی استفاده کنند، تنظیمات را در فایل /etc/ssh/ssh_config قرار می‌دهیم.

- اگر بخواهیم تنظیمات مربوط به کسانی که به سیستم ما SSH می‌زنند را تغییر دهیم، یا بخواهیم سیستم خود را آماده‌ی دریافت ارتباطات SSH کنیم، تنظیمات را در فایل `/etc/sshd_config` قرار می‌دهیم.

OpenSSH دستورالعمل‌های زیادی دارد که مطالعه‌ی آنها در manpage مربوط به `ssh_config` و `sshd_config` خالی از لطف نیست. ما در اینجا فقط به بررسی چندتا از مهم‌ترین دستورالعمل‌های موجود در فایل `sshd_config` خواهیم پرداخت:

- `AllowTcpFowarding`: این دستورالعمل به ما امکان انجام Port Forwarding را می‌دهد.
- `ForwardX11`: این دستورالعمل به ما امکان X11 Forwarding را می‌دهد (چیزی شبیه به Remote Desktop ویندوز).
- `PermitRootLogin`: این دستورالعمل به کاربر روت اجازه‌ی لاگین کردن از طریق ارتباط SSH را می‌دهد. این دستورالعمل به صورت پیش‌فرض دارای مقدار `yes` می‌باشد؛ اما در اکثر اوقات، بهتر است که به آن مقدار `no` داده شود.
- `Port`: شماره‌ی پورتی که daemon مربوط به OpenSSH روی آن برای ارتباطات OpenSSH گوش می‌کند را مشخص می‌کند.

خیلی از اوقات، ممکن است سرورها پورت پیش‌فرض SSH را عوض کرده و روی یک پورت دیگر برای ارتباطات SSH گوش کنند. در چنین حالتی، ما می‌توانیم با ادیت‌کردن فایل‌های `/etc/ssh/ssh_config` یا `~/.ssh/config`، پورت جدیدی که دستور SSH ارتباطات را به آن ارسال می‌کند را مشخص کنیم. اگر این کار را در فایل‌های تنظیمات انجام ندهیم، مجبور هستیم با استفاده از آپشن `-p` دستور `ssh`، پورتی که می‌خواهیم به آن متصل شویم را مشخص کنیم.

ایجاد کلیدهای SSH

OpenSSH معمولا به دنبال جفت کلید عمومی/خصوصی سیستم برای ایجاد و دریافت ارتباطات میگردد. اگر OpenSSH موفق به پیدا کردن این جفت‌کلیدها نشود، آنها را به صورت اتوماتیک ایجاد می‌کند. این جفت‌کلیدها که به آنها کلیدهای هاست یا Host Keys نیز می‌گویند، در دایرکتوری `/etc/ssh` در قالب چند فایل ذخیره می‌شوند. برای مثال:

```
[root@localhost ~]# ls -1 /etc/ssh/*key*
/etc/ssh/ssh_host_ecdsa_key
/etc/ssh/ssh_host_ecdsa_key.pub
/etc/ssh/ssh_host_ed25519_key
/etc/ssh/ssh_host_ed25519_key.pub
/etc/ssh/ssh_host_rsa_key
/etc/ssh/ssh_host_rsa_key.pub
```

همانطور که می‌بینید، روی سیستم ما چندین کلید وجود دارد. فایل‌هایی که دارای پسوند `.pub` می‌باشند، کلیدهای عمومی سیستم هستند و می‌توانیم آنها را با دیگران با اشتراک بگذاریم. کلیدهایی که پسوندی ندارند، کلیدهای خصوصی هستند و به هیچ عنوان نباید به اشتراک گذاشته شوند. کلیدهای عمومی موجود در این دایرکتوری، از فرمت نام‌گذاری زیر پیروی می‌کنند:

```
ssh_host_KeyType_key
```

به طوری که KeyType اشاره به الگوریتمی دارد که در ایجاد کلیدها از آن استفاده شده است. معمول‌ترین این الگوریتم‌ها به شرح زیر می‌باشند:

- rsa (Rivest-Shamir-Adleman) قدیمی‌ترین و پراستفاده‌ترین الگوریتم می‌باشد که تقریبا توسط همه‌ی سیستم‌ها پشتیبانی می‌شود.
- dsa (Digital Signature Algorithm) یک الگوریتم منسوخ می‌باشد که دیگر از آن استفاده نمی‌شود.
- ecdsa (Eliptical Curve Digital Signature Algorithm) اشاره به نوعی از پیاده‌سازی DSA با استفاده از روش Eliptic Curve دارد.
- ed25519 یکی از انواع متفاوت الگوریتم Edwards-cruve Digital Signature Algorithm (یا EdDSA) می‌باشد که امنیت بالاتری نسبت به dsa و ecdsa دارد.

