Безопасная аутентификация без использования HTTPS

Филимошин В.Ю., Давлеткиреева Л.З.

Аннотация **– в статье рассматривается алгоритм безопасной аутентификации для веб-ресурсов без использования HTTPS, который позволяет сохранить пароль защищённым от злоумышленника. Ключевая идея данного алгоритма состоит в том, чтобы не передавать в открытом виде пароль пользователя на сервер. Вместо пароля предлагается передавать зашифрованный хэш от пароля, который. Суть данного подхода в том, что если злоумышленник перехватит зашифрованный хэш пароля и если ему удастся его расшифровать, то он получит только хэш с солью, из которого уже невозможно получить исходный пароль. Показана часть реализации данного алгоритма с помощью языка php. Статья будет полезна для веб-разработчиков.**

Ключевые слова **— безопасная аутентификация, веб-ресурс, алгоритм, php**

I. Введение

Безопасность пользователей в сети Интернет – ключевая проблема для ИТ-специалистов, решение которой осуществляется за счёт различных комплексов мер, одним из которых является аутентификация.

Когда идет речь о защите информации, одним из важнейших аспектов является защита от несанкционированного доступа к ресурсам нашей сети. Разумеется, крайне важным вопросом является обеспечение процедуры безопасной аутентификации. Совершенно очевидно, что любое разграничение полномочий, настройка прав доступа на ресурсы системы имеет смысл только в том случае, если мы уверены в том, что тот, кто пытается получить доступ к нашим ресурсам, является легальным пользователем. Вопросы безопасной аутентификации являются весьма актуальными при попытке обеспечения безопасности организации в целом [1].

Рассмотрим, что происходит при входе пользователя на веб-ресурс. Вход состоит из трёх последовательно выполняемых процедур: идентификация, аутентификация, авторизация.

Статья получена 1 июня 2017

Филимошин В.Ю. ФГБОУ ВО «Магнитогорский государственный технический университет им. Г.И. Носова», 455000, Россия (тел: +7-9088274454; e-mail: flightofdeath@mail.ru).

Давлеткиреева Л.З. ФГБОУ ВО «Магнитогорский государственный технический университет им. Г.И. Носова», 455000, Россия (e-mail: ldavletkireeva@mail.ru).

Идентификация – процедура поиска, ранее (при регистрации), назначенного уникального идентификатора пользователя.

Аутентификация – проверка подлинности предоставленного идентификатора с помощью пароля или какой-либо другой информации, которая доступна только этому пользователю.

Авторизация – процедура предоставления прав доступа к закрытым разделам сайта и функциональным возможностям, которые не доступны для гостей.

В настоящее время большинство веб-ресурсов дают возможность пользователям регистрироваться и после чего пользоваться расширенным функционалом своего веб-ресурса. Как при регистрации, так и при аутентификации злоумышленник может перехватить пароль пользователя, если он передаётся на сервер в открытом виде (без использования протокола с шифрованием HTTPS[1]), либо с простым хэшированием[2] без соли[3].

С 2014 года компания Google рекомендует веб-мастерам использовать защищённый HTTPS протокол для своих сайтов, а совсем недавно объявила, что с начала января 2017 года их браузер chrome будет сообщать пользователям о небезопасности передачи данных по HTTP протоколу: «To help users browse the web safely, Chrome indicates connection security with an icon in the address bar. Historically, Chrome has not explicitly labelled HTTP connections as non-secure. Beginning in January 2017 (Chrome 56), we'll mark HTTP pages that collect passwords or credit cards as non-secure, as part of a long-term plan to mark all HTTP sites as non-secure» [2].

Использование HTTPS предотвращает перехват пароля в открытом виде, но не все могут использовать данный протокол по разным причинам, например, нет денег для покупки сертификата (речь идёт о физическом, а не юридическом лице) или нежелание использовать бесплатный сертификат (из-за его ограничений), либо использовать собственный сгенерированный сертификат. Собственный сертификат

[1] HTTPS (аббр. от англ. *HyperText Transfer Protocol Secure*) – расширение протокола HTTP, для поддержки шифрования в целях повышения безопасности [3].

