

Internet- ТЕХНОЛОГИИ

ЛЕКЦИИ №1-2
БЕЗОПАСНАЯ РАЗРАБОТКА
WEB-ПРИЛОЖЕНИЙ

Основы валидации и хранения данных

Содержание

- Валидация на клиентской (HTML5 и JavaScript) и на серверной стороне (PHP).
- Хранение и передача данных.
- Криптографические расширения PHP.
- Man-in-the-middle attack. OpenSSL.

HTML5 Валидация

Валидация – проверка введенных пользователем данных на соответствие заданным требованиям.

Валидация средствами браузера позволяет предотвратить передачу невалидных данных и отобразить встроенное сообщение об ошибке валидации.

`<form>`

`<p>Заполните форму (все поля обязательны)</p>`

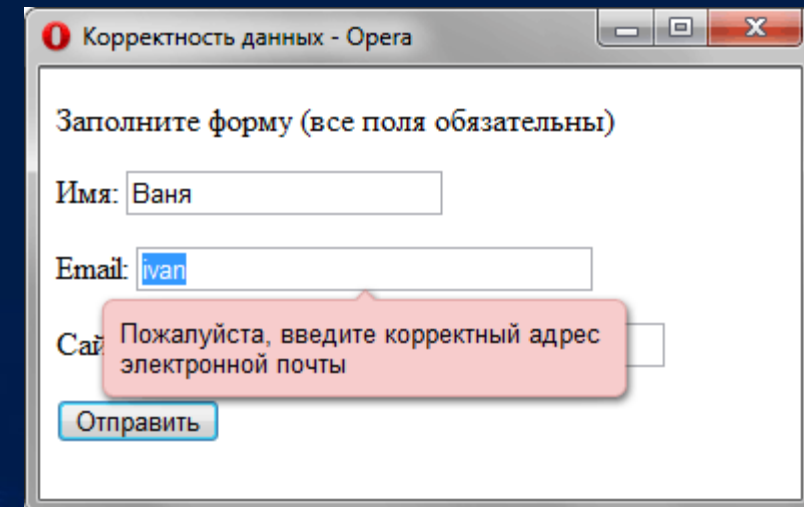
`<p>Имя: <input name="name" placeholder="Алла" title="Login is required!"
pattern="[a-zA-Z]{6}" required></p>`

`<p>Email: <input type="email" name="email" required></p>`

`<p>Сайт: <input type="url" name="site" required></p>`

`<p><input type="submit" value="Отправить"></p>`

`</form>`



Некоторые атрибуты input в HTML5 формах

- autofocus делает поле активным после загрузки страницы (*может использоваться со всеми типами input*).
- multiple указывает, что данное поле может принимать несколько значений одновременно, через запятую (*только для email и file*).
- placeholder отображает текст-подсказку в поле (*может использоваться с input следующих типов: text, search, url, tel, email и password*).
- **required** указывает, что данное поле должно быть обязательно заполнено перед отправкой.
- **pattern** задает регулярное выражение, с которым должно быть сверено значение элемента <input> перед отправкой.

```
<input type="email" id="email" name="email" multiple  
placeholder="name@domain.ru" required/>
```



The image shows a web form with a label "E-mail:" in a dark grey box. To the right is a light blue input field with a green border. Inside the field, the text "aaaaaa@www.df, qwerty@wer.cd|" is visible, demonstrating the use of the 'multiple' attribute. A green checkmark icon is located at the end of the input field, indicating that the field is valid and required.

HTML5 Валидация. Пример

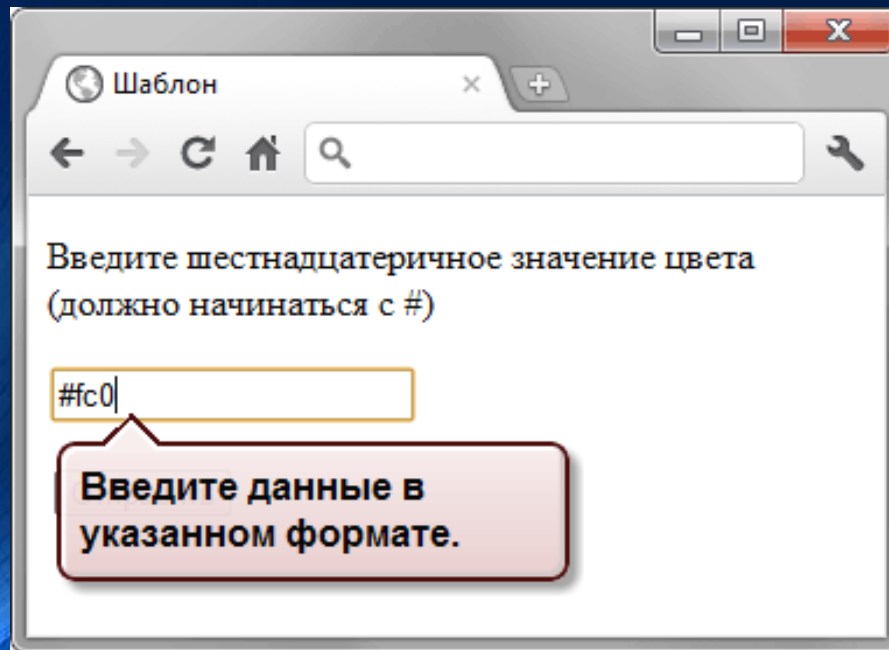
<form>

<p>Введите шестнадцатеричное значение цвета
(должно начинаться с #)</p>

<p><input name="digit" required pattern="#[0-9A-Fa-f]{6}"></p>

<p><input type="submit" value="Отправить"></p>

</form>



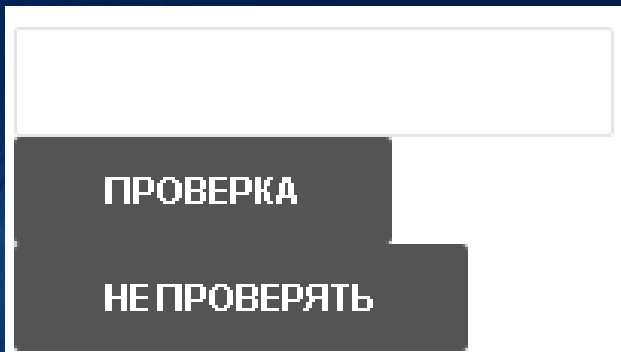
Отключение HTML5 валидации

- 1 способ `<form method="post" id="form" name="form" novalidate>`
- 2 способ: вариант отправки данных без проверки

```
<form><input type="text" required="" />
```

```
<input type="submit" value="Проверка" />
```

```
<input type="submit" formnovalidate value="Не проверять" /></form>
```

A screenshot of a web form. It features a white rectangular input field at the top. Below the input field are two dark gray rectangular buttons. The top button is labeled 'ПРОВЕРКА' in white uppercase letters. The bottom button is labeled 'НЕ ПРОВЕРЯТЬ' in white uppercase letters. The buttons are slightly offset to the left.