نکته: باز هم اشاره می‌کنیم که حفاظت از کلیدهای خصوصی SSH نیز، بسیار مهم می‌باشد. این فایل‌ها باید دارای مجوز اکتال 0640 یا 0600 باشند و مالک آنها، باید کاربر روت باشد.

بعضا ممکن است نیاز به ایجاد کلیدهای جدید داشته باشیم. برای این کار، از ابزار ssh-keygen استفاده می‌کنیم. انجام این کار بسیار ساده می‌باشد:

```
[root@localhost ~]# ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key
Generating public/private rsa key pair.
/etc/ssh/ssh_host_rsa_key already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /etc/ssh/ssh_host_rsa_key.
Your public key has been saved in /etc/ssh/ssh_host_rsa_key.pub.
The key fingerprint is:
[...]
```

دستور ssh-keygen آپشن‌های بسیاری دارد که در اینجا، ما فقط از دوتای آنها استفاده کرده‌ایم. آپشن t-، مشخص کننده‌ی KeyType یا نوع الگوریتم می‌باشد. همانطور که می‌بینید ما در این مثال از rsa استفاده کرده‌ایم. آپشن f-، موقعیت ذخیره و نام کلید خصوصی را مشخص می‌کند. کلید عمومی نیز در همان موقعیت ذخیره شده و تنها تفاوت آن وجود پسوند pub. می‌باشد. اگر توجه کرده باشید، این دستور هنگام ایجاد کلید، از ما درخواست یک Passphrase کرد. این Passphrase برای محافظت از کلید خصوصی می‌باشد؛ به طوری که قبل از استفاده از کلید خصوصی، از ما درخواست می‌شود که این Passphrase را وارد کنیم.

احراز هویت با کلیدهای SSH

توصیه می‌شود که به جای استفاده از پسوورد برای احراز هویت خود هنگام ایجاد ارتباط SSH با یک سرور، از کلیدهای SSH استفاده کنیم. برای استفاده از این روش احراز هویت، ابتدا باید سیستم را به صورت زیر آماده‌ی این کار کنیم:

۱- در سیستم خود یک جفت کلید SSH ID ایجاد کنیم.

۲- کلید عمومی SSH ID خود را به سرور ریموتی که می‌خواهیم به آن SSH بزنیم منتقل کنیم.

۳- کلید عمومی SSH ID خود را در موقعیت ~/.ssh/authorized_keys در سرور ریموت قرار دهیم.

بیایید با هم این مراحل را انجام دهیم.

ما ابتدا باید وارد سیستم خود شویم. ما به سیستم خود کلاینت می‌گوییم و در اینجا، باید جفت کلید SSH ID را با استفاده از ابزار ssh-keygen ایجاد کنیم. کلیدهای ایجاد شده باید دارای نام id_*TYPE* باشند، به طوری که *TYPE* می‌تواند dsa، rsa یا ecdsa باشد. برای مثال:

```
[behnam@localhost ~]$ ssh-keygen -t rsa -f ~/.ssh/id_rsa
Generating public/private rsa key pair.
Created directory '/home/behnam/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/behnam/.ssh/id_rsa.
Your public key has been saved in /home/behnam/.ssh/id_rsa.pub.
The key fingerprint is:
[...]
[behnam@localhost ~]$ ls .ssh/
id_rsa  id_rsa.pub
```

همنطور که می‌بینید، دستور ssh-keygen یک جفت کلید برای ما ایجاد کرد و آن را در موقعیتی که گفته بودیم، با نامی که خواسته بودیم، ذخیره کرد. همانطور که گفتیم، کلید دارای پسوند .pub کلید عمومی می‌باشد و کلیدی که پسوندی ندارد، کلید خصوصی می‌باشد.