[2] Хэширование – преобразование массива входных данных произвольной длины в (выходную) битовую строку фиксированной длины, выполняемое определённым алгоритмом [4].

[3] Соль – строка данных, которая передаётся хеш-функции вместе с паролем. Используется для удлинения строки пароля, чтобы увеличить сложность взлома [5].

может вызывать сомнение у пользователя, так как при каждом посещении он должен подтверждать своё согласие о риске использования непроверенного сертификата данного веб-ресурса, либо добавить этот сертификат в доверенную зону.

II. Алгоритм безопасной аутентификации

В связи с этим был реализован алгоритм безопасной аутентификации пользователя. Основная цель данного алгоритма заключается в том, чтобы злоумышленник не смог узнать исходный пароль пользователя и при перехвате, передаваемых данных, от пользователя к серверу, не смог подставить их к себе в браузер (cookie) без подмены дополнительных данных[4].

Схема алгоритма аутентификации представлена на рисунке 1.

Рассмотрим алгоритм безопасной аутентификации по шагам (с правильно введённым логином и паролем):

1. Пользователь вводит логин и пароль в форму на веб-ресурсе.
2. Через скрипт JavaScript (JS), на сервер (методом post) передаётся (через AJAX без перезагрузки страницы) логин пользователя без пароля в открытом виде (для идентификации).
3. На сервере обрабатывается логин (на запрещённые символы) и происходит поиск учётной записи по логину в базе данных.
4. Генерируются публичный и приватный ключи, в базу данных, к найденной учётной записи, добавляется время обращения к скрипту (UNIXTIME) и приватный ключ.
5. Время обращения и публичный ключ передаются на сторону клиента, которые получает скрипт JS посредством AJAX технологии.
6. На стороне клиента, с помощью JS скрипта, хэшируется пароль плюс время обращения, а так же с помощью открытого ключа шифруется полученный хэш криптографическим алгоритмом RSA.
7. В форме, где введён логин и пароль, пароль заменяется[5] на зашифрованный хэш с солью от пароля и только после этого происходит отправка на сервер.
8. Скрипт на сервере обрабатывает полученные данные от пользователя (на запрещённые символы).
9. Идёт поиск учётной записи в базе данных по логину пользователя.
10. С помощью закрытого ключа (который находится в БД у найденного пользователя) расшифровывается полученный хэш, после чего хэш пароля, который находится в базе, дополнительно хэшируется с солью (время обращения) и сравнивается с полученным хэшом от пользователя.
11. Аутентификация завершена.

[4] Дополнительные данные это данные, которые хэшируются на стороне сервера, то есть без доступа к скрипту, злоумышленник не будет знать о дополнительных данных.

[5] Замена происходит за счёт JS скрипта, на стороне клиента, и пароль в открытом виде, в итоге, никуда не передаётся.

Далее происходит запись данных в cookie и сессию, для дальнейшего распознавания пользователя, пока он находится на веб ресурсе и назначаются права (авторизация).

Если был введён не правильный логин, то скрипт выводит сообщение пользователю, что данного логина или пароля не существует. Если был введён не правильный пароль, то все данные, которые были добавлены в БД (время обращения и закрытый ключ) удаляются, а так же выводится сообщение пользователю, что данного логина или пароля не существует.

Перейдём к рассмотрению непосредственно самого класса, который обрабатывает входящие данные от пользователя.

При создании объекта класса конструктор обрабатывает входящие данные, в зависимости от действия, в него передаются данные о входе пользователя, либо не передаётся ничего (см. листинг 1).

Листинг 1. Конструктор

```
public function __construct(
                        $enter=false,
                        $login='',
                        $hash_password='',
                        $remember_me=''
){
 session_start();
 if($enter){
  $this->enter_check_user($login, $hash_password, $remember_me);
 }
 else{
  if(
     isset($_SESSION['login']) &&
     isset($_SESSION['random_number']) &&
     isset($_COOKIE['id']) &&
     isset($_COOKIE['hash'])
  ){
   $this->check_session(
              $_SESSION['login'],
              $_SESSION['random_number'],
              $_COOKIE['id'],
              $_COOKIE['hash']
   );
  }
  else{
   if(isset($_COOKIE['id']) && isset($_COOKIE['hash'])){
    $this->check_remember_user(
               $_COOKIE['id'],
               $_COOKIE['hash']
     );
    }
   }
  }
}
```

Рассмотрим четыре результата работы конструктора в таблице 1.

