Understanding of CGI Security Issues

После того как пользователь протестировал и отладил свой CGI скрипт и успешно запустил его в первый раз, ему, вероятно, захочется немедленно выложить его на свой сайт. Он несомненно будет доволен тем, что сделал и захочет, чтобы мир увидел его работу.

Подобные действия, хоть и оправданы, но могут быть опасными. Поскольку в мире существуют разные люди, занимающиеся в том числе и взломом, то, наверняка, кто-нибудь из них захочет взломать и ваш сайт. Хотя вероятность этого очень мала, но у вас может появиться много проблем.

Мстительные хакеры хорошо знакомы с компьютерами и особенно с интернетом, и, хотя большинство ВЕБ серверов специально запрограммированы так, чтобы быть защищенными от их хитрых приемов, отдельная ошибка безопасности в CGI скрипте может привести к тому, что они получат полный доступ к вашему компьютеру: к вашим файлам с паролями, вашей личной информации и т.д.

Если следовать небольшому количеству простых правил и быть всегда начеку, то можно сделать скрипты стойкими к атаке, что позволит вам спать спокойно по ночам.

Прочитав этот документ, вы узнаете:

- Преимущества и недостатки скриптов перед программами.
- Как безопасно отобразить пользовательские данные
- Как правильно запускать внешние программы
- Как защитить ваши скрипты от локальных пользователей
- Насколько опасно использовать чужие CGI скрипты.

Скипты против Программ

Когда вы садитесь, для того чтобы написать скрипт, существует несколько соображений, исходя их которых следует выбирать какой язык использовать. Одно из этих соображений – безопасность.

Shell скрипты, Perl и Си программы это наиболее часто используемые формы представления CGI скриптов, и каждая из которых имеет свои преимущества и недостатки, когда во внимание принимается еще и безопасность. Ни одна из них не является наилучшей, хотя в зависимости от других факторов каждая из них может иметь место.

Shell скрипты используются обычно для маленьких, быстрых и почти всегда одноразовых CGI программ, и как следствие почти всегда пишутся, не заботясь о безопасности. Такая небрежность может привести к «дырам», которыми могут воспользоваться даже те, кто обладает только общими знаниями о вашей системе.

Хотя и shell CGI программы часто легки в написании, их зачастую очень трудно бывает проконтролировать, поскольку их работа заключается в исполнении других внешних программ. Это может привести к нескольким возможным ловушкам, поскольку CGI скрипт полностью наследует все проблемы безопасности программ, которые он использует.

Например, наиболее часто используемая UNIX-утилита awk имеет довольно ограниченные пределы данных, которые она может обработать. И если в CGI используется awk, то все эти ограничения касаются и скрипта.

Perl это следующая ступенька после shell скриптов. Он обладает множеством преимуществ и достаточно безопасен. Но Perl достаточно гибкий, и может показаться что он недостаточно защищен.

Например, Perl – это интерпретатор. Это означает, что он компилируется и исполняется на каждом шаге, каждый раз при запуске. Это дает возможность данным, введенным, пользователем, стать частью исполняемого кода, неправильно быть интерпретированными и привести к сбою.

И наконец существует Си. Язык Си быстро стал стандартом для разработки приложений, и практически все UNIX и Windows NT были разработаны на нем. Может показаться, с точки зрания безопастности, что Си это то что нужно, пока вы не столкнетесь напрямую с этими проблемами, которые очень популярны и хорошо всем известны.

Например, Си очень плохо обрабатывает строки. Он не осуществляет автоматического выделения и очищения памяти для них, предоставляя возможность программистам следить за этим самим. Большинство программистов на Си, при использовании строк, просто устанавливают им заранее определенную длину, надеясь, что этого хватит с запасом, что бы ни ввел пользователь. Это, естественно, может оказаться очень небезопасным. Существет много примеров того, как были написаны эксплойты, основанные на переполнении буфера, с помощью которых осуществлялись дальнейшие взломы.