حال که جفت‌کلیدها در سیستم کلاینت ایجاد شدند، باید کلید پابلیک کلاینت را به سرور ریموت منتقل کنیم. بهتر است این کار را از طریق یک راه امن و مطمئن انجام دهیم. خوشبختانه ابزار ssh-copy-id یک روش امن و ساده برای انتقال کلید و همچنین قرار دادن آن در دایرکتوری ~/.ssh/authorized_keys سرور ریموت می‌باشد. ما به شکل زیر از این ابزار استفاده می‌کنیم:

```
[behnam@localhost ~]$ ssh-copy-id -n behnam@192.168.1.51
[...]
=-=-=-=-=-=-=-=
Would have added the following key(s):

ssh-rsa AAAAB3NzaC1yc[...]
=-=-=-=-=-=-=-=
[behnam@localhost ~]$ ssh-copy-id behnam@192.168.1.51
[...]
behnam@192.168.1.51's password:

Number of key(s) added: 1
[...]
```

همانطور که می‌بینید، ما ابتدا دستور ssh-copy-id را با استفاده از آپشن n- اجرا کردیم. این آپشن به ما اجازه می‌دهد که ببینیم دقیقا چه کلیدی را می‌خواهیم در سرور ریموت قرار دهیم. به عبارتی، این آپشن حکم اجرای یک تست را دارد و فقط به ما نشان می‌دهد که برای ارسال کلید عمومی ما به سرور ریموت، چه کاری را قرار است انجام دهد.

پس از این کار، ما دستور ssh-copy-id را بدون آپشن n- اجرا کردیم و در نتیجه، کلید عمومی ما به درستی به سرور ریموت منتقل شد و در دایرکتوری ~/.ssh/authorized_keys در سرور ریموت قرار گرفت. توجه کنید که برای این که کلید به درستی منتقل شود، باید پسوورد کاربری که در سرور ریموت قرار است به عنوان آن لاگین کنیم را وارد کنیم.

حال که کلید عمومی ما داخل سرور ریموت قرار گرفته، می‌توانیم بدون نیاز به ارائه‌ی پسورد به سرور ریموت SSH بزنیم. البته اگر برای کلید خصوصی خود یک Passphrase انتخاب کرده باشیم، باید آن Passphrase را وارد کنیم. بیایید این کار را امتحان کنیم:

```
[behnam@localhost ~]$ ssh behnam@192.168.1.51
Enter passphrase for key '/home/behnam/.ssh/id_rsa':
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-58-generic x86_64)
[...]
behnam@the-albatross:~$ exit
logout
Connection to 192.168.1.51 closed.
```

همانطور که می‌بینید، هنگام استفاده از احراز هویت با کلیدهای SSH، چگونگی استفاده از دستور ssh دچار تغییر نمی‌شود؛ فقط در این حالت اگر برای کلید خصوصی خود یک Passphrase ایجاد کرده باشیم، باید آن را وارد کنیم.

استفاده از دستور scp نیز دقیقا مانند قبل می‌باشد، با این تفاوت که نیازی به وارد کردن پسوورد سرور ریموت نداریم:

```
[behnam@localhost ~]$ scp awesome_script.sh behnam@192.168.1.51:~
Enter passphrase for key '/home/behnam/.ssh/id_rsa':
awesome_script.sh                             100%    0     0.0KB/s   00:00
```

حال بیایید از صحت انتقال فایل اطمینان حاصل کنیم:

```
[behnam@localhost ~]$ ssh behnam@192.168.1.51 "ls -l"
Enter passphrase for key '/home/behnam/.ssh/id_rsa':
total 0
-rwxrw-r-- 1 behnam behnam 0 May  8 06:09 awesome_script.sh
```

همانطور که می‌بینید، فایل ما به درستی منتقل شده است.

بهبود امنیت SSH

برای افزایش امنیت SSH، می‌توانیم چندین کار انجام دهیم. این کارها به شرح زیر می‌باشند:

- از پورتی به جز پورت پیش‌فرض SSH (پورت ۲۲) استفاده کنیم.
- اجازه‌ی لاگین کردن به عنوان کاربر روت را ندهیم.
- از استفاده از Protocol 2 اطمینان حاصل کنیم.

پیشنهاد می‌شود که در سیستم‌هایی که دارای آی‌پی پابلیک می‌باشند، یعنی از طریق اینترنت قابل دسترسی هستند، هیچگاه روی پورت ۲۲ منتظر دریافت ارتباطات SSH نباشیم. ما می‌توانیم پورت SSH را با تغییر شماره‌ی نوشته شده در مقابل دستورالعمل Port در فایل /etc/ssh/sshd_config تغییر دهیم.