Таблица 1. Результат работы конструктора

	Передаваемые переменные	cookie	session	Результат работы конструктора
1	-	-	-	Ничего не происходит
2	$enter, $login, $hash_password,	-	-	Вызывается метод проверки переданных переменных

	$remember_me			(enter_check_user())					)
3	-	+	-	Вызывается метод проверки пользователя только по cookie (check_remember_user	4	-	+	+	Вызывается метод проверки пользователя по cookie и сессии (check_session)

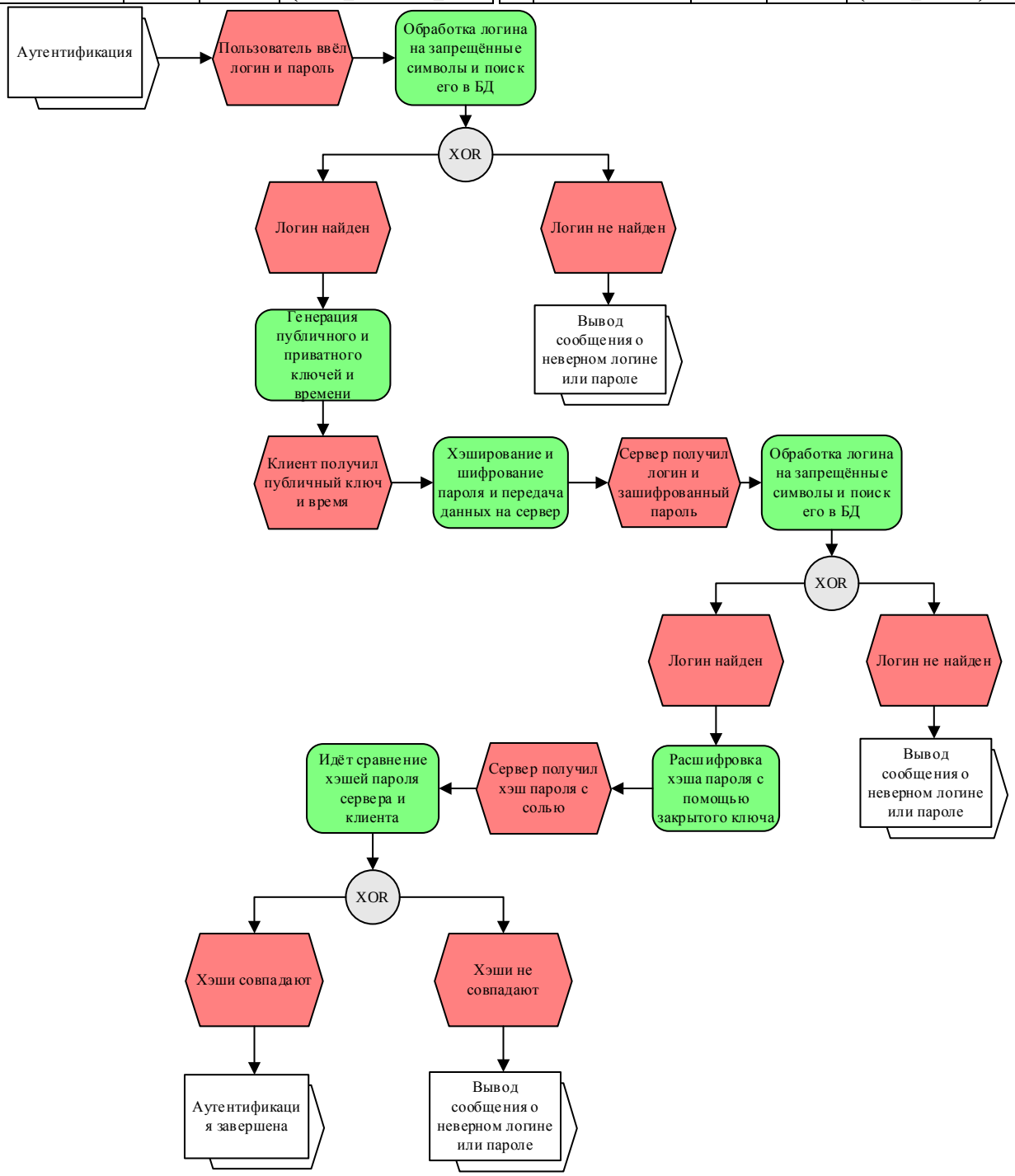

Рисунок 1 – Схема аутентификации

Необходимо добавить, что в первом случае, если объект создаётся в скрипте вызванный JavaScript'ом через AJAX, то вызывается публичный метод (action_before_entering(), см. листинг 2) для поиска логина в БД, генерированию ключей (приватного и публичного), добавлению данных в БД и возврату данных клиенту в JavaScript через AJAX.

Листинг 2. Метод action_before_entering и дополнительные приватные методы

```
public function action_before_entering($login){
  global $db;
  $login=rs('/[^A-Za-z0-9]/u',$login,20);
```

```
  $id_user=$this->check_login($login);
  if($id_user!='0'){
   $keys_array=array("","");
   if($this->use_RSA){
    $keys_array=$this->generate_keys();
   }
   $time=time();
   $db->update_query("
    UPDATE
     `users`
    SET
     `time`='".$time."',
     `private_key`='".$keys_array[1]."'
    WHERE
     `id`='".$id_user."'
   ");
   if($this->use_RSA){
    return 'RSA;'.$time.';'.$keys_array[0];
   }
   else{
    return 'not_RSA;'.$time;
