Эссе по курсу "Защита информации",
кафедра радиотехники,
Московский физико-технический институт (МФТИ ГУ)
http://www.re.mipt.ru/infsec

Обзор уязвимостей CGI

(The review of CGI vulnerabilities)

Студент: Калинченко Максим
Группа: 115

Преподаватель: Булыгин Юрий

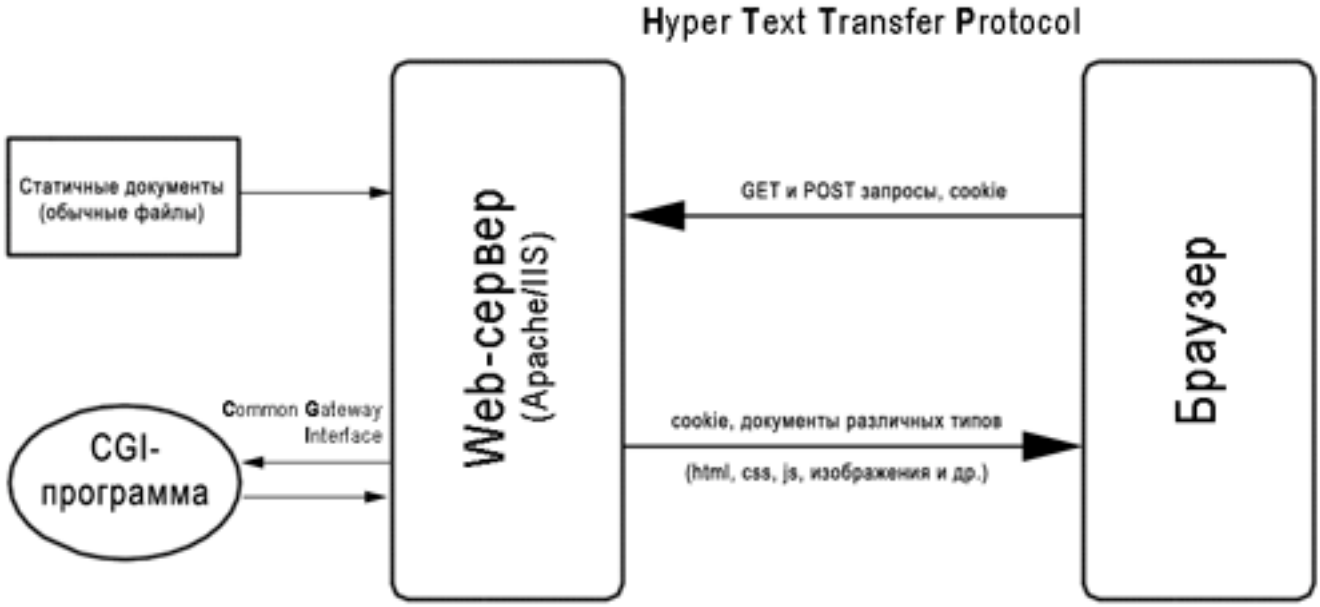
2005 г.

Содержание

Введение

Веб-сайт может содержать как статичные документы, так и динамичные, будь то html-форматированный текст, картинки или документ любого другого типа. Содержание статичного документа хранится в файле на сервере и в неизменном виде передается клиенту. Содержание же динамичного документа создается программой, выполняемой на стороне сервера, и естественно может изменяться в зависимости от различных факторов. Как правило, это параметры, передаваемые клиентской программой (браузером).

CGI (Common Gateway Interface – Общий Шлюзовой Интерфейс) – интерфейс связи веб-сервера с программой, запускаемой этим сервером и создающей динамический документ. Такая программа называется *шлюзом*. CGI подразумевает передачу параметров шлюзу посредством переменных окружения (REMOTE_ADDR, QUERY_STRING, REQUEST_URI и др.) и через STDIN (в случае POST-запроса). Созданный документ шлюз (далее CGI-программа) должен вернуть в STDOUT.



CGI-программа может быть написана на **любом** языке программирования, главное чтобы она могла быть выполнена сервером. Чаще всего используются интерпретируемые языки программирования (PHP, Perl, Python, ASP) ввиду многих причин. Это простота написания и понятность кода, отсутствие необходимости в компиляции, но в то же время чрезвычайная гибкость и достаточная для CGI мощность. Программы, написанные на интерпретируемых языках, называются скриптами. Скрипт (от англ. script - сценарий) — программа, исполняемая путём обработки нетранслированного исходного кода особой программой, которая переводит его в машинные инструкции. Файл с кодом также

называют скриптом, а программу, обрабатывающую код — интерпретатором.