- 3 способ

```
<input type="email" id="email" name="email" multiple  
placeholder="valera@domain.ru" size="10" formnovalidate />
```

Псевдоклассы в CSS3, полезные при валидации

- **:focus** — когда вы используете какой-то объект на страницы, то на нем устанавливается фокус (в случае с текстовым полем это постановка курсора в это поле);
- **:not()** — дает ограничение на применение стилей по селектору (то есть селектор `.red-block:not(div)` применит указанный стиль ко всем элементам с классом `.red-block`, только есть этот элемент не `<div>`);
- **:enabled** — выбирает активные `input`;
- **:disabled** — выбирает неактивные `input`;
- **:valid** — стиль для корректного `input` (когда указана `data type` в HTML 5);
- **:invalid** — когда `input` невалиден;
- **:required** — все обязательные поля.

Стилизация форм HTML5. Пример

```
.form1 input:not([type="submit"]):invalid {  
  background: #ff8080;  
  border: 1px solid #800000;  
  box-shadow: 0 0 5px #d45252;  
}
```

```
.form1 input:not([type="submit"]):valid {  
  background: #80ff00;  
  background-color: #80ffff;  
  box-shadow: 0 0 5px #5cd053;  
}
```

Validator Input *Denotes Required Field

Info about you

Login: ✓

Password:* ✓

Confirm password:* ⚠

Contact from you

Minimum 6 characters Your password meets our requirements, thank you.

DOM API для валидации. *willValidate* и *validity*

- свойство **willValidate** true, если узел (элемент управления) должен будет проходить валидацию:

```
<div id="one"></div>
<input id="two" type="text" />
<input id="three" type="text" disabled="disabled" />
<script>
  document.getElementById('one').willValidate; //undefined
  document.getElementById('two').willValidate; //true
  document.getElementById('three').willValidate; //false
</script>
```

- **validity** – объект *ValidityState*, имеющий ряд свойств:
 - **patternMismatch** true, если value не соответствует регулярному выражению

```
<input id="foo" type="text" pattern="[0-9]{4}" value="1234" />
<input id="bar" type="text" pattern="[0-9]{4}" value="ABCD" />
<script>
  document.getElementById('foo').validity.patternMismatch; //false
  document.getElementById('bar').validity.patternMismatch; //true
</script>
```

DOM API для валидации. Свойства объекта *validity*

- **rangeOverflow** (или **rangeUnderflow**) true, если value больше (или меньше) чем max

```
<input id="foo" type="number" min="2" max="3" value="2" />
<input id="bar1" type="number" min="2" max="3" value="4" />
<input id="bar2" type="number" min="2" max="3" value="1" />
<script>
  document.getElementById('foo').validity.rangeOverflow; //false
  document.getElementById('foo').validity.rangeUnderflow; //false
  document.getElementById('bar1').validity.rangeOverflow; //true
  document.getElementById('bar2').validity.rangeUnderflow; //true
</script>
```

- **stepMismatch** true, если значение свойства value не кратно step:

```
<input id="foo" type="number" step="2" value="4" />
<input id="bar" type="number" step="2" value="3" />
<script>
  document.getElementById('foo').validity.stepMismatch; //false
  document.getElementById('bar').validity.stepMismatch; //true
</script>
```

DOM API для валидации. Свойства объекта validity

- **typeMismatch** true, если value не соответствует атрибуту type
- **valueMissing** true, если элемент имеет атрибут require, но не имеет значение в value
- **customError**: true, если определено пользовательское сообщение об ошибке (был установлен setCustomValidity())
- **tooLong** true, если количество введенных символов больше, чем maxlength.
- **valid** true, если элемент валиден по всем параметрам (если все действия условий, перечисленных выше, являются false).

DOM API для валидации. Методы и свойства

- **validationMessage** – сообщение валидации. Можно посмотреть так `document.getElementById('foo').validationMessage;`

- **checkValidity()** – проверка валидации всей формы `document.getElementById('form-1').checkValidity();`

- **setCustomValidity()** – установить пользовательское сообщение об ошибке.

```
<input type="password" id="pass1" name="pass1" pattern="\w{6,}" required autofocus  
onchange="this.setCustomValidity(this.validity.patternMismatch ? 'Password must contain  
at least 6 characters' : '');"/>
```

Password:*	<input type="password" value="....."/>
Confirm password:*	<input type="password"/>

Password must contain at least 6 characters

DOM API для валидации. События

- **onblur** – потеря фокуса.
- **onchange** – изменение значения элемента формы.
- **onfocus** – получение фокуса
- **onload** – документ загружен.
- **onmouseover** – курсор наводится на элемент.
- **onreset** – форма очищена.
- **onselect** – выделен текст в поле формы.
- **onsubmit** – форма отправлена.
- **onunload** – закрытие окна.

Работа с событиями формы Javascript

- **Использование атрибута HTML**

```
<input type="password" id="pass1" name="pass1" pattern="\w{6,}" required  
autofocus onchange="this.setCustomValidity (this.validity.patternMismatch ?  
'Password must contain at least 6 characters' : ''); "/>
```

- **Назначение обработчиков по стандарту w3c** (позволяет добавлять несколько обработчиков на одно событие одного элемента)

Назначение обработчика

```
element.addEventListener( event, handler, phase);
```

Удаление обработчика

```
element.removeEventListener( event, handler, phase);
```

```
document.getElementById("myBtn").addEventListener("click", function(){  
    document.getElementById("demo").innerHTML = "Hello World";  
});
```

Javascript. Работа с рег. выражениями

- Объект String:

- search
- match
- replace
- split

```
function checkMasterCard(txtValue) {  
    var pattern = /^5[1-5][0-9]{14}$/i;  
    if(txtValue.search(pattern) == -1) {  
        alert('Ошибка в поле "CreditMasterCard");  
    }  
}
```

- Объект Regular Expression:

- exec
- test
- compile

```
function checkLogin(txtValue) {  
    var pattern = /^[a-z][a-zo-ya]{6,}$/;  
    if(!pattern.test(txtValue)) {  
        alert('Ошибка в поле "Login");  
    }  
}
```

Javascript. Модификаторы регулярных выражений

- **i** (ignore case). Не различать строчные и прописные буквы.
- **g** (global search). Глобальный поиск всех вхождений образца.

```
var pattern = /\w+/g;
```

- **m** (multiline). Многострочный ввод должен рассматриваться как несколько строк. Влияет на учет символов **^** и **\$**.
- Любые комбинации этих трех опций, например **ig** или **gim**.

