УДК 004.056
ББК 32.973-018
В 58

Власенко А.В.
Кандидат технических наук, доцент кафедры компьютерных технологий и информационной безопасности института информационных технологий и безопасности, начальник управления аспирантуры и докторантуры ФГБОУ ВПО «Кубанский государственный технологический университет», Краснодар, e-mail: Vlasenko@kubstu.ru
Дзьобан П.И.
Аспирант института информационных технологий и безопасности ФГБОУ ВПО «Кубанский государственный технологический университет», Краснодар, e-mail: antiemoboy@mail.ru

Разработка алгоритмов и программ выбора оптимального набора компонент нейтрализации актуальных угроз на основе описания модели и интеграции их в Web-приложение

(Рецензирована)

Аннотация. *Разработка алгоритмов и программы для выбора оптимального набора компонент нейтрализации актуальных угроз и обработки распространенных уязвимостей Web-приложений, организованных в структурированном списке, состоящего из шести классов: аутентификация, авторизация, атаки на клиентов, выполнение кода, разглашение информации, логические атаки, а так же разработка ресурса, максимально удовлетворяющего требованиям безопасности, которые зачастую заявляет заказчик/владелец Web-сайта, и находящегося в рамках закона соответственно, либо приведение уже имеющегося сайта к этим требованиям, прибегнув к анализу использующихся и потенциально опасных уязвимостей.*

Ключевые слова: *Web-приложения, идентификация пользователя, аутентификация, авторизация, модели угроз.*

Vlasenko A.V.
Candidate of Engineering Sciences, Associate Professor, Associate Professor of Department of Computer Technologies and Informative Safety of Institute of Information Technologies and Safety, Chief of Administration of Post-Graduate and Doctoral Study Courses, Kuban State University of Technology, Krasnodar, e-mail: Vlasenko@kubstu.ru
Dzoban P.I.
Post-graduate student of Institute of Information Technologies and Safety, Kuban State University of Technology, Krasnodar, e-mail: antiemoboy@mail.ru

Development of algorithms and programs to choose the optimal set of components of actual threat neutralization on the basis of model description and their integration in Web appendix

Abstract. *Algorithms and programs are developed to choose the optimal set of components of actual threat neutralization and to process widespread vulnerabilities of Web appendixes. The appendixes are organized in the structured list, consisting of six classes: authentification, authorizing, attacks on clients, code implementation, information disclosure and logical attacks. Also the resource is developed to meet the requirements of safety, that a customer / proprietor of Web-site declares frequently, and to be within the framework of the law, correspondingly. Or otherwise an already present Web-site is brought over to these requirements, resorting to the analysis of used and potentially dangerous vulnerabilities.*

Keywords: *Web appendixes, user identification, authentification, authorizing, threat models.*

Анализ тематики атак на различные базы данных и соответственно каналы, по которым передаются данные, особенно тех, где содержится финансовая информация, сегодня очень актуальна. Злоумышленник может добыть информацию о кредитной карте пользователя, подобрав пароль или обойдя систему безопасности сервера коммерческого предприятия, а так же выполнить какие-либо действия в сети от лица пользователя, последствия которых могут быть необратимыми. Задача разработчиков и системных администраторов любого сайта – уберечь конфиденциальную информацию от хакинга.

На сегодняшний день выделяют 6 классов атак на Web-приложения и сайты.

Большинство из них приходится на этап идентификации пользователя, а именно процесс передачи авторизационных и аутентификационных данных от пользователя к базе данных Web-приложения, минуя брандмауэры Web-приложения и различные системы защиты как программного, так и аппаратного уровня.

Рассмотрим методику реализации процесса безопасной авторизации и аутентификации.

Первоначально выполняется авторизация и вход пользователей.

Форма, с помощью которой пользователь сможет передать нам свой логин и пароль, представлена на рисунке 1.

```
<form action="login.php" method="post">
    <table>
        <tr>
            <td>Логин:</td>
            <td><input type="text" name="login" /></td>
        </tr>
        <tr>
            <td>Пароль:</td>
            <td><input type="password" name="password" /></td>
        </tr>
        <tr>
            <td></td>
            <td><input type="submit" value="Войти" /></td>
        </tr>
    </table>
</form>
```

Рис. 1. Форма передачи пользователем пароля и логина

После ввода пользователем данных, проверяется его логин и пароль. Очевидно, мы не будем хранить пароль в чистом виде, будем сравнивать хеш хеша введенного пароля с тем, что хранится в базе, как представлено на рисунке 2.