Область применения CGI очень широка – это новостные ленты, гостевые книги, форумы, чаты, интернет-магазины, различные системы управления содержанием сайта (CMS, Content Management System). Нечего и говорить о последствиях обнаружения злоумышленником уязвимостей в коде CGI-программы (далее CGI-уязвимости). Это может быть как «безобидный» дефейс (от англ. deface – изменение главной страницы сайта), так и более серьезный вред типа получения злоумышленником секретных данных (пароли, номера кредитных карт, информация личного характера) или использования злоумышленником ресурсов сервера в личных целях (рассылка спама, троянов, атаки на другие сервера). Последствия обнаружения злоумышленниками CGI-уязвимостей резко возрастают, если данная CGI-программа используется на большом количестве веб-серверов (яркий пример – различные платные и бесплатные форумы).

Основные типы CGI-уязвимостей

Простейшие уязвимости

Порой бывает, что уязвимость лежит на самой поверхности и так и просит ей воспользоваться. Это такого рода уязвимости, которые требуют от злоумышленника знания лишь самых основных принципов устройства веб-сайтов. Например, хранение и передача «стратегической» информации в cookie, hidden-полях формы и/или в параметрах GET-запроса. Рассмотрим несколько примеров.

1. Система авторизации на сайте может быть устроена таким образом, что после проверки введенных логина и пароля CGI-программа устанавливает cookie с идентификационным номером пользователя (id) и в дальнейшем производит аутентификацию только на основании этой информации. Таким образом, для доступа под чужим логином злоумышленнику нужно лишь изменить идентификатор пользователя в cookie.

2. Если форма авторизации на сайте использует метод GET, то значит логин и пароль передаются как часть URL и видны в строке браузера (т.е. могут быть подсмотрены «из-за спины»). Более того, этот URL скорее всего записывается в журнал посещений браузера и в последствии может быть просмотрен другим пользователем. Или заветный URL может даже быть предложен системой автозаполнения URL-адресов – пользователю остается лишь нажать Enter! Хранение открытого пароля в cookie или hidden-полях формы тоже далеко не безопасно, но потребует от злоумышленника чуть больших навыков.

3. Представьте ситуацию, когда CGI-программа интернет-магазина использует для окончательного составления и отправки заказа цену товара или общую сумму заказа, получаемую из hidden-поля формы подтверждения заказа. В этом случае злоумышленнику нужно лишь сфабриковать запрос с выгодной для него ценой.

4. К некоторой информации на сайте (каталог товаров, новости, фотогалерея и т.п.) доступ, как правило, осуществляется по уникальному номеру (id) материала в данном разделе. Если по какой-то причине тот или иной материал (товар в каталоге, новость, фотография) не должен отображаться на сайте (или именно данному пользователю), то он помечается в базе данных как скрытый и не показывается в общем списке материалов данного раздела. Но если при запросе такого материала не производится проверка на «скрытость», то пользователь (точнее злоумышленник, хотя сложно сказать где проходит эта грань) может получить доступ к неразрешенному материалу изменив уникальный номер запрашиваемого материала.

Подобных примеров можно придумать множество, но к счастью такие ошибки в CGI-программах присущи в основном начинающим веб-программистам, которые не создают серьезных программных продуктов, требующих большой ответственности.

Cross Site Scripting (XSS)

Если на сайте имеется возможность отображения произвольного текста, введенного пользователем (например, сообщения в чате, форуме или гостевой книге), то здесь кроется целый класс уязвимостей, называемый Cross Site Scripting (межсайтовый скриптинг). Во избежание путаницы с Cascading Style Sheets (CSS), для обозначения такого рода уязвимостей применяют аббревиатуру XSS. Суть уязвимости заключается в следующем. Если злоумышленнику удастся внедрить в свой текст HTML и/или JavaScript/VBScript код (т.е. если при отображении на сайте сообщение пользователя будет интерпретировано не как текст, а как код), то он сможет исполнить на стороне жертвы (другого пользователя) свой скрипт и/или украсть (отослать к себе на e-mail или передать в свой скрипт) секретные данные пользователя. Как правило, это аутентификационный номер сессии, с помощью которого злоумышленник сможет зайти на сайт от имени жертвы.

