Впервые опубликовано на developerWorks 13.06.2006

Подготовка к экзамену LPI: Системная безопасность

Средний уровень администрирования (LPIC-2) тема 212

Настройка маршрутизатора

О фильтре пакетов

Ядро Linux содержит инфраструктуру "netfilter", которая позволяет вам отфильтровывать сетевые пакеты. Обычно эта возможность компилируется как неотъемлимая часть ядра, но можно собрать в виде модуля. В любом случае, загрузка модуля дожна быть прозрачной (например, запуск `iptables` приведет к загрузке `iptables_filter.o`, если это потребуется).

В более новых системах Linux фильтрация пакетов управляется утилитой `iptables`; более старые системы использовали `ipchains`. А до этого применялась `ipfwadm`. Хотя, если требуется обратная совместимость, вы все еще можете использовать `ipchains` вместе с недавними версиями ядра, вы все равно предпочтете более широкие возможности и улучшенный синтаксис в `iptables`. То есть, большая часть понятий и опций в `iptables` являются совместимыми улучшениями `ipchains`.

В зависимости от точного сценария фильтра (firewall, NAT, и тд), фильтрация и перевод адреса может произойти либо до, либо после маршрутизации как таковой. В обоих случаях используется одна и та же программа `ipchains`, но используют различные правила ("цепочки") -- в основе, `INPUT` и `OUTPUT`. Однако, фильтрация также может повлиять на решение маршрутизации, из-за филтрации `FORWARD` цепочки; этот способ может привести к исчезновлению пакетов, вместо их маршрутизирования.

Маршрутизация

Наряду с фильтрацией при помощи `iptables` (или более ранней `ipchains`), ядро Linux производит маршрутизацию IP-пакетов, которых получает. Маршрутизация -- более простой процесс, нежели фильтрация, хотя они умозрительно являются связанными.

Во время маршрутизации, хост просто смотрит на IP-адрес назначения и решает, знает ли он, как доставить пакет непосредственно на этот адрес, или же ему доступен шлюз, который знает, как доставить на этот адрес. Если хост не может ни доставить пакет сам, ни знает шлюза, которому можно поручить доставку, пакет теряется. Однако типичная конфигурация содержит "шлюз по умолчанию", который

обрабатывает любой адрес, не определенный каким-нибудь способом.

Настройка и отображение информации по маршрутизации производится утилитой route. Однако маршрутизация может быть либо статической, либо динамической.

При статической маршрутизации, доставка определяется таблицей маршрутизации, которая явно настраивается вызовом команды route и ее командами add или del. Более полезной, однако, может оказаться настройка динамической маршрутизации с использованием демонов routed или gated, которые рассылают информацию о маршрутизации соседним демонам-маршрутизаторам.

Демон routed поддерживает Routing Information Protocol (RIP); демон gated вдобавок имеет поддержку других протоколов -- и может пользоваться многими протоколами одновременно -- таких как:

- Routing Information Protocol Next Generation (RIPng)
- Exterior Gateway Protocol (EGP)
- Border Gateway Protocol (BGP) и BGP4+
- Defense Communications Network Local-Network Protocol (HELLO)
- Open Shortest Path First (OSPF)
- Intermediate System to Intermediate System (IS-IS)
- Internet Control Message Protocol (ICMP и ICMPv6)/Router Discovery

Давайте взглянем на довольно типичную статическую таблицу маршрутизации:

Listing 1. Типичная статическая таблица маршрутизации

```
% /sbin/route
Kernel IP routing table
Destination   Gateway         Genmask         Flags Metric Ref  Use Iface
66.98.217.0   *               255.255.255.0   U     0      0      0 eth0
10.10.12.0    *               255.255.254.0   U     0      0      0 eth1
66.98.216.0   *               255.255.254.0   U     0      0      0 eth0
169.254.0.0   *               255.255.0.0     U     0      0      0 eth1
default       ev1s-66-98-216- 0.0.0.0         UG    0      0      0 eth0
```

Это означает, что адресы в диапазонах 66.98.217/24 и 66.98.216/23 напрямую доставляются через `eth0`. Диапазоны адресов 10.10.12/23 и 169.254/16 доставляются на `eth1`. Все, что осталось, посылается на шлюз `ev1s-66-98-216-1.ev1servers.net` (имя обрезано в выводе `route`; также можно использовать `route -n`, чтобы убедиться, что это имя имеет IP-адрес 66.98.216.1). Если вы хотите добавить другой шлюз для некоторых диапазонов адресов, вам следует выполнить следующее:

Listing 2. Добавление нового шлюза для других диапазонов адресов

```
% route add -net 192.168.2.0 netmask 255.255.255.0 gw 192.168.2.1 dev
eth0
```

Для машины, которая сама по себе является шлюзом, вообще говоря, используется динамическая маршрутизация, при помощи демонов `routed` или `gated`, которые

могут давать меньшее число статических маршрутов. Демон `routed` настраивается содержимым /etc/gateways. Демон `gated` является более современным, и, как уже говорилось, имеет больше возможностей; он настраивается файлом /etc/gated.conf. Вообще говоря, при использовании какого-нибудь из этих демонов, вы можете запускать через сценарии запуска. Но нельзя запускать демонов `routed` и `gated` на одной машине, поскольку результаты станут непредсказуемыми и, конечно же, нежелательными.

Фильтрация с помощью iptables

Ядро Linux хранит таблицу правил фильтрации IP-пакетов, которая образует некоторую разновидность машины состояний. Наборы правил объединяются в последовательности, известные как "цепочки". Если одна цепочка встречает условие, одним из возможных действий является передача управление на обработку другой цепочки, как делается в машине состояний. Перед добавлением правил или состояний, автоматически присутствуют три цепочки: `INPUT`, `OUTPUT`, и `FORWARD`. Цепочка `INPUT` работает когда пакет, адресованный машине-хосту, передаётся процессу локального приложения. Цепочка `FORWARD` используется, когда приходит пакет, адресованный другой машине, полагая, что здесь активирована переадресация, и система маршрутизации знает, как переслать пакет дальше. Пакет, порожденный на локальном хосте, посылается для фильтрации в цепочку `OUTPUT` -- если он проходит фильтры в цепочке `OUTPUT` (или любых связанных цепочек), он маршрутизуется за пределы сетевого интерфейса.

Одно действие, которое может предпринять правило, есть `DROP` (уничтожение) пакета; в этом случае для этого пакета больше никаких дальнейших обработок правил или переходов между состояниями не предпринимается. Но если пакет не уничтожается, то проверяется, соответствует ли пакету следующее правило в цепочке. В некоторых случаях соответствие правилу перенесёт процесс обработки на другую цепочку со своим набором правил. Создание, удаление или изменение правил и цепочек, где содержатся эти правила, производится утилитой `iptables`. В старых системах Linux, эти же функции выполняла утилита `ipchains`. Идеи, реализованные в этих утилитах, и даже в более древней `ipfwadm` похожи, но здесь мы обсудим синтаксис `iptables`.

Правило определяет набор условий, которым пакет может соответствовать, и то, какое действие следует произвести, если пакет не соответствует условию. Как упоминалось, одно общее действие состоит в `DROP` (уничтожении) пакетов. Например, предположим, что вам нужно (по каким-то причинам) отключить `ping` в петлевом интерфейсе (интерфейсе ICMP). Это можно осуществить с помощью:

Listing 3. Отключение петлевого интерфейса

```
% iptables -A INPUT -s 127.0.0.1 -p icmp -j DROP
```

Конечно, это правило глупое, и его, скорее всего, следует убрать после тестирования, примерно вот так:

Listing 4. Как убрать это глупое правило

```
% iptables -D INPUT -s 127.0.0.1 -p icmp -j DROP
```

Удаление правила с опцией `-D` требует либо в точности те же параметры, что были указаны при добавлении правила, либо описания с помощью номера правила (который нужно определять на первом месте) вот так:

Listing 5. Указание номера правила, так чтобы удаление сработало

```
% iptables -D INPUT 1
```

Более интересное правило может глядеть на адрес источника и назначения в пакетах. Например, представьте, что подозрительная удаленная сеть пытается использовать службы на определенной подсети вашей сети. Это можно заблокировать на машине вашего шлюза/межсетевого экрана таким образом:

Listing 6. Блокировка машины шлюза/межсетевого экрана

```
% iptables -A INPUT -s 66.98.216/24 -d 64.41.64/24 -j DROP
```

Это остановит все, что исходит из `66.98.216.*` IP-диапазона на локальную подсеть `64.41.64.*`. Конечно, использование этого метода занесения определенного IP-диапазона в черный список в качестве защиты довольно ограничено. Более вероятным сценарием будет разрешение доступа к локальной подсети только от определенного диапазона IP:

Listing 7. Разрешение указанному диапазону IP иметь доступ к локальной подсети

```
% iptables -A INPUT -s ! 66.98.216/24 -d 64.41.64/24 -j DROP
```

В этом случае только адреса из IP диапазона `66.98.216.*` могут иметь доступ к указанной подсети. Более того, для адреса можно использовать символическое имя, и можно указывать определенный протокол для фильтрации. Также можно выбрать для фильтрации определенный сетевой интерфейс (например, `eth0`), но это используется не так широко. Например, чтобы разрешить только отдельной удаленной сети обращаться к локальному web-серверу, следует использовать:

Listing 8. Разрешение отдельной удаленной сети обращаться к локальному Веб-серверу

```
% iptables -A INPUT -s ! example.com -d 64.41.64.124 -p TCP -sport 80 -j
DROP
```

Можно указать еще и некоторые другие опции для `iptables`, например, включая ограничения на число допустимых пакетов или фильтрацию по флагам TCP. За подробностями обращайтесь к man-страницам `iptables`.

Цепочки, определенные пользователем

Мы видели основы добавления правил в автоматическае цепочки. Но гибкая настраиваемость `iptables` возможна при добавлении цепочек, определенных пользователем и переход к ним при совпадении с шаблоном. Новые цепочки определяются с использованием опции `-N`; мы уже осуществляли переход с помощью специальной цели `DROP`. `ACCEPT` (разрешить) -- это тоже специальная цель с очевидным значением. Также доступны специальные цели `RETURN` (вернуть) и `QUEUE` (поставить в очередь). Первая означает остановку обработки данной цепочки и возврат к вызывавшему/породителю ее. Обработчик `QUEUE` позволяет передавать пакеты к процессам пользователя для дальнейшей обработки (это может быть журналирование, изменение пакета, или более сложная фильтрация, нежели та, что поддерживается `iptables`). Простой пример из книга Русти Рассела "Linux 2.4 Packet Filtering HOWTO" -- хорошая иллюстрация добавления пользовательских цепочек:

Listing 9. Добавление пользовательской цепочки

```
# Создание цепочки для блокировки всех соединенений за исключением
локальных или уже существующих
|-------- XML error:  The previous line is longer than the max of 90
characters ---------|
% iptables -N block
% iptables -A block -m state --state ESTABLISHED,RELATED -j ACCEPT
% iptables -A block -m state --state NEW -i ! ppp0 -j ACCEPT
% iptables -A block -j DROP  # Уничтожаем всё, что не было разрешено
(ACCEPT)
# Переключение на эту цепочку с цепочек INPUT и FORWARD
% iptables -A INPUT -j block
% iptables -A FORWARD -j block
```

Обратите внимание, что цепочка `block` принимает (`ACCEPT`) в ограниченном числе случаев, тогда как последнее правило пропускает (`DROP`) все, что не было принято ранее.

По мере того, как были созданы новые цепочки, либо путем добавления правил к автоматическим, либо при помощи новых пользовательских цепочек, можно использовать опцию `-L` для просмотра текущих правил.

Преобразование сетевых адресов в сравнении с межсетевыми экранами

Примеры, которые мы видели, в основном касались правил для межсетевых экранов. Но преобразование сетевых адресов (NAT) так же настраивается с помощью `iptables`.

Как правило, NAT -- это способ использования отслеживания соединений для

маскаранинга пакетов, исходящих из адреса локальной подсети в качестве внешнего адреса WAN, перед тем, как отправлятьиз дальше "по проводам" (на цепочке `OUTPUT`). Шлюз/маршутизатор, выполняющий NAT, должен запоминать, какой локальный хост соединен с каким удаленным хостом, и обращать преобразование адреса, если вдруг пакеты приходят обратно с удаленной машины.

с точки зрения системы фильтрации просто делается вид, что NAT не существует. Правила, определяемые нами, просто используют "настоящие" локальные адреса, безотносительно тому, как NAT замаскирует их для внешнего мира. Включение маскарадинга, такого как базовый NAT, просто использует описанную ниже команду `iptables`. Для этого следует сначала убедиться, что модуль ядра `iptables_nat` загружен, а затем включить IP-пересылку:

Listing 10. Включение маскарадинга

```
% modprobe iptables_nat    # Загрузка модуля ядра
% iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
% echo 1 > /proc/sys/net/piv4/ip_forward   # Включение марштуризации
```

Эта способность называется NAT источника -- адрес исходящего пакета изменен. Также существует NAT назначения (DNAT), позволяя осуществлять пересылку порта, разделение загрузки, и прозрачное проксирова. В этих случаях, входящие пакеты изменяются так, чтобы достигнуть требуемого хоста или подсети.