Javascript. Методы объекта String

Метод `string.search(searchvalue)`

```
var str= document.forms[o].searching.value;  
var pattern = /hun/;  
var result = str.search(pattern);  
if(result == -1) {  
    alert("Search not working");  
}  
else {  
    alert("Search working" +" "+str.search(pattern));  
}
```

*Search:	<input type="text" value="aaahun"/>
*Replace:	<input type="text"/>
*Match:	<input type="text"/>
*Split:	<input type="text"/>
*Test:	<input type="text"/>
*Exec:	<input type="text"/>
*Compile:	<input type="text"/>
<input type="button" value="Отправить"/>	

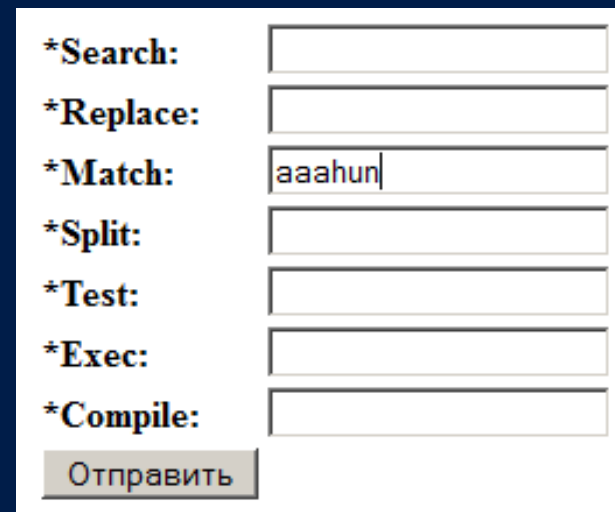
Search working 3

Javascript. Методы объекта String

Метод `string.match(regex)`

```
var str= document.forms[o].matching.value;  
var pattern = /hun/;  
var result = str.match(pattern);  
if(result == null) {  
    alert("Match not working");  
}  
else {  
    alert("Match working" + " " + str.match(pattern));  
}
```

Данная функция имеет возможность возвращать несколько вхождений только в случае использования глобального модификатора поиска g (пример на рис. снизу).



A screenshot of a web form with the following fields and labels:

- *Search:
- *Replace:
- *Match:
- *Split:
- *Test:
- *Exec:
- *Compile:
- Отправить (Send)

Match working hun

Match working hun,hun

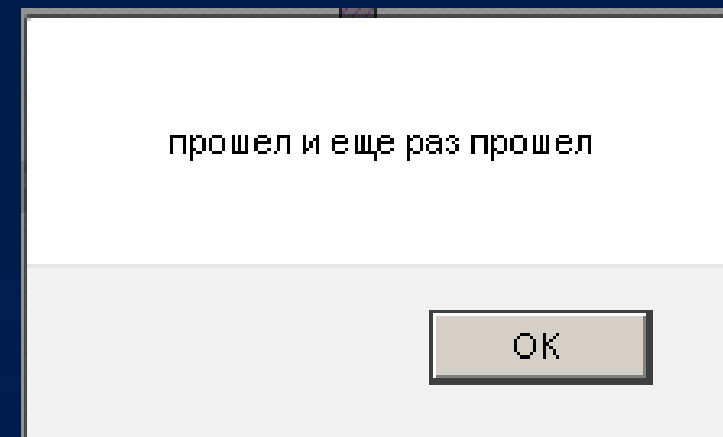
OK

Javascript. Методы объекта String

Метод *string.replace(regex, newSubStr|function)*

```
var str = "тест и еще раз тест";  
// var newstr=str.replace(new RegExp("тест",'g'),"прошел");  
var newstr=str.replace(/тест/g,"прошел");  
// "прошел и еще раз прошел"  
alert(newstr);
```

Этот метод не меняет вызывающую строку, а возвращает новую, после замены. Использование с флагом "g" позволяет заменить сразу несколько вхождений, как показано на рисунке.



Javascript. Методы объекта String

Метод `string.split(separator, limit)`

```
var str = document.forms[o].spliting.value;  
// разделяем на части с помощью запятых  
var arr = str.split(',');// сеператор  
document.forms[o].spliting.value = arr[o] + "||" + arr[1]+ "||" + arr[2];
```

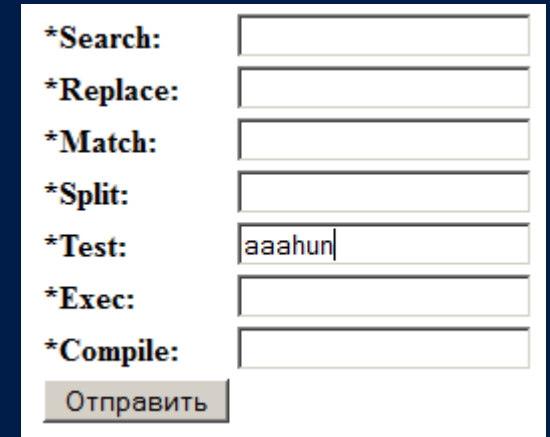
*Search:	<input type="text"/>
*Replace:	<input type="text"/>
*Match:	<input type="text"/>
*Split:	<input type="text" value="a,a,a"/>
*Test:	<input type="text"/>
*Exec:	<input type="text"/>
*Compile:	<input type="text"/>
<input type="button" value="Отправить"/>	

*Search:	<input type="text"/>
*Replace:	<input type="text"/>
*Match:	<input type="text"/>
*Split:	<input type="text" value="a a a"/>
*Test:	<input type="text"/>
*Exec:	<input type="text"/>
*Compile:	<input type="text"/>
<input type="button" value="Отправить"/>	

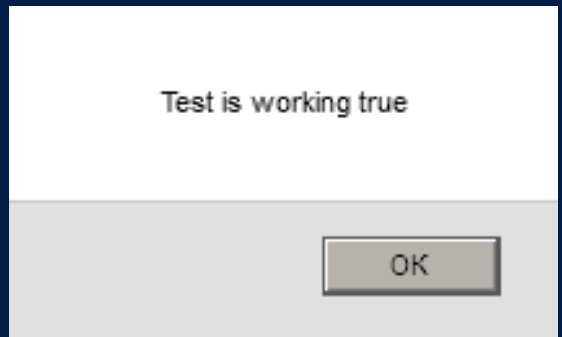
Javascript. Использование объекта Regular Expression