عدم اجازه‌ی لاگین کردن به عنوان کاربر روت هنگام ایجاد ارتباط SSH نیز بسیار مهم می‌باشد. به صورت پیش‌فرض، سیستم‌ها اجازه‌ی لاگین به عنوان کاربر روت را به همه می‌دهند. از آنجایی که روت مهم‌ترین کاربر سیستم می‌باشد، در صورت عدم غیرفعال کردن لاگین آن از طریق SSH، ممکن است کاربران مخرب با استفاده از حملات Brute Force اقدام به شکستن رمز آن کنند.

برای غیرفعال کردن لاگین به عنوان root، کافی است عبارت no را در مقابل دستورالعمل PermitRootLogin در فایل /etc/ssh/sshd_config قرار دهیم و سپس سرویس OpenSSH را ری‌استارت کرده یا فایل

تنظیمات آن را reload کنیم.

در نهایت، مهم است که از Protcol 1 استفاده نکنیم. Protocol 1، یک نسخه‌ی قدیمی‌تر از OpenSSH می‌باشد که از امنیت پایینی برخوردار است. با این که اکثر توزیع‌های مدرن از Protocol 2، که بعضا به آن OpenSSH 2 می‌گویند، استفاده می‌کنند، بهتر است که از صحت استفاده از آن اطمینان حاصل کنیم. برای این کار، کافی است به فایل /etc/ssh/sshd_config رفته و عدد مقابل دستورالعمل Protocol را بررسی کنیم. اگر عدد مقابل این دستورالعمل ۱ بود، آن را باید به ۲ تغییر دهیم و سرویس OpenSSH را ری‌استارت کرده یا فایل تنظیمات آن را Reload کنیم.

استفاده از GPG

خیلی از اوقات لازم است که یک فایل را رمزگذاری کرده و سپس آن را از طریق ایمیل یا روش‌های ارتباطی دیگر، به یک فرد ارسال کنیم. GPG یا GNU Privacy Guard می‌تواند ما را در این کار یاری دهد. این ابزار علاوه بر رمزگذاری فایل‌ها، می‌تواند امضای دیجیتال نیز به آنها اضافه کند.

از آنجا که GPG بر مبنای ابزار PGP (Pretty Good Privacy)، که یک ابزار Open Source و Free نبود ایجاد شده است، بعضا به آن OpenPGP نیز می‌گویند. GPG ابزار بسیار محبوب و معروفی است و در بسیاری از سیستم‌ها به صورت پیش‌فرض نصب می‌باشد. با این حال، اگر این ابزار در توزیع شما نصب نیست، می‌توانید آن را با استفاده از پکیج منیجر سیستم خود نصب کنید. این ابزار معمولا با نام gpg یا gnupg2 قابل نصب می‌باشد.

ایجاد کلیدهای GPG

برای استفاده از GPG، ابتدا باید یک جفت کلید عمومی/خصوصی ایجاد کنیم. GPG از رمزنگاری نامتقارن استفاده می‌کند و در نتیجه، کلید عمومیِ آن کلیدی است که در دسترس همگان قرار می‌گیرد و کلید خصوصیِ آن، کلیدی است که باید نزد کاربر محفوظ بماند (البته GPG به کلید خصوصی، Secret Key یا کلید مخفی می‌گوید).

برای ایجاد کلیدهای GPG از دستور gpg --gen-key استفاده می‌کنیم. به محض اجرای این دستور، GPG چندین سوال، اعم از نام کامل و آدرس ایمیل ما را از ما می‌پرسد و همچنین از ما می‌خواهد که یک Passphrase انتخاب کنیم. ما باید Passphrase و آدرس ایمیلی که وارد می‌کنیم را به خاطر بسپاریم، چرا که کلید عمومی ما با آدرس ایمیل وارد شده شناسایی می‌شود و کلید خصوصی، فقط با ورود Passphrase به طور صحیح کار می‌کند.

ما نتیجه‌ی اجرای این دستور را در اینجا نمی‌آوریم، چون کار کردن با این دستور بسیار ساه می‌باشد.

نکته: برای ایجاد کلیدهای قوی، نیاز به حجم زیادی از اطلاعات رندوم داریم. حین اجرای دستور gpg --gen-keys و پس از پاسخ به سوالات آن، این برنامه از ما می‌خواهد که دکمه‌های کیبورد خود را فشار دهیم، ماوس را تکان دهیم، اطلاعاتی را روی هارددیسک منتقل کنیم و... . این کار باعث ایجاد اطلاعات رندوم زیادی می‌شود، اما بسیار زمان‌گیر می‌باشد.