Но в большинстве случаев, когда пользователи или администраторы говорят о NAT, они имеют ввиду NAT источника. Если хотите настроить NAT назначения, следует указать `PREROUTING`, а не `POSTROUTING`. При DNAT, пакеты преобразовываются перед маршрутизацией.

Обеспечение безопасности FTP-серверов

FTP-сервера

Для Linux существует множество различных FTP-серверов, и разные дистрибутивы предлагают разные сервера. Естественно, настройки различных серверов различаются, хотя большинство склоняется использовать похожие конфигурационные директивы.

Популярным FTP-сервером является vsftpd (Very Secure FTP daemon). ProFTP также широко используется, равно как wu-ftpd и ncftpd.

Для многих целей FTP, в сущности, и не нужен. Например, безопасную передачу файлов для пользователей, имеющих учетные записи на машине сервера, можно часто совершать при помощи scp (secure copy), которая основывается на SSH, но в использовании очень похожа на знакомую команду cp.

Файл конфигурации для vsftpd -- это /etc/vsftpd.conf. Другие FTP-сервера используют

похожие файлы.

Конфигурационные опции FTP

Существует несколько опций для /etc/vsftpd.conf (и, возможно, вашего сервера, если у вас другой), которые следует помнить:

- `anonymous_enabled`: Позволяет анонимным пользователям входить, используя имена пользователей "anonymous" или "ftp".
- `anon_mkdir_write_enable`: Позволяет анонимным пользователям создавать каталоги (внутри открытых всем для записи родительских каталогов).
- `anon_upload_enable`: Позволяет анонимным пользователям закачивать файлы.
- `anon_world_readable_only`: По умолчанию "YES"; вряд ли менять его -- хорошая идея. Позволяет анонимному пользователю FTP считывать только к файлы, помеченные как общедоступные для чтения.
- `chroot_list_enable`: Помещает пользователей (перечисленных в /etc/vsftpd.chroot-list) в "тюрьму chroot" в их домашнем каталоге (они не смогу просмотреть файлы и каталоги выше в иерархии файловой системы, даже если имеют на них права доступа).
- `ssl_enable`: Поддерживает SSL-шифрованные соединения.

Ознакомьтесь с man-страницами для вашего FTP-сервера за более сложными опциями. Вообще говоря, запуск FTP-сервера настолько прост, насколько таковым является выбор конфигурационного файла и запуск сервера из сценариев инициализации системы.

Безопасная оболочка (SSH)

Клиент и сервер

Почти каждая Linux-машина (да и большинство других опреационных систем) должны иметь клиента безопасной оболочки (SSH). Часто используется версия OpenSSH, но также иногда применяют и других совместимых SSH клиентов. SSH клиент является необходимым для соединения, большие проблемы с безопасностью возникают при неправильной настройке SSH-сервера.

Поскольку клиент запускает соединение с сервером, ему часто приходится доверять серверу. Просто наличие SSH-клиента не дает никакого доступа внутрь машины; следовательно, это не может представлять опасности.

Настройка сервера тоже не особенно сложная; демон сервера был разработан содержащим и реализующим хорошую схему безопасности. Но, очевидно, именно сервер разделяет ресурсы с клиентами, основываясь на запросах от клиентов,

которым сервер решает доверять.

Протокол SSH имеет две версии, версию 1 и версию 2. В современных системах всегда предпочтительным является использования протокола версии 2, но, вообще говоря, как клиент, так и сервер поддерживают обратную совместимость с версией 1 (если только если это не отключено конфигурационными опциями). Это позволяет соединяться с все реже и реже встречающимся системами версии 1.

Протоколам версий 1 и 2 соответствуют несколько различные конфигурационные файлы. В протоколе версии 1 клиент сначала создает с помощью ssh-keygen пару ключей RSA, и хранит закрытый ключ в $HOME/.ssh/identity, а открытый ключ -- в $HOME/.ssh/identity.pub. Точно такой же identity.pub должен быть добавлен к удаленным файлам $HOME/.ssh/authorized_keys.

Очевидно, здесь присутствует проблема яйца и курицы: как можно скопировать файл на удаленную систему, до получения доступа к ней? К счастью, SSH также поддерживает метод аварийной идентификации, состоящий в отправке зашифрованных на лету паролей, которые вычисляются посредством обычного контроля входа на удаленную систему (таких как существует ли учетная запись пользователя, и введен ли правильный пароль).