Хотя данная уязвимость чаще всего встречается в различных расширенных возможностях чатов и псевдо-тегах (BBCode) форумов, но чатами и форумами ее область применения далеко не ограничивается. Она встречается в поисковиках, CMS, интернет-магазинах. Также очень серьезные последствия от подобных уязвимостей могут быть в публичных почтовых службах. Положение усугубляется прогрессом в области интернет-технологий. Так, флэш-ролик, внедренный злоумышленником, может исполнить произвольный JavaScript в контексте данной страницы сайта, а с помощью JavaScript'а, помимо похищения данных и мелких пакостей, можно исполнить ActiveX-объект, у которого, в отличие от скриптовых языков, есть полный доступ к компьютеру жертвы.

В настоящее время XSS является самым распространенным типом уязвимостей, как в детищах начинающих веб-программистов, так и в «профессиональных» программных продуктах. Это связано главным образом с трудностями по их выявлению. Порой встречаются совсем нетривиальные случаи, вызывающие трудности не только по выявлению, но даже и по предотвращению уязвимости.

SQL-инъекции (SQL-injection)

Также широко распространенными уязвимостями CGI-программ являются так называемые SQL инъекции. Суть уязвимости в следующем. Если CGI-программа использует для работы базу данных (а сейчас мало кто их не использует), то выборки из базы естественно зависят от переданных этой программе параметров. Это и аутентификация пользователя по логину/паролю, и получение полного текста статьи по ее уникальному номеру (id), и чтение/добавление записей в форумах и гостевых книгах. Итак, SQL инъекции – это внедрение SQL-кода злоумышленника в запрос к базе данных. Рассмотрим самый простой пример. Допустим на запрос браузера

```
http://www.***.ru/users.cgi?id=5
```

идет следующий SQL-запрос в базу:

```
SELECT * FROM users WHERE id=$id
```

Тогда передав в параметре id после значения пробел (шестнадцатеричный код %20), можно дополнить SQL-запрос своим кодом:

```
http://www.***.ru/news.cgi?id=5%20or%20admin%3D1
SELECT * FROM users WHERE id=5 or admin=1
```

Если значение параметра в SQL-запросе экранировано кавычками, но фильтрация значения на наличие кавычек не производится, то достаточно закрыть кавычку в значении параметра:

```
SELECT * FROM users WHERE login='$login' AND password='$pass'

login: "admin' OR "
password: "<>'"

SELECT * FROM users WHERE login='admin' OR ' AND password='<>''
```

Конечно не зная структуры таблицы в базе данных и полного текста SQL-зароса, злоумышленнику будет сложно составить синтаксически (и логически) правильный запрос. Но в этом ему могут здорово помочь сообщения об ошибках, возвращаемые в браузер:

```
You have an error in SQL-query:
"SELECT id, login, email, administrator WHERE login='' OR admin='..."
Unknown column 'admin' in 'where clause'
```

Также очень облегчит задачу злоумышленнику наличие исходного кода CGI-программы. Это либо свободно распространяемые веб-приложения, либо часть кода, полученная посредством другой уязвимости.

Помимо дополнения SQL-запроса своими выражениями в *WHERE*-условии, в зависимости от используемой на сервере базы данных,

можно воспользоваться более важными возможностями SQL-синтаксиса. Например, если используется MS SQL, то можно ввести несколько запросов, разделяя их точкой с запятой (";"):

```
SELECT * FROM articles WHERE id=5; DROP TABLE users
```

Также есть возможность исполнения внешних программ на стороне сервера, если MS SQL-сервер не настроен соответствующе.

В наиболее распространенном среди веб-приложений MySQL-сервере, начиная с версии 4, поддерживается синтаксис объединения нескольких запросов в один:

```
SELECT id, title, author, date FROM posts WHERE forum_id=5 UNION
SELECT id, password, login, email FROM users
```

В данном примере, список тем в открытом форуме будет дополнен новыми темами, содержащими в качестве заголовков пароли пользователей.

SQL-уязвимости могут представлять большую опасность, особенно если CGI-программа обращается к базе данных с привилегиями супер-пользователя (root). К примеру, таким образом был взломан один из крупнейших российских хостинг-провайдеров. Т.е. был получен доступ ко всем проектам, размещающимся на данном сервере! Просто форум с SQL-уязвимостью на сайте этого хостинг-провайдера обращался к базе данных от имени пользователя root.

Переполнение буфера (Buffer overflow)

Если CGI-программа написана на языке, подверженном переполнению буфера (например, Си), то эта уязвимость становится актуальной для данной CGI-программы. Суть уязвимости заключается в том, что при занесении в некоторую переменную данных, объем которых превышает выделенную под эту переменную память, "лишние" данные записываютя в "чужую" память, т.е. память, которая выделена для других целей. Рассмотрим один из способов использования данной уязвимости.