روش بهتر و ساده‌تر این است که به جای ایجاد اطلاعات رندوم به این طریق، از ابزار rngd استفاده کنیم. این ابزار از طریق پکیج rng-utils یا rng-tools قابل نصب می‌باشد. پس از نصب این ابزار و قبل از اجرای gpg --gen-keys، باید دستور sudo rngd -r /dev/urandom را اجرا کنیم. اجرای این دستور برای چند

لحظه کنترل شِل را از دست ما می‌گیرد. پس از بازپس‌گیری کنترل شِل، کافی است دستور `gpg --gen-key` را اجرا کنیم و مراحل را مانند قبل، دنبال کنیم. با این که باز هم در مرحله‌ی آخر از شما خواسته می‌شود که کارهای قبلی، نظیر فشردن دکمه‌های کیبورد و... را انجام دهید، ولی این مرحله بسیار سریع به پایان می‌رسد و کلیدهای شما با سرعت خیلی بالاتری ایجاد می‌شوند.

پس از ایجاد کلیدها، gpg آنها را در دایرکتوری `~/.gnupg/` ذخیره می‌کند. به مجموعه کلیدهای موجود در سیستم، Keyring می‌گویند.

استخراج کلید عمومی

همانطور که گفتیم، برای این که بتوانیم به سایرین امکان ارسال پیام‌های رمزگذاری شده به خودمان را بدهیم، باید کلید عمومی خود را برایشان ارسال کنیم. به این کار، استخراج کلید عمومی می‌گویند. ما می‌توانیم با استفاده از آپشن `--export`، از gpg بخواهیم که کلید عمومی ما را استخراج کرده و آن را درون یک فایل قرار دهد. به صورت زیر:

```
gpg --export EMAIL-ADDRESS > FILENAME.pub
```

به طوری که *EMAIL-ADDRESS* نشان دهنده‌ی آدرس ایمیلی است که یک کلید عمومی خاص را در Keyring ما شناسایی می‌کند (دقیقا همان آدرس ایمیلی که هنگام ایجاد کلید وارد کردیم) و *FILENAME* نشان دهنده‌ی نام فایلی است که می‌خواهیم کلید عمومی درون آن ذخیره شود. بیایید از این دستور استفاده کنیم:

```
[behnam@localhost ~]$ gpg --export the-albatross@outlook.com > gpg.pub
[behnam@localhost ~]$ ls -l
total 4
-rw-rw-r--. 1 behnam behnam 2386 May  9 11:30 gpg.pub
```

همانطور که می‌بینید، با اجرای این دستور، کلید عمومی ما استخراج شده و درون فایلی که نام آن را مشخص کرده بودیم قرار گرفته است.

پس از استخراج موفق کلید عمومی، می‌توانیم آن را در اختیار سایرین قرار دهیم تا بتوانند به ما اطلاعات رمزگذاری شده بفرستند. لازم به ذکر است که نیازی به مخفی کردن کلید عمومی نیست و ما می‌توانیم آن را به سایرین ایمیل کنیم و بدون هیچ نگرانی، حتی آن را در وبسایت خود قرار دهیم.

نکته: کلید عمومی که در بالا استخراج کردیم در فرمت باینری می‌باشد. برخی از سرویس‌های ایمیل اجازه‌ی ارسال فایل‌های باینری را به ما نمی‌دهند. در چنین حالتی، باید به دستور مربوط به استخراج کلید، `--armor` را اضافه کنیم تا کلید در فرمت ASCII استخراج شود.

ایمپورت کردن کلیدها

اگر بخواهیم فایلی را برای دیگران رمزگذاری کنیم، باید کلید عمومی آنها را داشته باشیم. به محض دریافت کلید عمومی آنها، باید این کلید را وارد Keyring خود کنیم. به این فرآیند، Importing می‌گویند. ما می‌توانیم این کار را با استفاده از دستور زیر انجام دهیم:

```
gpg --import FILENAME.pub
```

به طوری که *FILENAME* نشان دهنده‌ی نام فایلی است که کلید عمومی کاربر مورد نظر را درون خود دارد.