Протокол 2 поддерживает как ключи RSA, так и DSA, но RSA-идентификация немного улучшена по сравнению с протоколом 1. В протолоке 2, закрытые ключи хранятся в $HOME/.ssh/id_rsa и $HOME/.ssh/id_dsa. Протокол 2 также поддерживает некоторое количество алгоритмов экстра конфиденциальности и целостности: AES, 3DES, Blowfish, CAST128, HMAC-MD5, HMAC-SHA1, и так далее. Сервер можно настроить как на предпочтительные алгоритмы, так и задать проядок использования аварийных режимов.

Общие конфигурационные опции, в отличие от ключа, содержатся у клиента в /etc/ssh/ssh_config (или, если это доступно, в /$HOME/.ssh/config). Параметры клиента можно также настроить с помощью опции `-o`; часто используемой опцией является `-X` или `-x`, для включения или выключения переадресации X11. Если она активирована, порт X11 тунеллируется через SSH, чтобы осуществлять шифрованные X11 соединения.

Утилиты, подобные `scp` также используют похожий порт для переадреации через по SSH. Пусть, например, я работаю за локальной машиной, и я могу запустить увидеть на дисплее X11 приложение, которое работает только удаленно (в этом случае -- на сервере в локальной подсети):

Listing 11. запуск удаленного приложения X11

```
$ which gedit  # на локальной системе отсутствует
$ ssh -X dqm@192.168.2.2
Password:
Linux averatec 2.6.10-5-386 #1 Mon Oct 10 11:15:41 UTC 2005 i686
GNU/Linux
No mail.
Last login: Thu Feb 23 03:51:15 2006 from 192.168.2.101
dqm@averatec:~$ gedit &
```

Настройка сервера

Демон `sshd`, что особенно характерно для версии OpenSSH, включает безопасные шифрованные соединения между двумя ненадежными хостами по небезопасной сети. Основной `sshd` сервер обычно запускается во время инициализации системы и ждет сигнала от подключений пользователей, запуская новый экземпляр демона для каждого пользовательского соединения. Отделенные демоны осуществляют обмен ключами, шифрование, идентификацию, выполнение команд и обмен данными.

Сервер `sshd` поддерживает большое разнообразие опций командной строки, но обычно настраивается файлом /etc/ssh/sshd_config. Также используется некоторое число других конфигурационных файлов. Например, файлы контроля доступом /etc/hosts.allow и /etc/hosts.deny являются предпочтительными. Ключи хранятся похожим образом на стороне клиента, в /etc/ssh/ssh_host_key (протокол 1), /etc/ssh/ssh_host_dsa_key, /etc/ssh/ssh_host_rsa_key, а открытые ключи -- в /etc/ssh/ssh_host_dsa_key.pub и подобных. Также, на клиенте, для генерации ключей используется `ssh-keygen`. Обратитесь к man-страницам для `sshd` и `ssh-keygen` за деталями конфигурационных файлов и копирования сгенерированных ключей в соответствующие файлы.

Множество конфигурационных опций имеется в /etc/ssh/sshd_config, а значения по умолчанию, вообще говоря, чувствительны (и ощутимо безопаснее). Стоит упомянуть несколько опций:

- `AllowTcpForwarding` включает или выключает переадресацию портов, и по умолчанию выставлено "YES".
- `Ciphers` управляет списком и порядком шифровательных алгоритмов, которые будут использовться.
- `AllowUsers` и `AllowGroups` принимают шаблоны ввода и позволяет контролировать, какие пользователи могут сделать попытку в дальнейшей идентификации.
- `DenyGroups` и `DenyUsers` действуют. как и следовало ожидать, симметрично.
- `PermitRootLogin` позволяет пользователю root подключаться через SSH.
- `Protocol` позволяет указать, принимаются ли оба типа протоколов (и, если нет, (какой из них).
- `TCPKeepAlive` хороша, если у вас иногда обрываются SSH-соединения. Сообщение "keepalive" рассылается, чтобы проверить соединения, если опция включена. Но это может вызвать рассоединение, если при маршрутизации происходят редкие ошибки.