Булевый метод `RegExpObject.test(string)` проверяет строку на вхождение.

```
var str = document.forms[o].testing.value;  
var pattern = new RegExp("hun");  
// ищем по шаблону в строке  
if(pattern.test(str)) {  
    alert("Test is working" + " " + pattern .test(str));  
}  
else {  
    alert("Not found!");  
}
```



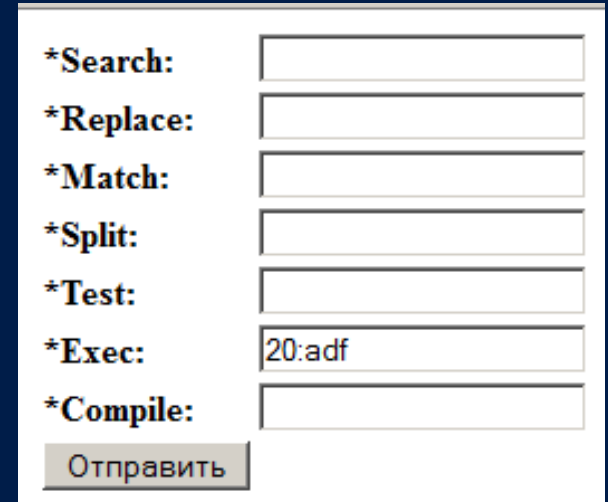
*Search:
*Replace:
*Match:
*Split:
*Test:
*Exec:
*Compile:



Javascript. Использование объекта Regular Expression

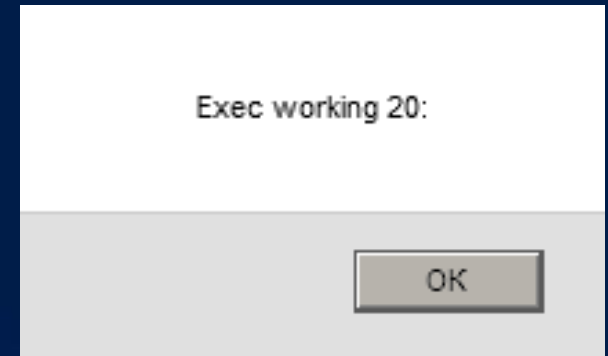
Метод `RegExpObject.exec(string)` выполняет поиск сопоставления регулярного выражения в указанной строке.

```
var place = document.forms[o].execing.value;  
var obj = /20:/; // указываем шаблон  
// ищем вхождение, помещаем результат в массив  
result = obj.exec(place);  
if(result == null) {  
    alert ("Not found!");  
}  
else {  
    alert("Exec working"+" "+ result);  
}
```



A screenshot of a web form with the following fields and labels:

- *Search:
- *Replace:
- *Match:
- *Split:
- *Test:
- *Exec:
- *Compile:
- Отправить



A screenshot of an alert dialog box with the text "Exec working 20:" and an "OK" button.

Свойства объекта *RegExp*

\$1 - \$9 – подстрока №1 - №9 *RegExp*.\$n

lastIndex – Индекс последнего совпадения. Это свойство устанавливается, только если регулярное выражение использует опцию g.

lastMatch – вся совпавшая строка

leftContext – строка перед совпадением

rightContext – строка после совпадения

lastParen – последняя совпавшая группа

Свойства объекта RegExp. Пример

```
var re = new RegExp("d(b+)(d)","ig");  
var str = "cdbBdbbsbdbdz";  
var arr = re.exec(str);
```

// output.

```
var s = ""  
s += "input: " + RegExp.input + "<br />";  
s += "$1: " + RegExp.$1 + "<br />";  
s += "$2: " + RegExp.$2 + "<br />";  
s += "$3: " + RegExp.$3 + "<br />";  
s += "lastMatch: " + RegExp.lastMatch + "<br />";  
s += "leftContext: " + RegExp.leftContext + "<br />";  
s += "rightContext: " + RegExp.rightContext + "<br />";  
s += "lastParen: " + RegExp.lastParen + "<br />";  
document.write(s);
```

```
input: cdbBdbbsbdbdz  
$1: bB  
$2: d  
$3:  
lastMatch: dbBd  
leftContext: c  
rightContext: bbsbdbdz  
lastParen: d
```


Валидация данных на клиентской стороне

Недостатки проверки на клиентской стороне:

- Сильная зависимость от типа браузера и настроек JavaScript
- Клиент может вообще отключить выполнение JavaScript
- Пользователь может намеренно отправлять некорректные данные с использованием специализированного ПО.

Выводы:

- Проверка на клиентской стороне необходима исключительно с целью **уменьшить нагрузку на сервер** при непреднамеренной отправке некорректных данных.
- Проверка на клиентской стороне не исключает необходимости проверки на стороне сервера.
- Проверка данных на стороне сервера **обязательна!**

Функции PHP для работы с регулярными выражениями

- PCRE (Perl-compatible regular expressions):
 - **preg_match** — поиск соответствий.
 - **preg_match_all** — поиск всех соответствий.
 - **preg_replace** — поиск и замена по регулярному выражению.
 - **preg_split** — разделение строки по регулярному выражению.
- POSIX-extended regular expressions (**устарели!**):
 - **ereg** — поиск соответствий.
 - **eregi** — поиск соответствий, игнорируя регистр.
 - **ereg_replace** — поиск и замена по рег. Выражению.
 - **eregi_replace** — поиск и замена по рег. выражению, игнорируя регистр.
 - **split** — разделение строки по регулярному выражению.
 - **spliti** — разделение строки по регулярному выражению, игнорируя регистр.

PCRE-функции. preg_match

```
int preg_match (  
    string pattern, // строка шаблона  
    string subject // строка поиска  
    [, array &matches // результат  
    [, int flags // флаг PREG_OFFSET_CAPTURE  
    [, int offset]]] ) // начальное смещение
```

```
if(!preg_match("/hun/", $log, $matches)) {  
    echo "<tr><td><u>Preg_match:</u></td><td>Тестовая строка, содержащая  
как минимум 7 символов букв и цифр, начинающихся с буквы латинского  
алфавита</td></tr>";  
}  
else {  
    echo "<tr><td rowspan=2><u>Preg_match:</u></td><td>Validation  
OK!</td></tr>";  
    echo "<tr><td>". print_r($matches) . "</td></tr>";  
}
```

Validation by PHP

<u>Preg_match:</u>	Validation OK!
	hun
<u>Ereg:</u>	Validation OK!
	tun
<u>Preg_match all:</u>	Validation OK!
	pun
	pun

Безопасное хранение данных в Web-приложениях

- Конфиденциальные регистрационные данные пользователя (логин, пароль, email, номера кредитных карточек), хранящиеся в базе данных
- Данные, хранящиеся в SESSION, COOKIES, WebStorage.