بیایید یک کلید عمومی در سیستم خود ایمپورت کنیم (برای تمرین، می‌توانید از کلید عمومی ریچارد استالمن استفاده کنید):

```
[behnam@localhost ~]$ gpg --import rms-pubkey.txt
gpg: key 2A8E4C02: public key "Richard Stallman <rms@gnu.org>" imported
gpg: Total number processed: 1
gpg:               imported: 1  (RSA: 1)
gpg: 3 marginal(s) needed, 1 complete(s) needed, PGP trust model
gpg: depth: 0  valid:   2  signed:   0  trust: 0-, 0q, 0n, 0m, 0f, 2u
```

پس از ایمپورت کردن یک کلید، بد نیست که نگاهی به کلیدهای موجود در Keyring خود بیاندازیم تا از صحت ایمپورت یک کلید، اطمینان حاصل کنیم:

```
[behnam@localhost ~]$ gpg --list-keys
/home/behnam/.gnupg/pubring.gpg
------------------------------
pub   2048R/0CCE3C03 2021-05-08
uid                  The Albatross <the-albatross@outlook.com>
sub   2048R/1ADB8360 2021-05-08

pub   2048R/BF6323E5 2021-05-09
uid                  Puppy McPupperson <the-puppster@woof.com>
sub   2048R/5D17D431 2021-05-09

pub   4096R/2A8E4C02 2013-07-20
uid                  Richard Stallman <rms@gnu.org>
sub   4096R/62853425 2013-07-20
```

همانطور که می‌بینید، در خروجی این دستور، علاوه بر کلید پابلیک خودمان، کلیه‌ی کلیدهای پابلیکی که در سیستم ایمپورت کرده‌ایم نیز نشان داده می‌شود.

رمزگذاری و رمزگشایی اطلاعات

پس از ایمپورت یک کلید عمومی در Keyring خود، می‌توانیم عملیات رمزگذاری را استارت بزنیم. ما ابتدا باید فایل مورد نظر را ایجاد کرده و سپس آن را با gpg رمزگذاری کنیم. این فایل، می‌تواند هر نوع فایلی، اعم از فایل متنی و... باشد. برای رمزگذاری فایل، به صورت زیر از دستور gpg استفاده می‌کنیم:

```
gpg --out ENCRYPTED-FILE --recipient EMAIL-ADDRESS --encrypt ORIGINAl-FILE
```

به طوری که *ORIGINAL-FILE* نام فایلی است که می‌خواهیم رمزگذاری کنیم، *EMAIL-ADDRESS* آدرس ایمیل مربوط به کلید عمومی می‌باشد که می‌خواهیم فایل را با آن رمزگذاری کنیم و *ENCRYPTED-FILE*، نامی است که می‌خواهیم فایل رمزگذاری شده به خود بگیرد. بیایید یک فایل رمزگذاری شده ایجاد کنیم:

```
[behnam@localhost ~]$ gpg --out encrypted_script --recipient the-puppster@woof.com
--encrypt script.sh
gpg: 5D17D431: There is no assurance this key belongs to the named user

[...]
```

با استفاده از این دستور، فایل `script.sh` را با کلید عمومی کاربر the-puppster@woof.net رمزگذاری کردیم و فایل رمزگذاری شده را با نام `encrypted_script.sh` ذخیره کردیم. قبل از انجام رمزنگاری، gpg به شما هشدار می‌دهد که ممکن است کلید عمومی استفاده شده، واقعا کلید عمومی کاربر مورد نظر

نباشد. اگر کلید عمومی را از یک موقعیت قابل اطمینان دریافت کرده باشید، لازم نیست نگران این هشدار باشید.

فردی که فایل رمزگذاری شده را دریافت کرده، می‌تواند آن را با کلید خصوصی خود، باز کند. برای این کار، او باید دستور زیر را اجرا کند:

```
gpg --out DECRYPRED-FILE --decrypt ENCRYPTED-FILE
```

به طوری که *ENCRYPTED-FILE* نشان دهنده‌ی فایل رمزگذاری شده‌ای است که می‌خواهیم رمزگشایی کنیم و *DECRYPTED-FILE* نام فایلی است که میخواهیم فایل رمزگشایی شده در آن ذخیره شود. لازم به ذکر است که هنگام رمزگشایی و در صورت قرار دادن یک Passphrase روی کلید خصوصی خود، gpg از ما می‌خواهد که Passphrase کلید خصوصی را برای رمزگشایی فایل، وارد کنیم.