При вызове любой функции, ее данные (передаваемые параметры, локальные переменные) записываются в стек, память которого распределена так, что адрес возврата данной функции (адрес памяти, на который передается управление после завершения данной функции) записан ниже памяти, выделенной под используемые в этой функции локальные переменные. Таким образом, если в программе не выполняется проверка на объем данных, записываемый в локальные переменные, то эти данные могут выйти за пределы выделенной под них памяти и изменить адрес возврата данной функции. Таким образом, после выполнения функции можно передать управление на произвольный код, который передаем в качестве содержимого переполняемых переменных. Т.е. произвольный код можно выполнить

от имени пользователя, из под которого запущена данная CGI-программа.

Также существуют другие механизмы доступа к "чужой" памяти, как например уязвимости format string и heap overflow, но не будем заострять на этом внимание и отошлем любознательного читателя к другой литературе.

Степень опасности данного типа уязвимостей очень высока, т.к. они дают доступ не только к "секретным" данным, но и к самому серверу, ресурсы которого могут быть использованы злоумышленниками для рассылки спама, DoS-атак или в качестве анонимного прокси-сервера для взлома других серверов.

Другие уязвимости

Также существуют уязвимости, основанные на синтаксисе языка программирования, на котором написана CGI-программа. Такие уязвимости, как правило, встречаются в скриптах. Рассмотрим несколько примеров.

1. NULL-символ.

Допустим сайт устроен таким образом, что CGI-скрипт на запрос

```
http://www.***.ru/script.cgi?part=about
```

возвращает содержимое текстового файла about.txt открывая его таким образом (язык Perl):

```
open FILE, "$part.txt";
```

Таким образом, злоумышленник может получить с сервера содержимое любого файла с расширением txt. Но расширение накладывает жесткие ограничения - мало интересного можно найти в txt-файлах. Как же это обойти?
В скриптовом языке Perl, в отличие от языка Си, NULL-символ (символ с кодом 0x00) не является признаком конца строки. Но функции системных вызовов (в частности open) реализованы на Си, в котором NULL - признак конца строки. Т.е. Perl-строка "about\0.txt" в функции open будет воспринята как "about". Таким образом, запрос

```
http://www.***.ru/script.cgi?part=/etc/passwd%00
```

вызовет

```
open FILE, "/etc/passwd\0.txt";
```

и будет считан файл паролей /etc/passwd.

2. Pipe-символ.

Вернемся к предыдущему примеру:

```
http://www.***.ru/script.cgi?part=about
open FILE, "$part.txt";
```

Согласно описанию функции open, вместе с именем файла можно использовать pipe-символ ("|") для организации пайпа (программного канала межпроцессного взаимодействия) между исполняемым CGI-скриптом и указанным совместно с pipe-символом **исполняемым** файлом. Т.е. вызов

```
open FILE, "|about.txt";
```

исполнит файл about.txt (если, конечно, у этого файла установлены соответствующие атрибуты на исполнение) и результат вернется в CGI-программу, как если бы это было содержимое файла "|about.txt". Таким образом, по запросу

```
http://www.***.ru/script.cgi?part=|ls%20-la%00
```

мы получим содержимое текущей директории, а запрос

```
http://www.***.ru/script.cgi?part=|rm%20-rf%20~%20/%00
```

полностью удалит домашнюю директорию пользователя, из-под которого исполняется CGI-скрипт и попытается удалить все остальное.

3. Code including.

Т.к. при исполнении скриптов интерпретатор выполняет непосредственно исходный нетранслированный код программы, существуют уязвимости CGI-скриптов, позволяющие злоумышленнику исполнить через уязвимый скрипт свой код. Это становится возможным, если в CGI-скрипте выполняется подключение файла с кодом, имя которого передается как CGI-параметр. Например,

```
Запрос: http://www.***.ru/script.cgi?module=news
Perl/PHP-код: require $module;
```

В данном случае опасность не велика, потому что вряд ли на сервере найдется файл с нужным кодом. Но в PHP имеется возможность подключения удаленного файла с кодом, указав его URL (за эту возможность отвечает опция *allow_url_fopen* в *php.ini*). Т.е. выкладываем на доступ по HTTP файл с нужным кодом и передаем его URL как параметр в уязвимый скрипт. Что касается Perl, то вызов нужной части кода может осуществляться так

```
sub news {
   ....
}

eval($module);
```

или так

```
sub news {
   ....
}

&$module;
```

В первом случае злоумышленник может исполнить любой свой код, передав его в параметре *module*. Во втором случае он сможет лишь исполнить какую-либо функцию, что, однако, может быть достаточно, например, для SQL-инъекции.