Конечно, shell скрипты, Perl и Си это далеко не полный список языков программирования, которые могут быть использованы для написания CGI скриптов. Единственное ограничение – это способ взаимодействия с ВЕБ сервером. На UNIX и Windows NT серверах, данные предоставляются скриптам через переменные окружения и стандартный поток ввода, так что любая программа, которая умеет читать данные из этих двух потоков данных и писать в стандартный поток вывода, может быть использована для написания CGI: awk, Fortran, C++, Basic и другие. Программисты под Windows могут использовать популярный Visual Basic.

Но из всех перечисленных shell, Perl и Си наиболее популярны. Это не означает что вы обязаны использовать именно их, но большиство библиотек уже написаны именно на этих языках, и ими пользоваться просто удобнее.

Не доверяй никому

Практически все «дыры» в безопасности CGI возникают при взаимодействии с пользователем. Это взаимодействие происходит через формы или через пути к файлам, что не только дает силу CGI скриптам, но также делает их потенциально опастной частью в работе ВЕБ серверов.

Написание безопастных CGI скриптов - это в значительной степени тренировка в творчестве и паранои. Вы должны творчески обдумать все способы, какими пользователь обдуманно или другим способом может передать вам данные, которые потенциально могут вызвать проблемы. И вы должны предусмотреть все возможные способы, поскольку пользователи непредсказуемы.

Два способа нарушения

Когда пользователи заходят на ваш ВЕБ сайт и начинают взаимодействовать с ним, это может принести вам головную боль двумя способами. Один состоит в том, чтобы не следовать установленным правилам, изгибая или ломая каждое ограничение или пределы, которые вы установили на вашей страничке. Второй - в исполнении того, что вы попросили сделать их сами.

Большинство CGI скриптов работают как внутренний интерфейс к HTML формам, обрабатывая информацию, введенную пользователями, для обеспечения осмысленного ответа. Поскольку это имеет место, большинство CGI скриптов написаны так, что ожидают данные в определенном особом формате. Они предполагают, что данные, введенные пользователем, будут соответствовать тому, что они ожидают. Но это, порой, не всегда так. Пользователь может обойти эти ограничения различными путями, посылая скрипту на вид случайную информацию. Ваш скрипт должен быть готов к этому.

Во-вторых, пользователи могут послать CGI скрипту данные в точности того типа, как и ожидается, заполняя все необходимые поля в форме в нужном скрипту формате. Такое поведение может быть у невинного пользователя, который взаимодействует с вашим сайтом так как вы и предполагали, или это может быть злобный хакер, использующий свои знания вашей операционной системы и ВЕБ-сервера для того, чтобы воспользоваться преимуществом общих ошибок CGI программирования. Подобные атаки, где все выглядит как и должно быть, наиболее опасны и трудны в выявлении. И безопасность вашего ВЕБ-сайта зависит от их предупреждения.

Не доверяйте данным формы

Одна из наиболее общих ошибок, возникающих в CGI программировании – доверие данным, которые передаются скрипту через форму. Достаточно непослушные пользователи непрочь найти некоторое количество способов послать данные, которые вы никогда не ожидаете, полагая, что это невозможно. Все ваши скрипты должны принимать это во внимание. Например, каждая из ситуаций ниже, и много других, подобных возможны:

- Выбор из группы радио кнопок на форме нескольких вариантов, когда предусмотрен один.
- Длина строки текстового поля может быть больше допустимого значения, определяемого полем `MAXLENGTH`.
- Имена переменных сами по себе могут не соответствовать тем, которые были указаны в форме.

Откуда появляются плохие данные

Эти ситуации могут иметь место в нескольких случаях, некоторые случайные, некоторые нет. Например, ваш скрипт может получить данные, которые он не ожидал, потому что кто-то еще написал форму, которая требует для ввода данные совершенно отличающиеся от тех, которые были описаны в вашей форме, и непредумышленно указал FORM ACTION на ваш CGI скрипт. Возможно он использовал вашу форму как образец, для написания своей собственной, и забыл исправить ACTION URL, прежде чем начать тестировать её. Это приведет к тому, что ваш скрипт получит данные, и не будет знать что с ними делать, это может вызвать его небезопасное поведение.