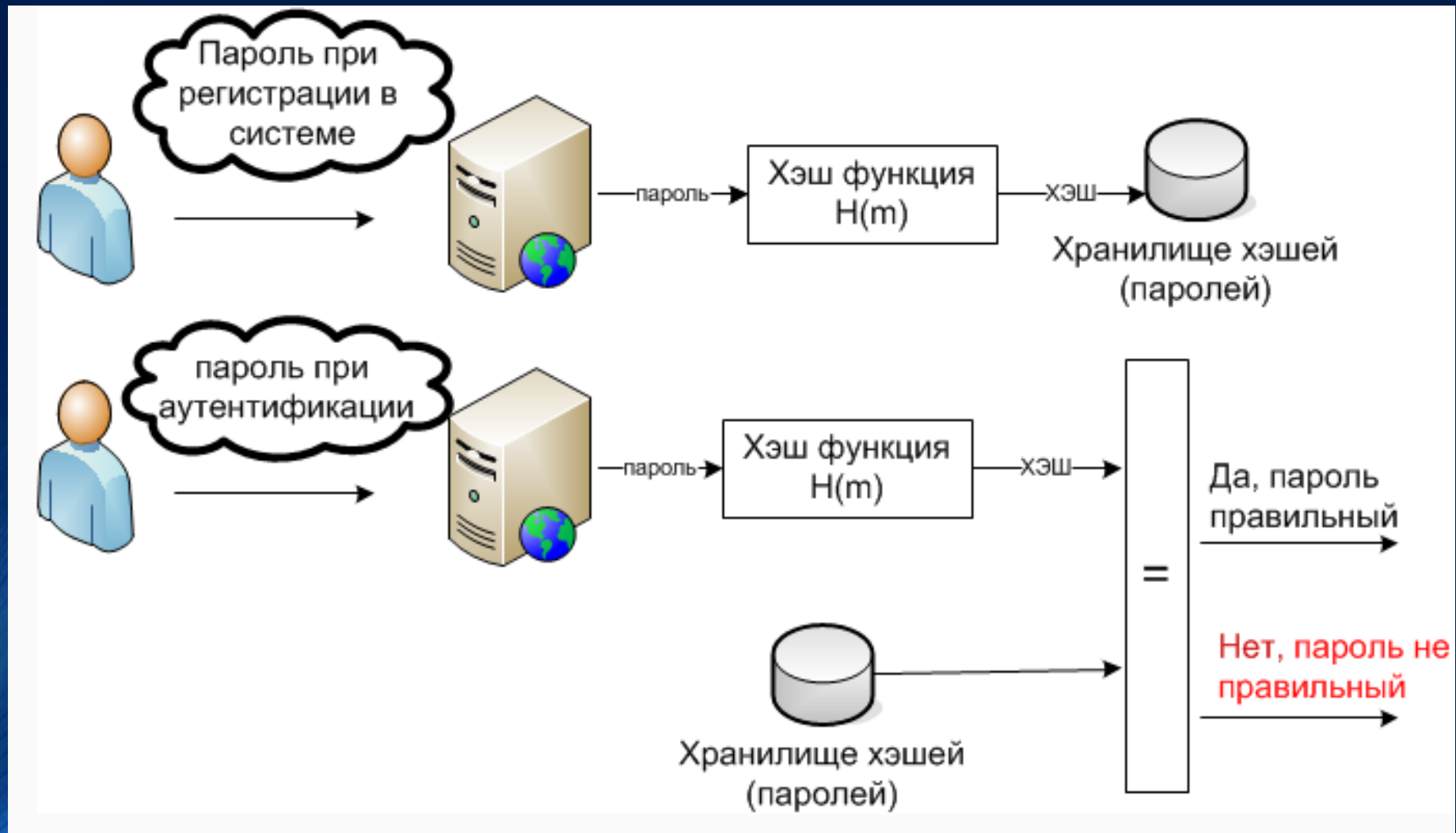
Шифрование – обратимое преобразование информации в целях сокрытия от неавторизованных лиц, с предоставлением авторизованным пользователям доступа к ней.

Примеры применения: шифрования файлов или потоков данных.

Хеширование – необратимое преобразование по детерминированному алгоритму входного массива данных произвольной длины в выходную строку фиксированной длины.

Примеры применения: контрольная сумма, CRC, хеширование паролей.

Хеширование. Примеры применения



Хеширование. Традиционные методы

```
$sha=sha1($email);  
echo "Sha1: 40-символьное шестнадцатеричное число ".$sha."<br>";  
$md5=md5($email);  
echo "MD5: 32-значное шестнадцатеричное число ".$md5."<br>";  
$crc32=crc32($email);  
echo "CRC32: ".$crc32."<br>"; printf(" CRC32: %u\n",$crc32);
```

ВЫВОД:

Sha1: 40-символьное шестнадцатеричное число

ed30750a981b20457eb05d8e265dodd428232a82

MD5: 32-значное шестнадцатеричное число

b235c544196b3fd731a33292def1f9e6

CRC32: -1795582690

CRC32: 2499384606

Хеширование. Коллизии

Коллизия хеш-функции — это равенство значений хеш-функции на двух различных блоках данных.

```
$i = 0;
while (true) {
    if (crc32(base64_encode($i)) == 323322056) {
        echo base64_encode($i);
        exit;
    }
    $i++;
}
```

Пример:

```
echo crc32('supersecretpassword'); //на выходе 323322056
echo crc32('MTIxMjY5MTAwNg=='); //на выходе 323322056
```

Хеширование. Соль

Соль — строка данных, которая передаётся хеш-функции вместе с паролем и применяется для его удлинения пароля, чтобы увеличить сложность взлома. Используется для борьбы с восстановлением паролей за один проход полного перебора или с помощью предварительно построенных радужных таблиц.

```
$salt = 'sflprt49fhi2';  
$hash1 = md5($password); $hash2 = md5($hash1 . $salt);  
echo "MD5 Соль Пример№1 ".md5($email.$salt). "<br>";  
echo "MD5 Соль Пример№2 ". md5(md5($email).$salt) . "<br>";  
echo "MD5 Соль Пример№3 ". md5(md5($salt).md5($email)) . "<br>";
```

ВЫВОД:

```
MD5 Соль Пример№1 1fc877dc5d351e0a4146bbo2023ac67d  
MD5 Соль Пример№2 od2d5cc3121ca221d73ece2f673cefd4  
MD5 Соль Пример№3 f314a60a9d9fa74f9a3c861ege03eaa1
```


Хеширование. Скорость выполнения

```
function myhash($password, $unique_salt)
{
    $salt = "f#@V)Hu^%Hgfds";
    $hash = sha1($unique_salt . $password);
    /* увеличиваем время исполнения функции в 1000 раз, заставив
    функцию сперва выполниться 1000 раз, и только затем вернуть
    результат*/
    for ($i = 0; $i < 1000; $i++) {
        $hash = sha1($hash);
    }
    return $hash;
}
```

Алгоритм хеширования не должен позволять его распараллелить.