SSH тунеллирование

OpenSSH позволяет создать туннель для инкапсулирования других протоколов в шифрованном SSH-канале. Эта возможность включается на сервере sshd по

умолчанию, но может быть отключена опциями командной строки или конфигурационными файлами. Полагая, что эта возможность включена, клиенты могут легко эмулировать любой порт/протокол, который пожелают, и использовать его для соединения. Например, для создания туннеля для telnet:

Listing 12. Прокладка туннеля для telnet

```
% ssh -2 -N -f -L 5023:localhost:23 user@foo.example.com
% telnet localhost 5023
```

Конечно же, этот пример бессмысленный, поскольку командная оболочка SSH делает то же в качестве оболочки shell. Однако можно создавать соединения POP3, HTTP, SMTP, FTP, X11, и по любому другому протоколу аналогичным образом. Основная идея состоит в том, что определенные порты локального хоста ведут себя так будто это есть удаленная служба с реальными коммуникационными пакетами, гуляющими в SSH соединении в зашифрованном виде.

Опции, которые мы использовали в примере, таковы:

- `-2` (использовать протокол 2),
- `-N` (нет команды/только туннель),
- `-f` (SSH фоновым), и
- `-L`, (описать туннель как "localport:remotehost:remoteport").

Также указывается сервер и имя пользователя.

TCP-wrappers

Что такое "TCP-wrappers"?

Первое, что следует уяснить про TCP-wrappers -- это то, что их не надо использовать, и они не являются активно развивающимися. Однако вы можете обнаружить, что демон tcpd из TCP-wrappers все еще выполняется в вашей системе. В свое время это было хорошим приложением, но теперь его функциональность сильно упала по сравнению с iptables и другими приложениями. Главной целью TCP-wrappers осталось отслеживание и фильтрация входящих запросов от SYSTAT, FINGER, FTP, TELNET, RLOGIN, RSH, EXEC, TFTP, TALK, и других сетевых служб.

TCP-wrappers можно настроить двумя путями. Один состоит в замене других служб на tcpd, обеспечивая аргументы, для передачи контроля определённому приложению после того, как tcpd выполнил свое журналирование и фильтрацию. Другой метод оставляет сетевых демонов в покое, но изменяет конфигурационный файл inetd. Например, такая строка:

```
tftp dgram udp wait root /usr/etc/tcpd in.tftpd -s /tftpboot
```

приведет к тому, что входящий запрос tftp запустится через оберточную программу

(tcpd) под именем процесса `in.tftpd`.

Задачи по безопасности

Чтобы выполнить поставленную цель - изучить методы обеспечения безопасности - надо изучить огромное количество вопросов (и каждый из них может оказаться принципиально важным). Поэтому я не надеюсь полностью раскрыть эту тему на страницах небольшого учебника. Всвязи с этим я рекомендую уделить время знакомству с ресурсами и утилитами, перечисленными в этом разделе.

web-сайты, на которые стоит заглянуть за статьями по безопасности и патчами:

- Security focus news: The Security Focus web-сайт -- один из лучших сайтов по докладам и обсуждению проблем безопасности и конкретных уязвимостей. Сайт содержит несколько новостных групп и объявлений, на которые можно подписаться, равно как и статьи общего характера и сообщения об ошибках, по которым можно выполнять поиск.
- The Bugtraq mailing list: Обширный модерируемый список рассылки для подробных обсуждений и объявлений уязвимостей в системах компьютерной безопасности: кто они, как с ними бороться, как их обнаружить.
- CERT Coordination Center: размещенный на сервере Carnegie Mellon University, CERT имеет совещательный коллектив, схожий с Security Focus site, но имеет больший уклон в сторону пособий и руководств. Обозревать несколько таких сайтов -- хороший способ убедиться, что вы в курсе всех проишествий, влияющих на вашу ОС, дистрибутив или определенные программы или серверы.
- Computer Incident Advisory Capability: Информационные бюллетени CIAC распространяются сообществом Department of Energy для извещения сайтов об уязвимостях в компьютерной безопасности и рекомендуемых действиях. Пободно тому, консультативные записки CIAC служат для предупреждения сайтов об опасной, уязвимости, требующей срочного решения, и разрешения этих проблем. Технические бюллетени CIAC охватывают вопросы технической безопасности и анализы, не столь чувствительные ко времени.

Утилиты для слежения за безопасностью, которые стоит запускать, таковы:

- Open Source Tripwire: Утилита. касающаяся безопасности и целостности данных, отслеживающая определенные изменения в файлах.
- `scanlogd` : Утилита, обнаруживающая сканирование TCP портов.
- Snort: Обнаружение и предотвращение вторжения в сеть, с помощью языка правил; использует методы, основанные на подписях, протоколах и аномалиях.