   }
  }
  else{
   return 'error';
  }
 }

 private function check_login($login){
  global $db;
  $user_check=$db->select_query("
   SELECT
    `id`
   FROM
    `users`
   WHERE
    `login`='".$login."'
   LIMIT 1
  ");
  if(is_array($user_check)){
   return $user_check[0]['id'];
  }
  else{
   return '0';
  }
 }

 private function generate_keys(){
  global $db;
  $config=openssl_pkey_new(
   array(
    'private_key_type'=>OPENSSL_KEYTYPE_RSA,
    'private_key_bits'=>1024
   )
  );
  $private_key='';
  openssl_pkey_export($config,$private_key);
  $public_key_array=openssl_pkey_get_details(
   $config
  );
  return array($public_key_array['key'],$private_key);
 }
```

В методе action_before_entering() обрабатывается на запрещённые символы переданный логин, после чего вызывается приватный метод поиска логина в БД (check_login()), далее вызывается приватный метод генерирования приватного и публичного ключей (generate_keys()), при условии, что классу разрешено использовать RSA шифрование (за это отвечает переменная use_RSA). После генерирования ключей в БД записывается приватный ключ и время в формате unixtime (будущая соль). После всех этих действий в JavaScript возвращается строка, содержащая информацию об использовании или не использовании шифрования, время (то, которое записалось в БД) и массив с ключами (либо заполненный, либо пустой).

Рассмотрим 3 метода, которые вызываются конструктором при условиях, которые описаны в таблице 1 (см. листинги 3, 4, 5).