Решения для безопасных передачи и хранения данных

PHP поддерживает большое количество алгоритмов шифрования и предоставляет расширения и библиотеки с готовыми методами, не требуя профессиональной подготовки в области криптографии и детального изучения алгоритмов шифрования, для решения такого рода задач.

Криптографические расширения PHP:

- Hash
- Password Hashing
- Crack
- MHash
- Mcrypt
- OpenSSL
- Библиотека phpseclib

Пакеты репозитория PEAR:

- PEAR_Encryption
- HTML_Crypt

Расширение hash. Список алгоритмов

array **hash_algos** (void)

Array ([0] => md2

[1] => md4

[2] => md5

[3] => sha1

[4] => sha224

[5] => sha256

[6] => sha384

[7] => sha512

[8] => ripemd128

[9] => ripemd160

[10] => ripemd256

[11] => ripemd320

[12] => whirlpool

[13] => tiger128,3

[14] => tiger160,3

[15] => tiger192,3

[16] => tiger128,4

[17] => tiger160,4

[18] => tiger192,4

[19] => snefru

[20] => snefru256

[21] => gost

[22] => adler32

[23] => crc32

[24] => crc32b

[25] => salsa10

[26] => salsa20

[27] => haval128,3

[28] => haval160,3

[29] => haval192,3

[30] => haval224,3

[31] => haval256,3

[32] => haval128,4

[33] => haval160,4

[34] => haval192,4

[35] => haval224,4

[36] => haval256,4

[37] => haval128,5

[38] => haval160,5

[39] => haval192,5

[40] => haval224,5

[41] => haval256,5)

Расширение hash. Прямой способ хеширования

```
$email=$_POST["email"];  
$key=$_POST["security"];  
$hash=hash('sha256', $email, FALSE);  
echo "Hash".$hash."<br>";  
$hmac=hash_hmac('sha256', $email, $key, FALSE);  
echo " HMAC с ключом". $hmac."<br>";
```

Вывод:

Hash 2d8e0c110b0b01569302bada31606c8e

HMAC с ключом ab7d28bfe05777c2abc368110d99322a

Расширение *hash*. Инкрементальное хеширование

resource **hash_init** (string \$algo [, int \$options = 0 [, string \$key = **NULL**]])

bool **hash_update** (resource \$context , string \$data)

bool **hash_update_file** (resource \$context , string \$filename
[, resource \$context = **NULL**])

int **hash_update_stream** (resource \$context , resource \$handle
[, int \$length = -1])

string **hash_final** (resource \$context [, bool \$raw_output = false])



Расширение hash. Инкрементальное хеширование

```
$ctx=hash_init('sha256', HASH_HMAC, $key);  
hash_update ($ctx, $email);  
echo "Hash_update с ключом ".hash_final($ctx, FALSE)."<br>"
```

Вывод:

Hash_update с ключом

16672a4c563eb51a143efdc2dfc5346f33db4f4d8a40165daa7caf38a1564ea

Password Hashing API

string **password_hash** (string \$password , integer \$algo [, array \$options])

array **password_get_info** (string \$hash)

boolean **password_verify** (string \$password , string \$hash)

boolean **password_needs_rehash** (string \$hash , string \$algo [, string \$options])

```
$options = [ 'cost' => 12,];
```

```
$hash = password_hash($pass, PASSWORD_BCRYPT, $options)."\n";
```

```
echo $hash; \\$2y$07$BCryptRequires22Chrcte/VlQHopiJtjXl.ot1XkA8pw9dMXTpOq'
```

```
if (password_verify($pass, $hash)) {
```

```
    echo 'Password is valid!';
```

```
} else {
```

```
    echo 'Invalid password.';}
```

Crack

resource crack_opendict (string \$dictionary)

bool crack_check (resource \$dictionary , string \$password)

bool crack_closedict ([resource \$dictionary])

string crack_getlastmessage (void)

Возвращаемое сообщение соответствует одному из следующих:

- it's WAY too short
- it is too short
- it does not contain enough DIFFERENT characters
- it is all whitespace
- it is too simplistic/systematic
- it is based on a dictionary word
- it is based on a (reversed) dictionary word
- strong password

Crack. Пример использования

```
<?php
// Открываем словарь CrackLib
$dictionary = crack_opendict('/usr/local/lib/pw_dict')
    or die('Unable to open CrackLib dictionary');

// Выполняем проверку пароля
$check = crack_check($dictionary, 'gx9A2sox');
$diag = crack_getlastmessage();
echo $diag; // 'strong password'

// Закрываем словарь
crack_closedict($dictionary);
?>
```

Библиотека Mhash

Библиотека Mhash поддерживает широкий спектр алгоритмов хеширования, таких как MD5, SHA1, ГОСТ, и многие другие.

```
string mhash_keygen_s2k ( int $hash , string $password , string $salt , int $bytes )  
string mhash ( int $hash , string $data [, string $key ] )
```

MHASH_ADLER32	MHASH_HAVAL128
MHASH_CRC32	MHASH_HAVAL160
MHASH_CRC32B	MHASH_HAVAL192
MHASH_GOST	MHASH_HAVAL224
MHASH_MD2	MHASH_HAVAL256
MHASH_MD4	MHASH_RIPEMD128
MHASH_MD5	MHASH_RIPEMD256
	MHASH_RIPEMD320

MHASH_SNEFRU256
MHASH_SNEFRU128
MHASH_TIGER
MHASH_TIGER128
MHASH_TIGER160
MHASH_WHIRLPOOL

MHASH_SHA1
MHASH_SHA192
MHASH_SHA224
MHASH_SHA256
MHASH_SHA384
MHASH_SHA512

Библиотека Mhash. Пример

```
$mhash_keygen=mhash_keygen_s2k(MHASH_GOST,$email,$salt, 20);  
echo "Mhash_keygen ".$mhash_keygen."<br>";  
echo "Mhash_keygen_bin2hex ".bin2hex($mhash_keygen)."<br>";
```

```
$mhash=mhash(MHASH_HAVAL128,$email, $mhash_keygen);  
echo "Mhash с ключом ".bin2hex( $mhash)."<br>";
```