Следующий код демонстрирует форму, которая посылает ненужные данные CGI скрипту, который осуществляет поиск по базе данных Yahoo. Скрипт хорошо написан и защищён, потому что он игнорирует данные, которые не может распознать.

```
<FORM METHOD="POST" ACTION="http://search.yahoo.com/bin/search">
      Enter your name, first then last:
      <INPUT TYPE="TEXT" NAME="first">
      <INPUT TYPE="TEXT" NAME="last">
</FORM>
```

Возможно пользователь случайно или нарочно исправил URL на ваш CGI скрипт. Когда браузер отправляет данные CGI программе, он просто добавляет данные, введенные в форме к URL скрипта (в случае метода GET) также, как и пользователь вводит адрес желаемой странички в браузере, он может легко изменить данные которые будут посланы скрипту.

Например, когда вы в форме нажимаете кнопку отправить, Netscape переместит в адресную строку URL CGI скрипта и следом за ней строку данных через символ “?”, которая будет представлять собой совакупность пар ИМЯ и ЗНАЧЕНИЕ, определенных в форме. Если вам угодно, вы можете свободно изменить эту строку данных, а также добавить туда новые пары.

И наконец честолюбивый хакер может написать программу, которая производит соединение с ВЕБ сервером и без посредничества WEB-браузера. Такая программа будет способна производить действия нехарактерные для обычного браузера, например, послать сотни мегабайт данных скрипту. Что произойдет, если количество данных, передаваемых скрипту будет неограничено ? Возможно произойдет сбой, который предоставит несанкционированный доступ к системе.

Борьба с плохими данными

Вы можете противостоять отправке вашему скрипту плохих данных несколькими способами. И при написании своих скриптов вы должны использовать любые или все из них.

Во-первых, ваш скрипт должен установить разумные пределы на размер данных, которые он собирается принимать, в равной степени для каждой пары ИМЯ \ ЗНАЧЕНИЕ. Если скрипт обрабатывает, например, POST метод, то необходимо проверить значение переменной окружения CONTENT_LENGTH, для того чтобы убедиться, что в данных содержиться что-то, что он можете ожидать. Если единственная функция скрипта – это получение имени пользователя, то было бы неплохо выдать ошибку, если CONTENT_LENGTH больше, скажем, 100 байт. Очень маловероятно, что длина имени будет больше чем 100 символов, и подобным ограничением можно защитить скрипт от слепого считывания неверных данных.

Замечание
Вам не надо заботиться об ограничении данных при отправке их методом GET, поскольку этот метод по стандарту не может послать данных более одного килобайта. Сервер автоматически ограничит размер данных и выставит длину передаваемой строки в переменной окружения `QUERY_STRING`.

Следующий шаг – это удостовериться, что скрипт знает что делать в случае, если он не может распознать вид полученных данных. Если, например, в форме пользователю предлагается сделать выбор между двумя радио-кнопками, скрипт не должен полагать, что, если не нажата первая из них, то нажата другая. Нижеследующий пример Perl-кода демострирует эту ошибку.

```
if ($form_Data{"radio_choice"} eq "button_one")
{
      # Button One has been clicked
}
else
{
      # Button Two has been clicked
}
```

Этот код содержит ошибку, поскольку полагает, что если при выборе из двух не выбрано первое, то обязательно выбрано второе. Однако это не всегда так. Хоть и в приведенном куске кода все выглядит безобидно, в некоторых случаях это может быть опасным.

Скрипт должен предусмотреть подобную ситуацию и обработать ее соответственно. И в случае неожиданной ситуации, можно выдать ошибку, как это сделано в нижеследующем примере:

```
if ($form_Data{"radio_choice"} eq "button_one")
{
      # Button One selected
}
elsif ($form_Data{"radio_choice"} eq "button_two")
{
      # Button Two selected
}
else
{
      # Error
}
```

Путем добавления второй конструкции ЕСЛИ-условие-ТО для проверки значения “radio_choice” на значение “button_two” скрипт теперь не делает никаких предположений.