Листинг 3. Метод аутентификации.

```
 private function enter_check_user(
  $login,
  $hash_password,
  $remember_me
 ){
  global $db;
  $login=rs('/[^A-Za-z0-9]/u',$login,20);
  $id_user=$this->check_login($login);
  if($id_user!='0'){
   if(!$this->use_RSA){
    $hash_password=rs('/[^0-9a-z]/u',$hash_password,32);
   }
   else{
    $hash_password=rs(
     '/[$\"\'\`\~\<\>\(\)\*\%\^\#\@\!]/u',
     $hash_password,
     172
    );
    $user_private_key_array=$db->select_query("
     SELECT
      `private_key`
     FROM
      `users`
     WHERE
      `id`='".$id_user."'
    ");
    $hash_password=$this->decrypt_RSA(
     $user_private_key_array[0]['private_key'],
     $hash_password
    );
    unset($user_private_key_array);
    $db->update_query("
     UPDATE
      `users`
     SET
      `private_key`=NULL
     WHERE
      `id`='".$id_user."'
    ");
   }
   if($this->check_hash_password($id_user,$hash_password)){
    $random_number=mt_rand(10,99).mt_rand(10,99);
    $crypt_id=$this->decryption_or_encryption_id(
     $id_user,
     $random_number,
     'encryption'
    );
    $this->id=$id_user;
    $this->login=$login;
    $this->permission();
    $_SESSION['login']=$login;
    $_SESSION['random_number']=$random_number;
    if($remember_me!=''){
     $hash=md5($_SERVER['HTTP_USER_AGENT']);
     setcookie('id',$crypt_id,time()+$this->time_remember);
     setcookie('hash',$hash,time()+$this->time_remember);
     setcookie('hash2',md5($login),time()+$this->time_remember);
     $db->insert_query("
```

```
        INSERT INTO
          `users_remember`(
            `id_user`,
            `crypt_id`,
            `hash`,
            `time`
          )
          values(
            '".$id_user."',
            '".$crypt_id."',
            '".$hash."',
            '".time()."'
          )
        ");
      }
      else{
        setcookie('id',$crypt_id);
        setcookie('hash',md5($_SERVER['HTTP_USER_AGENT']));
        setcookie('hash2',md5($login));
      }
    }
  }
}
```

В методе enter_check_user() происходит обработка логина на запрещённые символы, вызывается метод поиска логина в БД, расшифровывается хэш пароля с помощью приватного ключа (если используется шифрование), вызывается метод для сверки хэшей пароля (check_hash_password()). Стоит обратить внимание, что в БД лежит hash пароля без соли, и при сравнении он дополнительно хэшируется с солью и уже после этого сравнивается, таким образом при каждом новом входе у пользователя будет новая соль. Приватным методом назначаются права пользователя (permission()). Далее id пользователя шифруется с помощью случайного числа в приватном методе (decryption_or_encryption_id()). В сессию добавляются логин пользователя и случайное число для расшифровки id пользователя. В cookie добавляются зашифрованный id пользователя, хэш значения HTTP_USER_AGENT и хэш логина.

Листинг 4. Метод проверки по cookie

```
private function check_remember_user(
  $crypt_id,
  $hash,
  $hash2
){
  global $db;
  $crypt_id=rs('/[^0-9]/u',$crypt_id,10);
  $hash=rs('/[^0-9a-z]/u',$hash,32);
  $hash2=rs('/[^0-9a-z]/u',$hash2,32);
  if(md5($_SERVER['HTTP_USER_AGENT'])==$hash){
    $user_remember_array=$db->select_query("
      SELECT
        `id`,
        `id_user`,
        `time`
      FROM
        `users_remember`
      WHERE
        `crypt_id`='".$crypt_id."' and
        `hash`='".$hash."'
    ");
    if(is_array($user_remember_array)){
      if(
        $user_remember_array[0]['time']<
          time()-
          $this->time_remember
        ){
          $db->delete_query("
            DELETE FROM
              `users_remember`
            WHERE
              `id`='".$user_remember_array[0]['id']."'
          ");
          $this->delete_cookies();
        }
        else{
          $user_login_array=$db->select_query("
            SELECT
              `login`
            FROM
              `users`
            WHERE
              `id`='".$user_remember_array[0]['id_user']."'
          ");
          if($hash2==md5($user_login_array[0]['login'])){
            $random_number=mt_rand(10,99).mt_rand(10,99);
            $crypt_id=$this->decryption_or_encryption_id(
              $user_remember_array[0]['id_user'],
              $random_number,'encryption'
            );
            $db->update_query("
              UPDATE
                `users_remember`
              SET
                `crypt_id`='".$crypt_id."'
              WHERE
                `id`='".$user_remember_array[0]['id']."'
            ");
            $this->id=$user_remember_array[0]['id_user'];
            $this->login=$user_login_array[0]['login'];
            $this->permission();
            $_SESSION['login']=$user_login_array[0]['login'];
            $_SESSION['random_number']=$random_number;
            setcookie('id',$crypt_id,time()+$this->time_remember);
          }
          else{
            $this->delete_cookies();
          }
        }
      }
      else{
        $this->delete_cookies();
      }
    }
    else{
      $this->delete_cookies();
    }
  }
}
```

В методе check_remember_user() обрабатываются все данные из cookie пользователя и идёт их сверка не только с таблицей данных пользователей в БД, но и ещё со специальной таблицей для пользователей, которые поставили галочку «запомнить меня», где хранятся дополнительные данные о них. После чего назначаются права пользователя и записываются данные в сессию, а если данные не прошли проверку, то cookie на стороне пользователя немедленно удаляются.