//Возвращает наивысший доступный id хэша. Хэши нумеруются от 0 до этого hash id.

```
$nr = mhash_count();  
for ($i = 0; $i <= $nr; $i++) {  
echo sprintf("The blocksize of %s is %d\n",  
//Используется для получения имени специфицированного хэша  
mhash_get_hash_name($i),  
//Используется для получения размера блока специфицированного хэша hash.  
mhash_get_block_size($i));}
```


Библиотека Mhash

Вывод:

Mhash_keygen_bin2hex b3dcdd3770bba6973a68267c46aa6da734d6b03b

Mhash с ключом 8c2c35dc09b17cbd45306dd3b28ea8b5

The blocksize of CRC32 is 4 The blocksize of MD5 is 16 The blocksize of SHA1 is 20 The blocksize of HAVAL256 is 32 The blocksize of is 0 The blocksize of RIPEMD160 is 20 The blocksize of is 0 The blocksize of TIGER is 24 The blocksize of GOST is 32 The blocksize of CRC32B is 4 The blocksize of HAVAL224 is 28 The blocksize of HAVAL192 is 24 The blocksize of HAVAL160 is 20 The blocksize of HAVAL128 is 16 The blocksize of TIGER128 is 16 The blocksize of TIGER160 is 20 The blocksize of MD4 is 16 The blocksize of SHA256 is 32 The blocksize of ADLER32 is 4 The blocksize of SHA224 is 28 The blocksize of SHA512 is 64 The blocksize of SHA384 is 48 The blocksize of WHIRLPOOL is 64 The blocksize of RIPEMD128 is 16 The blocksize of RIPEMD256 is 32 The blocksize of RIPEMD320 is 40 The blocksize of is 0 The blocksize of SNEFRU256 is 32 The blocksize of MD2 is 16 The blocksize of FNV132 is 4 The blocksize of FNV1A32 is 0 The blocksize of FNV164 is 8 The blocksize of FNV1A64 is 0 The blocksize of JOAAT is 4

Библиотека Mcrypt. Список алгоритмов

```
array mcrypt_list_algorithms ([ string $lib_dir = ini_get("mcrypt.algorithms_dir") ] )
```

cast-128	saferplus
cast-256	wake
gost	des
rijndael-128	tripledes
rijndael-192	xtea
rijndael-256	enigma
twofish	serpent
blowfish	arcfour
blowfish-compat	rc2
loki97	

```
$algorithms = mcrypt_list_algorithms(  
"/usr/local/lib/libmcrypt");
```

```
foreach ($algorithms as $cipher) {  
echo "$cipher<br />\n"; }  
}
```

Mcrypt. Описание базовых методов

1. Инициализация алгоритма, используемого режима, вектора

resource **mcrypt_module_open** (string \$algorithm , string \$algorithm_directory ,
string \$mode , string \$mode_directory)

string **mcrypt_create_iv** (int \$size [, int \$source = MCRYPT_DEV_RANDOM])

int **mcrypt_generic_init** (resource \$td , string \$key , string \$iv)

2. Непосредственно шифрование и дешифрование

string **mcrypt_generic** (resource \$td , string \$data)

string **mdcrypt_generic** (resource \$td , string \$data)

3. Заккрытие модуля и деинициализация

bool **mcrypt_generic_deinit** (resource \$td)

bool **mcrypt_module_close** (resource \$td)

Mcrypt. Пример

```
$td = mcrypt_module_open('tripledes', '', 'cfb', '');  
$iv = mcrypt_create_iv (mcrypt_enc_get_iv_size($td), MCRYPT_RAND);  
mcrypt_generic_init($td, $email, $iv);
```

```
$encrypted_data = mcrypt_generic($td, $email);  
echo bin2hex($encrypted_data). "<br>";
```

```
mcrypt_generic_init($td, $email, $iv);  
$p_t = mdecrypt_generic($td, $encrypted_data);
```

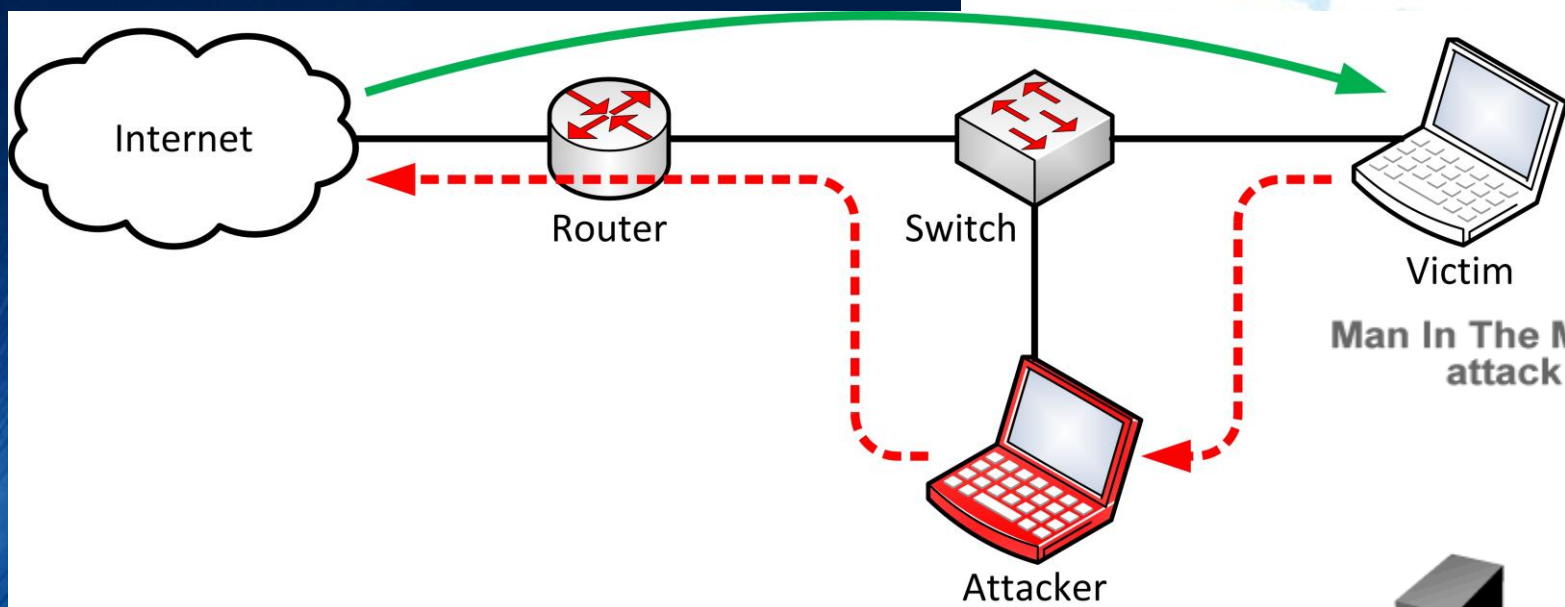
```
mcrypt_generic_deinit($td);  
mcrypt_module_close($td);
```