Например, следующий Си-код проверяет введенный текст на принадлежность к нескольким возможным вариантам, и устанавливает значение по-умолчанию, в случае, если не находит ни одного соответствия. Это может быть использовано для того, чтобы лучше объяснить пользователю, что именно от него ожидается.

```
if ((strcmp(help_Topic,"how_to_order.txt")) &&
    (strcmp(help_Topic,"delivery_options.txt")) &&
    (strcmp(help_Topic,"complaints.txt")))
{
      strcpy(help_Topic,"help_on_help.txt");
}
```

С другой стороны, скрипт может лучше попытаться исправить ошибки пользователя, чем просто выдать сообщение об ошибке или установить значение по-умолчанию. Если в форме пользователю предлагается ввести секретное слово, скрипт должен автоматически отбросить любые пробелы из строки ввода, прежде чем производить сравнение. Следующий фрагмент на Perl делает это.

```
$user_Input =~ s/\s//;

# Remove white space by replacing it with an empty string
if ($user_Input eq $secret_Word)
{
      # Match!
}
```

И наконец можно объединить обработку нескольких форм в один скрипт. В этом случае необходимо предвидеть все возможные варианты ввода для каждой формы и производить соответствующие проверки.

Например, если в форме для отправки данных CGI скрипту используется метод POST, то это означает, что данные буду переданы скрипту через стандартный поток ввода. Для того, чтобы проверить каким методом были отправлены данные и откуда их ожидать, необходимо проверить значение переменной окружения REQUEST_METHOD, которая будет содержать слово POST или GET соответственно. Правильно написанные скрипты должны уметь получать данные как при использовании метода POST, так и GET. И результат их работы не должен зависеть от способа получение данных, что делает их более защищенными. Ниже приведен пример скрипта на Perl, который читает данные как из стандартного потока ввода, так и при отправке данных методом GET.

```
# Takes the maximum length allowed as a parameter
# Returns 1 and the raw form data, or "0" and the error text
sub cgi_Read
{
      local($input_Max) = 1024 unless $input_Max = $_[0];
      local($input_Method) = $ENV{'REQUEST_METHOD'};

      # Check for each possible REQUEST_METHODs
      if ($input_Method eq "GET")
      {
            # "GET"
            local($input_Size) = length($ENV{'QUERY_STRING'});

            # Check the size of the input
            if ($input_Size > $input_Max)
            {
                  return (0,"Input too big");
            }

            # Read the input from QUERY_STRING
            return (1,$ENV{'QUERY_STRING'});
      }
      elsif ($input_Method eq "POST")
      {
            # "POST"
            local($input_Size) = $ENV{'CONTENT_LENGTH'};
            local($input_Data);

            # Check the size of the input
            if ($input_Size > $input_Max)
            {
                  return (0,"Input too big");
            }

            # Read the input from stdin
            unless (read(STDIN,$input_Data,$input_Size))
            {
                  return (0,"Could not read STDIN");
            }
```

```
            return (1,$input_Data);
        }

        # Unrecognized METHOD
        return (0,"METHOD not GET or POST");
}
```

Подводя итог, скрипт не должен делать никаких предположений о типе и форме данных, которые он получает. Надо ожидать неожидаемого, несмотря на противоречие в терминах, и обрабатывать их каким-любо образом. Провести тестирование скрипта всевозможными способами, прежде чем начать его использование; не принимать плохие данные и выдавать сообщение об ошибке, автоматически устанавливать значение по-умолчанию, если чего-то недостает или эти данные были введены неправильно; попытаться раскодировать введенные данные, которые, например, чувствительны к регистру.

Не доверяйте данным о путях

Другой тип данных, которые пользователь может изменить – это серверная переменная окружения PATH_INFO. Эта переменная содержит любую информацию о пути, соответствующей URL CGI. Например, если foobar.sh – это CGI shell скрипт, и URL http://www.server.com/cgi-bin/foobar.sh/extra/path/info, тогда в PATH_INFO будет помещено /extra/path/info при запуске foobar.sh

И если скрипт использует эту переменную окружения, то нужно проверять достоверность ее содержимого. Поскольку данные от формы могут быть быть изменены любым способом, то с таким же успехом может быть изменено и PATH_INFO. И скрипт, который вслепую использует значение PATH_INFO позволит злобным пользователям причинить ущерб на сервере.