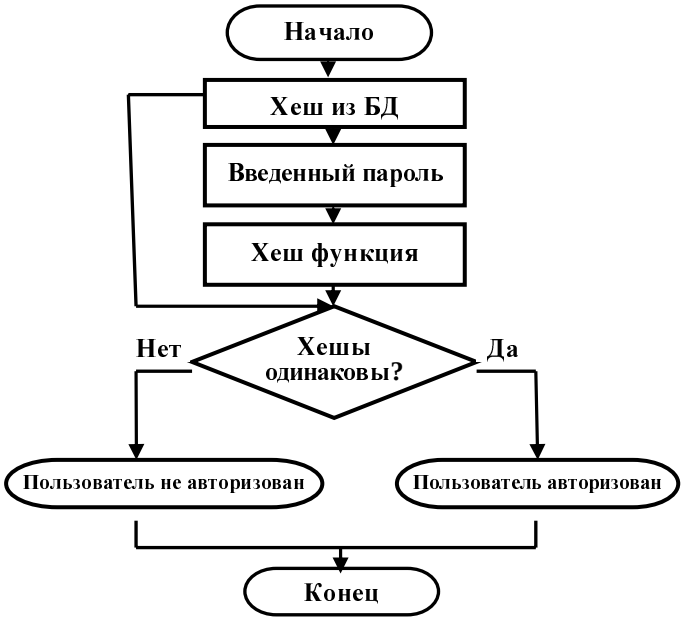

Рис. 2. Простой алгоритм процесса авторизации пользователя

Процесс хеширования пароля будет проводиться с использованием алгоритма хеширования md5. На рисунке 3 представлена процедура сравнения хеша пароля с хешом пароля, хранящегося в базе данных Web-приложения [1-3].

```
if (isset($_POST['login']) && isset($_POST['password']))
{
    $login = mysql_real_escape_string($_POST['login']);
    $password = md5($_POST['password']);

    // делаем запрос к БД
    // и ищем юзера с таким логином и паролем

    $query = "SELECT `id`
            FROM `users`
            WHERE `login`='{$login}' AND `password`='{$password}'
            LIMIT 1";
    $sql = mysql_query($query) or die(mysql_error());

    // если такой пользователь нашелся
    if (mysql_num_rows($sql) == 1) {
        // то мы ставим об этом метку в сессии (допустим мы будем ставить ID пользователя)

        $row = mysql_fetch_assoc($sql);
        $_SESSION['user_id'] = $row['id'];

        // не забываем, что для работы с сессионными данными, у нас в каждом скрипте должно присутствовать session_start();
    }
    else {
        die('Такой логин с паролем не найдены в базе данных. И даём ссылку на повторную авторизацию.');
    }
}
```

Рис. 3. Процесс сравнения хеша введенного пользователем пароля с хешом, хранящимся в БД Web-приложения

Итак, мы определились, что авторизованные пользователи – те, у которых указан $_SESSION['user_id']. Всех других пользователей будем считать гостями.

Пример дискриминирования посетителей Web-приложения или сайта приведен на рисунке 4.

```
if (isset($_SESSION['user_id'])) {
    // показываем защищенные от гостей данные.
}
else {
    die('Доступ закрыт, даём ссылку на авторизацию.');
}
```

Рис. 4. Пример разделения посетителей на пользователей и гостей

Механизм их установки и получения следующий:

1. При запросе страницы браузером сервер формирует ответ, в заголовках которого указывает, что следует установить соответствующие cookie.

2. Браузер получает ответ и сохраняет значения cookie.

3. При последующем запросе страницы того же сайта браузер в заголовках запроса посылает cookie на сервер.

По сути, это аналог переменных, которые можно хранить у пользователя.

Подходят они (cookie) также для сбора общей усредненной статистики по достаточно большому количеству человек. Например, при отслеживании уникальных пользователей на сайте. Если их около 100000, то скорее всего, подавляющее большинство не будет отключать или подделывать cookie.

Для хранения конфиденциальных данных такой способ не подходит. Для идентификации пользователей подходят с оговорками. Например, после авторизации вы мо-

жете положить пользователю в cookie его логин и на следующих страницах идентифицировать его уже по нему. Но ничто (кроме технической грамотности) не помешает человеку, зайдя под логином «user», потом исправить соответствующую cookie на «admin». Обычно в cookie помещают идентификатор сеанса, представляющий собой случайную 32-разрядную строку. Тут уже надежда на то, что пользователь не сможет подобрать чужой идентификатор, значительно выше.

Еще один момент – число cookie одного сайта ограничено (обычно пара десятков). Их суммарный объем также ограничен (несколько килобайт). Поэтому хранить много информации не получится. Если это необходимо, используйте cookie для идентификации пользователя, а связанную с этим пользователем информацию храните на сервере в вышеуказанном захешированном виде.

Рассмотрим процедуру установки cookie.

В PHP cookie устанавливаются функцией setCookie().

Установка времени меньше текущего так же позволяет удалить cookie. Обратите внимание, что это именно время устаревания, а не время жизни. То есть для установки cookie на час следует делать так, как представлено на рисунке 5.

Код: PHP

```php
setCookie("cook", "value", time() + 3600); // Правильно
setCookie("cook", "value", 3600); // Не правильно
```

Рис. 5. Установка времени устаревания Cookie на 1 час

Сессии не постоянны и данные в них хранятся только на время работы пользователя со скриптом. Поэтому, чтобы пользователь оставался залогиненым после того, как он закрыл и снова открыл на нашей странице свой браузер, необходимо сохранять его авторизационные данные в cookie (они хранятся на его компьютере в браузере).