Вывод:

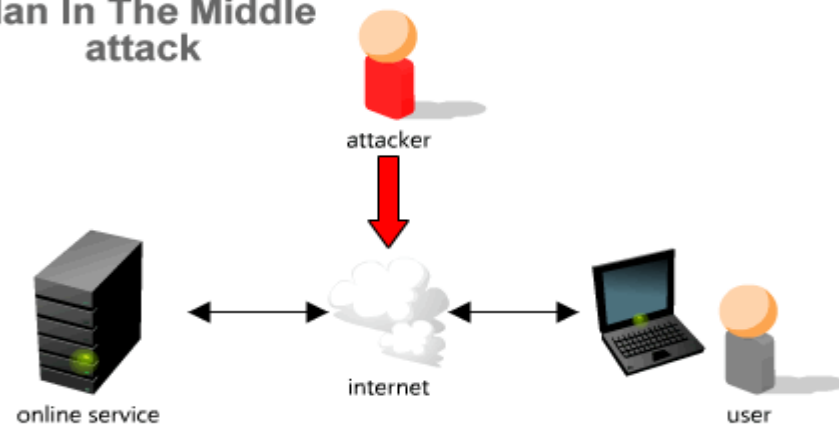
b16579f82b678ebff2759bfo801d

Man-in-the-middle attack

Traceroute Path 1: from Guadalajara, Mexico to Washington, D.C. via *Belarus*



Man In The Middle attack



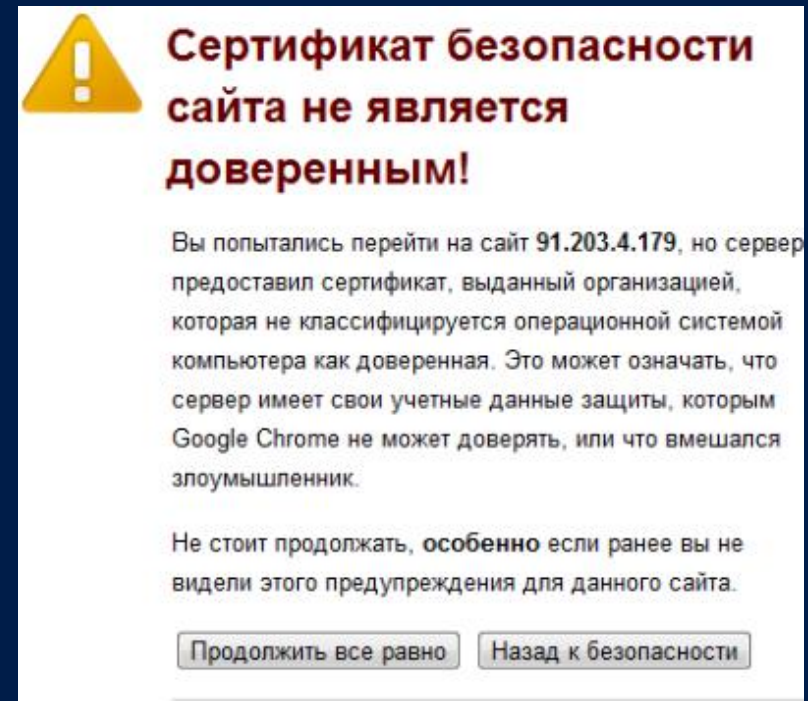
Man-in-the-middle attack

Сценарии атаки:

- Обмен открытыми ключами
- Внедрение вредоносного кода
- Downgrade Attack (SSH V1 вместо SSH V2)

Обнаружение MITM-атаки:

- IP-адрес сервера
- DNS-сервер
- X.509-сертификат сервера (подписан ли сертификат центром сертификации, менялся ли сертификат недавно).



OpenSSL

OpenSSL — криптографический пакет с открытым исходным кодом. В пакете реализованы различные криптографические алгоритмы, форматы и протоколы, что позволяет использовать OpenSSL для широкого круга прикладных задач.

```
resource openssl_pkey_get_private ( mixed $key [, string $passphrase = "" ] )  
bool openssl_private_encrypt ( string $data , string &$encrypted , mixed $key [, int  
$padding = OPENSSL_PKCS1_PADDING ] )  
bool openssl_private_decrypt ( string $data , string &$decrypted , mixed $key [, int  
$padding = OPENSSL_PKCS1_PADDING ] )
```

```
resource openssl_pkey_get_public ( mixed $certificate )  
bool openssl_public_encrypt ( string $data , string &$encrypted , mixed $key [, int  
$padding = OPENSSL_PKCS1_PADDING ] )  
bool openssl_public_decrypt ( string $data , string &$decrypted , mixed $key [, int  
$padding = OPENSSL_PKCS1_PADDING ] )
```

OpenSSL. Пример

```
// Absolute location of private key and certificate
$key_1 = "C:\privatekey_18G1E.pem";
// $priv_key можно получить после открытия файла privatekey_18G1E.pem
$cert = "C:\certificate_18G1E.pem";
// $pub_key можно получить после открытия файла certificate_18G1E.pem

$res_private = openssl_get_privatekey($priv_key,$passphrase);
$res_encrypt=openssl_private_encrypt($credit, $crypttext, $res_private);
echo "Result of encryption is ".$res_encrypt;

$res_public = openssl_get_publickey($pub_key);
openssl_public_decrypt($crypttext,$newmessage,$res_public);
echo "Text decrypted: $newmessage";
```


Вопросы

- Недостатки валидации на стороне клиента.
- Возможности валидации данных стандарта HTML5.
- Как отключить валидацию формы?
- Способы назначения сообщения об ошибке для HTML5.
- Какой объект JavaScript используется для проверки соответствия введенных данных?
- Какие функции PHP используются для проверки соответствия введенных данных?
- В чем состоит отличие процесса шифрования от хеширования?
- В чем заключается проблема коллизий?
- Что представляет собой криптографическая соль?
- В чем отличие прямого и инкрементального способа хеширования?
- Какие вы знаете криптографические расширения и библиотеки?
- В чем заключается MITM-атака?