Например, если CGI скрипт разработан для того, чтобы вывести содержимое файла, на который ссылается PATH_INFO, тогда пользователь сможет прочитать содержимое практически любого файла на вашем компьютере, как это продемонстрировано в следующем скрипте.

```
#!/bin/sh

# Send the header
echo "Context-type: text/html"
echo ""

# Wrap the file in some HTML
#!/bin/sh
echo "<HTML><HEADER><TITLE>File</TITLE></HEADER><BODY>"
echo "Here is the file you requested:<PRE>\n"
cat $PATH_INFO
echo "</PRE></BODY></HTML>"
```

Хотя этот скрипт хорошо работает только в случае, если пользователь кликает на определенные заранее ссылки, скажем http://www.server.com/cgi-bin/foobar.sh/public/faq.txt Более смышленый пользователь может запросить любой файл на сервере. И если запросить http://www.server.com/cgi-bin/foobar.sh/etc/passwd, тогда приведенный скрипт успешно выдаст файл паролей, что очень нежелательно и крайне опасно.

Более безопасно использовать переменную окружения PATH_TRANSLATED. В ней автоматически формируется путь из корня ВЕБ-сервера плюс содержимое PATH_INFO, и считается, что все файлы, на которые ссылается PATH_TRANSLATED, доступны для браузера и без посредства скрипта и считаются безопасными.

В одном случае, однако, файлы могут быть недоступны через браузер, но могут быть доступны, если PATH_TRANSLATED используется в CGI скрипте. И вы должны знать о таких случаях.

Файл .htaccess, который может существовать в поддиректориях, контролирует доступ к файлам, лежащим в этих директориях. Он может быть использован, например, для ограничения доспутности группы ВЕБ-страниц для сотрудников компании.

Несмотря на то, что сервер понимает, как интерпретировать файл .htaccess и может определить, кому позволено просматривать ВЕБ-страницы, а кому нет, CGI скрипт об этом не знает. И если скрипт использует PATH_TRANSLATED для доступа к файлам, в директории, которая защищена .htaccess, то тем самым можно обойти защиту сервера.

Все бы казалось хорошо, НО...

Теперь, после того, как были обрисованы способы, которыми пользователь может передать скрипту данные, которые не ожидались, можно занять более серьезным вопросом. Как проверить достоверность данных, которые пользователь послал.

В большинстве случаев, корректно, но с умом отправленные данные из формы, могут принести больше неприятностей, чем выходящие за границы допустимого объемы данных. Легко проигнорировать бессмысленный ввод пользователя, но определить неподдельный, правильно сформированный ввод – трудная задача.

Поскольку CGI скрипты настолько гибки, что могут делать практически все, что может компьтер, маленькая брешь в безопастности может привести к очень серьезным последствиям.

Обработка имен файлов

Имена файлов, например, это пример данных, которые могут быть посланы CGI скрипту и вызвать бесконечное множество проблем, если не быть аккуратным.

Каждый раз, когда вы пытаетесь открыть файл, имя которого было получено от пользователя, вы должны тщательно проверить это имя на наличие в нем каких-либо трюков. Если вы спрашиваете у пользователя имя файла, и, затем, пытаетесь открыть его без каких либо проверок – у вас могут возникнуть большие неприятности.

Например, что будет, если пользователь ввел имя файла, которое содержит элементы пути, такие как слэши и двойные точки ? Хотя вы и ожидаете простое имя файла, скажем file.txt, можно получить и /file.txt, и ../../../file.txt. В зависимости от того, как настроен ВЕБ сервер и что вы собираетесь сделать с принятым именем файла, теоретически можно выдать любой файл умному взломщику.

Далее, что если пользователь введет имя уже существующего файла или имя файла, важного для запуска системы ? Что если введенное имя это /etc/passwd или

c:\winnt\system32\krnl32.dll ? В зависимости от того, что делает скрипт, эти фалы могут быть либо отправлены взломщику, либо перезаписаны, либо повреждены.