برای مثال:

```
[puppy@localhost]~% whoami
puppy
[puppy@localhost]~% ls
encrypted_script
[puppy@localhost]~% gpg --out the_script.sh --decrypt encrypted_script

You need a passphrase to unlock the secret key for
user: "Puppy McPupperson <the-puppster@woof.com>"
2048-bit RSA key, ID 5D17D431, created 2021-05-09 (main key ID BF6323E5)

gpg: encrypted with 2048-bit RSA key, ID 5D17D431, created 2021-05-09
      "Puppy McPupperson <the-puppster@woof.com>"
[puppy@localhost]~% ls the_script.sh
the_script.sh
[puppy@localhost]~% cat the_script.sh
HA HA fooled ya, there is no script lmao
```

امضا کردن پیام‌ها و تصدیق امضاها

با این که رمزگذاری یک پیام، محتوای آن را برای سایرین غیرقابل درک می‌کند و به نحوی، حریم شخصی شما را حفظ می‌کند، اما این پیام در برابر دستکاری شدن حین ارسال مصون نیست. فرض کنید بهنام می‌خواهد پیامی را به صورت رمزگذاری شده به رضا ارسال کند. یک کاربر مخرب می‌تواند کلید عمومی رضا را به دست آورده، پیامی را با آن رمزگذاری کرده و به رضا ارسال کند و ادعا کند که آن پیام از طرف بهنام ارسال شده است.

خوشبختانه هنگام استفاده از gpg، ما می‌توانیم فایل‌های رمزگذاری شده را امضا کنیم. در این حالت اگر فایل به هر طریقی حین ارسال دستکاری شود، gpg ما را از دستکاری شدن فایل مطلع می‌سازد.

برای امضای یک فایل، از آپشن `--sign`، یا اگر بخواهیم امضا به صورت ASCII باشد، از آپشن `--clearsign` استفاده می‌کنیم. پروسه‌ی امضای دیجیتال کمی با پروسه‌ی رمزگذاری فایل متفاوت می‌باشد. برای رمزگذاری امضای دیجیتال، ما از کلید خصوصی خودمان استفاده می‌کنیم، در حالی که برای رمزگذاری یک فایل برای یک کاربر، از کلید عمومی کاربری که می‌خواهیم فایل را به او ارسال کنیم استفاده می‌کنیم. بیایید یک فایل را رمزگذاری کرده و سپس آن را امضا کنیم:

```
[behnam@localhost ~]$ cat new_file.txt
Hi Puppy,
It's me, Behnam.
No, I still don't know why I spent a year writing this shit.
```

```
Signed,
Behnam
[behnam@localhost ~]$ gpg --out unsigned_file --recipient the-puppster@woof.com --encrypt new_file.txt
[...]
[behnam@localhost ~]$ gpg --output signed_file --sign unsigned_file
[...]
[behnam@localhost ~]$ ls signed_file
signed_file
```

توجه کنید که در صورت انتخاب Passphrase برای کلید خصوصی خود، هنگام امضای فایل، gpg از ما درخواست وارد کردن Passphrase کلید خصوصیمان را می‌کند.

زمانی که گیرنده‌ی فایل، فایل رمزگذاری شده را دریافت می‌کند، می‌تواند با استفاده از آپشن --verify، صحت و یکپارچگی فایل دریافتی را بررسی کند. به صورت زیر:

```
gpg --verify RECEIVED-SIGNED-FILE
```

به طوری که *RECEIVED-SIGNED-FILE*، فایل رمزگذاری شده و امضا شده‌ی دریافتی می‌باشد.

نکته: توجه داشته باشید که چون امضای دیجیتال با استفاده از کلید خصوصی فرستنده رمزگذاری می‌شود، گیرنده‌ی پیام باید کلید عمومی فرستنده را در Keyring خود داشته باشد، در غیر این صورت، gpg با پیام Can't check signature: No public key کاربر را از عدم وجود کلید عمومی مربوطه مطلع می‌سازد.

چک کردن امضای فایل و رمزگشایی آن یک پروسه‌ی دو مرحله‌ای می‌باشد. ما ابتدا باید امضای دیجیتال را رمزگشایی و تصدیق کنیم و پس از آن، خود فایل را رمزگشایی کنیم. به صورت زیر:

```
[puppy@localhost]~% gpg --out recieved_msg --verify signed_file
gpg: Signature made Mon 10 May 2021 12:34:02 PM +0430 using RSA key ID 0CCE3C03
gpg: Good signature from "The Albatross <the-albatross@outlook.com>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: 31FA C88C 0598 C173 B43C  AF68 AAAC 70B6 0CCE 3C03
```

همانطور که می‌بینید، ما با استفاده از این دستور، امضای دیجیتال را رمزگشایی کرده، صحت فایل را تایید کردیم و خروجی مربوطه را در فایل received_msg ریختیم. در محیط تستی، نیازی به توجه به جملاتی که پس از WARNING آمده نیست (در اینجا gpg به ما می‌گوید که نمی‌تواند صحت مالکیت کلید عمومی مورد استفاده توسط کاربری که فایل را امضا کرده را تایید کند).