Вариантов подобных уязвимостей можно придумать множество, вспомним хотя бы нашумевшую уязвимость популярного freeware форума phpbb, связанную с параметром highlight. А некоторые начинающие веб-программисты даже умудряются так построить работу своего скрипта, что намеренно передают исполняемый код через внешние параметры (post, get или cookie).

Основные методы предупреждения CGI-уязвимостей

Как видно, CGI-уязвимости несут в себе реальную опасность и требуют внимания при создании веб-приложений. Четких инструкций на тему "как не допустить CGI-уязвимость", конечно, не существует, но можно отметить несколько основных моментов.

1. Для аутентификации пользователя следует использовать сессии, причем менять их следует как можно чаще. При завершении пользователем сеанса работы с разделом сайта, защищенным авторизацией, следует обязательно закрывать сессию. Это позволит сократить время действия сессии, номер которой может узнать злоумышленник.

2. Пароли нужно передавать методом POST, по возможности используя защищенное SSL соединение. Не следует передавать или хранить пароли (как в открытом, так и шифрованном виде) в GET-параметрах, cookie или hidden-полях веб-форм.

3. Не стоит передавать на сторону клиента (пусть даже в скрытой форме – cookie или hidden-полях) информацию "не для посторонних глаз" – чужие e-mail'ы, механизмы расчета цены и пр.

4. Нельзя доверять информации, полученной от клиента. Нужно всегда проверять ее. Если параметр числовой, то нужно проверить, что содержимое – число, если это *id* новости – проверить наличие этой новости и параметры доступа/отображения (например, скрытая еще не опубликованная или недоступная этому пользователю), если это ссылка на веб-страницу, то она должна начинаться с `http://`. По возможности не принимать от клиента информацию, которая может быть получена непосредственно на стороне сервера (цены, наличие товара и др.).

5. При любых входных данных CGI-программа должна возвращать верный результат, даже если это сообщение об ошибке ("Новость не найдена" вместо "Internal server error").

6. Для защиты от SQL-инъекций, XSS и некоторых других уязвимостей, следует фильтровать входные данные на наличие таких символов, как `<`, `>`, `&`, `|` прямые/обратные/двойные кавычки. Как правило их заменяют escape-последовательностями – в этом случае браузер отображает символ без изменения, но синтаксически он состоит из нескольких символов. Например, текст `<text>` будет записан в виде `&lt;text&gt;`. Также для защиты от XSS-уязвимостей часто требуется фильтровать входные данные на наличие таких слов, как `script`, `javascript`, `onload`, `onmouseover`, `onerror` и т.п.

7. Веб-сервер (например, Apache) и сервер баз данных (например, MySQL) должны быть запущены от имени отдельных пользователей. CGI-программы по возможности также должны исполнятся от отдельного пользователя (например, посредством suexec). Доступ CGI-программы к серверу баз данных должен быть от имени пользователя с, по возможности, максимально ограниченными правами (хотя бы в пределах выделенной базы данных).

8. Такие параметры исполнения CGI-программы, как максимальное время выполнения и общий размер входных данных, также желательно ограничить в разумных пределах.

9. Подключаемые программные модули, файлы конфигураций и другие файлы, к которым не требуется удаленный HTTP-доступ, следует размещать в каталогах, не доступных извне по HTTP-протоколу. Для всех доступных по HTTP каталогов следует прописать директиву веб-сервера *Options –Indexes*, чтобы запретить листинг содержимого каталогов. Например, однажды, просматривая листинг одного из каталогов, был найден файл с параметрами FTP-доступа к этому сайту.

Также существуют различные программы, тестирующие сайты на наличие уязвимостей. На некоторые упущения в безопасности они могут указать, но все же взлом сайта не механическая, а скорее творческая работа. Поэтому если вы всерьез заботитесь о безопасности Вашего веб-проекта, гораздо надежней самому иметь представление о различных типах уязвимостей и не допускать их при написании CGI-программ.

В любом случае, нужно всегда внимательно относится к проектированию веб-приложений и почаще смотреть на свой сайт глазами злоумышленника.

Ссылки по теме

1. http://www.void.ru
2. http://www.securitylab.ru
3. http://www.opennet.ru
4. http://www.antichat.ru
5. http://www.web-hack.ru/