Листинг 5. Метод проверки по cookie и сессии

```
private function check_session(
  $login,
  $random_number,
```

```
  $crypt_id,
  $hash,
  $hash2
){
  global $db;
  $login=rs('/[^A-Za-z0-9]/u',$login,20);
  $random_number=rs('/[^0-9]/u',$random_number,4);
  $crypt_id=rs('/[^0-9]/u',$crypt_id,10);
  $hash=rs('/[^0-9a-z]/u',$hash,32);
  $hash2=rs('/[^0-9a-z]/u',$hash2,32);
  if(
    !empty($login) &&
    !empty($random_number) &&
    !empty($crypt_id) && !empty($hash) &&
    !empty($hash2)
  ){
    if($hash==md5($_SERVER['HTTP_USER_AGENT'])){
      if($hash2==md5($login)){
        $id_user=$this->check_login($login);
        if($id_user!='0'){
          if($this->decryption_or_encryption_id(
            $crypt_id,
            $random_number,
            'decryption')==$id_user
          ){
            $this->id=$id_user;
            $this->login=$login;
            $this->permission();
          }
          else{
            $this->delete_cookies();
          }
        }
        else{
          $this->delete_cookies();
        }
      }
      else{
        $this->delete_cookies();
      }
    }
    else{
      $this->delete_cookies();
    }
  }
  else{
    $this->delete_cookies();
  }
}
```

В методе check_session() так же как и в методе check_remember_user() обрабатываются данные из cookie, а также дополнительно данные из сессии, после чего идёт сравнение хэша логина из cookie с логином из сессии (который хэшируется перед сравнением). Идёт поиск пользователя в БД по расшифрованному id пользователя из cookie, если какая-либо из проверок не прошла, то данные из cookie немедленно удаляются.

Полную реализацию данного алгоритма на php с примером его работы можно посмотреть по ссылке https://github.com/flightofdeath/secure_authentication.

III. Заключение

В итоге мы получаем безопасную аутентификацию, при которой злоумышленник может только перехватить логин (при аутентификации) и зашифрованный хэш пароля, который он не сможет расшифровать без приватного ключа. Если злоумышленник сможет его каким-либо образом расшифровать, то у него всё равно окажется не пароль в открытом виде, а только хэш от пароля с солью.

БИБЛИОГРАФИЯ

[1] Шапиро Л. Двухфакторная аутентификация в Службе Каталога Active Directory Domain Services [Электроныйй ресурс]. URL: http://itband.ru/2010/09/authentication-part1/

[2] Google Security Blog Moving towards a more secure web [Электронный ресурс]. URL: https://security.googleblog.com/2016/09/moving-towards-more-secure-web.html

[3] Wikipedia HTTPS [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/HTTPS

[4] Wikipedia Хеширование [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/%D0%A5%D0%B5%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5

[5] Wikipedia Соль (криптография) [Электронный ресурс]. URL: https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D0%BB%D1%8C_(%D0%BA%D1%80%D0%B8%D0%BF%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F)

Secure authentication without using HTTPS

Filimoshin V.Y., Davletkireeva L.Z.

Abstract - The purpose of this article is to present a secure authentication algorithm for web resources without using HTTPS. The main idea of the algorithm is to avoid transferring a password in open way. So the password is presented to the server hashed and encrypted. If someone manages to intercept and decrypt the password hash, he will receive only a salted password hash and won't be able to receive the initial password. Some implementation results of the algorithm written in PHP are described to demonstrate how to protect the password from being compromised. The article could be useful for web developers.

Keywords - secure authentication, web resource, algorithm, php