Мы будем записывать в cookie пользователя, его логин и хеш хеша пароля – для более полной криптостойкости. На рисунке 6 представлен скрипт по проверке пользовательских данных в cookie:

```php
// если пользователь не авторизован
if (!isset($_SESSION['user_id'])) {
    // то проверяем его куки
    // вдруг там есть логин и пароль к нашему скрипту

    if (isset($_COOKIE['login']) && isset($_COOKIE['password'])) {
        // если же такие имеются
        // то пробуем авторизовать пользователя по этим логину и паролю
        $login = mysql_real_escape_string($_COOKIE['login']);
        $password = mysql_real_escape_string($_COOKIE['password']);

        // и по аналогии с авторизацией через форму:

        // делаем запрос к БД
        // и ищем юзера с таким логином и паролем

        $query = "SELECT `id`
                FROM `users`
                WHERE `login`='{$login}' AND `password`='{$password}'
                LIMIT 1";
        $sql = mysql_query($query) or die(mysql_error());

        // если такой пользователь нашелся
        if (mysql_num_rows($sql) == 1) {
            // то мы ставим об этом метку в сессии (допустим мы будем ставить ID пользователя)

            $row = mysql_fetch_assoc($sql);
            $_SESSION['user_id'] = $row['id'];

            // не забываем, что для работы с сессионными данными, у нас в каждом скрипте должно присутствовать session_start();
        }
        else {
            // только мы не будем давай ссылку на форму авторизации
            // вдруг человек и не хочет был авторизованым
            // а пришел просто поглядеть на наши страницы как гость
```

Рис. 6. Скрипт Php по проверке пользовательских данных в cookie

Заметим, что на данном скрипте метка в сессии пользователя ставится по ID-пользователя, который меняется при каждом посещении Web-приложения или сайта.

Так же следует уделить внимание безопасному хранению паролей в базе данных. Пароль не следует хранить в открытом виде. Всегда существует опасность SQL инъекции, при которой злоумышленник может получить ваш пароль. Лучше его хешировать (например, с помощью функции md5()) несколько раз – именно поэтому мы отводим на пароль 32 символа.

И следовательно, при авторизации мы сверяем не пароли, а их хеши хешей. В нашем случае это было так: мы сравнивали md5 (md5 ('введенного пароля')) с хэшем хеша пароля, хранящимся в БД.

Пароль можно подобрать по его хешу, зная алгоритм хеширования. Существует много баз, где пароли сопоставлены с их хешами. Поэтому мы можем заметно усложнить задачу злоумышленникам, которые соберутся подбирать пароль по его хешу.

Мы будем использовать двойной хеш (например, md5(sha1('password'))) или использовать так называемую «соль» (salt).

Пример использования соли: md5(md5('password'). 'secret_code'); secret_code – это и есть соль, то есть мы к паролю добавляем какой-то набор символов, заданный по случайному алгоритму, который можно менять раз в период, для увеличения криптостойкости.

Так же, помимо соли и *n*-разового хеширования пароля, актуально в разработанной методике с учетом повышенного интереса злоумышленников к таким видам атак использовать дополнительные инструменты:

– «Зависимый хэш» – зависит от уникальной переменной (например, логина);

– «Фарш» – перемешать значение хэша;

– «Интеллектуальный хэш» – хэш меняет алгоритм в зависимости от длины и значений.

Для каждого пользовательского пароля (при его регистрации) следует генерировать «свою соль» и записывать в базу рядом с паролем, чтобы использовать при хешировании [4, 5].

У каждого Web-приложения свой уровень риска и чувствительности.

Для расстановки приоритетов в защите пользователей необходимо определить степень риска и угроз организации. Способ построения Web-приложения в значительной степени будет влиять на степень вовлеченности пользователей в обеспечение безопасности, поэтому на наш взгляд данное направление исследований является очень актуальным и перспективным.

Примечания:

1. Кришнамурти Б., Рексфорд Дж. Web-протоколы. Теория и практика. HTTP/1.1, взаимодействие протоколов, кэширование, измерение трафика. М.: Бином, 2002. 592 с.
2. Кузнецов С.Д. Базы данных. Модели и языки. М.: Бином-Пресс, 2008. 720 с.
3. Менаске Д., Алмейда В. Производительность Web-служб. Анализ, оценка и планирование. СПб: ДиаСофтЮП, 2003. 480 с.
4. Кузнецов С.Д. Проектирование и разработка корпоративных информационных систем. Центр Информационных Технологий, 1998.
5. Ларман К. Применение UML и шаблонов проектирования. М.: Вильямс, 2001. 485 с.

References:

1. Krishnamurti B., Reksford G. Web protocols. Theory and practice. NTTR/1.1, interaction of protocols, caching, traffic measurement. M.: Binom, 2002. 592 pp.
2. Kuznetsov S.D. Models and languages. M.: Binom-Press, 2008. 720 pp.
3. Menaske D., Almeida V. Productivity of Web services. Analysis, assessment and planning. SPb: DiaSoftYuP, 2003. 480 pp.
4. Kuznetsov S.D. Projecting and development of corporate information systems. Center of Information Technologies, 1998.
5. Larman K. Application of UML and design patterns. M.: Williams, 2001. 485 pp.