Под операционной системой Windows 95 и Windows NT, если не проверять на наличие обратных слэшей, то можно позволить браузеру получить доступ к файлам, которые надохятся даже не на ВЕБ машине.

Что произойдет, если пользователь введет недопустимые символы в имени файла ? Под операционной системой UNIX любой файл начинающийся с точки становится невидимым. Под Windows оба типа слэшей являются разделителями директорий. Можно написать программу на Perl неаккуратно, что позволить внешним программам исполняться, если при открытии файла, имя начинается с pipe. Даже специальные управляющие символы могут присутствовать в имени, если пользователь знает как.

Еще хуже с shell скриптами, в которых точка с запятой разделяет команды. И если скрипт выполняет команду cat, а пользователь набрал file.txt; rm –rf / в качестве имени фала, тогда помимо распечатывания файла произойдет очистка всего жесткого диска, без запроса на подтверждение.

In with the Good, Out with the Bad

Для того, чтобы избежать всех этих проблем и закрыть все потенциальные дыры, необходимо просматривать каждое имя файла, которое было получено от пользователя.

Наилучший способ состоит в посимвольном сравнении имени файла со списком приемлемых символов, и возвращать ошибку, в случае несоответствия. Это гораздо безопаснее, чем искать по списку всех недопустимых символов.

Пример, приведенный ниже производит такую проверку на Perl. Он допускает наличие любых букв верхнего и нижнего регистра, любые числа, подчеркивание и точку. Также он проверяет, что имя файла не начинается с точки. Таким образом, этот пример не допускает наличие слэшей, для смены директории, точки с запятой и pipe.

```
if (($file_Name =~ /[^a-zA-Z_\.]/) || ($file_Name =~ /^\./))
{
  # File name contains an illegal character or starts with a period
}
```

Обработка HTML

Другой тип на-вид безобидного ввода от пользователя может принести неприятности, в случае, если он содержит HTML. Пусть, например, скрипт запросил у пользователя имя, а затем выводит приветствие, обращаясь к пользователю по имени. Пример на Perl ниже просто выводит приветствие.

```
print("<HTML><TITLE>Greetings!</TITLE><BODY>\n");
print("Hello, $user_Name!  It's good to see you!\n");
print("</BODY></HTML>\n");
```

Представьте, что произойдет, если помимо имени пользователь введет <hr><h1> John Smith </h1><hr> очевидно результат будет не таким, как вы желаете.

Но вводя HTML в данные, пользователь может не только изменить внешний вид страницы. Представьте, что он ввел <img src=”/secret/project/cutekid.gif”> вместо своего имени. Если будет использоваться скрипт, описанный выше, то потенциальные хакер сможет увидеть секретную картинку.

Если сервер поддерживает включения, то пользователь может ввести <!--#include file=”/secret/project/plan.txt”--> вместо своего имени, и увидит целиком текст вашего секретного плана. Или он может ввести <!--#include file=”/etc/passwd”--> для того, чтобы получить доступ к файлу с паролями. И в худшем случае хакер может ввести `<!-- #exec cmd="rm -rf /" -->` вместо своего имени и тогда все данные на диске будут уничтожены.

Есть два способа удаления HTML из данных, полученных от пользователя.

- Быстрый и небрежный способ – это удалить все символы < > из строки. Сделать это можно с помощью команты на Perl, приведенно ниже.

```
$user_Input =~ s/<>//g;
```

- Более правильный способ, это не удалять эти символы, а заменять их на их коды. Знак меньше заменяется на < а знак больше заменяется на > Ниже приведен пример на Perl.

```
$user_Input =~ s/</&lt;/g;
$user_Input =~ s/>/&gt;/g;
```

Заключение

В любом случае при исполнении скрипта необходимо позаботиться о том, чтобы любые данные полученные от пользователя, были предварительно обработаны и проверены на ниличие недопустимых символов, на размер этих данных, и в случае обнаружения несоответствия выдавать ошибку. Это поможет защититься от злобных хакеров.