حال می‌توانیم فایل received_msg را رمزگشایی کنیم. این کار را بارها انجام داده‌ایم، پس:

```
[puppy@localhost]~% gpg --out decrypted_msg --decrypt recieved_msg
[...]
[puppy@localhost]~% cat decrypted_msg
Hi Puppy,
It's me, Behnam.
No, I still don't know why I spent a year writing this shit.
Signed,
Behnam
```

نکته: اگر بخواهیم یک فایل را امضا کنیم اما رمزگذاری نکنیم، می‌توانیم از آپشن --deatch-sig استفاده کنیم. این کار باعث ایجا یک فایل جداگانه‌ی امضا می‌شود. ما باید این فایل را همراه فایل مورد نظر به گیرنده ارسال کنیم.

ابطال یک کلید

اگر کلید خصوصی ما گم شود یا دزدیده شود، باید کلید عمومی خود را ابطال کنیم. این کار، یک پروسه‌ی ۳ مرحله‌ای می‌باشد:

۱- ایجاد یک سند ابطال

۲- ایمپورت سند ابطال داخل Keyring خود

۳- در دسترس گذاری یا ارسال سند ابطال به همه‌ی افرادی که کلید عمومی ما را دارند.

برای ایجاد سند ابطال، باید از آپشن --gen-revoke یا --generate-revocation دستور gpg استفاده کنیم. انجام این کار باعث می‌شود که gpg چندین سوال کلی از ما در مورد دلیل ایجاد این سند ابطال بپرسد. پس از پاسخ به این سوالات، gpg به ما اجازه می‌دهد که اطلاعات بیشتری در مورد دلیل ابطال، ارائه دهیم.

بیایید کلید عمومی خود را ابطال کنیم:

```
[behnam@localhost ~]$ gpg --out key-revocation.asc --gen-revoke the-albatross@outlook.com

sec  2048R/0CCE3C03 2021-05-08 The Albatross <the-albatross@outlook.com>

Create a revocation certificate for this key? (y/N) y
Please select the reason for the revocation:
  0 = No reason specified
  1 = Key has been compromised
  2 = Key is superseded
  3 = Key is no longer used
  Q = Cancel
(Probably you want to select 1 here)
Your decision? 1
Enter an optional description; end it with an empty line:
> Someone stole my bike! And me private key! Oh no!
>
Reason for revocation: Key has been compromised
Someone stole my bike! And me private key! Oh no!
Is this okay? (y/N) y

You need a passphrase to unlock the secret key for
user: "The Albatross <the-albatross@outlook.com>"
2048-bit RSA key, ID 0CCE3C03, created 2021-05-08

ASCII armored output forced.
Revocation certificate created.

Please move it to a medium which you can hide away; if Mallory gets
access to this certificate he can use it to make your key unusable.
It is smart to print this certificate and store it away, just in case
your media become unreadable.  But have some caution:  The print system of
your machine might store the data and make it available to others!
```

بیایید از صحت ایجاد سند ابطال اطمینان حاصل کنیم:

```
[behnam@localhost ~]$ ls key-revocation.asc
key-revocation.asc
```

حال که سند ابطال ما ایجاد شد، باید آن را داخل Keyring خود ایمپورت کنیم. ما قبلا در مورد چگونگی این کار صحبت کردیم:

```
[behnam@localhost ~]$ gpg --import key-revocation.asc
[...]
```

حال باید این سند ابطال را در اختیار کلیه‌ی افرادی که کلید عمومی ما را داخل Keyring خود دارند قرار دهیم. چگونگی این کار بستگی به چگونگی به اشتراک گذاری کلید پابلیک خود دارد. برای مثال اگر کلید عمومی خود را در وبسایت خود قرار داده باشیم، اکنون باید این سند را نیز داخل وبسایت خود قرار دهیم. اگر کلید عمومی خود را ایمیل کرده باشیم، اکنون باید سند ابطال را نیز به سایرین ایمیل کنیم، یا اگر از یک سرور کلید GPG استفاده کرده باشیم، باید اکنون این سند ابطال را به آن سرور ارسال کنیم.